EZ Test
Language Reference Manual

American Systems
P.O. Box 93747
Southlake, TX 76092
Phone: (817) 485 6547
Fax: (817) 485 2193

Table of Contents

Chapter 1. Language Overview 13

About EZ Test Scripts 13

Script Language Overview 13

Access to External Information 14

Asynchronous Decision Making 14

External Actions 15

Chapter 2. Script Structure *15*

Script Elements 15

Structure 16

Commands, Functions, and Variables 16

Comments 18

Public, Private, and Local Variables 18

Constants 19

Reserved Words 19

Strings 19

String Constants 19

'C' Escape Sequences 20

Keystrokes 20

String Variables 20

String Assignment 21

String Expressions 21

Boolean String Expressions 21

String System Variables 22

String System Functions 22

Other String Functions 23

Numbers 23

Numeric Constants 23

Numeric Variables 23

Numeric Assignment 24

Numeric Expressions 24

Boolean Numeric Expressions 24

Numeric System Variables 25

Numeric System Functions 26
Other Numeric Functions 26
String/Number Type Conversion 26
Arrays 27
Single-Key Arrays 28
Multi-Key Arrays 28
Events 29
Test Data 30
SQL Commands 31
Chapter 3. Script Command Groups 34
Checks 34
Clocks 34
Date/Time 34
Dialog Control 35
File Access 35
Language 36
Menu Control 36
Menu Information 36
Miscellaneous 36
Mouse Control 37
Mouse Information 37
Number Manipulation 37
Performance Monitoring 37
Program Flow 37
SQL Commands 38
String Manipulation 38
Synchronization 39
System Information 40
Testdata Handling 40
Window Control 41
Window Information 41
Chapter 4. *43*
Script Commands 43
Abbrev( ) 43
Abs( ) 44

ActiveName( ) 44
ActiveWindow( ) 44
AnchorSelect( ) 45
AppActivate( ) 46
ArrayPush( ) 46
Arrays 47
ArraySize( ) 48
Asc( ) 49
Assignment 49
Attach( ) 50
AttachAtPoint( ) 51
AttachMouseX( ) 52
AttachMouseY( ) 52
AttachName( ) 52
AttachWindow( ) 53
Beep( ) 53
BitMapSelect( ) 54
Boolean Expressions 54
Break 56
BrowserToolbarCtrl( ) 56
Button( ) 57
ButtonDefault( ) 58
CalendarCtrl( ) 58
CalendarRange( ) 59
CalendarToday( ) 59
Cancel( ) 60
Capture( ) 60
CaptureBox( ) 61
CaretPosX( ) 62
CaretPosY( ) 62
Cesc( ) 62
Chain( ) 63
ChDir( ) 64
Check( ) 64
CheckBox( ) 65

CheckExists( ) 65
Chr( ) 66
ClipBoard( ) 66
Clng( ) 66
Clock( ) 67
ClockReset( ) 67
ClockStart( ) 68
ClockStop( ) 68
Close( ) 68
CloseCom( ) 69
CmdLine( ) 69
ComboBox( ) 70
ComboText( ) 71
Compare( ) 71
Const 72
Continue 72
ControlFind( ) 72
Control Labels 73
ConvertCurrency( ) 74
CopyFile( ) 75
Create( ) 75
CreateDate( ) 76
CtrlChecked( ) 78
CtrlEnabled( ) 79
CtrlFocus( ) 79
CtrlLabel( ) 80
CtrlPressed( ) 80
CtrlSelText( ) 80
CtrlText( ) 81
CtrlType( ) 81
CurDir( ) 83
CurTime( ) 83
DataCtrl( ) 84
DataType( ) 84
DataWindow( ) 85

Date( ) 85
DateTimeCtrl( ) 86
DateTimeMode( ) 86
DateTime( ) 87
DateVal( ) 87
Day( ) 88
dbAddNew( ) 88
dbBOF( ) 89
dbClose( ) 89
dbConnect( ) 90
dbDisconnect( ) 91
dbEdit( ) 91
dbEOF( ) 92
dbExecute( ) 92
dbGetField( ) 93
dbMove( ) 93
dbMoveFirst( ) 94
dbMoveLast( ) 94
dbMoveNext( ) 94
dbMovePrev( ) 95
dbRecordCount( ) 95
dbSelect( ) 96
dbSetField( ) 97
dbUpdate( ) 97
Delete ArrayName[Element] 98
DeleteFile( ) 99
DeleteStr( ) 99
DestroyEvent( ) 100
Dialog( ) 101
Dir( ) 103
DLLFunc 104
Do...Loop While 105
EditClick( ) 106
EditLine( ) 107
EditLineCount( ) 107

EditText( ) 108
Err 109
ErrFile 109
ErrFunc 110
ErrLine 110
ErrMsg 110
Error 111
Event( ) 111
Exec( ) 112
Exit( ) 113
ExitWindows( ) 113
Fatal( ) 114
FileExists( ) 114
FilePos( ) 115
FileStatus( ) 115
FileTime( ) 116
FillArray( ) 116
FindChar( ) 117
FindStr( ) 118
Fix( ) 119
Focus( ) 119
FocusName( ) 119
FocusWindow( ) 120
For…Next 120
FormatDate( ) 121
Function…End Function 122
Get4GLInfo( ) 125
GetEnv( ) 125
GetProperty ( ) 125
GetReadyState 126
HeaderCtrl( ) 127
Hotkey 128
HotspotCtrl( ) 128
Hours( ) 129
If…Else…Endif 129

IgnoreCase( ) 130
ImageSelect( ) 131
Include 132
InsertStr( ) 133
InStr( ) 133
IPControl( ) 134
IsFile( ) 134
IsMenu( ) 135
IsRunning( ) 136
IsWindow( ) 137
JulianDate( ) 137
JulianDateVal( ) 138
LabelCtrl( ) 138
LastKey( ) 139
LastKeyStr( ) 139
Left( ) 140
Length( ) 140
LinkCheck( ) 140
ListBox( ) 141
ListCount( ) 142
ListFindItem( ) 143
ListFocus( ) 143
ListItem( ) 144
ListTopIndex( ) 145
ListViewCtrl( ) 146
Log.Checks 147
Log.Commands 147
LogComment( ) 147
Log.Comments 148
Log.DLLCalls 148
Log.Enable 148
Log.Name 149
LogOff( ) 149
LogOn( ) 150
LogOpen( ) 150

Log.System 150
LowerCase( ) 151
LtrimStr( ) 151
MakeCheck( ) 151
MakeDir( ) 153
MakeEvent( ) 153
Max( ) 159
Maximize( ) 159
MenuCount( ) 160
MenuCtrl( ) 161
MenuFindItem( ) 161
MenuItem( ) 162
MenuSelect( ) 163
MessageBox( ) 164
Mid( ) 165
Min( ) 165
Minimize( ) 165
Mins( ) 166
Month( ) 167
MouseClick( ) 167
MouseCursor( ) 168
MouseHover( ) 169
MouseMove( ) 170
MouseWindow( ) 170
MouseX( ) 170
MouseY( ) 171
Move( ) 171
NCMouseClick( ) 172
NotifyEvent( ) 173
On Error 173
Open( ) 175
OpenCom( ) 176
Operators 177
OverlayStr( ) 178
PadStr( ) 179

Pause( ) 179
PictureCtrl( ) 180
PopUpMenuSelect( ) 180
Print( ) 181
PromptBox( ) 181
Public 182
PurgeCom( ) 183
RadioButton( ) 184
Random( ) 184
RandomSeed( ) 185
Read( ) 185
ReadCom( ) 185
Readini( ) 186
ReadLine( ) 187
RemoveDir( ) 187
RenameFile( ) 188
RepeatStr( ) 188
Repeat...Until 189
ReplaceStr( ) 189
Restore( ) 190
Resume 190
Resume Next 191
Return 191
Reverse( ) 192
RfindStr( ) 192
Right( ) 193
RtrimStr( ) 193
Run( ) 193
ScrollBar( ) 194
ScrollBarPos( ) 194
ScrollBarWindow( ) 195
Secs( ) 196
SendToEditor( ) 196
SetDate( ) 197
SetFocus( ) 197

SetStrLen( ) 198
SetTime( ) 198
Size( ) 199
Sleep( ) 199
SplitPath( ) 200
Sqr( ) 200
Str( ) 200
StrCat( ) 201
SubStr( ) 201
Suspend 202
Switch...End Switch 202
SysMenuSelect( ) 202
SystemInfo( ) 203
TabCtrl( ) 204
TableColumns( ) 204
TableItem( ) 205
TableRows( ) 205
TableSelect( ) 206
TerminateApp( ) 207
TestData( ) 207
TestDataClose 208
TestDataCurField( ) 209
TestDataCurRecord( ) 209
TestData Expressions 210
TestDataField( ) 211
TestDataFieldCount( ) 211
TestDataIndex( ) 212
TestDataRecordCount( ) 212
TestDataTransform( ) 213
TestValue 214
TextPanel( ) 215
TextSelect( ) 215
Time( ) 216
TimeVal( ) 217
ToolBarCtrl( ) 217

TopWindow( ) 218
Transpose( ) 218
TreeViewCtrl( ) 219
TypeToControl 220
Trset( ) 221
Type( ) 221
UpDownCtrl( ) 222
UpDownPos( ) 223
UpperCase( ) 223
UserCheck( ) 223
ViewPortClear( ) 225
Val( ) 225
Wait( ) 226
WeekDay( ) 227
Whenever 228
While...Wend 229
WinClose( ) 230
WindowText( ) 230
WinGetPos( ) 231
WinVersion( ) 231
WndAtPoint( ) 232
Word( ) 232
Words( ) 232
Write( ) 233
WriteCom( ) 233
Writeini( ) 234
WriteLine( ) 234
Year( ) 235
Index 235

Chapter 1. Language Overview

Welcome to the *EZ Test Language Reference Manual*. This manual is one component of the documentation set which, collectively, explains all aspects of using *EZ Test*. Specifically, this manual provides a reference to the commands you can use in your *EZ Test* scripts. It is intended for experienced *EZ Test* users who wish to exploit the scripting language to develop robust, sophisticated test procedures. To gain an understanding of automated testing using *EZ Test*, you should work through the exercises in the accompanying *EZ Test GUI Testing Getting Started Guide* or *EZ Test Character-Based Testing Getting Started Guide*. They introduce you to the basics of working with *EZ Test* — setting up the system, learning scripts, building checks (test cases), and viewing the results. The *EZ Test User's Guide* provides a complete reference for using *EZ Test*. It contains detailed explanations on how to set up and configure the system, develop scripts, define checks (test cases), and view the results of a test run. It also describes how to use external testdata files and DBC-compliant data sources, debugging scripts, and other advanced features. The main body of this manual (Chapter 4, "Script Commands") contains an alphabetic listing of script commands. Related script commands are cross-referenced. The operation of each command, with all its options, is explained and followed by at least one usage example.

This chapter describes methods for using *EZ Test* scripts and provides a brief overview of the script language.

About EZ Test *Scripts*

EZ Test is an automated testing tool that helps you plan, develop, and run tests on application software in order to identify application problems. *EZ Test* is used to create and execute these test procedures. Once you have developed a test procedure using *EZ Test*, you can run it as many times as you wish — automatically. Automated test processes run in a fraction of the time it takes to manually test. Automated tests can be run overnight or on weekends when machine time is easier to schedule. All actions taken by the test script and all reactions of the target application are recorded in a log, which can be reviewed at your convenience. *EZ Test* helps you meet your testing requirements by automating the testing process. *EZ Test* works by mimicking the actions of a human tester. All the actions that you would perform to test an application — making menu selections, typing in data, checking the way it is processed, and so on — are recorded in a script. Maintaining your test procedures in script format has the following advantages. You can; Modify your test procedures as the target application changes; scripts do not need to be rewritten from scratch, Build new test procedures by copying and modifying existing scripts, Make scripts "loop" to repeat a process over and over again, Build "intelligence" into your tests to handle unexpected situations, Divide the work among many people and merge their efforts together, Use scripts as the documentation of test processes. *EZ Test* scripts are written in a simple, but powerful, programming language. By using the available features, you have total control over any application running in the Windows® environment. You can create scripts automatically using *EZ Test*'s Learn facility. Your actions — and the responses of the applications you work with — are translated into script commands and pasted into the editor. However, creating your initial script is just the beginning. One of the advantages to using *EZ Test*, is that it gives you the ability to modify and tailor your scripts to accommodate changes in the testing environment or target application. For example, you may need to modify or enhance the original script in order to; Display a dialog to receive input, such as a user ID or password, from a user, Repeat a process many times with variable data., Add routines to deal with errors in the target application. To make these types of modifications, you would require more than just *EZ Test*'s Learn facility — you need to make some advanced alterations to the original script. You can accomplish this using *EZ Test*'s script language. The *EZ Test* script language contains the functionality of a high-level programming language and a number of features designed specifically for software control and testing.

Script Language Overview

EZ Test's script language is designed to be easy to use, but powerful enough to cope with any automation or testing requirement. It is less "formal" than some other programming languages. For example, it is not mandatory to "declare" a variable before you begin using it; if you need to store information as a variable, just make up a name and assign the information to it. There's no need to think about "data type" either. All numbers in *EZ Test* are floating point and can be very large. Type conversion from string to number and vice versa is automatic. An automation language has a number of capability requirements beyond those of normal programming languages (such as BASIC or PASCAL). For example, a automation language must be able to:

- Access external information
- Perform asynchronous decision making
- Conduct external actions.

The sections that follow discuss how *EZ Test* handles these advanced requirements.

Access to External Information

For any software automation process to succeed, the following information is required:

- Status of other programs running in the system
- Contents of windows and dialogs
- User activity (keyboard, mouse activity etc.)
- Operating system data (the time, existence of files, etc.)

Some of this information is public (for example, the text displayed in a window), while other information can only be obtained by querying another program's Application Program Interface (API), if one is available. An API is an interface that one program offers to others, allowing them to communicate. This type of information is made available in a *EZ Test* script by using *system variables* and *system functions* with values that change according to changes in external conditions. Information about external conditions can be determined by defining and testing *events*. An event is a condition (external to *EZ Test*, but internal to the computer system) with a status that can be tested by the script. For example, if the following statement is true, it indicates that it's the afternoon:

```
Time( ) > "12:00:00"
```

Asynchronous Decision Making

Knowledge of events enables a script to become *intelligent* – allowing it to make decisions about the flow of the automation process based on external conditions. Events can be used in conventional flow-of-control statements, for example:

```
If <Event is true>
  <do one thing>
Else
  <do another>
Endif
```

Or:

```
Repeat
  <instructions>
Until <Event is true>
```

In addition to using events in conventional constructions, a good automation language must also be able to respond to events that may happen that are outside of the control of the program. For example:

```
Wait <until Event is true>
```

Or:

```
Whenever <Event is true>
  <instructions>
Endwhen
```

A Wait statement simply halts script execution until an external event occurs. A Whenever statement defines a task (the <instructions>) that will be executed *whenever* the external event occurs. Using a Whenever statement is similar to using the telephone. You don't need to keep checking for someone on the line; you simply react whenever the phone rings. You then follow a sequence of actions; pick up the handset, talk to the caller, and hang up. You can then continue what you were doing before the call or, as a result of the conversation, decide to do something else. A Whenever statement is a multi-tasking concept that is fundamental to automation. Whenevers enable an automation application to respond to asynchronous external events.

Note
Some other programs use the term *event* to mean an action triggered by a user (such as a key press, a mouse click, or a menu selection). However, the action is still *internal* to the program itself. In *EZ Test*, an event is considered to be *external* to *EZ Test* (a key press, mouse click, or menu selection in the *target* application or in the computer system itself).

External Actions

Access to external information, and the ability to make decisions based on that information, enables an automation process to take appropriate action. The automation language must support commands for specifying those actions. In addition to the usual actions common to any programming system (such as arithmetic, string manipulation, and reading and writing files), automation systems must be able to *drive* other processes inside the environment. This may be accomplished using APIs or by simulating a human operator. Consequently, actions appropriate to an automation language fall into the following three categories:

- Traditional actions — such as file access, memory access, and string and numeric manipulation.
- Using human simulation — keystrokes, mouse movements, etc.
- Calling other APIs.

Chapter 2. Script Structure

This chapter provides a detailed explanation of the *EZ Test* script structure and describes the purpose of each type of command, function, and variable. The following concepts are covered in detail:

- Script elements
- Strings
- Numbers
- String/Number type conversion
- Arrays
- Events
- Testdata
- SQL commands

Script Elements

When you are developing your scripts, you should consider issues of maintainability, portability, and usability. It is important that your script structure is both flexible and comprehendible to others. The development techniques — your use of the following script elements — that you employ as you create your scripts contribute to these factors.

Structure

EZ Test's scripting language is designed to develop "block-structured" scripts. A well designed script should contain a number of short blocks (called functions), each performing some simple action. These individual functions may then call lower-level functions or may themselves be called by higher-level functions. A script consists of a top-level function (called the Main function) that contains a series of calls to lower-level functions, and it also passes data to those functions. The lower level functions process the data and return the results, which may be used in the next call. The primary advantage to block structure is that commonly used routines only need to be written once. Then, they can be called from other functions. Block-structured scripts are very adaptable and easily enhanced or expanded because a test script can include additional scripts or can call other scripts as if they were "external blocks." This means that once a script is developed, it is available for use within other scripts. New scripts can be built quickly by assembling the existing blocks.
Another advantage of this structure is that a single, reusable block of instructions presents a single point of maintenance. If a process changes due to a change in environment or an update to the target application, only the block that handles that specific process needs be modified. Consequently, it is not necessary to change all scripts that make use of that block.

The Learn facility can be used to create scripts quickly and easily by capturing keystrokes, mouse actions, and the target application's responses. However, you should plan your work to avoid learning long, unstructured scripts that will be difficult to understand, reuse, and maintain. We recommend learning scripts in short bursts and structuring them as you go along. The time spent building common routines into separate blocks will most certainly be recouped when you find yourself creating new scripts or changing old ones.

An example of a block-structured script is shown below. This diagram demonstrates how various blocks of scripts interact:

Commands, Functions, and Variables

A script consists of functions that perform actions on other applications or that handle data in the form of strings and numbers and return the result. Most commands in *EZ Test*'s scripting language are themselves functions. The general form of a *EZ Test* command is:

return value = Function(string / number values, options); comment

The return value often gives an indication of the success (or failure) of the command. If you want to know the return value from a function, you must use parentheses around the parameters that are passed to it. If you are not interested in the return value, the parentheses are optional. Comments are optional too — but they help to explain the purpose of the script. For example, the following two lines are equivalent:

```
CopyFile "c:\netwk.log", "c:\backup\netwk.log" ; backup the log
ret = CopyFile( "c:\netwk.log", "c:\backup\netwk.log" )
```

The first variant simply issues a command to copy one file to another. The second variant also issues a copy file command, but assigns a value of 1 to the variable ret if the copy

```
Function Main
Make Tea
Make Coffee
End Function
Function Make Coffe
Prepare Water
Get Cup
Add Coffee
Pour Water
End Function
Function Make Tea
Prepare Water
Get Cup
```

```
Add Tea
Pour Water
End Function
Function Prepare Water
Fill Kettle
Switch on Kettle
Wait to Boil
Switch off Kettle
End Function
Function Get Cup
Get Cup from Cupbaord
Check it's Clean
Place Cup on Table
End Function
Function Pour Water
Carry Kettle to Table
Pour Water into Cup
Stop When Full
Return Kettle
End Function
Function Add Tea
Get Tea
Add 1 Spoonful to Cup
Return Tea
End Function
Function Add Coffee
Get Coffee
Add 1 Spoonful to Cup
Return Coffee
End Function
```

was successful or 0 if the copy failed. This allows you to verify that your script is proceeding as expected and to take recovery action if it is not.

All commands, labels, procedure names, function names, user-defined variable names, system variables, and system functions can be entered in upper, lower, or mixed case characters. The standard convention throughout this document is to capitalize the initial letter for each word in a multi-word command CopyFile, ListCount, ScrollBarWindow, etc.). This technique is not necessary for your script to process correctly; however, it makes your scripts easier to read. Lines of script can span more than one line, as white space is ignored. White space includes new lines, tabs, and space characters. For example, the MessageBox( ) function takes the general form:

```
MessageBox( "title" , "message" , "buttons" )
```

If the "title" and "message" are too long to display on one line, they can be split over multiple lines. For example:

```
MessageBox( "This is the title" ,"This is the message that will be" + " displayed within the messagebox" , "OKCancel")
```

Note how, in this example, the "message" and "buttons" lines are indented to indicate that they are part of the MessageBox( ) command above. Using indents is a good way to show that lines of script are grouped together. It is particularly useful within Function( ) definitions, in If...Else...Endif constructions, and inside loops. For example:

```
Function WaitFor( a , f )
getout = 0
Repeat
  If ActiveName( ) = a
    If FocusName( ) = f
      getout = 1
    Endif
  Endif
  pause 5 "ticks"
```

```
Until getout = 1
End Function
```

Likewise, spaces that fall between function parentheses and parameter commas are not required, but they are recommended to make the script easier to read. For example, the following two lines are processed the same:

```
ret = BitMapSelect ( "ImageMapName", "options" )
ret=bitmapselect("ImageMapName","options")
```

Also, string and number values can be stored by the script as constants or as variables. A variable can switch from string to numeric type and back again depending on the last assignment made. These concepts are explained in more detail later in this chapter.

Comments

All text after a ; , or // to the end of the line is treated as a comment. *All* text between a /* and a */ is ignored. For example:

```
; this is a comment line
// This is a comment line too
/* This is a block of text spanning
more than one line which is also
treated as a comment */
```

Commenting your script helps to explain the logic. Doing so helps you trace problems if your script does not work as you expected, and it helps others understand your scripts. If a comment appears on a line, there cannot be any other command following it on the same line. The following is a valid construction:

```
part = Mid( name, ; load part with part of name
1, ; starting from the beginning
12 ) ; for 12 characters
```

Public, Private, and Local Variables

Numeric and string variables can be public, private, or local. Variables declared as public can be accessed by all child scripts executed using the Run( ) function. They must be declared as public in the parent script and in all child scripts. Public variables cannot be declared within function definitions:

Parent Script:

```
Public a, b, names[] ; declare as public variables
Function Main
  a = 10 ; assign a value to a public variable
  Run "Child" ; run a child script
End Function
```

Child Script:

```
Public a, b, names[] ; declare as public variables
Function Main
  MessageBox( "Child", a ); display value of public variable
End Function
```

Variables declared outside of functions are private to the current script. The Var statement can be used to declare private variables, but it isn't mandatory (in other words, variables are, by default, private). Private variables can be accessed and updated anywhere within a script — but not by child scripts.

```
Var a, b, c ; this line is optional
Function Main
Setup ; call Setup function
MessageBox( "a is" a ) ; display value of private variable
```

```
End Function
Function Setup
a = 10 ; this variable is private
End Function
```

Variables declared inside function definitions are local to that function. They can be declared anywhere inside the definition and are accessible from the point of declaration. The Var statement is used to declare local variables:

```
Function Main
Setup
MessageBox( "a is" a ) ; a is uninitialized in this function
End Function
Function Setup
var a ; a is local to this function
a = 10
MessageBox( "a is" a ) ; and so shows a value here
End Function
```

Constants

Constants are like variables, except their value can't be changed when the script is run. Constants are always private to the script. They can be either string or numeric, but they must be declared before they can be used:

```
Const TRUE = 1
Const FALSE = 0
Const FileName = "session.log"
```

Once declared, a constant cannot be redefined.

Reserved Words

The following words are reserved and cannot be used as variable names.

Strings

A string is simply a group of one or more characters — words rather than numbers. A string is an item of data that can be used in many of the *EZ Test* commands. Strings can be constant (or literal), which means that they take a fixed value, or they can be variable, meaning their value can change during the execution of a script.

String Constants

A string constant is a sequence of characters inside single quotes (') or double quotes ("). For example:

```
'EZ Test. "Software Testing Software" from American Systems'
```

Or:

```
"All the world's a stage"
```

The maximum size of a string constant is 32 K (32,767) characters. You could put your name into a string constant and have the LogComment command write it to the log whenever the script was run:

```
LogComment "This script was developed by Derek Amitri"
```

Special representations are used for non-printing characters, special characters, and keystrokes.

Table 2-1. *EZ Test* Reserved Words

and const delete elseif func loop or select var
break continue dllfunc end function main public step wend
call declare do endif if next ref then whenever
case default else for include not return to while

'C' Escape Sequences

Table 2-2 details the valid escape sequences.

Example 1:

MsgBox("", c"*EZ Test*\r\nAmerican Systems")

Displays the following Message Box:

Example 2:

c"Window\x73" ; "Windows" (letter "s" has ASCII value 115 = hex 73)

Use 'C' escape sequences to display or to write text to disk.

Table 2-2. *EZ Test* Valid Escape Sequences

\n 10 New Line
\r 13 Carriage Return
\a 7 Bell
\b 8 BackSpace
\f 12 Form Feed
\t 9 Horizontal Tab
\v 11 Vertical Tab
\' 39 Single Quote
\" 34 Double Quote
\\ 92 BackSlash
\oOO ASCII character Number 'OO' is the octal representation
of the character
\xhh ASCII character Number 'hh' is the hexadecimal representation
of the character

Keystrokes

For typing, special keys are represented as keynames within {} braces. Key combinations are shown within nested braces.

Examples:

Type "Hello World{Return}"
Type "American Systems{Tab}*EZ Test*{Control r}"

The easiest way to generate the correct keystroke syntax is to Learn it into a script.

String Variables

A string variable is a holder for a string, the value of which may change from time-totime. You give the variable a name and assign values to it. The variable name must begin with an alpha character and may be up to 128 characters (spaces are not permitted). The maximum size of a string variable is only limited by the available memory in your PC. If you use a terminal emulation program to access applications running on a large, remote computer, and you want your log to record the screen title each time the display changes, you could write a routine that captures the screen title into a string variable and writes it to the log; then call that routine each time the display changes:

```
Function ArrivedAt
  screen_title = CaptureBox( "my application", 70, 0, 400, 20 )
  LogComment( "Arrived at the " + screen_title + " screen" )
End Function
```

Each time this routine is called, the string variable screen title acquires a different value — the title of the application screen at that point.

String variable names are not case sensitive; they are always treated as uppercase, whether defined in upper, lower, or mixed case. Table 2-3 provides some example string variables and their interpretations. Strings are allocated dynamically. If a string variable is assigned a value by one statement and reset by the next, the space used is released automatically.

Table 2-3. *EZ Test* String Variable Interpretation Examples

D D
address ADDRESS
ADDRESS1 ADDRESS1
A_Very_Long_String_Name A_VERY_LONG_STRING_NAME

String Assignment

String variables are null ("") until assigned a value. To assign a value to a variable, enter the following:

stringvariable = <string expression>

You cannot assign a value to a read-only system variable or function. To copy the contents of one string to another, enter the following:

thisstring = thatstring

The maximum length of a string variable is only limited by available memory.

String Expressions

All string types (constant, variable, system variable, string array variable) can be combined using the + operator to give a new string expression. For example:

longstring = Log.Name + " is the current log for " + name[4] + " and " + department + " is the department for group " + Left(group , 7)

Note that the assignment of the string can span more than one line, which makes it easier to read. Each broken line must end with a + to indicate a continuation. If a function requires a string parameter, the parameter can also be an expression. For example:

part = SubStr(first + last , startpos , endpos)

In this example "first + last" is evaluated before SubStr( ) is executed to load part.

Boolean String Expressions

Boolean expressions take the following form:

string1 operator string2

The value of the expression is either 1 (true) or 0 (false). Table 2-4 details the available Boolean operators.

Examples

Example 1:

```
a = "Apples"
b = "Pears"
If a = b ; if "Apples" and "Pears" are the same
MessageBox( "Result",
"Values match" ) ; display this
Else ; if "Apples" and "Pears" are different
MessageBox( "Result",
"Values do not match" ); display this
```

```
Endif
```

= Equals True if string1 is equal to string2, otherwise
false
<> (or !=) Not Equal To True if string1 is not equal to string2,
otherwise false
> Greater Than True if string1 is greater than string2,
otherwise false
< Less Than True if string1 is less than string2,
otherwise false
>= Greater Than or Equal To True if string1 is greater than or equal to
string2, otherwise false
<= Less Than or Equal To True if string1 is less than or equal to
string2, otherwise false

Example 2:

```
a = "Apples"
b = "Pears"
If a <> b ; if "Apples" and "Pears" are different
MessageBox( "Result",
"Values do not match" ) ; display this
Else ; otherwise
MessageBox( "Result",
"Values match" ) ; display this
Endif
```

In addition to the above operators, AND, OR, and NOT can be used to build compound expressions:

```
; read surname and initial from the screen
surname = CaptureBox( "my application", 20, 40, 100, 10 )
initial = CaptureBox( "my application, 140, 40, 20, 10 )
If initial = "T" and surname = "Jones"
<process this record>
Else
<display next record>
Endif
```

Boolean expressions can be used with If...Else...Endif and to evaluate exit conditions for Do...Loop While, Repeat...Until, and While...Wend loops.

String System Variables

String system variables are string variables with values that are predetermined by the system (either by Windows or by *EZ Test*) or with values that can be set to modify the way *EZ Test* performs its tasks. The system variable Log.Name is a system variable that allows you to view the name of the log currently used by *EZ Test*:

```
MsgBox( "Logging to..." "The Current Log is " + Log.Name )
```

String System Functions

These functions' meanings are predetermined by the system. Their return values are set either by Windows or by *EZ Test*. Some are *read-only,* meaning that you can read their current value, but cannot assign a value to them, and some are *read/write,* allowing you to both read their current value and to assign a new one. The ActiveName( ) function is an example of a read-only system function:

```
active_window = ActiveName( )
```

This function returns the name of the window that currently has focus. It supplies information about the current state of the system, and you cannot assign a value to it.

The ClipBoard( ) function is an example of a read/write system function. It enables you to read the current contents of the clipboard and, optionally, replace it with new data:

old_clipboard_contents = ClipBoard("New ClipBoard Contents")

Other String Functions

In addition to the + and = operators, there are many functions that process a string and return a result. For example, the Left( ) function takes a string and a length and returns the left-most portion of that string:

leftmost = Left("CALIFORNIA", 2); loads leftmost with "CA"

The Right( ) function can be used in a similar way to extract the right-most characters from a string.
The FindStr( ) function can be used to check for the occurrence of one string within another and, if found, return its position. For example:

pos = FindStr(astring , "partstring")

This looks in astring for the occurrence of the word "partstring"; if "partstring" is found, the position of its first character is placed in the numeric variable named pos. If "partstring" is not found, pos is loaded with 0. The Length( ) function can be used to return the length of a string. For example:

thelen = Length(longstring)

This loads the numeric variable thelen with the length of the string longstring.

Numbers

Numbers, like strings, may be constant (meaning that they take a fixed value), or they may be variable (meaning that their value can change during script execution).

Numeric Constants

Numeric constants can be integers or fractional and can be written in decimal, hexadecimal, or exponential form.

Examples
123 Integer
1234.672 Fractional
0x1234 Hexadecimal (equivalent to 4660 decimal)
12.67e-2 Exponential (equivalent to 0.1267)
12.67e+8 Exponential (equivalent to 1267000000)
The maximum value of a floating point number is 1.7976931348623158e+308.
The maximum value of a hexadecimal constant is 0xffffffff (equivalent to 4294967295).
Hexadecimal numbers cannot have fractional parts.

Numeric Variables

A numeric variable is a holder for a number, the value of which may change from time-to-time. You give the variable a name and assign values to it. The variable name must begin with an alpha character and may be up to 128 characters (spaces are not permitted). Numeric variable names are not case sensitive. It does not matter if they are defined in upper or lower case, they are treated as if uppercase. Table 2-5 provides some examples of numeric variable interpretation.

D23 D23
x X
Salary SALARY
A_LONG_NUMERIC_VARIABLE A_LONG_NUMERIC_VARIABLE

Numeric Assignment

Numeric variables are 0 (zero) until assigned a value. To assign a value to a variable, enter the following:

numeric_variable = <numeric expression>

You cannot assign a value to a read-only system variable or function. To copy the contents of one numeric variable to another, enter the following:

this_number = that_number

Numeric Expressions

You can use the operators +, -, *, / together with parentheses to build numeric expressions in the usual way. For example:

x = n * ((54 / c) + (a + 75)) + 4

The priority of executions is * and / before + and -. Operations at the same level are evaluated from left to right. Parentheses can be used to change the sequence of precedence. Therefore:

result = 5 + 3 * 2

loads result with 11, because 3 * 2 = 6, plus 5 equals 11. Whereas:

result = (5 + 3) * 2

loads result with 16, because 5 + 3 = 8, multiplied by 2 equals 16. The % operator returns the remainder when the integer part of one number is divided by the integer part of another. For example:

remainder = 23%7; loads remainder with 2
remainder = 107.95%12.35; returns 11 (equivalent to 107%12)

Binary arithmetic is possible using the & (AND), | (OR) and ^ (NOT) operators and the << and >> bit-shift operators. For example:

ret = 21 & 14; returns 4 (binary 10101 & 01110 = 00100)
ret = 21 | 14; returns 31 (binary 10101 | 01110 = 11111)
ret = 21 ^ 14; returns 27 (binary 10101 ^ 01110 = 11011)
ret = 21 << 2; returns 84 (binary 10101 shifted 2 left = 1010100)
ret = 21 >> 2; returns 5 (binary 10101 shifted 2 right = 101)

If a function requires a numeric parameter, the parameter can also be an expression. For example:

part = SubStr(search , spos+2 , epos-3)

This example evaluates spos+2 and epos-3 before SubStr( ) is executed to load part.

Boolean Numeric Expressions

Boolean expressions take the following form:

number1 operator number2

The value of the expression is either 1 (true) or 0 (false). The operators are described in

Table 2-6.

Examples

Example 1:

a = 0x1234

```
b = 4660
If a = b ; if 0x1234 is equivalent to 4660
MessageBox( "Result", ; display this
"Values match" )
Else ; if 0x1234 does not equal 4660
MessageBox( "Result", ; display this
"Values do not match"
Endif
```

Table 2-6. Boolean Numeric Expressions

= Equals True if number1 is equal to number2, otherwise false
<> (or !=) Not Equal To True if number1 is not equal to number2, otherwise false
> Greater Than True if number1 is greater than number2, otherwise false
< Less Than True if number1 is less than number2, otherwise false
>= Greater Than or Equal To True if number1 is greater than or equal to number2, otherwise false
<= Less Than or Equal To True if number1 is less than or equal to number2, otherwise false

Example 2:

```
a = 21 >> 2 ; binary 10101, shifted 2 right
b = sqr( 25 ) ; b is the square root of 25
If a <> b ; if a & b have different values
MessageBox( "Result", ; display this
"Values do not match" )
Else ; otherwise
MessageBox( "Result", ; display this
"Values match" )
Endif
```

Boolean expressions can be used with If...Else...Endif and to evaluate exit conditions for Do...Loop While, Repeat...Until, and While...Wend loops.

In addition to the above operators AND, OR , and NOT can be used to build compound expressions:

```
a = 3
b = 4
c = 5
If a < b and c <> ( a + b )
MsgBox("Result", "3 is less than 4 and 5 does not equal 7")
Else
MsgBox("Result", "Check your maths")
Endif
```

Numeric System Variables

These are numeric variables whose values can be set to modify the way *EZ Test* performs its tasks. System variables take the following form:

```
System.Action = value
```

The system variable Log.Enable is an example of a system variable which controls the logging of *EZ Test*'s actions, while Replay.AttachExact controls the way *EZ Test* processes Attach names. The variables can be set as follows:

```
Log.Enable = 1 ; switches logging on
```

```
Replay.AttachExact = 0 ; enables "near match" attaching
```

Numeric System Functions

These are functions whose meanings are predetermined by the system. Their values are set either by Windows or by *EZ Test*. Some are *read-only,* meaning that you can read their current value but cannot assign a value to them, and some are *read/write,* allowing you to both read their current value and assign a new one. ActiveWindow( ) is an example of a read only system function.

```
whandle = ActiveWindow( )
```

This function returns the handle (a numeric value assigned by Windows) of the window that currently has focus. It supplies information about the current state of the system and you cannot assign a value to it. The FilePos( ) function is an example of a read/write system function. It enables you to read the current value of a file pointer and, optionally, to reset it:

```
oldfilepos = FilePos( "filename", newfilepos )
```

Other Numeric Functions

In addition to the numeric operators, there are many functions that process numbers and return a result. For example, Max( ) takes a series of numbers and returns the biggest:

```
maxval = Max( 10, 100, 1000, 20, 2 ); loads maxval with 1000
```

The Min( ) function can be used in a similar way to extract the minimum value from a series of values. Random( ) can be used to generate random numbers between a minimum and maximum value.

String/Number Type Conversion

A variable can switch from string to numeric depending on the last assignment made. For example:

```
a = 10 ; a is numeric.
b = "hello world" ; b is a string.
a = b ; a is now a string.
a = 15 ; a is numeric again.
b = a ; b is now numeric.
```

The result of a mixed string/numeric expression is determined by the left side of the expression. Strings containing *leading* numeric characters are converted to numeric values. For example:

```
; string expressions
a = "1234" ; a is a string (value "1234")
b = a + "9" ; b is a string (value "12349")
c = a + "xyz" ; c is a string (value "1234xyz")
; mixed string / numeric expressions
d = a + 9 ; d is string (value “12349”)
e = 1234 + "xy" ; e is numeric (value 1234)
f = 123 + "123" ; f is numeric (value 246)
g = 1234 + "3xy" ; g is numeric (value 1237)
h = 1234 + "x3y" ; h is numeric (value 1234)
```

In Boolean expressions, both sides are converted to the type on the left before the comparison is performed. For example:

```
a = "123" ; a is a string
b = "76 trombones" ; b is a string
If a > b ; returns false (because "123" < "76 ")
```
but:

```
a = 123 ; a is numeric
b = "76 trombones" ; b is a string
If a > b ; returns true (because 123 > 76)
```

The '+' operator returns a value according to the type of the left-most part of the expression. For example:

```
a = 10 ; a is numeric
b = "20" ; b is a string
c = a + b ; c is numeric (30)
```

The '-', '*' and '/' operators always return numeric values. If applied to a string variable, its type is changed to numeric before the operation. For example:

```
a = "1234" ; a is a string
b = a/2 ; b is numeric (617)
```

The Str( ) function converts a number into a string. For example:

```
a = 1234 ; a is a numeric (value 1234)
b = a + “9” ; b is numeric (value 1243)
c = Str( b) + "xyz" ; c is a string
; (value "1243xyz")
```

Arrays

The various data that you work with are, quite often, related. It is convenient to collect these related data into a group and refer to them by the same name. This can be done by the use of arrays. An array is a collection of related data values referred to by a single variable name. Each value in an array is called an element. It is distinguished from other elements in the array by an identifier called a key, which is enclosed in [ ] square brackets. The key indicates an element's position in an array.

EZ Test supports multi-dimensional arrays of strings and numbers. The arrays are implemented as associative lists rather than vectors. The number of elements that an array has is dynamic and is limited only by available memory. It is not necessary, therefore, to dimension the array but it must be declared. Arrays which are declared as public can be accessed by child scripts which are executed using the Run( ) function:

```
Public globala[ ], globalb[ ]
```

Arrays declared outside of functions are private to the current script. Use the Var statement to declare private arrays:

```
Var ArrayName[ ]
Var privatea[ ], privateb[ ], c[ ]
```

Local arrays are declared inside function definitions and are accessible from the point of declaration. The Var statement is used to declare local arrays.

```
Function Main
Var locala[ ], localb[ ]
End Function
```

Arrays can be copied to other arrays using the = assignment statement:

```
thisarray = thatarray
```

Note that when referring to whole arrays, the [ ] square brackets are omitted. Individual array elements are accessed using the following syntax:
arrayname[<keys>]
where <keys> is a list of comma separated expressions. These expressions can be

constant or variable, string or numeric, or a combination. For example:

```
surname = name[ "fred" ]
age[ "fred", "bloggs" ] = 65
x = list[ 1, 2, "bloggs", occupation ]
```

Range checking is performed by *EZ Test* and any attempt to access an element outside the array size causes a runtime error to occur. The maximum number of array elements is limited only by the available memory in your PC. The maximum length of a key is 256 bytes. There are a number of functions that allow you to set up and manipulate arrays. Table 2-7 provides some examples that allow you to manipulate arrays.

Single-Key Arrays

This example shows how you can use a single-key array:

```
var exefile[] ; declare the array
Function Main
FillArray( exefile, ; fill the array with .EXE
"c:\windows\*.exe" ) ; filenames
no_of_elements = ArraySize( exefile ); return the number of
; array elements
c = 1 ; initialize a counter
While c <= no_of_elements
MsgBox( "Next .EXE file", exefile[ c ] ); display the values
c = c + 1
Wend
End Function
```

Table 2-7. Functions for Array Manipulation

ArraySize( )	Returns the number of elements in an array.
DelArray( )	Deletes items from an array.
FillArray( )	Fills an array with filenames matching a file specification.
Var ArrayName[ ]	Declares local and private arrays.
Public ArrayName[ ]	Declares global arrays.

Multi-Key Arrays

The following example (a simple phone book program) demonstrates how you can use a multi-key array:

```
var phone[] ; declare the array
Function Main
Setup ; call setup routine
Inquire ; call inquiry routine
End Function
Function Setup
phone[ "Development", "London", ; Index the array
"Jim" ] = 148 ; elements with Department
phone[ "Sales", "London", ; Location and Name
"Bob" ] = 137 ; details and assign each
phone[ "Support", "London", ; person's phone number
"Anne" ] = 127 ; to an element
phone[ "Development", "New York",
"Dave" ] = 12016
phone[ "Sales", "New York",
"Rick" ] = 13179
End Function
Function Inquire
promptbox( "Enter Department", ; Display PromptBoxes
"Department", dept ) ; requesting the three
promptbox( "Enter Location", ; elements of the search.
"Location", loc ) ; Information entered is
```

```
promptbox( "Enter Name", ; stored in variables
"Name", name ) ; dept, loc and name.
tel = phone[ dept, loc, name ] ; Use the variables to
; retrieve the required
; phone number.
msgbox( "The phone number you " + ; Display the result
"require is", tel )
End Function
```

Events

An event is a condition which occurs outside of *EZ Test*, but within the PC. For example:

- A key is struck within the target application
- A menu selection is made
- Some text is displayed in a window
- The internal clock reaches a particular time of day

Events like these that are crucial to the successful execution of a script can be defined within *EZ Test*. The script can be made to wait for defined events to happen or to perform some action when they occur. The occurrence of a defined event can be determined by the Event( ) function.

The language supports the definition of events within a script. However, you are strongly advised to define events using **Insert>Event** from the script editor's menu. This defines the event within the ***event map*** — which has significant advantages over defining events within a script. For example:

- It removes the event definition from the script — making it easier to read.
- It makes the defined event available to other scripts and other users — avoiding duplication of effort.
- It provides a single point of maintenance should the event definition need to be altered in future.

The only significant advantage of defining an event within a script is that it permits the use of variables within the definition. Should you wish to define an event within a script, use the syntax described below.

```
Eventname = MakeEvent( "EventType [event throwaway]", "window", "Action" )
```

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call. EventType Is the event type. This can be Keyboard, Mouse, Menu, Window, Screen or Date/Time. This may be followed by the optional word [event]. throwaway Prevents the keys or mouse clicks defined in Keyboard or Mouse events from reaching the application.window Is one of the following forms:

"anywindow" Indicates that the event can be triggered in any application window.
"module <ModuleName>" Instructs the event to use the application's EXE name. The "<ModuleName>" specifies the module name.
"<AttachName>" The Attach Name (or object map name) of the window where the event must occur.
"Action" Defines the activity that triggers the event. This will be a list of keystrokes, mouse clicks, menu selections, window states, text strings or dates/times.
See the MakeEvent( ) entry in Chapter 4, “Script Commands” for a full explanation of event definitions.

Example

The following script waits for you to hit the {F9} key. If you press {F9} within 5 seconds, an acknowledgment is displayed; if you don't, a reminder is displayed.

```
Function Main
; define the event to wait for
F9Key = MakeEvent( "keyboard throwaway", "anywindow",
"{F9}" )
Wait("5", "until", F9Key); allow 5 secs for event to occur
If Event( "F9Key" ) = 1 ; if it has
MessageBox( Event( F9Key ), "You struck the {F9} key" )
Else ; otherwise
MessageBox( Event( F9Key ), "Wake up please" )
Endif
End Function
```

Similar syntax can be used for screen, mouse, keyboard, time, menu, and window events.

Test Data

Testdata files provide an efficient way for scripts to access external data. The use of testdata files enables the *logic* of a script to be separated from its *data*. For example, to input 500 entries into a database application, you only need to script a single entry. The 500 sets of input data can be read by the script from an external testdata file at runtime. A testdata file is a comma separated variable (CSV) file when each line constitutes a *record.* Each record contains a number of *fields* that are separated by commas. For example:

Example 1 (A Testdata File with 3 Records, Each with 5 Fields):

```
Tom,Jones,24,Software Development,4227
Dick,Tracy,36,Quality Assurance,1044
Harry,Hawk,52,Product Planning,2128
```

Example 2 (A Testdata File with 'm' Records, Each with 'n' Fields):

```
R1F1,R1F2,R1F3,……R1Fn
R2F1,R2F2,R2F3,……R2Fn
…
RmF1,RmF2,RmF3, RmFn
```

Testdata files can be created with a text editor or they can be produced from any spreadsheet or database program that can export or save files in CSV format. Fields containing commas may be included within a testdata file if enclosed in double quotes:

```
American Systems,"123 Seventh Street","Fort Worth, TX 76180"
American Systems Inc.,"456 Main Street","Addison, TX 75001"
```

Testdata files are indexed by *EZ Test* to ensure quick location of individual fields. If the testdata file does not have an index file, or the existing index file is older than the testdata source file, a new index file is created automatically. Testdata files should be “rectangular” — that is, each record should contain the same number of fields. Records which contain fewer data fields should be padded with blank fields or indexation will fail (the indexing process assumes that all records contain the same number of fields as the first record). There are a number of functions that allow access to testdata files. These functions permit the relevant testdata file to be selected, interpret strings containing testdata expressions, and return the corresponding field values. Each testdata string expression is of the form:

```
"{<R>.<F>}"
```

Note
There should be no spaces between the fields and their comma separators.

Where <R> can be:

<N> A number specifying the index of a record.
= Retrieve from the current record.
+ Retrieve from the next record.

- Retrieve from the previous record.
* Retrieve from a record selected at random.
Where <F> can be:
<N> A number specifying the index of a field.
= Retrieve from the current field.
+ Retrieve from the next field.
- Retrieve from the previous field.
* Retrieve from a field selected at random.

A testdata file containing the names of composers may look like this:

Ives,Copland,Gershwin,Bernstein,Joplin,Berlin
Satie,Milhaud,Faure,Saint-Saens,Debussy,Ravel
Elgar,Britten,Vaughan-Williams,Walton,Tippett,Delius
Shostakovich,Tchaikovsky,Prokofiev,Stravinsky,Rimsky-
Korsakov,Glinka
Bach,Handel,Mozart,Schubert,Brahms,Hindemith

The testdata expression "{3.4}" refers to Walton (third record, fourth field)
"{+.-}" then refers to Prokofiev (fourth record, third field)
"{1.*}" refers to an American composer (selected at random from the first record)

The number of records in a testdata file and the number of fields within each record can be obtained from the TestDataRecordCount( ) and TestDataFieldCount( ) functions. The current record and field positions may be obtained from the TestDataCurRecord( ) and TestDataCurField( ) functions. Both functions are updated whenever a testdata expression is evaluated by a testdata function. The contents of testdata files may be Typed directly into an application. Alternatively, the TestDataTransform() and TestDataField( ) functions may be used to extract values from a testdata file, either to use in controls that cannot be "typed" to (such as edit controls) or to enable processing of the value to take place before passing it on to the target application. If no path is specified, a testdata file is assumed to be located in the directory containing the current *EZ Test* database.

SQL Commands

You can access data from a Microsoft Access database (.MDB file) or from an ODBC (Open Database Connectivity) data source using Structured Query Language (SQL) statements. Before you can access data using an ODBC driver, you must add a data source for it using the ODBC icon in the Windows Control Panel. Once a data source has been established, you can extract information using commands within your *EZ Test* script. For example, the EZ TESTDemo sample application shipped with *EZ Test* uses a Microsoft Access database containing three tables:

CarList Customers UserDetails
Ref Account Number User
Make Name Password
Engine size Address UserName
Year
Doors
Colour
Cost
Discount
QuantityA
QuantityB
QuantityC
Cond1
Cond2
Cond3
Cond4
Cond5
Cond6

To use the information within the EZ TESTDemo database, you must connect to the data source using the dbConnect( ) function:

```
dbConnect( "DSN=EZ TESTDemo" ) ; ODBC connection
```

Or:

```
dbConnect( "EZ TESTDemo.MDB" ) ; Microsoft Access .MDB file
```

Once connected, you can use SQL commands to extract information from the database. The dbSelect( ) command creates a set of records that satisfy selection criteria. The selected records may be a *dynaset* (a set of records which can be used to manipulate values in the underlying database tables) or a *snapshot* (a set of records which can be used to examine, but not update, values in the underlying database tables). For example:

```
; connect to data source
dbConnect( "c:\Program Files\EZ Test.32\demos\EZ Testdemo.mdb" )
; select records from the data source using SQL statement
dbSelect( "SELECT Make, Year FROM CarList WHERE Cost >” +
“15000", "dynaset" )
; move to the first matching record
dbMoveFirst( )
While dbEOF( ) = 0 ; while not at end of result set
Print dbGetField("Make") ; print next entry in "Make" field
Print dbGetField("Year") ; print next entry in "Year" field
Print "" ; print a blank line
dbMoveNext( ) ; move to next record
EndWhile ; end of loop
dbDisconnect( ) ; disconnect from data source
```

There are also commands to count the number of records in a record set, to move through them, and to retrieve and set field values. SQL statements may also be executed directly on records within a data source using the

dbExecute( ) function:

```
; connect to a data source using ODBC driver
dbConnect( "DSN=EZ TESTDemo" )
; update records which match particular selection criterion
dbExecute( "UPDATE Carlist SET Cost=Cost-1000 WHERE
Make='Ford'" )
; disconnect from the data source
dbDisconnect( )
```

You cannot refer to a variable directly within a SQL statement. To use a variable you must construct your SQL expression so that *EZ Test* resolves the value before it is passed to the SQL function. In this example, a new entry is made in the EZ TESTDemo database. The reference number is captured from the screen and used to check that the database fields have been updated correctly:

```
Function Main
Add_New_Entry
Check_New_Entry
End Function
Function Add_New_Entry
; attach to EZ TESTDemo "New" dialog
Attach "New PopupWindow"
; click the "OK" button to Add Car
Button "OK", 'Left SingleClick'
; attach to the "Add Car" dialog
Attach "Add Car ChildWindow~1"
```

```
; read Reference from "Ref" field
RefNo = WindowText( "~N~EZ TESTDEMO.EXE~Edit~&Ref :" )
; enter new car name
EditText "&Make :", "Test"
; and value
EditText "Sale &Price :", "20000"
; click "Add" button to save
Button "Add", 'Left SingleClick'
; close dialog
Button "Close", 'Left SingleClick'
End Function
Function Check_New_Entry
; connect to database
dbConnect( "c:\Program Files\EZ Test.32\demos\EZ Testdemo.mdb" )
; enter SQL statement, resolving reference variable
dbSelect( "SELECT Cost FROM CarList WHERE Ref='" +
RefNo + "'" )
; check "Cost" field and log result
If dbGetField("Cost") = "20000"
UserCheck( "dbUpdate", 1, "DataBase updated correctly" )
Else
UserCheck( "dbUpdate", 0, "DataBase not updated" )
Endif
End Function
```

Chapter 3. Script Command Groups

This chapter lists the script commands in their respective command groups. A command group is simply an arrangement of related commands. This chapter groups the commands by *concept* rather than by alphabetic listing. Use the command group to search and locate the command you are looking for. You may then use the alphabetic command listing in Chapter 4, “Script Commands” to find specific information related to syntax, variants, operation, and examples. The command groups are as follows:

- Checks
- Clocks
- Date/Time
- DDE Commands
- Dialog Control
- File Access
- Language
- Logging
- Menu Control
- Menu Information
- Miscellaneous
- Mouse Control
- Mouse Information
- Number Manipulation
- Performance Monitoring
- Program Flow
- SQL Commands
- String Manipulation
- Synchronization
- System Information
- Testdata Handling
- Window Control
- Window Information
- 4GL Commands

Checks

Check( ) Runs a check on the target application.
CheckExists( ) Verifies the existence of a check.
LinkCheck( ) Reports on the existence of a link.
MakeCheck( ) Dynamically creates a new check using an existing check as a template.
TestValue Assigns a value to set the current test status.
UserCheck( ) Sends user-defined check entry to the log.

Clocks

Clock( ) Retrieves the current value of a clock in milliseconds.
ClockReset( ) Resets a clock to zero.
ClockStart( ) Starts or resumes a clock.
ClockStop( ) Stops a clock.

Date/Time

CreateDate( ) Enters a dynamically generated date into the target application at replay.
CurTime( ) Represents the current date and time as a number.
Date( ) Converts a date value into a string.
DateVal( ) Converts a date into a numerical representation.
Day( ) Returns the day of the month.

FormatDate( ) Formats a date and time into a string.
Hours( ) Returns the specified hour.
JulianDate( ) Returns the number of seconds since 12:00 a.m. December 31, 1899.
JulianDateVal( ) Returns the number of days since the beginning of the year (1 - 366).
Mins( ) Returns the specified minutes.
Month( ) Returns the month number.
Secs( ) Returns the specified seconds.
SetDate( ) Sets the PC's internal date.
SetTime( ) Sets the PC's internal time
Time( ) Converts a time value into a string.
TimeVal( ) Converts a time into a numerical representation.
WeekDay( ) Returns the day of the week.
Year( ) Returns the year.

Dialog Control

AnchorSelect( ) Selects Web objects that are created using the "A" HTML tag.
BitMapSelect( ) Clicks the mouse on a bitmap.
BrowserToolbarCtrl( ) Selects options from a Web browser's toolbar.
Button( ) Processes a button control.
CalendarCtrl( ) Sets the date on a Windows Month calendar control.
CalendarRange( ) Returns the start and end date of a range of dates selected in a Windows Calendar control.
CalendarToday( ) Returns the "today" date of a Windows Calendar control.
CheckBox( ) Processes a check box control.
ComboBox( ) Selects a string from a combo box.
ComboText( ) Enters text into the edit control of a combo box.
DateTime( ) Returns the numerical representation of a date time control.
DateTimeCtrl( ) Sets the date or time of a date/time control.
DateTimeMode( ) Returns a string indicating if the date/time picker control is operating in date or time mode.
EditClick( ) Clicks the mouse in an edit control.
EditText( ) Enters text into an edit control.
HeaderCtrl( ) Selects a column header control.
Hotkey( ) Simulates the pressing of a shortcut key.
ImageSelect( ) Selects Web objects that are created using the "IMG" HTML tag.
IPContol() Sets the IPAddress value on a Windows IPAddress control.
ListBox( ) Selects a string from a list box.
ListViewCtrl( ) Drives the file list area in a dialog.
MenuCtrl( ) Processes a menu control on Web-based applications.
RadioButton( ) Processes a radio button control.
ScrollBar( ) Drives the scroll bars or slider controls of the currently attached window.
ScrollBarWindow( ) Drives the scroll bars of the currently attached window.
TabCtrl( ) Selects a tab control in a dialog box.
TableSelect( ) Selects an item in a Java application's table control.
TextSelect( ) Clicks the mouse on a string of text.
ToolBarCtrl( ) Selects options from a toolbar.
TreeViewCtrl( ) Drives a directory list area in a dialog box.
Type( ) Types a string of keys to the currently attached window.
TypeToControl Learns typing actions on known controls without requiring repeated Attach statements.
UpDownCtrl( ) Drives the Up and Down spin control found on some dialogs.

File Access

ChDir( ) Changes the current working directory.
Close( ) Closes a file previously opened with the Open( ) function.

CopyFile( ) Copies a file to a given destination.
Create( ) Creates a new file or resets an existing one.
CurDir( ) Returns the current working directory.
DeleteFile( ) Deletes a specified file.
Dir( ) Returns next file in a folder matching a given criteria.
FileExists( ) Checks if a file exists.
FilePos( ) Returns or sets the position of the file pointer.
FileStatus( ) Returns the status of a previously opened file.
FileTime( ) Gives the date and time a file was last modified.
FillArray( ) Fills an array with file names matching a filespec.
IsFile( ) Checks for file existence and attributes.
MakeDir( ) Creates a new directory (folder).
Open( ) Opens a file for reading, writing or both.
Read( ) Reads a number of characters from a file.
Readini( ) Returns a value from an INI file.
ReadLine( ) Reads a line from a file.
RemoveDir( ) Removes a directory, or folder, at the specified path.
RenameFile( ) Renames a file.
Write( ) Writes a string to a file.
Writeini( ) Writes a value to an INI file.
WriteLine( ) Writes a line to a file.

Language

Arrays Uses of public, private and local arrays.
Assignment Assigning values to variables.
Boolean Expressions Tests the relationship between values.
Control Labels Identify controls in a dialog.
Const Declares a constant.
Declaration of Variables Declaring variables as public, private or local.
Operators Perform operations on values.
Public Declares public variables.
Testdata Expressions Handling of testdata files.
Var Declares private or local variables.

Menu Control

MenuSelect( ) Selects item from the currently attached window's menu.
PopUpMenuSelect( ) Selects a menu item from a pop-up menu.
SysMenuSelect( ) Selects an item from the attached window's system menu.

Menu Information

IsMenu( ) Returns the menu state of a menu item.
MenuCount( ) Returns the number of menu items on a specified menu level.
MenuFindItem( ) Returns either the position of menu item or the name of a menu item found in a specified position.
MenuItem( ) Returns the text of a specific menu item.

Miscellaneous

ArrayPush( ) Inserts a value at the end of an existing array.
ArraySize( ) Returns the number of elements in an array.
Beep( ) Plays a note on the speaker.
ClipBoard( ) Captures text from or places text into the clipboard.
CmdLine( ) Returns the command line string.

ConvertCurrency( ) Converts the value of one European currency into the value of another specified European currency based on the value of the EURO.
DeleteArrayName[Element] Deletes a whole array or an element of an array.
Dialog( ) Calls a dialog box.
DLLFunc Calls an external DLL function.
Err Reports the current error code.
ErrFile Reports the name of the script file that generated an error.
ErrFunc Reports the function that caused a runtime error.
ErrLine Reports the script line number where the error generated.
ErrMsg Reports a textual description of the current error.
GetProperty( ) Retrieves a property from a Java control.
Include Adds another script to the script during compilation.
MessageBox( ) Creates a message box.
Print( ) Sends output to the viewport window.
PromptBox( ) Defines a simple dialog box requiring user input.
Rem Adds a comment to the script.
TextPanel( ) Creates a panel with message text.
ViewPortClear( ) Clears the viewport window.

Mouse Control

MouseClick( ) Simulates the clicking of a mouse button in the currently attached window.
MouseHover( ) Moves the mouse pointer to the control specified and "hovers" for the specified seconds.
MouseMove( ) Moves the mouse pointer to the position specified.
NCMouseClick( ) Executes a mouse click in a non-client window.

Mouse Information

AttachMouseX( ) Returns the x-position of the mouse pointer within the currently attached window.
AttachMouseY( ) Returns the y-position of the mouse pointer within the currently attached window.
MouseX( ) Returns the x-position of the mouse pointer, in pixels, relative to the left of the screen.
MouseY( ) Returns the y-position of the mouse pointer, in pixels, relative to the top of the screen.

Number Manipulation

Abs( ) Returns the absolute value of a number.
Clng( ) Converts a number to a long integer.
Fix( ) Removes the fractional part of a number.
Int( ) Returns the integer part of a number.
Max( ) Returns the maximum value from a list of numbers.
Min( ) Returns the minimum value from a list of numbers.
Random( ) Generates a random number between two values.
RandomSeed( ) Seeds the random number generation function.
Sqr( ) Returns the square root of a number.

Performance Monitoring

NotifyEvent( ) Generates an event that can be monitored by an external application, such as ClientVantage, to time round-trip transactions.

Program Flow

Break Exits the current loop and continues execution on the line following the loop.
Chain( ) Executes another script. Caller and called script run concurrently.
Continue Returns to the top of a loop, ignoring following statements within the loop.
Do...Loop While Repeats a series of instructions while a condition is true.
Error Aborts the current error handler and calls the previous one.

Exec( ) Executes a program.
Exit( ) Exits the current script.
ExitWindows( ) Shuts down Windows.
Fatal( ) Generates a fatal runtime error and aborts the script.
For…Next Repeats a series of instructions a number of times.
Function…End Function Declares a user-defined function.
Goto Causes program execution to jump to a specified label.
If...Else...Endif Allows the script to perform runtime decisions.
On Error Handles runtime errors in scripts.
Repeat...Until Repeats a series of instructions until a condition is true.
Resume Restarts execution of a suspended script.
Resume Next Resumes script execution following an error.
Return Returns from a function with an optional return value.
Run( ) Runs another script from this script. This script is suspended until the other finishes.
Stop Stops the current script and all its parents.
Suspend Suspends the current script leaving Whenevers active.
Switch...End Switch Creates a CASE statement to switch on a value.
Whenever Executes a function whenever an event occurs.
While...Wend Repeats a series of instructions while a condition is true.

SQL Commands

dbAddNew( ) Permits addition of a new record to the current result set.
dbBof( ) Determines if the record pointer is at the start of the current result set.
dbClose( ) Closes the record set associated with the last dbSelect( ).
dbConnect( ) Connects to a SQL data source.
dbDisconnect( ) Disconnects from a SQL data source.
dbEdit( ) Permits editing of a field in the current record of the current result set.
dbEOF( ) Determines if the record pointer is beyond the last record in the current result set.
dbExecute( ) Executes a SQL command on the current data source.
dbGetField( ) Retrieves a field from the current record in the current result set.
dbMove( ) Moves the pointer within the current result set.
dbMoveFirst( ) Moves the record pointer to the first record in the current result set.
dbMoveLast( ) Moves the record pointer to the last record in the current result set.
dbMoveNext( ) Moves the record pointer to the next record in the current result set.
dbMovePrev( ) Moves the record pointer to the previous record in the current result set.
dbRecordCount( ) Returns the number of records in the current result set.
dbSelect( ) Selects records from a SQL data source.
dbSetField( ) Sets the value of a field within the current record in the current result set.
dbUpdate( ) Commits an edited field in the current record of the current result set to the data source.

String Manipulation

Abbrev( ) Checks the leading characters in a string.
Asc( ) Returns the ANSI value of a character.
Cesc( ) Converts 'C' escape sequences into characters.
Chr( ) Converts a value into an ANSI character.
Compare( ) Compares the contents of two strings.
DataType( ) Checks if characters in a string are of a particular type.
DeleteStr( ) Deletes a string within a target string.
FindChar( ) Scans a string for the first character that is not in a search list.
FindStr( ) Returns the position of one string within another.
IgnoreCase( ) Sets case sensitivity for string comparisons and searches.
InsertStr( ) Inserts a string into a target string.
InStr( ) Returns the position of one string within another.
Left( ) Extracts a number of characters from the start of a variable.

Length( ) Returns the length of a string.
LowerCase( ) Converts a string to lowercase.
LtrimStr( ) Removes leading spaces from a string.
Mid( ) Extracts a substring from the middle of another string.
OverlayStr( ) Overlays one string onto another at a given position.
PadStr( ) Pads a string with spaces or a specific character.
RepeatStr( ) Creates a string consisting of another repeated string.
ReplaceStr( ) Replaces characters within a string.
Reverse( ) Reverses a string.
RfindStr( ) Returns the position of the last occurrence of one string within another.
Right( ) Extracts a number of characters from the end of a string.
RtrimStr( ) Removes trailing spaces from a string.
SetStrLen( ) Prepares a string to use in a DLL function.
SplitPath( ) Returns part of a path string.
Str( ) Converts a number into its string equivalent.
StrCat( ) Concatenates strings (with an optional separator).
SubStr( ) Returns part of a string.
Transpose( ) Performs actions on characters in a string.
Trset( ) Expands a string containing a range of characters.
UpperCase( ) Converts a string to uppercase.
Val( ) Converts a string into its numeric equivalent.
Word( ) Extracts words from a string.
Words( ) Returns the number of words in a string.

Synchronization

Cancel( ) Cancels an event.
DestroyEvent( ) Destroys the specified MakeEvent from memory.
Event( ) Checks the status of an event.
MakeEvent( ) Defines a keyboard, mouse, window, screen, time or menu event.
Pause( ) Pauses the current script for a specified length of time.
Replay.ActionKeys Specifies the list of keys to be used in conjunction with Replay.AutoWait.
Replay.AttachDelay Specifies the time *EZ Test* should wait before processing the currently attached window.
Replay.AttachExact Controls the way *EZ Test* processes an attach statement.
Replay.AttachTimeOut Sets the maximum time *EZ Test* should allow to attach to a window.
Replay.AutoWait Specifies the time to pause after an action key is typed.
Replay.BitmapSelectDelay Determines the time *EZ Test* should pause before performing a BitmapSelect command.
Replay.BrowserTimeOut Determines the maximum number of seconds that *EZ Test* will wait for a Web browser to load a page.
Replay.CtrlRetries Specifies the time allowed to attach to a control.
Replay.Delay Specifies the time to wait after executing each statement.
Replay.DoubleQuotesInCSV Allows *EZ Test* to process double quotes in TestData files according to the industry standard.
Replay.EditBySetText Inserts text into an edit control by sending the control a Windows message.
Replay.ExactEvents Forces *EZ Test* to use exact attach names when waiting for events.
Replay.ExactListItems Forces *EZ Test* to use an exact match during replay of combo and list boxes.
Replay.InternetProfile Specifies the internet settings to use when *EZ Test* attempts to connect to a Web site during check verification.
Replay.MenuByCmd Controls the way *EZ Test* selects menu items.
Replay.MenuWaitTime Specifies the maximum time to wait for a pop-up menu.
Replay.MouseCmdDelay Specifies the time to wait, in milliseconds, after mouse commands.
Replay.MouseDelay Specifies the time, in milliseconds, between mouse events.
Replay.MouseHoverTime Determines the amount of time in seconds that the mouse hovers over a specified control.

Replay.PauseMode Determines whether pause statements should be ignored.
Replay.RunEnvironment Determines the run environment to be used during script replay.
Replay.ScreenEventCount Determines the number of cyclic seconds to elapse before *EZ Test* attempts to test each screen event.
Replay.TodaysDate Determines the value to be used as "today" during script replay.
Replay.TypeDelay Inserts a delay between keystrokes other than those defined as action keys.
Replay.WaitTimeout Specifies the time before a Wait statement expires.
Sleep( ) Pauses the script for a specified length of time.
TerminateApp( ) Terminates an application.
Wait( ) Pauses the script until an event occurs.

System Information

Focus( ) Determines the application that has focus.
GetEnv( ) Gets the value of an environment setting.
IsRunning( ) Determines whether the specified application is running.
LastKey( ) Returns the virtual key code of the last key pressed.
LastKeyStr( ) Returns the keytop string of the last key pressed.
SystemInfo( ) Retrieves system information.
WinVersion( ) Returns the Windows version as a numeric value.

Testdata Handling

TestData( ) Sets the current testdata file.
TestDataClose Closes the current testdata file and releases the handle of the corresponding index file.
TestDataCurField( ) Sets or retrieves the current field number in the testdata file.
TestDataCurRecord( ) Sets or retrieves the current record number in the testdata file.
TestDataField( ) Retrieves a field from the currently open testdata file.
TestDataFieldCount( ) Returns the maximum number of fields per record in the current testdata file.
TestDataIndex( ) Creates an index to a testdata file.
TestDataRecordCount( ) Returns the number of records in the current testdata file.
TestDataTransform( ) Transforms testdata expressions and retrieves the corresponding field value.

Window Control

Attach( ) Attaches to a window.
AppActivate( ) Attaches to a window using the z-order to determine which window to activate.
Maximize( ) Maximizes a window.
Minimize( ) Minimizes a window.
Move( ) Moves the currently attached window to the specified position.
Restore( ) Restores the currently Attached window.
SetFocus( ) Sets focus to the specified window.
Size( ) Sizes the currently attached window.
TypeToControl Learns typing actions on know controls without requiring repeated Attach statements.
WinClose( ) Closes the specified or currently attached window.

Window Information

ActiveName( ) Returns the name of the currently active window.
ActiveWindow( ) Returns the handle of the currently active window.
AttachAtPoint( ) Returns the name of the window at a point.
AttachName( ) Returns the attach name of a window.
AttachWindow( ) Returns the handle of the currently attached window.
ButtonDefault( ) Determines if a button is the default button.
Capture( ) Returns the text currently displayed in a window.
CaptureBox( ) Returns the text currently displayed in an area of a window.
CaretPosX( ) Returns the x-position of the Windows caret within the attached window.

CaretPosY( ) Returns the y-position of the Windows caret within the attached window.
ControlFind( ) Returns the window handle of a control.
CtrlChecked( ) Determines whether the specified control is checked.
CtrlEnabled( ) Determines whether the specified control is enabled.
CtrlFocus( ) Determines if the specified control has the keyboard focus.
CtrlLabel( ) Retrieves the label associated with the specified control.
CtrlPressed( ) Determines whether the specified control is currently depressed.
CtrlSelText( ) Retrieves the selected text from an edit control.
CtrlText( ) Retrieves text from a control.
CtrlType( ) Returns a number indicating the type of control referred to by the passed window's handle.
EditLine( ) Retrieves a line of text from a multi line edit control.
EditLineCount( ) Returns the number of lines in a multi line edit control.
FocusName( ) Returns the name of the currently active window.
FocusWindow( ) Returns the handle of the parent window in focus.
Get4GLInfo( ) Retrieves the browser name and version number.
GetReadyState( ) Returns the ready state of the browser window.
IsWindow( ) Determines if a window is in a specified state.
ListCount( ) Returns the number of items in a list control.
ListCount( )-ID Based Returns the number of items in a list control.
ListFindItem( ) Returns the position of an item in a list control.
ListFindItem( )-ID Based Returns the position of an item in a list control.
ListFocus( ) Returns the position of the selected item in a list control.
ListFocus( )-ID Based Returns the position of the selected item in a list control.
ListItem( ) Retrieves the text from an item in a list control.
ListItem( )-ID Based Retrieves the text from an item in a list control.
ListTopIndex( ) Returns the position of the first visible item in a list control.
ListTopIndex( )-ID Based Returns the position of the first visible item in a list control.
MouseCursor( ) Returns the shape of the windows cursor.
MouseWindow( ) Returns the attach name of the window beneath the mouse pointer.
ScrollBarPos( ) Retrieves the position of a slider control.
TableColumns( ) Returns the number of columns in a PowerBuilder or Java application's table.
TableItem( ) Returns data from a cell within a PowerBuilder or Java application's table.
TableRows( ) Returns the number of rows in a PowerBuilder or Java application's table.
TopWindow( ) Gets the attach name of the topmost window.
UpDownPos( ) Retrieves the value of a spin control.
WindowText( ) Gets the text from a window.
WinGetPos( ) Gets the position and size of a window.
WndAtPoint( ) Retrieves the handle of a window at a point on the screen.

Chapter 4.

Script Commands

This chapter describes all the commands, functions, and system variables in the *EZ Test* scripting language (the script commands). The commands are arranged in alphabetical order, ignoring non-alphabetic characters such as periods and underscore characters. The title heading for each command indicates the group to which the command belongs. Refer to Chapter 3, “Script Command Groups” for more details related to the command groups.

As applicable, the following headings are provided for each command:

- **Syntax:** Specifies the method used to write the command, including any optional arguments and parameters.
- **Variants:** Specifies any alternative syntax that is acceptable.
- **See Also:** Lists other related commands, which you may want to refer to or use.
- **Operation:** Describes the way the command operates, including any assumed default values.
- **Examples:** Provides at least one example of the command used in a practical context.

Some of the examples are simple one-line demonstrations of the command that show both the way the command is used in a program function and the result it produces. Other examples are longer extracts from program coding that show the command in the context of the function in which it is being used. The actual command demonstrated in the example is shown in bold typeface.

Abbrev( )

String Manipulation
Checks the leading characters in a string.

Syntax
ret = Abbrev("targetstring", "shortstring")
See Also
FindStr( )
Operation
This function compares the leading characters in "targetstring" with those in "shortstring" and returns 1 if they match or 0 if they don’t. The Abbrev( ) function automatically converts numeric parameters to strings.
Examples
ret = Abbrev("abcdef", "abcdef") ; returns 1
ret = Abbrev("abcdef", "abc") ; returns 1
ret = Abbrev("abcdef", "b") ; returns 0
ret = Abbrev("abcdef", "abd") ; returns 0
ret = Abbrev("abcdef", "abcdefg") ; returns 0
ret = Abbrev("abcdef", "") ; returns 0
ret = Abbrev(123456, 123) ; returns 1
ret = Abbrev(123456, 246) ; returns 0
ret = Abbrev("hello", hello) ; compares the value of the
; variable hello with the
; string "hello"; if hello is
; uninitialized, returns 0)
ret = Abbrev(hello, world) ; returns 1 if hello and
; world are both
; uninitialized variables

Abs( )

Number Manipulation
Returns the absolute value of a number.
Syntax

ret = Abs(value)

Operation
This function returns the absolute (positive) value of a number.

Examples
ret = Abs(10.12) ; returns 10.12
ret = Abs(-10.12) ; returns 10.12

ActiveName( )

Window Information
Returns the name of the active window.

Syntax
ret = ActiveName( )

See Also
ActiveWindow( ), TopWindow( ), MouseWindow( ), IsWindow( ), WinGetPos( ), FocusWindow( ), FocusName( ), AttachName( )

Operation
This function returns the attach name of the active window. Note that, if the window contains child windows (such as a dialog window containing edit controls and buttons), the name of the parent is returned, not that of the child which has focus. To determine the name of the window that has focus, use the FocusName( ) function.

Examples
```
; prevent the "Bootlog.txt" file from being opened
a = 1 ; set up a counter
While a = 1 ; eternal loop
ret = ActiveName( ) ; get active window name
result = FindStr( ret, "Bootlog.txt" ); search name for text
If result <> 0 ; if found in attach name
Attach ret ; attach to window
WinClose ; and close it
Endif
Wend
```

ActiveWindow( )

Window Information
Returns the handle of the active window.

Syntax
ret = ActiveWindow( )

See Also
ActiveName( ), TopWindow( ), MouseWindow( ), IsWindow( ), WinGetPos( ), FocusWindow( ), FocusName( ), AttachName( )

Operation
This function returns the handle of the active window. Note that, if the window contains child windows (such as a dialog window containing edit controls and buttons), the handle of the parent is returned, not that of the child that has focus. To determine the handle of the window that has focus, use the FocusWindow( ) function.

Examples

```
a = 1 ; set up a counter
ret = ActiveWindow( ) ; get active window handle
While a = 1 ; an eternal loop
If ret <> ActiveWindow( ) ; if window changes focus
ret = ActiveWindow( ) ; update the reference
<Process Instructions> ; do something
Endif
Wend
```

AnchorSelect( )

Dialog Control
Selects Web objects that are created using the “A” HTML tag.

Syntax

ret = AnchorSelect("ControlId", "Options", x, y)

Variants

AnchorSelect("ControlId", "Options")

See Also

ImageSelect( )

Operation

This function processes an HTML anchor in the currently attached dialog box. The action is specified in "Options". The function parameters are as follows: "ControlId" Specifies the anchor text, which appears between the opening and closing anchor tags.

For example, in the hypertext link, <A HREF = “home.htm”> Home </A>, “Home” is the anchor text learned during the capture.

"Options" The options are as follows:

"left" Use the left mouse button to select the anchor.
"right" Use the right mouse button to select the anchor.
"middle" Use the middle mouse button to select the anchor.
"down" Press the mouse button down to select the anchor.
"up" Release the mouse button to select the anchor.
"doubleclick" Double-click the button to select the anchor.
"singleclick" Click the button once to select the anchor.
"control" Press the control key before clicking the button.
"shift" Press the shift key before clicking the button.
"with" Use in conjunction with "control" and "shift".
x , y These optional parameters specify where on the anchor the mouse button will be clicked. If omitted, the anchor is clicked in the topleft corner of the first line of anchor text.
The function returns 1 if the anchor is successfully selected, and it returns 0 if the anchor is not successfully selected.
When this command is generated by the Learn facility, the parentheses are omitted.

Note

The ControlId parameter is the control’s text identification (i.e., the anchor’s text), not an actual number.

Examples

```
Function Main
Attach "Program Manager PopupWindow"
ListViewCtrl "~1", "Internet Explorer", 'Left SingleClick'
Attach "http://compuweb.American Systems.com/ - Microsoft Internet
Explorer MainWindow"
Attach "~P~IEXPLORE.EXE~Edit~Compuweb Home - Microsoft Internet
```

```
Explorer"
EditClick "~0", 'Left SingleClick', 218, 7
Attach "~P~IEXPLORE.EXE~ComboBox~Compuweb Home - Microsoft
Internet Explorer"
ComboText "~0", "www.American Systems.com"
Attach "~P~IEXPLORE.EXE~Edit~Compuweb Home - Microsoft Internet
Explorer"
TypeToControl "Edit", "~0", "{Return}"
Attach "American Systems Corporation Home Page - Microsoft Internet
Explorer ChildWindow~1"
ImageSelect "American Systems Alliances", 'Left SingleClick'
Attach "American Systems Alliances - Microsoft Internet Explorer Child-
Window~1"
AnchorSelect "index.htm~6", 'Left SingleClick'
End Function
```

AppActivate( )

Window Control

Attaches to a window using the z-order to determine which window to activate.

Syntax

```
AppActivate( "windowname", Zorder )
```

Variants

```
AppActivate( "windowname" )
```

See Also

Attach( )

Operation

Brings an application window to the foreground. This command is similar to Attach( ), except it can be used to attach to applications that are overlapped in the z-order.

Examples

```
Function Main
Exec "NOTEPAD.EXE" ; start three copies of notepad
Exec "NOTEPAD.EXE"
Exec "NOTEPAD.EXE"
; Activate top-most copy
AppActivate( "Untitled - Notepad MainWindow", 1 )
Type "Number 1{Return}"
; Activate the one underneath
AppActivate( "Untitled - Notepad MainWindow", 2 )
Type "Number 2{Return}"
; Activate the bottom one
AppActivate( "Untitled - Notepad MainWindow", 3 )
Type "Number 3{Return}"
; Activate first one again ( now at the bottom, so Zorder=3 )
AppActivate( "Untitled - Notepad MainWindow", 3 )
Type "Number 1{Return}"
; Activate the third one ( now in the middle, so Zorder=2 )
AppActivate( "Untitled - Notepad MainWindow", 2 )
Type "Number 3{Return}"
; Activate the second one ( now at the bottom, so Zorder=3 )
AppActivate( "Untitled - Notepad MainWindow", 3 )
Type "Number 2{Return}"
End Function ; Main
```

ArrayPush( )

Miscellaneous

Inserts a value at the end of an existing array.

Syntax

ArrayPush(arrayname, value)

See Also

Var, ArraySize( )

Operation

This command is used to add an item to the end of an array declared using the Var command. The value added can be numeric or alphabetic.
ArrayPush( ) uses its own index, which is initially set to 1. If element 1 already exists, it will be overwritten.

Examples

Example 1:

```
Var myarray[]
ArrayPush( myarray, "Hello" )
ArrayPush( myarray, "World" )
Before = ArraySize( myarray ) ; Before contains 2
ArrayPush( myarray, "Everyone" )
After = ArraySize( myarray ) ; After contains 3
```

Example 2:

```
Var myarray[]
myarray[1] = "Hello"
myarray[2] = "World"
ArrayPush( myarray, "Everyone" )
; Note myarray[1] now equals "Everyone"
```

Arrays

Language
Use of Public, Private, and Local arrays.

Syntax

Var Variable1[ ] [, Variable2[ ], …, VariableN[ ]]
value = arrayname[<key>]

Variants

Public Variable1[ ] [, Variable2[ ], …, VariableN[ ]]

See Also

Const, Public, Var

Operation

An array is a collection of related data values referred to by a single variable name. Each value in an array is called an element. It is distinguished from other elements in the array by a key, which is enclosed in [ ] square brackets. The key indicates an element's position in an array.
EZ Test supports multi-dimensional arrays of strings and numbers. The arrays are implemented as associative lists, rather than vectors. Array elements are accessed using the syntax:

element = arrayname[<key>]

Where <key> is a list of string or numeric expressions separated by commas. For example:

```
surname = personal[ "fred", 10723 ]
list[ "fred", "smith" ] = 65
```

The maximum length of a key is 256 bytes.
The number of elements that an array has is dynamic and is only limited by available memory. It is not necessary, therefore, to dimension the array — but it must be declared.
Arrays that are declared as public can be accessed by child scripts that are executed using the Run( ) function. Public arrays must be declared outside of functions.
Use the Var statement to declare private and local arrays.
Arrays declared outside of functions are private to the current script. Arrays declared inside a function definition are local to that function. The maximum number of array

elements is only limited by available memory.

Examples

Example 1:

```
; "master" script
Public globala[ ], globalb[ ] ; declaration of public arrays
Function Main
FillArray( globala, "c:\*.bat" )
Run "Child"
End Function
; "child" script
c = 1
While c < ArraySize( globala )
MessageBox( "", globala[c] )
c=c+1
EndWhile
```

Example 2:

```
Var privateA[ ], privateB[ ] ; declaration of private arrays
Function Main
FillArray( privateA, "*.exe" ) ; fill privateA w/ .EXE filenames
ret=ArraySize(privateA) ; get size of array
Call Show
End Function
Function Show
MsgBox( "", privateA[ret - 1] ) ; value shown here
End Function
```

Example 3:

```

Function Main
Var locala[ ], localb[ ] ; declaration of local arrays
FillArray( locala, "*.exe" ) ; fill locala with .EXE filenames
ret=ArraySize(locala) ; get size of array
MsgBox( "", locala[ret - 1] ) ; local value shown here
Call Show
End Function
Function Show
Var locala[ ], localb[ ] ; declaration of local arrays
MsgBox( "", locala[ret - 1] ) ; no value shown here
End Function
```

ArraySize( )

Miscellaneous

Returns the number of elements in an array.

Syntax

```
ret = ArraySize( arrayname )
```

See Also

Var, FillArray( ), Delete ArrayName[Element]

Operation

This function returns the number of elements contained in the specified array. You must declare the array using the Var statement before using the ArraySize( ) function.

Examples

```
; find the number of .EXE files in a directory
var exefile[] ; declare the array variable
ret = FillArray( exefile,
"c:\windows\*.exe" ) ; fill the array with .EXE filenames
ret = ArraySize( exefile ) ; return the number of array
elements
MsgBox( "No. of EXE files", ret ); display the result
```

Asc( )

String Manipulation
Returns the ANSI value of a character.

Syntax
ret = Asc("String")
See Also
Chr( )
Operation
This function returns the ANSI value of string. If the string is more than 1 character, only the first character will be processed. The Asc( ) function automatically converts a numeric parameter to a string. Return values are in the range 0 to 255 (x00 to xFF).
Examples
ret = Asc("x") ; returns 120
ret = Asc("A") ; returns 65
ret = Asc("A LONG STRING") ; returns 65
ret = Asc("5") ; returns 53
ret = Asc(5) ; returns 53

Assignment

Language
Assigns values to variables.
Syntax
VarName = <Expression>
Variants
VarName[element] = <Expression>
VarName operator = <Expression>
VarName[element] operator = <Expression>
Operation
Assigns the result of <Expression> to the variable VarName. The optional operator can be one of the following:
+ Add the result of <Expression> to the current value of VarName.
- Subtract the result of <Expression> from the current value of VarName.
* Multiply the current value of VarName by the result of <Expression>.

/ Divide the current value of VarName by the result of <Expression>.
% Calculate the remainder when dividing the integer part of VarName by the integer part of <Expression>.
<< Bit-shift the current value of VarName by the result of <Expression> bits to the left.
>> Bit-shift the current value of VarName by the result of <Expression> bits to the right.
& Perform a bitwise "and" operation on the current value of VarName and the result of <Expression>.
^ Perform a bitwise "or" operation on the current value of VarName and the result of <Expression>.
| Perform a bitwise "not" operation on the current value of VarName and the result of <Expression>.
In all cases, the value of VarName is updated.
Examples
a = 100 ; a is 100
a+ = 100 ; a is increased by 100
b[10] = 20 ; element 10 of array b[] is 20
b[10] = c[10] ; element 10 of array b[] has same
; value as element 10 of array c[]

str1 = str2 ; string1 equals string2

Attach( )

Window Control
Attaches to a window.

Syntax

Attach("windowname", ["options"])

Variants

Attach(wndhandle, ["options"])

See Also

ActiveName( ), TopWindow( ), MouseWindow( ), FocusWindow( ), FocusName( ), AttachName( ), AttachWindow( ), Replay.AttachExact

Operation

This command causes *EZ Test* to attach to the window specified by "windowname". Attaching to a window makes it the recipient of all input from the script until the next Attach( ) function. The parameters are:

windowname The attach name of the window to attach to. The attach name can be any of the following formats:

"~<n>~<progname>~<classname>~<windowtitle>"
"~<n>~<progname>~<classname>~<windowtitle>~<pos>"
"<ObjectName>"

Where <n> can be:

N A normal window. The windowtitle parameter refers to the title of the window itself.
S The window is a child window with the same title as its parent window.
P The window has no title. The windowtitle parameter refers to the title of the parent window.
U An untitled window. No parent of the window has a title.
H The window is hidden.

<progname> Is the name of the window's executable module.
<classname> Is the window's class type. Wildcards can be used in this parameter. An asterisk (*) wildcards any number of characters, and a question mark (?) wildcards a single character.
<windowtitle> Is the window's title. This can be the title of the actual window or its parent, depending on the value of the <n> parameter. Wildcards can be used in this parameter. An asterisk (*) wildcards any number of characters, and a question mark (?) wildcards a single character.
<pos> Is the position of the window relative to other windows that have the same attach string.
<ObjectName> Is the name of the window as defined in the currently active Object Map. The ObjectName must start with an alphabetic character and cannot contain the '~' character.
wndhandle The window handle of the window to attach to.
options Specifies optional attach options. These can be:
exact Do exact attach name matching. This applies to raw attach names only. Use Significant fields to perform the same in the Object Map.
nearest Attach to the window with the closest matching attach name. This applies to raw attach names

only. Use wildcards to perform the same in the
Object Map.
activate Activate the window by giving it focus. Applies to
both raw attach names and Object Map names.
The function returns 1 if the attach was successful. It will return a runtime error if the attach was not successful.

Examples

Example 1:

```
; attach to Notepad's untitled edit window to enter text
Attach "~P~NOTEPAD.EXE~Edit~Untitled - Notepad"
Type "The quick brown fox"
; attach to Notepad's main window to select a menu item
Attach "~N~NOTEPAD.EXE~Notepad~Untitled - Notepad"
MenuSelect "File~Open..."
; attach to Notepad's Save warning dialog
Attach "~N~NOTEPAD.EXE~32770~Notepad"
Button "&No", 'Left SingleClick'
; attach to Notepad's Open File dialog
Attach "~N~NOTEPAD.EXE~32770~Open"
Button "Cancel", 'Left SingleClick'
```

Example 2:

```
; use the right mouse button to display the taskbar popup menu
Attach "~U~EXPLORER.EXE~Shell_TrayWnd~"
NCMouseClick 826, 7, 'Right Down'
NCMouseClick 826, 7, 'Right Up'
PopupMenuSelect "Properties"
Attach "~N~EXPLORER.EXE~32770~Taskbar Properties"
Button "Cancel", 'Left SingleClick'
```

Example 3:

```
; use the handle of the active window for the attach
hwnd = ActiveWindow( )
Attach hwnd
Maximize( )
```

Example 4:

```
; use the Object Map window name
Attach "Untitled - Notepad MultiLineEdit~1"
Type "the quick brown fox"
Attach "Untitled - Notepad MainWindow"
MenuSelect "File~Open..."
Attach "Notepad PopupWindow"
Button "@&No PushButton", 'Left SingleClick'
Attach "Open PopupWindow"
Button "@Cancel PushButton", 'Left SingleClick'
```

AttachAtPoint( )

Window Information
Returns the name of the window at a point.

Syntax

```
attachname = AttachAtPoint( x, y )
```

See Also

AttachMouseX( ), AttachMouseY( )

Operation

This function returns the name of the window at the point specified by the coordinates x, y. The value x, y is an absolute pixel value measured from the top-left corner of the screen.

Examples

```
Attach " popupwindow" ; attach to the taskbar
NCMouseClick 697, 9, 'Right Down' ; right mouse click
NCMouseClick 697, 9, 'Right Up'
```

PopupMenuSelect "Tile Vertically" ; tile all windows
leftwindow = AttachAtPoint(10, 10) ; window on the left
rightwindow = AttachAtPoint(790, 10) ; and window on the right

AttachMouseX( )

Mouse Information
Returns the x-position of the mouse within the currently attached window.

Syntax
Xpos = AttachMouseX( )
See Also
AttachMouseY( ), MouseX( ), MouseY( )
Operation
This function returns the x-position (horizontal) of the mouse pointer relative to the currently attached window.
Examples
; attach to the target application
Attach "~P~INVOICE.EXE~Edit~Raise Invoice"
; anchor the mouse on a field heading
TextSelect "Transaction No.", 'Left SingleClick'
; read the mouse X and Y coordinates
x = AttachMouseX( )
y = AttachMouseY( )
; capture the transaction number displayed 130 pixels to the right
transaction_no = CaptureBox("~P~INVOICE.EXE~Edit~Raise
Invoice", x+130, y-5, 200, 10)

AttachMouseY( )

Mouse Information
Returns the y-position of the mouse within the currently attached window.

Syntax
Ypos = AttachMouseY( )
See Also
AttachMouseX( ), MouseX( ), MouseY( )
Operation
This function returns the y-position (vertical) of the mouse pointer relative to the currently attached window.
Examples
; attach to the target application
Attach "~P~INVOICE.EXE~Edit~Raise Invoice"
; anchor the mouse on a field heading
TextSelect "Transaction No.", 'Left SingleClick'
; read the mouse X and Y coordinates
x = AttachMouseX( )
y = AttachMouseY( )
; capture the transaction number displayed 130 pixels to the right
transaction_no = CaptureBox("~P~INVOICE.EXE~Edit~Raise
Invoice", x+130, y-5, 200, 10)

AttachName( )

Window Information
Returns the attach name of a window.
Syntax
name = AttachName(hWnd)

Variants
name = AttachName( )
See Also
AttachWindow( )
Operation
This function returns the name of the window with window handle hWnd. If a window handle is not specified, the attach name of the currently attached window is returned.
Examples
ret = ActiveWindow( ) ; get window handle of parent
ParentName = AttachName(ret) ; get parent window name

AttachWindow( )

Window Information
Returns the handle of a window.
Syntax
ret = AttachWindow("windowname")
Variants
ret = AttachWindow( )
See Also
Attach( ), AttachMouseX( ), AttachMouseY( ), ActiveWindow( ), AttachName( )
Operation
This function returns the handle of the window specified by "windowname". If "windowname" is not specified, the handle of the currently attached window is returned. A value of 0 is returned if there is no attach.
When a window is created, Microsoft Windows assigns a handle (an integer value) to it. This handle is valid until the window is destroyed. If the application is restarted, the window handle may be different.
You can use this function to find a window handle to pass to a DLL call.
Examples
```
; use the attached window's handle to find the handle of its menu
Attach "My Application Window"
hwnd = AttachWindow( )
If hwnd = 0
Stop
Else
hmenu = DLLCALL( "USER.EXE", "GetMenu", hwnd )
If hmenu = 0
MessageBox( "Error", "Window does not have a menu", 0 )
Else
MessageBox( "Message", "Handle to the Menu = " + hmenu, 0)
Endif
Endif
```

Beep( )

Miscellaneous
Plays a note on the speaker.
Syntax
Beep(freq, time)
Variants
Beep(freq)
Beep( )
Beep
Operation
This command plays a note on the system speaker. If no parameters are specified, a message beep is performed.

The parameters are:
freq The frequency of the note to play.
time The length of time, in milliseconds, to play the note. If not specified, a default time of 20 ms is used.

Examples

```
Beep
Beep 440
Beep( 240, 40 )
```

BitMapSelect( )

Dialog Control
Clicks the mouse on a bitmap.

Syntax

ret = BitMapSelect("ImageMapName", "options")

See Also

TextSelect( )

Operation

This command moves the mouse pointer to the center of "ImageMapName" and performs the action specified by "options".
The "ImageMapName" parameter denotes an existing bitmap defined with the image map.
The "options" parameter can be any combination of the actions supported by the MouseClick( ) function:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the mouse button.
"singleclick" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with “control” and “shift”.
The function returns 1 if the bitmap is found, or it returns a 0 if the bitmap is not found.
BitMapSelects are generated automatically by the Learn facility if the Learn BitmapSelects option is selected in the Configure Learn dialog box.

Examples

```
; this example is based on a (user configurable) toolbar. Initially
; designed to look like this:
: each of the toolbar buttons is registered within the Image Map,
; enabling selection within a script:
BitMapSelect "Bold" SingleClick
; if the toolbar is subsequently redesigned as follows:
; the existing script continues to replay correctly
```

Boolean Expressions

Language
Tests the relationship between values.

Syntax

value1 operator value2

See Also

Operators

Operation

A Boolean expression allows you to test the relationship between two values. A Boolean expression takes the form:

value1 operator value2
If either side of the expression is numeric, both sides are promoted to numeric values before the operation is performed. The result is always a numeric value, 1 (true) or 0 (false). The operators are:
In addition to the above operators AND, OR, and NOT can be used to build compound expressions. Boolean expressions can be used with If...Else...Endif, Repeat...Until, and While...Endwhile commands to allow your script to make decisions about what to do next.

Examples

Example 1:

```
a = "EZ Test"
b = "EZ TEST"
If a = b
MessageBox( "Result", "Values match" )
Else
MessageBox( "Result", "Values do not match" )
Endif
```

Example 2:

```
a = "EZ Test"
b = "EZ TEST"
If a <> b
MessageBox( "Result", "Values do not match" )
Else
MessageBox( "Result", "Values match" )
Endif
```

= Equals True if value1 is equal to value2.
<> (or ! =) Not Equal To True if value1 is not equal to value2.
> Greater Than True if value1 is greater than value2.
< Less Than True if value1 is less than value2.
> = Greater Than or Equal To True if value1 is greater than or equal to value2.
< = Less Than or Equal To True if value1 is less than or equal to value2.

Example 3:

```
a = 100
b = 200
If a > b
MessageBox( "Result", "a is more than b" )
Else
MessageBox( "Result", "a is less than or equal to b" )
Endif
```

Example 4:

```
a = 100
b = 200
If a < b
MessageBox( "Result", "a is less than b" )
Else
MessageBox( "Result", "a is not less than b" )
Endif
```

Example 5:

```
a = 100
```

```
b = 200
If a <= b
MessageBox( "Result", "a is less than or equal to b" )
Else
MessageBox( "Result", "b is more than a" )
Endif
```
Example 6:
```
a = 100
b = 200
c = 300
d = 400
If a < = b AND c <= d
MessageBox( "Result", "b is more than a and d is more than c" )
Else
MessageBox( "Result", "a is more than b or c is more than d" )
Endif
```

Break

Program Flow
Exits the current loop and continues execution on the line following the loop.

Syntax
Break

See Also
Continue, Do...Loop While, Repeat...Until, While...Wend

Operation
This command immediately exits the current loop and continues script execution at the line following the loop. The statement can only be used inside a loop.

Examples
```
i = 0
While I <> 10 ; while i is not 10
Print i ; print 0, 1, 2, ...
If i = 5 ; on i = 5
Break ; break
Endif
i = i+1 ; increment i
Wend ; end of loop
MsgBox( "Break at ", i )
```

BrowserToolbarCtrl( )

Dialog Control
Selects options from a Web browser's toolbar.

Syntax
ret = BrowserToolbarCtrl("Button", "Options" , [x, y])

Operation
This function is used to make a selection from a standard Web browser's toolbar — such as that found in Netscape.
In cases where the text on a similarly functioning toolbar button is different from browser-to-browser, the learned text will work against other browsers.
The parameters are as follows:
"Button" The button to select from the tool bar. This value is the text of the button.
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.

"up" Release the mouse button.
"double" Double-click the mouse button.
"click" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Used in conjunction with "control" and "shift".
x , y These optional parameters specify where on the control the mouse button will be clicked. If omitted, the button is clicked in the center.
The function returns 1 if the selection is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, you must use parentheses.

Examples

Example 1:

```
; In Internet Explorer, select the Home Button
Function Main
Attach "Compuweb Home - Microsoft Internet Explorer MainWindow"
BrowserToolbarCtrl "Home", 'Left SingleClick'
End Function ; Main
```

Button( )

Dialog Control
Processes a button control.

Syntax

ret = Button("ControlId", "Options", x, y)

Variants

Button("ControlId", "Options")

See Also

CheckBox( ), ComboBox( ), ComboText( ), EditText( ), ListBox( ), RadioButton( ), ScrollBar( )

Operation

This function processes a push button in the currently attached dialog box. The action is specified in "Options". The function parameters are as follows:
"ControlId" Specifies the button label, such as “OK”, “Cancel”, “Yes”, “No”. If the ControlId is numeric (for example, if the button does not contain text), it represents the index value of the button, such as “~1”.
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the button.
"singleclick" Click the button once.
"control" Press the control key before clicking the button.
"shift" Press the shift key before clicking the button.
"with" Use in conjunction with "control" and "shift".
x , y These optional parameters specify where on the control the mouse button is clicked. If omitted, it is clicked in the center.
The function returns 1 if the button is successfully clicked, and it returns 0 if the button

is not successfully clicked.
When this command is generated by the Learn facility, the parentheses are omitted.

Examples

```
; close Notepad and abandon changes to the document
Attach "~N~NOTEPAD.EXE~Notepad~Untitled - Notepad"
MenuSelect "&File~E&xit"
Attach "~N~NOTEPAD.EXE~#32770~Notepad"
Button "&No", "SingleClick"
```

ButtonDefault( )

Window Information
Determines if a button is the default button.

Syntax

```
ret = ButtonDefault( hCtrl )
```

See Also

ControlFind( ), IsWindow( )

Operation

This function determines if the button whose window handle is hCtrl is the default button. The window handle can be obtained by using the ButtonFind( ) function — one of the ControlFind( ) group of functions.
The function returns 1 if the specified button is the default and returns 0 if the button is not the default.

Examples

```
; this script attaches to a dialog and tabs through the controls
; until the Cancel button is the default control
Attach "MyDialog" ; attach to the dialog
hCtrl = ButtonFind( "Cancel" ) ; get handle of Cancel button
While ButtonDefault( hCtrl ) != 1 ; while Cancel button is not
; default
Attach FocusName( ) ; attach to current control
Type "{Tab}" ; press {Tab} key
Wend
```

CalendarCtrl( )

Dialog Control
Sets the date on a Windows Month calendar control.

Syntax

```
ret = CalendarCtrl ( "ControlID" , "StartDateVal", "EndDateVal" )
```

CalendarFind( ), CalendarRange( ), CalendarToday( ), DateTimeCtrl( )

Operation

This function sets the date of the Windows Month calendar control specified with the ControlId parameter. The StartDateVal and EndDateVal parameters may be used to select a range of dates. The parameters are:
"ControlId" The index value of the calendar control.
"StartDateVal" A value for the start date of any range of dates selected in a calendar control. The format must be "mm-dd-yyyy" for dates.
"EndDateVal" A value for the end date of any range of dates selected in a calendar control. The format must be "mm-dd-yyyy" for dates.
ret A value of 1 is returned if the operation is successful. A value of 0 is returned for failure.

Examples

```
Function Main
Attach "Microsoft Control Spy - Date and Time Picker PopupWindow"
```

```
DateTimeCtrl "~1", "8-4-1999"
Attach "Microsoft Control Spy - Month Calendar PopupWindow"
CalendarCtrl "~1", "8-1-1999", "8-1-1999"
CalendarCtrl "~1", "8-7-1999", "8-7-1999"
CalendarCtrl "~1", "8-14-1999", "8-14-1999"
CalendarCtrl "~1", "8-8-1999", "8-8-1999"
CalendarCtrl "~1", "8-8-1999", "8-14-1999"
End Function ; Main
```

CalendarRange( )

Dialog Control
Returns the start and end date of a range of dates selected in a Windows Calendar control.

Syntax

CalendarRange (hCtrl , StartDate, EndDate)

See Also

CalendarCtrl( ), CalendarFind( ), CalendarToday( ), DateTimeCtrl( )

Operation

This function returns the start and end dates selected in the calendar control specified in the hCtrl parameter. The function also returns the total number of days spanned within the indicated range. The control default is 7. If the StartDate and EndDate parameter are the same, then the number of days returned is 1. The parameters are:
hCtrl The handle of the control.
StartDate A variable that receives the start date of the selected dates.
EndDate A variable that receives the end date of selected dates.
DaySpan A value indicating the total number of days in the selected range.

Examples

```
Function Main
Attach( "Microsoft Control Spy - Month Calendar PopupWindow" )
hCtrl = CalendarFind( "~1" )
TodayVal = CalendarToday( hCtrl )
fmt = FormatDate( "dd-mm-yyyy" , TodayVal )
msgbox( "hCtrl = " + hCtrl , "" )
msgbox( "Today = " + fmt , "" )
DayRange = CalendarRange( hCtrl , StartDate , EndDate )
fmtS = FormatDate( "dd-mm-yyyy" , StartDate )
fmtE = FormatDate( "dd-mm-yyyy" , EndDate )
msgbox( "DayRange = " + DayRange , fmtS + ", " + fmtE )
End Function ; Main
```

CalendarToday( )

Dialog Control
Returns the “today” date of a Windows Calendar control.

Syntax

Ret = CalendarToday (hCtrl)

See Also

CalendarCtrl( ), CalendarFind( ), CalendarRange( ), DateTimeCtrl( )

Operation

This function returns the active today setting of a Windows calendar control. This value can then be used in conjunction with *EZ Test*’s other commands found in the Date/Time command group (for example, FormatDate( )). The parameters are:
hCtrl The handle of the control.
ret The numerical representation of the today setting.
This value can then be used in the FormatDate( )

command to display the information.

Examples

```
Function Main
Attach( "Microsoft Control Spy - Month Calendar PopupWindow" )
hCtrl = CalendarFind( "~1" )
TodayVal = CalendarToday( hCtrl )
fmt = FormatDate( "dd-mm-yyyy" , TodayVal )
msgbox( "hCtrl = " + hCtrl , "" )
msgbox( "Today = " + fmt , "" )
DayRange = CalendarRange( hCtrl , StartDate , EndDate )
fmtS = FormatDate( "dd-mm-yyyy" , StartDate )
fmtE = FormatDate( "dd-mm-yyyy" , EndDate )
msgbox( "DayRange = " + DayRange , fmtS + ", " + fmtE )
End Function ; Main
```

Cancel( )

Synchronization

Cancels an event.

Syntax

```
Cancel( EventId )
```

Variants

```
Cancel( All )
Cancel
```

See Also

MakeEvent( ), Whenever

Operation

This function cancels the event specified by the EventId parameter. It is used to cancel Whenevers that are no longer required.

The Cancel("All") variant cancels all active events. The Cancel variant on its own can be used inside a Whenever event handler to cancel the triggered event. This function has no return value.

Examples

```
Whenever "npmove" call NPMOVE ; set up three window
Whenever "npmin" call NPMIN ; whenevers
Whenever "nprestore" call NPRESTORE
pause 10 ; after 10 seconds
Cancel( "npmove" ) ; cancel this whenever
suspend
Function NPMOVE;
MsgBox( "", "moved" )
End Function;
Function NPMIN;
MsgBox( "", "minimize" ) ; cancel this whenever after
Cancel ; it has triggered once
End Function;
Function NPRESTORE;
MsgBox( "", "restored" )
End Function;
```

Capture( )

Window Information

Returns the text currently displayed in a window.

Syntax

```
ret = Capture( "windowname", "style" )
```

Variants

```
ret = Capture( "windowname" )
```

Operation

This function returns all the text currently displayed inside the client area of the window specified by the "windowname" parameter. The parameters are:
"windowname" The attach name of the window to capture. This window and all the windows within its client area are captured.
The "style" parameter can be any combination of the following values:
"lf" Separate with a carriage return / line feed each “textout” that the application performs.
"noerase" Do not erase the contents of the window before refreshing it. This helps reduce “flicker” when capturing text.

Examples

Example 1:

```
tips = Capture( "~N~NOTEPAD.EXE~Notepad~Tips.txt - Notepad" )
MsgBox( "Tips", tips )
```

Example 2:

```
details = Capture( "orderdetails", "lf noerase" )
If FindStr( details, "O/N 1234" ) <> 0 ; search for order number
< process order >
Else
WriteLine( "audit.log", "O/N 1234 not found" )
Endif
```

CaptureBox( )

Window Information
Returns the text currently displayed in an area of a window.

Syntax

```
ret = CaptureBox( "windowname", x, y, width, height )
```

See Also

Capture( ), WindowText( )

Operation

This function captures the text currently displayed in a rectangular area of the window specified by "windowname". The parameters are:
"windowname" The attach name of the window from which to capture.
x The x-coordinate of the left edge of the area to capture, relative to the left edge of "windowname".
y The y-coordinate of the top edge of the area to capture, relative to the top edge of "windowname".
width The width of the area to capture, in pixels.
height The height of the area to capture, in pixels.

Examples

Example 1:

```
; attach to the target application
Attach "~P~INVOICE.EXE~Edit~Raise Invoice"
; anchor the mouse on a field heading
TextSelect "Transaction No.", 'Left SingleClick'
; read the mouse X and Y coordinates
x = AttachMouseX( )
y = AttachMouseY( )
; capture the transaction number displayed 130 pixels to the right
transaction_no = CaptureBox( "~P~INVOICE.EXE~Edit~Raise Invoice",
x+130, y-5, 200, 10 )
```

Example 2:

```
; this example captures the screen title from "MyApp" and pastes a
; MakeEvent statement into current script ("MyScript") each time
; the developer presses {F12}
Function Main
Whenever "Paste" Call Paste
End Function ; Main
Function Paste
```

```
Title = CaptureBox("MyApp", 0,140,1000,10)
EventName = Left( Title, 10 )
MyPaste = EventName + ' = MakeEvent( "Screen", "MyApp", "' +
Title + '", "0, 140, 1000, 10" )'
SendToEditor( MyPaste, "MyScript")
SendToEditor( chr(13)+chr(10), "MyScript" )
SendToEditor( 'Wait(30, "", "' + EventName + '")', "MyScript" )
SendToEditor( chr(13)+chr(10), "MyScript" )
End Function ; Paste
```

CaretPosX( )

Window Information
Returns the x-position of the Windows caret within the current window.

Syntax

```
ret = CaretPosX( )
```

See Also

CaretPosY( )

Operation

This function returns the x-position of the Windows caret within the current window. The caret is actually the text cursor, and the position returned is relative to the left edge of the client area.
If the window does not contain a caret, the function returns a value of 0.

Examples

```
Repeat
ret = CaretPosX ( ) ; Get the x-position of the caret
Pause 1 "ticks" ; Wait a while
Until ret = 100 ; Exit loop when it is at 100 pixels
```

CaretPosY( )

Window Information
Returns the y-position of the Windows caret within the current window.

Syntax

```
ret = CaretPosY( )
```

See Also

CaretPosX( )

Operation

This function returns the y-position of the Windows caret within the current window. The caret is actually the text cursor, and the position returned is relative to the top edge of the client area.
If the window does not contain a caret, the function returns a value of 0.

Examples

```
Attach( "Untitled - Notepad Edit~1" )
While CaretPosY( ) < 100 ; Check y-position of the caret
Type "{Return}" ; Go down to the next line
Wend
Type "Past Y Position 100{Return}" ; Indicate that we got there
```

Cesc( )

String Manipulation
Converts 'C' escape sequences into characters

Syntax

```
ret = Cesc( "string" )
```

See Also

Asc( ), Chr( )

Operation

This function converts 'C' style escape sequences within string into characters. The following codes are supported:
"\n" New line character (LF, ascii code {x0a})
"\r" Return character (CR, ascii code {x0d})
"\t" Horizontal tab character (HT, ascii code {x09})
"\v" Vertical tab character (VT, ascii code {x0b})
"\f" Form feed character (FF, ascii code {x0c})
"\a" Bell character (BEL, ascii code {x07})
"\b" Backspace character (BS, ascii code {x08})
"\"" " (double quote) character
"\'" ' (single quote) character
"\\" \ (backslash) character
"\x0f" Hexadecimal notation
"\o12" Octal notation

Examples

Example 1:
heading = Cesc("Name:\tAddress:"); returns "Name: Address:"
Example 2:
s = Cesc("EZ Test\n\"Software Testing Software\"")
MsgBox("Slogan", s)
Example 3:
x = Cesc("\x32\x33\x34") ; returns "234"
Note
Single and double quote characters in strings can also be represented by “double quotes”:
"The ""quick"" fox"; expands to The "quick" fox
'The "quick" fox'; expands to The 'quick' fox

Chain( )

Program Flow
Executes another script. The caller and called scripts run concurrently.

Syntax

ret = Chain("scriptname" [, "parameters"])

Variants

Chain("scriptname")

See Also

Run( )

Operation

This function launches another script. When the second script is started, the original script continues processing from the next line and both scripts run concurrently.
The "parameters" may be retrieved by the receiving script using the CmdLine( ) function.
Child scripts launched by the Chain() command cannot use a Public variable that is implemented in the launching/driver script. Instead, it is recommended you pass the variable as a parameter to the Chain command, and retrieve its value in the child script using the CmdLine command. For more information see the CmdLine command.

Examples

Example 1:
<Instructions> ; process these instructions
<Instructions> ; and these
<Instructions> ; and these
Chain("Account Update Test") ; launch this test script
<Instructions> ; process these instructions
<Instructions> ; and these
Chain("Invoice Created Test") ; launch this test too
Example 2:
Function Main ; This is the driver script

```
a = "hello"
Chain("Child Script", a)
End Function
Function Main ; This is the child script
a = CmdLine( 2 ) ;The first parameter received is
always the name of the script
MsgBox "", a, 'ok'
End Function
```

ChDir( )

File Access
Changes the current working directory.

Syntax

```
ret = ChDir( "newdirectory" )
```

Variants

```
CD( )
```

See Also

MakeDir( ), RemoveDir( )

Operation

This function allows the script to change the working directory or folder. The return value, ret, contains the name of the current directory or folder.

Examples

```
; change to another directory or folder
ret = ChDir( "c:\Sam's working folder" )
<Instructions> ; do something
; return to the previous directory or folder
CD( ret )
```

Check( )

Checks
Runs a check on the target application.

Syntax

```
result = Check( "CheckName" )
```

See Also

CheckExists( ), MakeCheck( ), Replay.CheckCurrentAttach, Replay.CheckExit, Replay.CheckRetry, Replay.CheckTimeout

Operation

This functions executes the CheckName check from the Checks Map. Checking compares the target application's actual state to the expected status defined within the check. If the actual and expected states match, the check passes. If they are different, the check fails.
The function returns 1 if the check passes and 0 if it fails.
If logging is on, the check result is written to the Log. If the check fails, both expected and actual states are written to the Log, so a differences analysis can be carried out.
If Replay.CheckExit is set to 1, a runtime error is generated on check failure.

Examples

```
ClockReset "LoadTime" ; reset clock
Attach "PopupWindow" ; select Run dialog
Button "Start", 'Left SingleClick'
PopupMenuSelect "Run..."
Attach "Run PopupWindow"
ComboText "&Open:", "MyApp" ; enter application name
Button "OK" SingleClick ; and run it
ClockStart "LoadTime" ; start a clock
Attach "MyApp MainWindow" ; attach to application
ClockStop "LoadTime" ; stop the clock
```

Check("LoadTime") ; check loading time
Check("MyApp Main Menu") ; check application menu
MenuSelect "File~Open..." ; select menu item
Check ("MyApp Open Dialog") ; check File~Open dialog
Attach "Open PopupWindow" ; attach to dialog
Button "Cancel", 'Left SingleClick'; close it
Attach "MyApp MainWindow" ; attach to main window
WinClose ; and close it

CheckBox( )

Dialog Control
Processes a check box control.

Syntax

ret = CheckBox("ControlId", "Options", x, y)

Variants

CheckBox("ControlId", "Options")

See Also

Button( ), ComboBox( ), ComboText( ), EditText( ), ListBox( ), RadioButton( ), ScrollBar( )

Operation

This function processes a check box control in the currently attached dialog box. The action is determined by the "Options" parameter. The parameters are:
"ControlId" Specifies the label shown to the side of the check box. If the ControlId is numeric, it represents the index value of the control.
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press and hold the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the radio button.
"singleclick" Click the radio button once.
"control" Press and hold the control key before clicking the radio button.
"shift" Press and hold the shift key before clicking the radio button.
"with" Used in conjunction with “control” and “shift”.
"on" Select the check box.
"off" Deselect the check box.
x, y These optional parameters specify where on the control the mouse button will be clicked.
The function returns 1 if the control is selected successfully and returns 0 if it is not. When this command is generated by the Learn facility, the parentheses are omitted.

Examples

; from the communications settings dialog select the correct
; communications speed
Attach "~N~KERNEL32.DLL~#32770~Configure Session"
Checkbox "&Use fast communications", "On"

CheckExists( )

Checks
Verifies the existence of a check.

Syntax

ret = CheckExists("CheckName")

See Also

Check( ), MakeCheck( )

Operation

The function verifies the existence of an a check before the MakeCheck( ) command is used to generate the new check at runtime. The CheckName parameter is the check name that will be searched for in the *EZ Test* database. If the check already exists, the function returns a value of 1. If the check name does not exist, the function returns and value of 0, and the MakeCheck command can be used to create the check at runtime.

Examples

```
If CheckExists( "New" ) = 0 ; Check does NOT exist
; Create the check
MakeCheck( "Template" , "New" , "Descr" )
Endif
; Execute the check
Check( "New" )
```

Chr( )

String Manipulation

Converts a value into an ANSI character.

Syntax

ret = Chr(value)

See Also

Asc( )

Operation

This function takes a numeric value from 0 to 255 and returns its corresponding ANSI character.

Examples

```
the_char = chr( 120 ) ; returns "x"
the_char = chr( 247 ) ; returns "÷"
```

ClipBoard( )

Miscellaneous

Captures text from, or places text into, the Clipboard.

Syntax

ret = ClipBoard("Text to insert")

Variants

ret = ClipBoard( )

Operation

This function places "Text to insert" on the clipboard and returns the previous contents. If the clipboard was empty or contained non-textual data, a null ("") is returned. To return the current contents of the clipboard without replacing it with new text, use the ClipBoard( ) function with no parameter.

Examples

Example 1:

```
a = ClipBoard( ) ; get the contents of the Clipboard
MsgBox( "The Result", a ) ; display the result
```

Example 2:

```
ClipBoard( "This is new text to be placed on the Clipboard" )
```

Clng( )

Number Manipulation

Converts a number to a long integer.

Syntax

ret = Clng(value)

Variants

ret = Chlong(value)

See Also

Int( ), Fix( )

Operation

This function converts a value to a long integer. The value is rounded before the conversion. This function has a maximum value of 2 billion.

Examples

ret = Clng(10.12) ; returns 10
ret = Clng(10.55) ; returns 11
ret = Clng(-10.50) ; returns -11
ret = Clng(12345678.90) ; returns 12345679

Clock( )

Clocks

Retrieves the current value of a clock in milliseconds.

Syntax

ClockVal = Clock("ClockName")

See Also

ClockReset( ), ClockStart( ), ClockStop( )

Operation

This function retrieves the current value (in milliseconds) of the clock specified by the ClockName parameter. All clocks are global to all scripts — so a script run from another script is able to read a clock started by its parent and vice versa.

Examples

Example 1:

```
ClockReset "MyClock" ; reset a clock
ClockStart "MyClock" ; start the clock
Run "TimeTest" ; run another script
the_time = Clock( "MyClock" ) ; check total time to execute
; "TimeTest" and return
```

Example 2:

```
ClockReset "myclockcheck" ; reset a clock
ClockStart "myclockcheck" ; start the clock
Run "timetest" ; run another script
ClockStop "myclockcheck" ; stop the clock
Check( "myclockcheck" ) ; check total time to execute
the_time = clock( "myclockcheck" ) ; check total time to execute
MessageBox( "clock check", the_time , 'ok' ); display the time
```

ClockReset( )

Clocks

Resets a clock to zero.

Syntax

ClockReset("ClockName")

Variants

ClockReset("ClockCheckName")

See Also

Clock( ), ClockStart( ), ClockStop( )

Operation

This function resets the clock specified by the "ClockName" parameter to zero. To reset a clock used within a check, use the clock check's name. All clocks are global to all scripts — so a script run from another script can reset a clock started by its parent and vice versa. This function has no return value.

Examples
ClockReset "MyClockCheck" ; reset a clock
ClockStart "MyClockCheck" ; start the clock
Run "TimeTest" ; run another script
ClockStop "MyClockCheck" ; stop the clock
Check("MyClockCheck") ; check total time to execute
; "TimeTest" and return

ClockStart( )

Clocks
Starts or resumes a clock.
Syntax
ClockStart("ClockName")
Variants
ClockStart("ClockCheckName")
See Also
Clock( ), ClockReset( ), ClockStop( )
Operation
This function starts the clock specified by the "ClockName" parameter. If the named clock is already running, no action is taken. To start a clock used within a check, use the clock check's name.
This function has no return value.
All clocks are global to all scripts — so a script run from another script can restart a clock started by its parent and vice versa.
Examples
ClockReset "MyClockCheck" ; reset a clock
ClockStart "MyClockCheck" ; start the clock
Run "TimeTest" ; run another script
ClockStop "MyClockCheck" ; stop the clock
Check("MyClockCheck") ; check total time to execute
; "TimeTest" and return

ClockStop( )

Clocks
Stops a clock.
Syntax
ClockStop("ClockName")
Variants
ClockStop("ClockCheckName")
See Also
Clock( ), ClockReset( ), ClockStart( )
Operation
This function stops the clock specified by the "ClockName" parameter. The clock is not reset to zero and may be re-started from its current value. To control a clock used within a check, use the clock check's name. This function has no return value.
All clocks are global to all scripts — so a script run from another script can stop a clock started by its parent and vice versa.
Examples
ClockReset "MyClockCheck" ; reset a clock
ClockStart "MyClockCheck" ; start the clock
Run "TimeTest" ; run another script
ClockStop "MyClockCheck" ; stop the clock
Check("MyClockCheck") ; check total time to execute
; "TimeTest" and return

Close( )

File Access
Closes a file that was previously opened using the Open( ) function.

Syntax
Close("filename")

Variants
Close(variable)

See Also
Open( ), Read( ), Write( )

Operation
This function closes a file that was previously opened using the Open( ) function or one of the read/write functions.

Examples
Example 1:
Close("c:\scripts\customer.dat")
Example 2:
File = "c:\scripts\customer.dat"
Close(File)

CloseCom( )

Serial Communications
Closes the PC's specified COM port.

Syntax
ret = CloseCom(Port)

See Also
OpenCom( ), PurgeCom( ), ReadCom( ), WriteCom( )

Operation
This function closes the specified COM port on the PC.
The options are:
Port A number from 1 - 255.
The function has the following return values:
1 Success
0 Failure
-1 Bad Port

Examples
```
Var y[ ]
Var x[ ]
y[1] = 41
y[2] = 41
y[3] = 41
y[4] = 41
y[5] = 41
z = OpenCom( 4, 9600, 8, 0, 1 ) ;open up COM port 4
z = PurgeCom ( 4 ) ;purge data in COM 4
z = WriteCom( 4, y, 5 ) ;writes 5 bytes of data to COM 4
z = ReadCom ( 4, x, 5, 1 ) ;reads back 5 bytes of data
z = CloseCom( 4 ) ;Close COM port 4
Print x[1]
Print x[2]
Print x[3]
Print x[4]
Print x[5]
```

CmdLine( )

Miscellaneous

Returns the command line string.

Syntax

command_string = CmdLine(word)

Variants

command_string = CmdLine( )

Operation

This function returns the values entered on the command line when a script is executed using RunAWL or the parameters passed in a Run( ) or Chain( ) function. The word parameter specifies which word to extract from the string; the first word is always the script name itself. If the word parameter is omitted, the entire command line string is returned.

If a script is executed from the editor or spawned from another script using the Run or Chain commands, the CmdLine( ) function returns the script name.

Examples

Example 1:

; if the following command is entered from the Windows Run dialog:
Runawl names tom dick harry
; then, in the "names" script:
CmdLine(1) ; returns "names"
CmdLine(3) ; returns "dick"
CmdLine( ) ; returns "names tom dick harry"

Example 2:

Chain "do_update", "Tom Dick Harry"
; in the "do_update" script
CmdLine(1) ; returns "do_update"
CmdLine(3) ; returns "Dick"

ComboBox( )

Dialog Control

Selects a string from a combo box.

Syntax

ret = ComboBox("ControlId", "Item", "Options")

Variants

ComboBox("ControlId", "Item")
Combobox("ControlId", "@ItemPosition", "Options")
Combobox("ControlId", "@ItemPosition")

Note

The CmdLine() function processes spaces in parameters as delimiters. If the parameter "window attach" is passed, for example, this function treats it as two separate parameters: "windows" and "attach".

See Also

Button( ), CheckBox( ), ComboText( ), EditText( ),ListBox( ), RadioButton( ), ScrollBar( )

Operation

This function selects the item specified by the "Item" parameter from the combo box specified by the "ControlId" parameter in the currently attached dialog.

The parameters are:

"ControlId" Specifies the index value of the combo box; "~1" for the first combobox, "~2" for the second, etc.

"Item" Determines the item to select from the combo box. This value can be literal or by position in the list. To select the third item in the list use "@3" in place of a literal value.

"Options" Determines how the item in the combobox is selected. The options are "SingleClick" or "DoubleClick".

The function returns 1 if the item is selected successfully and generates a runtime error if it is not. See the On Error command for information on processing runtime errors in

scripts.
When this function is generated by the Learn facility, the literal value (not position) of the item selected is inserted into the script; the mouse click and the parentheses are omitted.

Examples

Example 1:

```
; select the files to display from the File Open dialog
Attach "~N~KERNEL32.DLL~#32770~File Open"
ComboBox "~1", "Access Files"
; select the drive from the second combobox
ComboBox "~2", "h: host for c"
```

Example 2:

```
; always select the third item from the first combobox
Attach "~N~KERNEL32.DLL~#32770~File Open"
ComboBox "~1", "@3", "SingleClick"
```

ComboText( )

Dialog Control
Enters text into the edit control of a combo box.

Syntax

```
ret = ComboText( "ControlId", "Text" )
```

Variants

```
ComboText "ControlId", "Text"
```

See Also

Button( ), CheckBox( ), ComboBox( ), EditText( ), ListBox( ), RadioButton( ), ScrollBar( )

Operation

This function enters the text specified by the "Text" parameter into the edit control of the combo box specified by the "ControlId" parameter within the currently attached dialog box.
The parameters are:
"ControlId" Specifies the index value of the combo box; "~1" for the first combobox, "~2" for the second, etc.
"Text" Specifies the text to enter into the edit control of the selected combo box. This value can be literal or variable data.
The function returns 1 if the operation is successful and returns 0 if it is not.
When this command is generated by the Learn facility, the parentheses are omitted.

Examples

```
; from the Run dialog, execute a copy of the Address Book program
Attach "~N~EXPLORER.EXE~#32770~Run"
ComboText "~1", "Address"
Button "OK", "SingleClick"
```

Compare( )

String Manipulation
Compares the contents of two strings.

Syntax

```
ret = Compare( string1, string2, case )
ret = Compare( string1, string2 )
```

Operation

This function compares the contents of two strings. The parameters are as follows:
string1 First string to compare.
string2 Second string to compare.
case Optional setting to compare upper or lowercase characters.
If set to 1, case is ignored; if set to 0, comparison is case

sensitive. The default value is 1 (ignore case).
The Compare( ) function automatically converts numeric parameters to strings.
The function returns 0 if the two strings are equal, or it returns a positive number to indicate precisely where in the string the comparison failed.

Examples

ret = Compare("abc", "ABC") ; returns 0, (case ignored)
ret = Compare("abc", "ABC", 0) ; returns 1-fails at 1st character)
ret = Compare("abcd", "abcf") ; returns 4-fails on 4th character)
ret = Compare("abc", "abcdefg") ; return 4-fails on 4th character)
ret = Compare(this, that) ; compares current value of this
; with current value of that
; returns 0 if this and that are
; uninitialized
this = "this"
that = "that"
ret = Compare(this, that) ; returns 3

Const

Language
Declares a Constant.

Syntax

Const <ConstId> = <Constant>

See Also

Arrays, Var

Operation

Constants are like private variables, but their value cannot be changed when the script is run. Constants are always private to the rest of the script; they cannot be local to a function nor public to child scripts. They can be either string or numeric, but must be declared before they can be used. Once declared, a constant cannot be redefined.

Examples

Const TRUE = 1
Const FALSE = 0
Const FileName = "session.log"

Continue

Program Flow
Returns to the top of a loop, ignoring following statements within the loop.

Syntax

Continue

Variants

Continue

See Also

Break, Do...Loop While, While...Wend

Operation

This function performs the next iteration of the current loop immediately, ignoring any statements that occur between the Continue function and the end of the loop. The statement can only be used inside a loop.
The Continue function has no return value; the variant Continue can be used with no difference.

Examples

i = 0
While I <> 10 ; while i is not 10
i = i+1 ; increment i
If I%2 = 0 ; if i is even
Continue ; back to the top
Endif

```
MsgBox( "i is", I ) ; show odd numbers only
Wend ; end of loop
```

ControlFind( )

Window Information
Returns the window handle of a control.

Syntax

```
hCtrl = ButtonFind( "ControlId" )
hCtrl = CheckboxFind( "ControlId" )
hCtrl = RadioFind( "ControlId" )
hCtrl = ScrollbarFind( "ControlId" )
hCtrl = ComboFind( "ControlId" )
hCtrl = EditFind( "ControlId" )
hCtrl = ListboxFind( "ControlId" )
hCtrl = ListviewFind( "ControlId" )
hCtrl = HeaderFind( "ControlId" )
hCtrl = TabFind( "ControlId" )
hCtrl = ToolbarFind( "ControlId" )
hCtrl = UpdownFind( "ControlId" )
hCtrl = GridFind( "ControlId" )
hCtrl = DateTimeFind( "ControlId" )
hCtrl = CalendarFind( "ControlId" )
```

See Also

AttachWindow( )

Operation

This group of functions returns the window handle of the control specified by the ControlId parameter within the currently attached window.
The ControlId parameter specifies the control's label (see Control Labels for more information). The parameter can specify the control's label, attach name, or position. For example, if a button named "Properties", with the following attach name:
"~N!PROGRAM.EXE~Button~&Properties"
Then, either of the following commands would replay the button:

```
ButtonFind( "&Properties" )
ButtonFind( "@~N~PROGRAM.EXE~Button~&Properties" )
```

You can also use the "!" character to get the handle of a control by position alone. For example, using ButtonFind("!~3") returns the handle of the third button, regardless of its text.
The functions return 0 if the specified control is not found.

Examples

Example 1:

```
Attach "~N~EZ TESTDEMO.EXE~AfxFrameOrView40~Customer Invoice"
hCtrl = EditFind( "~2" )
MsgBox( "Control Handle", hCtrl )
hCtrl = EditFind( "Tax Rate %" )
MsgBox( "Control Handle", hCtrl )
hCtrl = ButtonFind( "&Print" )
MsgBox( "Control Handle", hCtrl )
```

Example 2:

```
; Return the handle of the third button on the dialog
Attach "~N~EZ TESTDEMO.EXE~Afx~EZ TESTDemo - Customer Invoice"
hCtrl = buttonfind( "!~4" )
MsgBox( "control Handle", hctrl )
```

Control Labels

Language
Identifies controls in a dialog.

Syntax

Control <Control Label> "Action"

Operation

Controls in dialog windows may be referenced in any of the following ways:
<id> By numeric control id.
"<name>" By unique control name.
"~<pos>" By position of the control relative to other unnamed controls of the same class in the same window.
"<name>~<pos>" By the position of the control relative to the other controls of the same class and with the same name in the same window.
"@<id>" By the index of the control as stored in the API window structure.
"@<Attachname>" By attach name or alias in the Object Map.

Examples

Attach "~N~NOTEPAD.EXE~32770~Open"
ComboBox 1137, " (C:)", 'Left SingleClick'
Button 2, "SingleClick"
Attach "~N~NOTEPAD.EXE~32770~Open"
Button "@~N~NOTEPAD.EXE~Button~Cancel", "SingleClick"
Attach "~N~KERNEL32.DLL~32770~Open"
ComboBox "List files of &type:", "Text Files (*.TXT)"
ListBox "~1", "PB.TXT", 'Left SingleClick'
Button "OK", 'SingleClick'

ConvertCurrency( )

Miscellaneous
Converts the value of one European currency into the value of another specified European currency based on the value of the euro.

Syntax

ret = ConvertCurrency("OriCurrency" , Amount, "NewCurrency" ,[Options])

Operation

The ConvertCurrency( ) command conducts currency conversions for the following thirteen countries: Austria, Belgium, British, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, Netherlands, Portugal, and Spain.
The function can be used to perform both triangulation exchanges and direct exchanges. Triangulation exchanges convert OriCurrency into the euro and then into the NewCurrency. The exchange rate for the euro is defined using the euro Currency dialog box, which is accessible from *EZ Test*'s Options menu. The function returns the converted value of the new currency.
A direct exchange converts the value of one currency directly into another currency. To conduct a direct exchange, the exchange rate must be supplied using the options parameter.
The parameters are as follows:
"OriCurrency" This parameter specifies the original currency type (i.e., the currency that the conversion will be converted *from*).
Acceptable values are:
ATS Austria
BEF Belgium
GBP Britain
FIM Finland
FRF France
DEM Germany
GRD Greece
IEP Ireland
ITL Italy
LUF Luxembourg

NLG Netherlands
PTE Portugal
ESP Spain
Amount The numeric quantity, in the original currency, to be converted.
"NewCurrency" This parameter specifies the new currency type (i.e., the currency that the conversion will be converted *into*). Acceptable values are the same as those listed for the OriCurrency parameter.
"Options" The exchange rate to be used in a direct conversion. If the Options parameter is not specified, the currency will be triangularly converted based on the value of the euro (defined on the euro Currency dialog box).

Examples

Example 1:

```
ret = ConvertCurrency("ESP" , 34 , "ATS" )
msgbox ("", ret) ; triangulation conversion
```

Example 2:

```
ret = ConvertCurrency( "ESP" , 34 , "ATS" ,4.7)
msgbox ("", ret) ; direct conversion
```

CopyFile( )

File Access
Copies a file to a given destination.

Syntax

```
ret = CopyFile( source, destination )
```

Variants

```
ret = Copy( source, destination )
```

See Also

Read( ),Write( ), DeleteFile( ), Isfile( ), FileExists( )

Operation

Copies a source file to a destination file. Wildcard characters can be used. The function returns 1 if the copy is successful and returns 0 if it is not.

Examples

Example 1:

```
; copy a file to a different directory
CopyFile( "c:\network.log", "c:\mynet\network.log" )
```

Example 2:

```
; copy a file to a new name in the same directory
ret = Copy( "daily.rpt", "daily.old" )
if ret = 0 ; if copy fails
Create( "daily.old" ) ; create an empty file
endif
```

Example 3:

```
; copy all data files to a backup directory on the network drive
Copy( "c:\*.dat", "y:\backup\*.bak" )
```

Create( )

File Access
Creates a new file or resets an existing file.

Syntax

```
Create( "filename" )
```

See Also

Read( ), Write( ), IsFile( ), FileExists( )

Operation

This function creates an empty file that may be written to using the Write( ) function. If

the specified file already exists, it is overwritten and its current contents are destroyed. If no path is specified in filename, the current directory is assumed.
The Create( ) function is a good way to clear the contents of a file before a script starts writing to it.
The function returns 1 if filename is successfully created and returns 0 if it is not.

Examples

; create file "dummy.dat" in the "c:\tests" directory
Create("c:\tests\dummy.dat")
; create "report.dat" in the current directory
Create("report.dat")

CreateDate( )

Date/Time
Enters a dynamically generated date into the target application at replay.

Syntax

ret = CreateDate(actualdate, dateformat, state, datestring)

Variants

ret = CreateDate(actualdate, dateformat, state, datestring, ageret)

See Also

Replay.TodaysDate, SetDate( ), SetTime( )

Operation

The CreateDate( ) command allows you to conduct Year 2000 date aging testing by capturing and recording dates that are entered into the target application using the CreateDate( ) command. The CreateDate( ) command can be used by a script at runtime to enter a dynamic date into the target application. The dynamically generated date that is used during script replay is calculated based on the PC's internal clock or a usersupplied date that represents the base date ("today"). This value must reflect dates occurring after January 1, 1900.
The parameters are as follows:
actualdate The date that was actually entered using the CreateDate hotkey while learning the script. This value provides a reference and allows you to use the originally recorded "un-aged" date.
dateformat The format of the date that will be entered into the target application. These are the same formats available through text checks (for example, MM-DD-YY). The following building blocks can be used to create valid date formats:
D The 1-digit number for the day.
DD The 2-digit number for the day.
Mmm The 3-letter abbreviation for the month (Jan, Feb, Oct, fetc.). The capitalization of this building block should reflect the expected check capitalization.
MM The 2-digit number for the month.
YY The last 2-digits of the year.
YYYY The 4-digit year.
Www The 3-letter weekday abbreviation (Mon, Tue, Wed, etc.). The capitalization of this building block should reflect the expected check capitalization.
Month The complete month name.
Weekday The full name of a day of the week (i.e., Monday).
state Determines whether the date that is entered into the target application will be aged or fixed. Acceptable values are:
aged The date entered into the target application will be aged based on the formula specified by dateval.
fixed The date entered into the target application will be fixed

based on the value specified by actualdate.
datestring The formula used to calculate the date that will be entered into the target application. The date is relative to the date determined by the computer's internal clock or the date specified as "Today" in the Run Environment Settings.
For example, if datestring is "today + 3 years +2 days" and "Today" is set as 12/31/99, the date replayed to the target application would be 01/02/03. Valid formula keywords for "today" are:
today The date specified as the "Today's" Date value in the Run Environment settings. The default value is the current date of the PC's internal clock.
year Adds or subtracts a number of years from the value of "today" to generate an aged date. A value of years is also acceptable.
week Adds or subtracts weeks from the value of "today" to generate an aged date. A value of weeks is also acceptable.
month Adds or subtracts a number of months from the value of "today" to generate an aged date. A value of months is also acceptable.
day Adds or subtracts a number of days from the value of "today" to generate an aged date. A value of days is also acceptable.
After *EZ Test* calculates the value for the aged date, the following keywords can be used to make additional weekday calculations within the month based on the calculated aged date.
next or > Uses the calculated aged date and advances to the next weekday specified. For example:
If today is January 1, 2000 and datestring is:
"today +1 week > saturday" , the date would be January 15, 2000.
prev or < Uses the calculated aged date and uses the previous weekday specified. For example:
If today is January 1, 2000 and datestring is:
"today +1 week prev tuesday" , the date would be January 4, 2000.
>= Uses the calculated aged date and advances to the next weekday specified. If the aged date falls on the date specified, that date is used. For example:
If today is January 1, 2000 and datestring is:
"today +1 week >= saturday" , the date would be January 8, 2000.
<= Uses the calculated aged date and uses to the previous weekday specified. If the aged date falls on the date specified, that date is used. For example:
If today is January 1, 2000 and datestring is:
"today +1 week <= saturday" , the date would be January 8, 2000.
final Uses the calculated aged date and advances to the final weekday of the month specified. For example:
If today is January 1, 2000 and datestring is:
"today +1 week final saturday" , the date would be January 29, 2000.
Uses the calculated aged date to calculate the month and returns the #nth weekday of that month (#2

Monday would return the second monday of the
determined month). For example:
If today is January 1, 2000 and datestring is:
"today +2 week #4 sunday" , the date would
be January 23, 2000.
ageret This is a variable that accepts the “date value” of the calculated date. It is the same format that the DateVal( ) command generates. (i.e., the number of seconds since 12:00 a.m. 31 Dec 1899). It can be used in any *EZ Test* function that takes a date value. CreateDate returns a string in the format of the aged date.

Examples

Example 1:

```
Function Main
counter = 1
repeat
; Using Explorer, find all files modified between
; Yesterday and today
Attach "Find: All Files MainWindow"
TabCtrl "~1", "Date Modified", 'Left SingleClick'
RadioButton "&between", 'Left SingleClick'
Type "{Tab}"
; Insert yesterday’s date using an aged date
DateRet = CreateDate("04/09/98","MM/DD/YY","aged",
"today +0 weeks -1 day")
Type DateRet
Type "{Tab}"
;Insert today's date using an aged date based on "today"
DateRet = CreateDate("04/10/98","MM/DD/YY","aged",
"today +0 years +0 months ")
Type DateRet
Attach "Find: All Files MainWindow"
Button "F&ind Now", 'Left SingleClick'
; Set the date to the 1st of January 2000
ret = SetDate( 2000 , 1 , 1 )
; repeat the activity with the date set to 1/1/00
counter = counter + 1
until counter = 3
End Function ; Main
```

Example 2:

```
Function Main
; Create a message box that displays the next Tuesday after
; The calculated date.
Replay.TodaysDate = "default"
DateRet = CreateDate("04/22/98","MM/DD/YYYY","aged","today > tuesday")
MsgBox( "calculated" , DateRet )
End Function ; Main
```

Example 3:

```
Function Main
; Calculate the 4th Friday of the month after the
; Calculated date.
DateRet = CreateDate("04/22/98","MM/DD/YYYY","aged","today +1 day + 2
weeks #4 Friday")
MsgBox( "calculated" , DateRet )
```

Example 4:

```
; Example using dateret parameter
DateRet = CreateDate( "20-04-98" , "DD-MM-YY" , "aged" ,
"today +1 day" , ageret )
FormattedDate = FormatDate( "yyyy, mmmm, d dddd", ageret )
msgbox( DateRet , FormattedDate )
msgbox( "" , ageret )
```

CtrlChecked( )

Window Information
Determines whether the specified control is checked.

Syntax

ret = CtrlChecked(hCtrl)

See Also

ControlFind( ), IsWindow( )

Operation

This function determines if the control whose window handle is hCtrl is checked (selected). The function can only be used on RadioButton and CheckBox controls, otherwise a runtime error is generated. The hCtrl can be determined from one of the ContrlFind( ) group of functions.
The function returns 1 if the specified control is enabled and returns 0 if it is not.

Examples

```
Attach "~P~EZ TESTDEMO.EXE~32770~Transfer Car" ; attach to dialog
hCtrl = RadioFind( "London" ) ; get handle to London
If CtrlChecked( hCtrl) = 1 ; if it's selected
hCtrl = RadioFind( "New York" ) ; try New York
If CtrlChecked( hCtrl) = 1 ; if it's selected
RadioButton "Paris", 'Left SingleClick' ; select Paris
Else ; otherwise
RadioButton "New York", 'Left SingleClick' ; select New York
Endif
Else ; otherwise,
RadioButton "London", 'Left SingleClick' ; select London
Endif
```

CtrlEnabled( )

Window Information
Determines if the specified control is enabled.

Syntax

ret = CtrlEnabled(hCtrl)

See Also

ControlFind( ), IsWindow( )

Operation

This function determines if the control whose window handle is hCtrl is enabled. The hCtrl can be determined from one of the ContrlFind( ) group of functions.
The function returns 1 if the specified control is enabled and returns 0 if it is not.

Examples

```
Attach "~P~EZ TESTDEMO.EXE~32770~Transfer Car" ; attach to dialog
hCtrl = RadioFind( "New York" ) ; get handle to button
If CtrlEnabled( hCtrl ) = 1 ; if it's enabled
RadioButton "New York", 'Left SingleClick'; select it
Else ; otherwise
RadioButton "London", 'Left SingleClick' ; select another
Endif
```

CtrlFocus( )

Window Information
Determines whether the specified control has the keyboard focus.

Syntax

ret = CtrlFocus(hCtrl)

See Also

CtrlEnabled( ), ControlFind( ), IsWindow( )

Operation

This function determines whether the control whose window handle is hCtrl has the keyboard focus. hCtrl can be determined from one of the ContrlFind( ) group of functions.
The function returns 1 if the specified control has focus, 0 otherwise.

Examples

```
Attach "~P~EZ TESTDEMO.EXE~32770~Transfer Car" ; attach to dialog
hCtrl = ButtonFind( "&Transfer" ) ; get handle to button
While CtrlFocus( hCtrl) <> 1 ; if not in focus
Attach FocusWindow
Type "{Tab}" ; tab to next control
EndWhile
Type "{Return}" ; select it
```

CtrlLabel( )

Window Information
Retrieves the label associated with the specified control.

Syntax

```
ret = CtrlLabel( hCtrl )
```

See Also

CtrlEnabled( ), ControlFind( ), CtrlFocus( ), IsWindow( )

Operation

This function retrieves the label of the control whose window handle is hCtrl. hCtrl can be determined from one of the ContrlFind( ) group of functions.

Examples

```
Attach "Transfer Car ChildWindow" ; attach to dialog
Repeat ; start of loop
Type "{Tab}" ; tab to next control
Attach FocusWindow ; attach to it
hCtrl = AttachWindow( ) ; get its handle
Until CtrlLabel( hCtrl ) = "&Quantity :" ; until required label
```

CtrlPressed( )

Window Information
Determines if the specified push button is currently pressed.

Syntax

```
ret = CtrlPressed( hCtrl )
```

See Also

ControlFind( ), IsWindow( )

Operation

This function determines if the push button control with window handle hCtrl is currently pressed. The function can only be used on push button controls, otherwise a runtime error is generated. The hCtrl can be determined from one of the ContrlFind( ) group of functions.
The function returns 1 if the specified control is enabled and returns 0 if it is not.

Examples

```
While ActiveName( ) = "Customer Invoice Parent"; while active
Attach "Customer Invoice Dialog" ; attach to dialog
hCtrl = ButtonFind( "&Find" ) ; get Find button
repeat ; start endless loop
if CtrlPressed( hCtrl ) = 1 ; check button down
MsgBox( "Find", "This function not implemented")
endif
until 1=2 ; end loop
EndWhile
```

CtrlSelText( )

Window Information
Retrieves the selected text from an edit control.

Syntax

Text = CtrlSelText(hCtrl)

See Also

ControlFind( )

Operation

This function retrieves the currently selected (highlighted) text from the edit control with window handle hCtrl. The window handle can be obtained from the EditFind( ) function— one of the ControlFind( ) group of functions.
The function returns an empty string if the edit control contains no selected text.
The CtrlSelText( ) function may only be used on edit controls. Using the function on any other type of control will cause a runtime error. A runtime error is also generated if the window handle specified is invalid.

Examples

```
Attach "Customer Letter" ; attach to letter to customer
hCtrl = EditFind( "Customer Edit" ); get handle of the edit control
TextSelect "O/N", 'Left DoubleClick'; locate start of order number
; referred to, select whole item
ret = CtrlSelText( hCtrl ) ; retrieve full order number
MsgBox( "", ret ) ; and display result
```

CtrlText( )

Window Information
Retrieves text from a control.

Syntax

Text = CtrlText(hCtrl)

See Also

ControlFind( ), CtrlSelText( )

Operation

This function retrieves the text from the control whose window handle is hCtrl. The window handle can be obtained from one of the ControlFind( ) group of functions. The currently displayed text is returned; in the case of list boxes and combo boxes, the currently selected item is returned.
The function returns an empty string if the control contains no text. A runtime error is generated if the window handle specified is invalid.

Examples

```
Attach "Open PopupWindow" ; attach to the File Open dialog
lCtrl = ListViewFind( "~1" ) ; get handle of list control
eCtrl = EditFind( "File &name:" ) ; get handle of File name: control
ltext = CtrlText( lCtrl ) ; get current contents
etext = CtrlText( eCtrl ) ;
MsgBox( ltext, etext ) ; and display result
```

CtrlType( )

Window Information
Returns a number indicating the type of control referred to by the passed window's handle.

Syntax

Ret = CtrlType (hWnd)

See Also

AttachWindow( ), AttachName( )

Operation

This command takes a window handle, hWnd, and returns a number that indicates the type of control. CtrlType( ) can return the following numbers. Each number and its associated control is listed below:

0 Unknown
1 Main
2 Child
3 Dialog
4 Popup
5 Static
6 GroupBox
7 PushButton
8 CheckBox
9 Radio
10 ScrollBar
11 ListBox
12 ComboBox
13 Edit
14 ComboLBox
15 UpDown
16 ListView
17 Header
18 TreeView
19 Toolbar
20 StatusBar
21 TabControl
22 Grid
23 DataWindow
24 Label
25 Picture
26 Hotspot

Examples

Example 1:

```
Exec "NOTEPAD.EXE" ; Start Notepad
Ret = CtrlType( FocusWindow( ) ) ; Get the control type of
; the active control
If ret = 13 ; its Edit control
Attach FocusWindow( ) ; Do something
Type "Hello World"
Else ; Do nothing
Endif
```

Example 2:

```
; Example of a mini "Identify"
Var ctrl[] ; Setup names of controls
Ctrl[0] = "Unknown"
Ctrl[1] = "Main"
Ctrl[2] = "Child"
Ctrl[3] = "Dialog"
Ctrl[4] = "Popup"
Ctrl[5] = "Static"
Ctrl[6] = "GroupBox"
Ctrl[7] = "PushButton"
Ctrl[8] = "CheckBox"
Ctrl[9] = "Radio"
Ctrl[10] = "ScrollBar"
Ctrl[11] = "ListBox"
Ctrl[12] = "ComboBox"
Ctrl[13] = "UpDown"
Ctrl[14] = "ComboLBox"
```

```
Ctrl[15] = "UpDown"
Ctrl[16] = "ListView"
Ctrl[17] = "Header"
Ctrl[18] = "TreeView"
Ctrl[19] = "Toolbar"
Ctrl[20] = "StatusBar"
Ctrl[21] = "TabControl"
Ctrl[22] = "Grid"
Ctrl[23] = "DataWindow"
Ctrl[24] = "Label"
Ctrl[25] = "Picture"
Ctrl[26] = "Hotspot"
Repeat
mw = MouseWindow( ) ; Get attach name of window
If mw <> last ; Check if different to last time
last = mw ; Store "last" value for next loop
Hwnd = AttachWindow( mw ); Get handle of the attach window
ret = CtrlType( hwnd ) ; Extract the control type
; Display the name using the appropriate array element
Textpanel 1, ( str(ret) + ", " + ctrl[ret] + ", " + mw )
Endif
Until 1 = 2
```

CurDir( )

File Access

Returns the current working directory.

Syntax

```
ret = CurDir( )
```

Variants

```
ret = Path( )
```

See Also

ChDir( ), Read( ), Write( )

Operation

This function returns the name of the current working directory. Its primary function is to check the current directory before file access.

Examples

Example 1:

```
ret = CurDir( ) ; get current directory
if ret<>"C:\" ; check its value
ChDir("C:\") ; change it if necessary
ret = CurDir( ) ; and get updated value
endif
MsgBox( "Current Directory", ret ) ; display the result
```

Example 2:

```
ret = CurDir( ) ; get current directory
ChDir( "C:\reports" ) ; change directory
DeleteFile( "DAILY.RPT" ) ; delete this file
ChDir( ret ) ; and change back
```

CurTime( )

Date/Time

Represents the current date and time as a number.

Syntax

```
ret = CurTime( )
```

Variants

```
ret = CurTime
```

See Also

Date( ), FormatDate( ), Time( )

Operation

This function converts the current date and time into a number. The value returned is the number of seconds elapsed since 01-01-1970. This can be used as a parameter to other date and time functions.

Examples

Example 1:

n = CurTime( ) ; returns current date/time value
MsgBox("Date is...", Date(n)); display current date
MsgBox("Time is...", Time(n)); display current time

Example 2:

n = CurTime ; returns current date/time value

DataCtrl( )

4GL Commands
Replays a Visual Basic Data Control.

Syntax

ret = DataCtrl("ControlID" , "Options")

Operation

This command processes a Visual Basic DataCtrl control in the currently attached dialog box. The action is specified using the "Options" parameter. The parameters are as follows:
"ControlID" Specifies the DataCtrl logics object name.
"Options" The Options are as follows:
"first" Goto the first database record.
"last" Goto the last database record.
"next" Goto the next database record.
"prev" Goto the previous database record.

Examples

Attach "Form1 MainWindow"
DataCtrl "~1", "prev"
DataCtrl "~1", "last"
DataCtrl "~1", "next"
DataCtrl "~1", "first"

DataType( )

String Manipulation
Checks if characters in a string are of a particular type.

Syntax

ret = DataType("string", "type")

Operation

This function returns 1 if all the characters in string are of the specified type, and it returns 0 if any character does not match. The types are:
Lowercase Lowercase characters in the range a-z.
Uppercase Uppercase characters in the range A-Z.
Alpha Alphabetic characters in the range a-z, A-Z.
alphanu**M**eric Alphanumeric characters in the range a-z, A-Z, 0-9.
Integer Integers in the range 0-9.
Hex Hexadecimal characters in the range 0-9, a-f.
Binary Binary characters 0 or 1.
The “type” parameter may be abbreviated as a single character. The acceptable abbreviations are indicated in bold typeface in the list above.
Numeric parameters are converted into strings before the analysis is performed.

Examples
ret = DataType("hello", "lowercase") ; returns 1
ret = DataType("Hello", "lowercase") ; returns 0
ret = DataType("Hello", "uppercase") ; returns 0
ret = DataType("Hello", "alpha") ; returns 1
ret = DataType("123", "integer") ; returns 1
ret = DataType(123, "integer") ; returns 1
ret = DataType("123.45", "i") ; returns 0
ret = DataType("PWR56r", "m") ; returns 1
ret = DataType("b7ff", "hex") ; returns 1

DataWindow( )

4GL Commands
Replays a double mouse-click to a PowerBuilder DataWindow.
Syntax
ret = DataWindow("ControlId", "DataWindow Location ID" 'Options' x, y)
See Also
dwClick( ), dwDblClick( )
Operation
This command replays a mouse-click into the DataWindow control specified by the "ControlId" parameter within the currently attached dialog box. The mouse is clicked at a location defined by "DataWindow Location ID".
The parameters are:
"ControlId" Specifies the internal PowerBuilder name of the control. If the ControlId is numeric, it represents the index value of the edit control: "~1" for the first, "~2" for the second, etc.
"DataWindow Location ID" Specifies an internal PowerBuilder name of the location within the DataWindow to click on.
'Options' The 'options' are as follows:
'Left' Use the left mouse button.
'Right' Use the right mouse button.
'Middle' Use the middle mouse button.
'SingleClick' Perform a single-click on the mouse button.
'DoubleClick' Perform a double-click on the mouse button.
x Specifies to override the coordinates of the "DataWindow Location ID".
y Specifies to override the coordinates of the "DataWindow Location ID".
Examples
Attach "~N~INSTBLDR.EXE~FNWND050~w_instbldr"
DataWindow"dw_components", "dw_components.detail.description.1"
, 'Left SingleClick'

Date( )

Date/Time
Converts a date value into a string.
Syntax
ret = date(dateval)
Variants
ret = date( )
See Also
DateVal( ), CurTime( )

Operation
This function returns a date value as a string in “mm-dd-yyyy” format. The dateval parameter is a date value, which can be obtained from the DateVal( ) or CurTime( ) functions.
If the dateval parameter is not specified, the current date is returned.
If the dateval parameter is invalid, the string “InvalidDate” is returned.
Examples
date_str = date(0) ; returns "12-30-1899"
date_str = date(817776000) ; returns "12-01-1995"
date_str = date(9999999999) ; returns "09-07-2014"
date_str = date( ) ; returns today's date
date_str = date(-12345678) ; returns "12-30-1899"

DateTimeCtrl( )

Dialog Control
Sets the date or time of a date/time control.
Syntax
ret = DateTimeCtrl ("ControlID" , "DateTimeVal")
See Also
DateTime( ), DateTimeFind( ), DateTimeMode( )
Operation
This function drives the selection of a date/time control based on the value specified by the ControlId parameter. The parameters are:
"ControlId" The index value of the date/time control.
"DateTimeVal" A string value that denotes either the date or time value of the control. The format must be "mm-ddyyyy" for dates and "hh:mm:ss" for times.
Examples
Function Main
Attach "Microsoft Control Spy - Date and Time Picker PopupWindow"
DateTimeCtrl "~1", "8-4-1999"
Attach "Microsoft Control Spy - Month Calendar PopupWindow"
CalendarCtrl "~1", "8-1-1999", "8-1-1999"
CalendarCtrl "~1", "8-7-1999", "8-7-1999"
CalendarCtrl "~1", "8-14-1999", "8-14-1999"
CalendarCtrl "~1", "8-8-1999", "8-8-1999"
CalendarCtrl "~1", "8-8-1999", "8-14-1999"
End Function ; Main

DateTimeMode( )

Dialog Control
Returns a string indicating if the date/time picker control is operating in date or time mode.
Syntax
ret = DateTimeMode (hCtrl)
See Also
DateTime( ), DateTimeCtrl( ), DateTimeFind( )
Operation
This parameter returns a string indicating if the operation mode of the date/time control is in date or time.
Examples
Function Main
Attach "Microsoft Control Spy - Date and Time Picker PopupWindow"
hCtrl = DateTimeFind("~1")
DateTimeVal = DateTime(hCtrl)
DTPMode = DateTimeMode(hCtrl)

```
if DTPMode = "date"
fmt = FormatDate( "dd-mm-yyyy" , DateTimeVal )
endif
if DTPMode = "time"
fmt = FormatDate( "hh:mm:ss" , DateTimeVal )
endif
msgbox( "Mode=" + DTPMode + ", DateTime = " + DateTimeVal , fmt )
End Function ; Main
```

DateTime( )

Dialog Control
Returns the numerical representation of the date/time control.

Syntax

```
ret = DateTime ( hCtrl )
```

See Also

DateTimeCtrl( ), DateTimeFind( ), DateTimeMode( ), DateVal( ), TimeVal( )

Operation

This function returns a numerical representation of the date/time control. This number can then be used in conjunction with the other *EZ Test* date and time command to format the output of the date/time information.

Examples

```
Function Main
Attach "Microsoft Control Spy - Date and Time Picker PopupWindow"
hCtrl = DateTimeFind( "~1" )
DateTimeVal = DateTime( hCtrl )
DTPMode = DateTimeMode( hCtrl )
if DTPMode = "date"
fmt = FormatDate( "dd-mm-yyyy" , DateTimeVal )
endif
if DTPMode = "time"
fmt = FormatDate( "hh:mm:ss" , DateTimeVal )
endif
msgbox( "Mode=" + DTPMode + ", DateTime = " + DateTimeVal , fmt )
End Function ; Main
```

DateVal( )

Date/Time
Converts a date into a numerical representation.

Syntax

```
ret = DateVal( yyyy, mm, dd )
```

See Also

Date( ), CurTime( ), FormatDate( )

Operation

This function returns a numerical representation of a date. The parameters are:
yyyy The year (1899 onwards)
mm The month (01 - 12)
dd The day (1 - 31)
The value returned is the number of seconds elapsed from 12:00 a.m. 30 Dec 1899 to 12:00 a.m. of the date entered in the function.
This function can be used to calculate a future date, where one day = 24*60*60 seconds. The result can be used by the Date( ) and FormatDate( ) functions to produce the date as a string. It can be combined with the value returned by the TimeVal( ) function to make a date/time value.
The function returns -1 if the date format is invalid or prior to 31 Dec 1899.

Examples

```
; get number of seconds since
```

```
; 12:00 midnight on the morning of 31 December 1899
; (ie one day before the new century started)
; 1 day = 86400 Seconds
; DateVal format is (yyyy,mm,dd)
ret = DateVal( 1899,12,31)
msgbox ("Seconds since 31 Dec 1899", ret )
; convert a date value into a date in format mm-dd-yyyy
; ie 24 Mar 1997 will return 03-24-1921
msgbox( "Date if 24 Mar 1921, date( DateVal( 1921,3,24 )) )
; get today's date
msgbox ("Today is", date( ) )
```

Day( )

Date/Time
Returns the day of the month.

Syntax

ret = Day(dateval)

Variants

ret = Day( )

See Also

DateVal( ), CurTime( )

Operation

This function returns the day of the month specified by the dateval parameter. The dateval parameter is a date value that can be derived from the DateVal( ) or CurTime( ) functions.
If the dateval parameter is not specified, the current system date is used.

Examples

Example 1:

```
n = DateVal( 1995, 11, 15 ) ; returns 816393600
Day_of_Month = Day( n ) ; returns 15
```

Example 2:

```
Day_of_Month = Day( ) ; current day of the month
```

dbAddNew( )

SQL Commands
Permits addition of a new record to the current result set.

Syntax

dbAddNew( )

See Also

dbEdit( ), dbExecute( ), dbSelect( ), dbUpdate( )

Operation

This function permits the creation of a new record within the current result set. The primary field value within the new record can then be set with the dbSetField( ) function. Following the edit, the new record must be confirmed with the dbUpdate( ) function. The new record is added to the end of the existing result set.
The dbAddNew( ) function may only be used with records in a result set obtained from the dbSelect( ) function. To modify fields in a data source directly, use the dbExecute( ) function.
The function has no return value.

Examples

```
; this example opens the EZ TESTDemo database, extracts the last
; reference number and generates a new entry
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveLast( ) ; move to last record
```

```
Last = dbGetField( "Ref" ) ; get last reference
Last = Right( Last, 4 ) ; extract numeric part
Next = Last + 10001 ; add 10000 + 1
NewRef = "C-a-" + Right( Next, 4 ) ; make new reference
dbAddNew( ) ; edit mode
dbSetField( "Ref", NewRef ) ; set new value
dbUpdate( ) ; commit new value
```

dbBOF( )

SQL Commands

Determines if the record pointer is at the start of the current result set.

Syntax

ret = dbBOF( )

See Also

dbEOF( ), dbMove( ), dbSelect( )

Operation

This function is used to determine if the record pointer has been moved before the first record in the current result set.

The function returns 1 if the pointer is positioned before the first record, 0 otherwise.

Examples

```
; connect to data source
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
If dbEOF = 1 ; if no records
Exit ; quit
Else ; otherwise
dbMoveLast( ) ; move to last record
While dbBOF( ) = 0 ; until the start
Cost = dbGetField( "Cost" ) ; get current value
dbEdit( ) ; edit mode
dbSetField( "Cost", Cost-100 ) ; set new value
dbUpdate( ) ; commit new value
dbMovePrev( ) ; previous record
EndWhile
Endif
```

dbClose( )

SQL Commands

Closes the record set associated with the last dbSelect( ).

Syntax

dbClose( )

See Also

dbDisconnect( ), dbSelect( )

Operation

This function closes the result set created by the previous dbSelect( ) function. The result set must be closed before another dbSelect() function can be used.

Examples

Example 1:

```
dbConnect( "c:\msoffice\access\mydb.mdb" ); connect to datasource
dbSelect( "select * from Products" ) ; select all records
dbMoveLast( ) ; move to last record
count=dbRecordCount( ) ; count no. of records
MsgBox( "No. of Records", count ) ; display result
dbClose( ) ; close result set
```

Example 2:

```
dbConnect( "DSN=EZ Testdemo" )
; select cars of a specific make
```

```
dbSelect( "SELECT * FROM CarList WHERE Make Like '*Ford*'" )
; move to first record in result set
dbMoveFirst( )
While dbEOF = 0 ; start of loop
dbEdit( ) ; edit fields
Cost=dbGetField( "Cost" ) ; get current cost
dbSetField( "Cost", Str(Cost-1000) ) ; reduce Cost by 1000
dbUpdate( ) ; update database
dbMoveNext( ) ; next record
EndWhile
dbClose( ) ; close result set
```

dbConnect( )

SQL Commands
Connects to a SQL data source

Syntax

dbConnect("filename.mdbc",[Library],[QueryTimeout],[Login-Timeout])

Variants

dbConnect("DataSource")

See Also

dbDisconnect( )

Operation

This function establishes a connection to a SQL data source. The "DataSource" parameter can be one of the following formats:
<filename.mdb> Where <filename.mdb> is a Microsoft Access MDB file. (No longer recommended)
DSN=<datasource> Where <datasource> is an ODBC compliant data source. (Strongly recommended)
"default" Opens the default database libraries.
"cursors" Opens an Oracle database using the ODBC cursor library. If the cursors option is specified, dynasets are not supported. Only snapshot-type record sets can be used with this library.
QueryTimeout Number of seconds EZ Test waits for query to complete.
LoginTimeout Number of seconds EZ Test waits for login to complete for ODBC connections only.
UID=<userID> Where <userID> is the user ID to data source
PWD=<Password> Where<Password> is the user password to data source
You must connect to a data source before SQL inquiries can be executed upon it. The function has no return value. If the data source is not located, a runtime error is generated.
The dbConnect command supports a direct file path to the MDB for Microsoft Access 97 and 2000, as well as ODBC connections to both Access and other types of datasources. However, note that the direct file path connection to Access uses Microsoft DAO to negotiate the connection. DAO has known limitations with the size and number of records it can handle. As a result, American Systems strongly recommends using ODBC with the dbConnect command (DSN=[ODBC DSN Name]).
To use the default timeout value, parameters may be omitted. To use an infinite timeout value, specify "0" seconds.
Examples

Example 1:

```
dbConnect( "c:\msoffice\access\mydb.mdb" ); connect to
; datasource
dbSelect( "select * from Products" ) ; select records
dbMoveLast( ) ; move to last record
count=dbRecordCount( ) ; count no.of records
dbDisconnect( ) ; disconnect datasource
MsgBox( "No. of Records", count ) ; display result
```

Example 2:
dbConnect("DSN=accounts") ; ODBC connection
Example 3:
dbConnect("DSN=OracDB" ,"cursors") ; Opens Oracle database
; using ODBC cursor library
Example 4:
dbConnect("DSN=accounts","default",24,25); Opens ODBC connection
; using default database
libraries
;Seconds before query
timeout
;seconds before login
timeout
Example 5:
DbConnect("DSN=Accounts;UID=Login;PWD=xyz");Opens ODBC connection
;using user ID
;and password

dbDisconnect( )

SQL Commands
Disconnects from a SQL data source
Syntax
dbDisconnect( )
See Also
dbConnect( )
Operation
This function disconnects from the currently connected SQL data source. Disconnecting from a data source frees resources. The connection will be cleared automatically when the script terminates.
The function has no return value.
Examples
dbConnect("c:\msoffice\access\mydb.mdb") ; connect to datasourse
dbSelect("select * from Products") ; select records
dbMoveLast( ) ; move to last record
count=dbRecordCount( ) ; count no.of records
dbDisconnect( ) ; disconnect datasource
MsgBox("No. of Records", count) ; display result

dbEdit( )

SQL Commands
Permits field editing in the current record of the current result set.
Syntax
dbEdit( )
See Also
dbAddNew( ), dbExecute( ), dbSelect( ), dbUpdate( )
Operation
This function sets the current result set into edit mode, allowing a field value in the current record to be set with the dbSetField( ) function. Following the edit, the change must be confirmed with the dbUpdate( ) function.
The dbEdit( ) function may only be used with records in a result set obtained from a dbSelect( ) function. To modify fields in a data source directly, use the dbExecute( ) function.
The function has no return value.
Examples
; connect to data source
dbConnect("c:\EZ Testcenter.32\demos\EZ Testdemo.mdb")

```
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveFirst( ) ; move to first record
count = dbRecordCount ; count no. of records
While dbEOF( ) = 0 ; while not at end
Cost = dbGetField( "Cost" ) ; get current value
dbEdit( ) ; edit mode
dbSetField( "Cost", Cost-100 ) ; set new value
dbUpdate( ) ; commit new value
dbMoveNext( ) ; move to next record
EndWhile
```

dbEOF( )

SQL Commands

Determines if the record pointer is beyond the last record of the current result set.

Syntax

```
ret = dbEOF( )
```

See Also

dbMove( ), dbSelect( )

Operation

This function is used to determine if the current result set contains any records or to check if the record pointer has been moved beyond the last record.

The function returns 1 if the result set contains no records or if the last record has been passed. It returns 0 if the result set contains records and the current record pointer is valid.

Examples

```
; connect to data source
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
If dbEOF = 1 ; if no records
Exit ; quit
Else ; otherwise
dbMoveFirst( ) ; move to first record
While dbEOF( ) = 0 ; until the end
Cost = dbGetField( "Cost" ) ; get current value
dbEdit( ) ; edit mode
dbSetField( "Cost", Cost-100 ) ; set new value
dbUpdate( ) ; commit new value
dbMoveNext( ) ; move to next record
EndWhile
Endif
```

dbExecute( )

SQL Commands

Executes a SQL command on the current data source

Syntax

```
dbExecute( "SQL" )
```

See Also

dbClose( ), dbConnect( ), dbEdit( ), dbSelect( ), dbSetField( ), dbUpdate( )

Operation

This function causes the SQL command to be executed directly on all the records of the currently connected data source. The function has no return value.

Examples

Example 1:

```
; connect to data source
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
; increase Cost of each car by 250
dbExecute( "UPDATE CarList SET Cost=Cost+250 )
dbClose( )
```

Example 2:
```
; connect to data source using ODBC driver
dbConnect( "DSN=EZ Testdemo" )
; increase cost of some cars by 50
dbExecute( "UPDATE CarList SET Cost=Cost+50 WHERE Make Like" +
"'*Ford*'" )
dbClose( )
```

dbGetField( )

SQL Commands

Retrieves a field from the current record of the current result set.

Syntax

```
ret = dbGetField( "FieldName" )
```

See Also

dbConnect( ), dbMove( ), dbSelect( ), dbSetField( )

Operation

This function retrieves the value of the "FieldName" parameter from the current record of the current result set. Date/Time fields are returned as strings in "MM:DD:YYYY hh:mm:ss" format.

A runtime error is generated if the result set contains no records, if the result set has been closed with a dbClose( ) functions, or if the "FieldName" parameter is invalid.

Examples

```
dbConnect( "DSN=EZ Testdemo" ) ; ODBC connection to datasource
dbSelect( "SELECT * FROM CarList" ); select records
dbMoveFirst( ) ; move to first record
While dbEOF = 0 ; while not at end of result set
MsgBox( "", dbGetField( "Make" ) ); display record
dbMoveNext( ) ; move to next record
EndWhile
```

dbMove( )

SQL Commands

Moves the record pointer within the current result set.

Syntax

```
dbMove( count )
```

See Also

dbMoveFirst( ), dbMoveLast( ), dbMoveNext( ), dbMovePrev( ), dbRecordCount( )

Operation

This function moves the record pointer count records in the current result set. Set the count parameter to a positive integer to move the record pointer forward; set the count parameter to a negative integer to move the record pointer backwards.

Attempting to use the dbMove( ) function to move the record pointer beyond the start or end of the record set causes a runtime error to be generated. The function has no return value.

Examples

Example 1:
```
dbConnect( "c:\msoffice\access\mydb.mdb" ); connect to datasource
dbSelect( "select * from Products" ) ; select all records
dbMoveFirst( ) ; move to first record
While dbEOF = 0 ; while not at the end
Print dbGetField( "Product Name" ) ; print product name
dbMove( 1 ) ; move to next record
EndWhile
```
Example 2:
```
dbConnect( "c:\msoffice\access\mydb.mdb" ); connect to datasource
dbSelect( "SELECT * from CarList" ) ; select all records
```

```
dbMoveLast( ) ; move to last record
count = dbRecordCount( ) ; get number of records
Repeat ; start of loop
Print dbGetField( "Product Name" ) ; print product name
dbMove( -1 ) ; move back one record
count = count-1 ; decrement counter
Until count = 0 ; until no more records
```

dbMoveFirst( )

SQL Commands
Moves the record pointer to the first record of the current result set.

Syntax

```
dbMoveFirst( )
```

See Also

dbMove( ), dbMoveLast( ), dbMoveNext( ), dbMovePrev( )

Operation

This function moves the record pointer to the first record of the current result set. A runtime error is generated if the result set contains no records or has been closed using the dbClose( ) function.
The function has no return value.

Examples

```
dbConnect( "DSN=EZ Testdemo" ) ; ODBC connection to datasource
dbSelect( "SELECT * FROM CarList" ); select records
While dbEOF = 0 ; while not at end of result set
dbMoveNext( ) ; move to next record
EndWhile
MsgBox( "", dbGetField( "Make" ) ; display last record
dbMoveFirst( ) ; move to first record
MsgBox( "", dbGetField( "Make" ) ; display first record
```

dbMoveLast( )

SQL Commands
Moves the record pointer to the last record of the current result set.

Syntax

```
dbMoveLast( )
```

See Also

dbMove( ), dbMoveFirst( ), dbMoveNext( ), dbMovePrev( )

Operation

This function moves the record pointer to the last record of the current result set. A runtime error is generated if the result set contains no records or has been closed using the dbClose( ) function.
The function has no return value.

Examples

```
dbConnect( "DSN=EZ Testdemo" ) ; ODBC connection to datasource
dbSelect( "SELECT * FROM CarList" ); select records
dbMoveLast( ) ; move to last record
MsgBox( "", dbGetField( "Make" ) ) ; display last record
dbMoveFirst( ) ; move to first record
MsgBox( "", dbGetField( "Make" ) ) ; display first record
```

dbMoveNext( )

SQL Commands
Moves the record pointer to the next record of the current result set.

Syntax

```
dbMoveNext( )
```

See Also
dbMove( ), dbMoveFirst( ), dbMoveLast( ), dbMovePrev( )
Operation
This function moves the record pointer to the next record of the current result set. A runtime error is generated if the result set contains no records or if the result set has been closed using the dbClose( ) function.
If the record pointer is already at the last record, the dbMoveNext( ) function moves the pointer to the dbEOF( ) marker. If you attempt to use the dbMoveNext( ) function to move beyond the dbEOF( ) marker, a runtime error is generated.
The function has no return value.
Examples
```
dbConnect( "DSN=EZ Testdemo" ) ; ODBC connection to datasource
dbSelect( "SELECT * FROM CarList" ); select records
dbMoveFirst( ) ; move to first record
While dbEOF = 0 ; while not at end of result set
MsgBox( "", dbGetField( "Make" ) ); display record
dbMoveNext( ) ; move to next record
EndWhile
```

dbMovePrev( )

SQL Commands
Moves the record pointer to the previous record of the current result set.
Syntax
dbMovePrev( )
See Also
dbMove( ), dbMoveFirst( ), dbMoveLast( ), dbMoveNext( )
Operation
This function moves the record pointer to the previous record of the current result set. A runtime error is generated if the result set contains no records or if the result set has been closed using the dbClose( ) function.
If the record pointer is already at the first record, the dbMovePrev( ) function generates a runtime error.
The function has no return value.
Examples
```
; connect to data source
dbConnect( "c:\Program Files\EZ Test.32\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveLast( ) ; move to last record
count = dbRecordCount ; count no. of records
While count > 0 ; while not at start
MsgBox( "", dbGetField( "Make" ) ) ; display record
dbMovePrev( ) ; move to previous record
count = count - 1
EndWhile
```

dbRecordCount( )

SQL Commands
Returns the number of records in the current result set.
Syntax
count = dbRecordCount( )
See Also
dbEOF( ), dbMoveFirst( ), dbMoveLast( ), dbMoveNext( ), dbSelect( )
Operation
This function returns the number of records in the current result set following a dbSelect( ).

You must move the result set pointer to the end of the result set before executing a dbRecordCount( ). If the data source is a Microsoft Access .MDB file, use dbMoveLast( ). If the data source is accessed via an ODBC driver, visit each record using dbMoveNext( ).

Examples

Example 1:

```
; using a Microsoft Access .MDB file as a data source
dbConnect( "c:\msoffice\access\mydb.mdb" ) ; connect to datasource
dbSelect( "select * from Products" ) ; select all records
dbMoveLast( ) ; move to last record
count = dbRecordCount( ) ; count no. of records
MsgBox( "No. of Records", count ) ; display result
```

Example 2:

```
; connecting to a data source using an ODBC driver
dbConnect( "DSN=EZ Testdemo" )
dbSelect( "select * from CarList" ) ; select all records
dbMoveFirst( ) ; move to first record
While dbEOF( ) = 0 ; while not at the end
dbMoveNext( ) ; move to next record
EndWhile
count = dbRecordCount( ) ; count no. of records
MsgBox( "No. of Records", count ) ; display result
```

dbSelect( )

SQL Commands

Selects records from a SQL data source.

Syntax

dbSelect("SQL" [, "option"])

See Also

dbConnect( ), dbExecute( ), dbClose( )

Operation

Performs an SQL Select query on the current data source. The dbSelect command retrieves data that satisfies a selection criteria and holds the result in a result set. The result set can be analyzed using the various dbMove( ) functions and dbGetField( ) functions.

You may specify one of the following options:

"dynaset" The fields within the result set may be used to update values in the underlying database. This is the default setting.

"snapshot" The result set can be used to examine values in the underlying database only.

The function has no return value.

Examples

Example 1:

```
dbConnect( "c:\msoffice\access\mydb.mdb" ); connect to datasource
dbSelect( "select * from Products" ) ; select all records
dbMoveLast( ) ; move to last record
count=dbRecordCount( ) ; count no. of records
MsgBox( "No. of Records", count ) ; display result
dbClose( ) ; close result set
```

Example 2:

```
; connect to a data source using an ODBC driver
dbConnect( "DSN=EZ Testdemo" )
; select records whose Cost is > 900
dbSelect( "SELECT Make, Year FROM CarList WHERE Cost>900", "snapshot" )
; move to first record in the result set
dbMoveFirst( )
While dbEOF( ) = 0 ; while not at end of result set
print dbGetField( "Make" ) ; print "Make" field
print dbGetField( "Year" ) ; print "Year" field
```

```
print "" ; print blank line
dbMoveNext( ) ; move to next record
dbClose( ) ; clear record
dbDisconnect( ) ; disconnect from datasource
EndWhile
```

Example 3:

```
; connect to a data source using an ODBC driver
dbConnect( "DSN=EZ Testdemo" )
; select cars of a specific make
dbSelect( "SELECT * FROM CarList WHERE Make Like '*Ford*'" )
; move to first record in result set
dbMoveFirst( )
While dbEOF = 0 ; start of loop
dbEdit( ) ; edit fields
Cost=dbGetField("Cost") ; get current cost
dbSetField( "Cost", Str(Cost-1000) ) ; reduce Cost by 1000
dbUpdate( ) ; update database
dbMoveNext( ) ; next record
dbClose( ) ; close result set
EndWhile
```

dbSetField( )

SQL Commands

Sets a field's value within the current record of the current result set.

Syntax

dbSetField("FieldName", value)

See Also

dbAddNew( ), dbEdit( ), dbGetField( ), dbSelect( ), dbUpdate( )

Operation

This function sets the value of the "FieldName" parameter in the current record of the current result set to value. The value parameter may be a string or numeric value, depending on the field data type of the "FieldName" parameter.

Before you can assign a value to a field, you must enter edit mode using the dbEdit( ) or dbAddNew( ) functions, and you must immediately commit the edit with a dbUpdate( ) function.

The dbSetField( ) function may only be executed on fields in a result set obtained from a dbSelect( ). To modify fields in a data source directly, use the dbExecute( ) function.

The function has no return value.

Examples

Example 1:

```
; connect to data source
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveFirst( ) ; move to first record
While dbEOF( ) = 0 ; while not at end
Cost = dbGetField( "Cost" ) ; get current value
dbEdit( ) ; edit mode
dbSetField( "Cost", Cost-100 ) ; set new value
dbUpdate( ) ; commit new value
dbMoveNext( ) ; move to next record
EndWhile
```

Example 2:

```
; this example opens the EZ TESTDemo database, extracts the last
; reference number and generates a new entry
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveLast( ) ; move to last record
Last = dbGetField( "Ref" ) ; get last reference
Last = Right( Last, 4 ) ; extract numeric part
```

```
Next = Last + 10001 ; add 10000 + 1
NewRef = "C-a-" + Right( Next, 4 ) ; make new reference
dbAddNew( ) ; edit mode
dbSetField( "Ref", NewRef ) ; set new value
dbUpdate( ) ; commit new value
```

dbUpdate( )

SQL Commands

Commits an edited field in the current record of the result set back to the data source.

Syntax

dbUpdate( )

See Also

dbAddNew( ), dbEdit( ), dbExecute( ), dbSelect( )

Operation

This function commits an edit to a field within the current result set to the underlying data source. Field values within a result set may be set with a dbSetField( ) function, providing that edit mode has been entered with a dbEdit( ) or dbAddNew( ) function. Changes are not saved unless a dbUpdate( ) is executed.

The dbUpdate( ) function may only be used with records in a result set obtained from a dbSelect( ). To modify fields in a data source directly, use the dbExecute( ) function.

The function has no return value.

Examples

Example 1:

```
; connect to data source
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveFirst( ) ; move to first record
While dbEOF( ) = 0 ; while not at end
Cost = dbGetField( "Cost" ) ; get current value
dbEdit( ) ; edit mode
dbSetField( "Cost", Cost-100 ) ; set new value
dbUpdate( ) ; commit new value
dbMoveNext( ) ; move to next record
EndWhile
```

Example 2:

```
; this example opens the EZ TESTDemo database, extracts the last
; reference number and generates a new entry
dbConnect( "c:\EZ Testcenter.32\demos\EZ Testdemo.mdb" )
dbSelect( "SELECT * FROM CarList" ) ; select records
dbMoveLast( ) ; move to last record
Last = dbGetField( "Ref" ) ; get last reference
Last = Right( Last, 4 ) ; extract numeric part
Next = Last + 10001 ; add 10000 + 1
NewRef = "C-a-" + Right( Next, 4 ) ; make new reference
dbAddNew( ) ; edit mode
dbSetField( "Ref", NewRef ) ; set new value
dbUpdate( ) ; commit new value
```

Delete ArrayName[Element]

Miscellaneous

Deletes a whole array or an element of an array.

Syntax

Delete ArrayName[Element]

Variants

Delete ArrayName

See Also
Var, FillArray( ), ArraySize( )
Operation
This command is used to delete a whole array or an element within an array.
ArrayName is the name of the array and Element is the number of the element to delete.
If Element is not defined, the entire array contents are deleted.
Examples
```
Var rgg[] ; declare array
rgg[1] = "Hello" ; allocate values
rgg[2] = "There"
rgg[3] = "World"
ret = StrCat( " ", rgg[1], rgg[2], rgg[3] ) ; concatenate the
; elements
MsgBox( "Contents", ret ) ; display the result
Delete rgg[2] ; delete element 2
ret = StrCat( " ", rgg[1], rgg[2], rgg[3] ) ; concatenate and
MsgBox( "Contents", ret ) ; re-display result
Delete rgg ; delete all elements
ret = StrCat( " ", rgg[1], rgg[2], rgg[3] ) ; concatenate and
MsgBox( "Contents", ret ) ; re-display result
```

DeleteFile( )

File Access
Delete a specified file.
Syntax
ret = DeleteFile("filespec")
See Also
Create( ), IsFile( ), FileExists( )
Operation
This function deletes “filespec” from disk. It is not placed in the recycle bin.
“filespec” can be a literal or a variable string, and wildcard characters can be used.
The function returns 1 if the operation is successful, and returns 0 if it fails because "filespec" is invalid.
A runtime error is generated if the file exists but cannot be deleted (for example, the READ-ONLY attribute is set).
Examples
Example 1:
```
ret = DeleteFile( "C:\MYFILE.TXT" ) ; delete this file
MessageBox( "Result", ret ) ; display result
```
Example 2:
```
target = "C:\Bob's working folder\dummy.txt"
DeleteFile( target )
```
Example 3:
```
DeleteFile( "c:\*.old" ) ; delete all old files
```

DeleteStr( )

String Manipulation
Deletes a string within a target string.
Syntax
ret = DeleteStr(target, start, length)
Variants
ret = DeleteStr(target, "text", count)
See Also
Left( ), Right( ), Mid( ), InsertStr( )
Operation
Deletes a string contained within a target string. The target is updated. The function has

two forms depending on the type of the second parameter.

- If the second parameter is numeric, it denotes the start position for the deletion. It can optionally be followed by the length of the string to delete.
- If the second parameter is a string, it can be followed by a number specifying the number of matching substrings to delete.

The parameters are as follows:

target The string from which to delete.

start The position in target to start the deletion.

length The number of characters to delete. If omitted, all characters to the end of the target are deleted.

"text" The text to be deleted.

count The number of instances matching "text" to delete. If set to 0, every instance is deleted. The default value is 1.

The function returns the number of deletions made.

Examples

```
target = "the quick brown fox"
DeleteStr( target, 5, 6 ) ; target becomes "the brown fox"
target = "the quick brown fox"
DeleteStr( target, 4 ) ; target becomes "the"
target = "the quick brown fox"
ret = DeleteStr( target, 0, 4 ) ; returns 0, target is unchanged
; 0 is an invalid start position
target = "a b c a a c d a a e "
DeleteStr( target, "a ", 0 ) ; delete all "a "
; target becomes "b c c d e "
target = "a b c a a c d "
DeleteStr( target, "a " ) ; delete first instance of "a "
; target becomes "b c a a c d "
target = "a b c a a c d a a e "
DeleteStr( target, "a ", 4 ) ; delete 4 instances of "a "
; target becomes "b c c d a e "
target = "a b c a a c d a a e "
ret = DeleteStr( target, "x ", 4 ); delete 4 instances of "x "
MsgBox( ret, target ) ; returns 0 (no deletions)
; target is unchanged
target = "a b c a a c d a a e "
ret = DeleteStr( target, "a ", 7 ); delete 7 instances of "a "
MsgBox( ret, target ) ; returns 5 (5 "a "s deleted)
; target becomes "b c c d e "
```

DestroyEvent( )

Synchronization

Destroys the specified MakeEvent from memory.

Syntax

DestroyEvent(Eventname)

See Also

MakeEvent( )

Operation

This command destroys a specific event created using the MakeEvent( )command and releases it from memory. This command can be used to free memory resources if your script contains many MakeEvent commands. After the DestroyEvent( )command is issued, the specified event can no longer be triggered. As a general rule, the DestroyEvent( )command should be issued after the event trigger is no longer applicable. This command does *not* apply to the events that are created using the event wizard and subsequently inserted into the script *without* selecting the **Paste MakeEvent** option from *EZ Test*'s Insert Event dialog box.

Example

Example 1:
```
Function Main
Keyboard0001 = MakeEvent( "Keyboard event throwaway",
; Event Type
"anywindow", ; Attach
"{F12}" ) ; Key list )
Wait(30, "", Keyboard0001 )
DestroyEvent( Keyboard0001 )
; The following will cause a runtime error to occur,
; Because the event has been destroyed from memory
Wait(30, "", Keyboard0001 )
End Function ; Main
```
Example 2:
```
Function Main
ScreenNotepad = MakeEvent( "Screen event", ; Event Type
"Run PopupWindow", ; Attach
"notepad", ; Search text
"erase" ) ; Screen options
Wait(1, "", ScreenNotepad )
DestroyEvent(ScreenNotepad) ;Destroy the event from memory
MessageBox( "" , "ok" , 'ok' )
End Function ; Main
```

Dialog( )

Miscellaneous
Calls a user-defined dialog box.

Syntax

Dialog "dialog name", arrayofcontrols

Variants

Dialog "dialog name", arrayofcontrols, "center"
Dialog "dialog name", arrayofcontrols, x, y

See Also

MessageBox( ), PromptBox( ), TextPanel( )

Operation

This command is used to display a dialog that is defined using the *EZ Test* dialog editor. Normally, the dialog command is created using the **Insert>Dialog** menu item.

dialog name The name of the dialog to view.

arrayofcontrols The name of the array that will receive the values of the controls used on the dialog.

center This optional parameter centers the dialog box on the screen.

x , y These optional parameters position the dialog box at the x- and y- position relative to the top-left corner of the screen.

When the Dialog command is pasted into the Editor, the array of controls and examples of the array names (as comments) are pasted before the command. If the elements are modified, the dialog that is displayed reflects these changes. If no changes are made, the details of the definition are used.

Once the dialog has executed, the array elements reflect the changes the user made to the dialog (filling in text, selecting radio buttons, etc.). For example, if the dialog contained push buttons, an extra array element is created named "Push Button Clicked". For example:

arrayofcontrols["Push Button Clicked"]Contains the name of the push button selected by the user.

arrayofcontrols["Selection Count"] Contains the number of entries selected for multi-line list boxes.

arrayofcontrols["Last X Position"] Returns the x-position of the

dialog when it was destroyed.
This allows you to replace the
dialog at its last position.
This is useful for validating values
that a user entered when the dialog
is displayed a “second” time in the
same location (assuming the
window was moved).
arrayofcontrols["Last Y Position"] Returns the y-position of the
dialog when it was destroyed.
This allows you to replace the
dialog at its last position.
This is useful for validating values
that a user entered when the dialog
is displayed a “second” time in the
same location (assuming the
window was moved).
arrayofcontrols["Control Name","Selected Items", position]
Returns the value of the selected
items for a particular list or
combobox control. Useful for
handling lists that allow you to
select multiple items.
The following details the data types of the different controls used by the dialog.
Edit Controls Text
Push Buttons Numeric, 1 selected, 0 not selected
Radio Buttons Numeric, 1 selected, 0 not selected
Check Boxes Numeric, 1 selected, 0 not selected
List Boxes Text
Combo Boxes Text

Loading Control Arrays Before Executing a Dialog:

```
Var Controls[ ]
; Assigning controls
Controls[ "Edit1" ] = "Fred" ; Assign edit control to "Fred"
Controls[ "Radio1" ] = 1 ; Turn radio button on
Controls[ "Check1" ] = "Fred" ; Select check box
; Load list box array (multi dimensional)
Controls[ "List1", 1 ] = "Fred"
Controls[ "List1", 2 ] = "Bill"
Controls[ "List1", 3 ] = "John"
; Load combo array (multi dimensional)
Controls[ "Combo1", 1 ] = "Fred"
Controls[ "Combo1", 2 ] = "Bill"
Controls[ "Combo1", 3 ] = "John"
Controls[ "Picture" ] = "EZ Test.bmp" ; Specify the bitmap/icon
```

Retrieving Values After a Dialog Executes:

```
; Retrieving
textval = controls[ "Edit1" ] ; Edit controls
numval = controls[ "Radio1" ] ; Radio buttons
numval = controls[ "Check1" ] ; Check boxes
textval = controls[ "List1" ] ; List boxes
textval = controls[ "Combo1" ] ; Combo boxes
numval = controls[ "Ok" ] ; Push buttons
```

Examples

Example 1:

```
; Setup the array to use
Var arrayofcontrols[]
; Load the elements to override the preset values
arrayofcontrols[ "Userid" ] = "User Text"
```

```
arrayofcontrols[ "Edit1" ] = "Defaultuserid"
arrayofcontrols[ "Password" ] = "Pass Text"
arrayofcontrols[ "Edit5" ] = "Default password"
// arrayofcontrols[ "OK" ]
// arrayofcontrols[ "Cancel" ]
; Call the actual dialog
Dialog "System Access dialog", arrayofcontrols
Switch( arrayofcontrols[ "Push Button Clicked" ] )
Case "OK"
MsgBox( "Dialog Result", , "OK selected, " +
"User Id Entered = " +
arrayofcontrols[ "Edit1" ] )
Case "Cancel"
MsgBox( "Dialog Result", "Cancel Selected" )
Default
MsgBox( "Dialog Result", "Nothing Selected" )
End Switch
```

Example 2:

```
var dig1_Controls[]
// dig1_Controls[ "1stBox1" ]
// dig1_Controls[ "Push button1" ]
; Load list box array (multi dimensional)
dig1_Controls[ "1stBox1", 1 ] = "Fred"
dig1_Controls[ "1stBox1", 2 ] = "Bill"
dig1_Controls[ "1stBox1", 3 ] = "John"
dialog "dig1", dlg1_Controls
ret = dig1_Controls[ "1stBox1", "Selected Items", 1 ]
msgbox "", ret, 'ok' //display first selected item in the list
```

Dir( )

File Access

Returns the next file in a folder matching a given criteria.

Syntax

```
target = Dir( "filespec", "filter", "format" )
```

Variants

```
target = Dir( )
target = Dir( "filespec" )
target = Dir( "filespec", "filter" )
```

See Also

FillArray( )

Operation

This function returns the name of the next file that matches that defined in “filespec”. Use the syntax Dir(“filespec”) to get the first file matching the required file specification. To get the next matching file, call the Dir( ) function again without specifying any parameters.

The parameters are as follows:

target A string or array variable.

filespec The search pattern used to find the filenames; this can include a path name and wildcard characters.

filter Specifies which type of filenames to include; if omitted the default is “fdr”. See below for valid filters.

format The display format for the filenames. This can contain normal characters as well as embedded codes. See below for formatting options.

The filter options are:

"f" Show normal files excluding hidden and system files. This is part of the default.

"d" Show directories. This is part of the default.

"h" Show hidden files.

"s" Show system files.
"r" Include read-only files. This is part of the default.
If omitted, or a null value is specified, the default options “fdr” (directories and normal files, including read-only files, but excluding system and hidden files) are used.
The format options are:
"<b>" Expand to the complete filename, including extension.
"<n>" The filename without the extension.
"<A>" Display the access time in the format “dd-mm-yyyy hh:mm:ss”.
"<M>" Display the modified time.
"<C>" Display the creation time.
"<a>" Display the file attributes.
"<s>" Display the file size up to 10 digits.
All format options are case sensitive. If omitted or a null value is specified, and then default format “<b>” is taken. To include a format, a filter must also be specified, even if it is a null.

Examples

Example 1:

```
ret = Dir( "c:\windows\*.txt" ) ; what files to find
While ret ; while ret is not empty
MsgBox( "Result", ret ) ; display the result
ret = Dir( ) ; update ret with next value
Wend ; end the while loop
```

Example 2:

```
Var filename[] ; set up array of filenames
c = 1 ; initialize counter
; get name and last modified date/time of hidden files
; in the root directory
filename[c] = Dir( "c:\*.*", "sh", "<b> <M>" )
While filename[c] ; while not empty
MsgBox( "the_file", filename[c] ) ; display details
c = c+1 ; increment counter
filename[c] = dir( ) ; get next file
Wend ; end of loop
```

DLLFunc

Miscellaneous
Calls an external DLL function.

Syntax

Declare DLLFunc "<prototype>" <LanguageName>

See Also

SetStrLen( )

Operation

This command allows a *EZ Test* script to call a function from an external DLL. The DLLFunc must be declared outside of a function.
The <LanguageName> is the name that you use to call the function from your script. It does not have to be the same as the actual DLL function name.
The "<prototype>" is the prototype of the DLL function. The format of the string is:
"<retval> <functionname>(<parameters>) <DLLpath>"
Where:
<retval> Defines the return type of the function. This can be:
char A signed char.
byte An unsigned char.
short A short (16-bit integer).
ushort An unsigned short (16-bit integer).
int A signed integer (32-bit integer).
uint An unsigned integer (32-bit integer).
long A signed long (32-bit integer).

ulong An unsigned long (32-bit integer).
str A null terminated string.
void The function does not return a value.
<functionname> Is the name of the DLL function to import.
<parameters> Is a list of the function parameters. This will be similar to the real prototype of the API function. Each parameter can be:
char A signed char.
byte An unsigned char.
short A short (16-bit integer).
ushort An unsigned short (16-bit integer).
int A signed integer (32-bit integer).
uint An unsigned integer (32-bit integer).
long A signed long (32-bit integer).
ulong An unsigned long (32-bit integer).
str A null terminated string.
Parameters can also be passed by reference. This means that the DLL function expects a pointer to the parameter, and it will modify the value. A '*' after the parameter type signifies pass by reference. String parameters are always passed by reference so you do not need a '*' after a 'str' type.
<DLLpath> Is the name of the DLL in which the function resides. If an extension is included, the script will not compile. If no path is specified, Windows searches for the file in the following sequence:
1. *EZ Test* .exes directory
2. Current directory
3. Windows system directory
4. Windows directory
5. Directories in the path.
When you declare a DLLFunc, you must ensure that the definition matches the actual function you are calling. Failure to do this may cause the system to hang.
Some DLL functions take a string buffer as a parameter and fill it with data. When you pass string variables to such functions, you must ensure the string you pass is large enough. Use SetStrLen( ) to prepare the variable so that it can be used as an argument to a DLL function. Do not use PadStr( ) (or any other string function) to set the size of a string for a DLL call. If the function takes a string parameter, but does not modify it, there is no need to use SetStrLen( ).
Most function in kernel32 or user32 that deal with strings have two forms — to deal with Unicode strings or ASCII strings. These are distinguished by a code (W or A) at the end of the function name. Strings in *EZ Test* are ASCII-based, so you should only call functions that work on ASCII strings.

Examples

Example 1:

```
; declare a DLLFunc to set the name of a window
Declare DllFunc "int SetWindowTextA( uint, str ) user32" SetWindowText
Function Main
; get the handle of the window to pass to the DLLFunc
hWnd = AttachWindow( "Untitled - Notepad MainWindow" )
```

Note

Incorrect use of DLL calls may cause the system to hang. Use this command only if you fully understand the implications of using DLL calls.

```
; pass the window handle and new title to the function
SetWindowText( hWnd, "New Title" )
End Function ; Main
```

Example 2:

```
; declare a DLLFunc to get the name of the computer
Declare DllFunc "int GetComputerNameA( str, ulong* ) kernel32" GetName
Function Main
; prepare a string to receive the name
```

```
len=100
SetStrLen( Name, len )
; call the DLL function
GetName( Name, len )
; display the result
MsgBox( "Computer Name", Name )
End Function ; Main
```

Do...Loop While

Program Flow
Repeats a series of instructions while a condition is true.

Syntax

```
Do
<Instructions>
Loop While <Boolean Expression>
```

See Also

Break, Continue, For…Next, Repeat…Until, While…Wend

Operation

This command executes the <Instructions> between the Do and Loop While statements repeatedly until <Boolean Expression> is false. Execution of the script then continues on the statement following the Loop While. The <Boolean Expression> can contain literals or variables, including return values from functions.
The command is very similar to the Repeat...Until structure, the only difference being that Do...Loop While exits the loop when <Boolean Expression> is false while Repeat...Until exits the loop when <Boolean Expression> is true.
Because <Boolean Expression> is evaluated after <Instructions> are executed, the loop always executes at least once.

Examples

```
i = 1
Do
MsgBox( "i is now", i )
i = i+1
Loop While i<6
Do
MsgBox( "Random Number", Random( ) )
Loop While MsgBox( "Run Again?", "Pick another number?", "yesno" ) = 6
Do
text = Capture( "~P~KERNEL32.DLL~ReportWnd~PARTS.LST" )
ScrollBarWindow 1, "Page Vert"
Loop While FindStr( text, "More..." )<>0
```

EditClick( )

Dialog Control
Clicks the mouse in an edit control.

Syntax

```
EditClick( "ControlId", "Options", x, y )
```

See Also

EditText( ), MouseClick( )

Operation

This function simulates clicking the mouse button in an edit control, to give it focus.
The parameters are as follows:
ControlId Specifies the label shown to the side of the edit control. If the ControlId is numeric, it represents the index value of the edit control: "~1" for the first edit control, "~2" for the second, etc.
x The x-position, relative to the left edge of the edit control, where the click is performed.

y The y-position, relative to the top of the edit control, where
the click is performed.
The "Options" are the standard MouseClick options:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the mouse button.
"singleclick" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Used in conjunction with "control" and "shift".
The function returns 1 if the control is selected, and it returns 0 if it is not.

Examples

```
; this example attaches to the "Find All Files" dialog, selects the
; "Advanced" search tab, clicks in the "Containing text" edit
; control and enters the search string "EZ Test".
Attach "Find: All Files MainWindow"
TabCtrl "~1", "Advanced", 'Left SingleClick'
EditClick "&Containing text:", 'Left SingleClick' 95, 10
EditText "&Containing text:", "EZ Test"
```

EditLine( )

Window Information
Retrieves a line of text from a multi-line edit control.

Syntax

```
ret = EditLine( hCtrl, LineNo )
```

See Also

ControlFind( ), EditLineCount( )

Operation

This function retrieves the text specified by the LineNo line of the multi-line edit control whose window handle is hCtrl. The window handle can be obtained by using the EditFind( ) function — one of the ControlFind( ) group of functions.
The function can only be used on an edit control.
An empty string is returned if the specified control contains no text. A runtime error is generated if the control is not an edit control or if the window handle is invalid.

Examples

```
; this example works through a list of names which have been
; read from a data file into Notepad
Attach "Names - Notepad MainWindow" ; attach to Notepad
hCtrl = FocusWindow( ) ; get handle of edit window
total = EditLineCount( hCtrl ) ; get number of lines in file
count = 0 ; initialize counter
While count != total ; while not equal to total
nextname = EditLine( hCtrl, count ) ; get the next line
< Process name > ; process it
count = count + 1 ; increment counter
Wend ; until all done
```

EditLineCount( )

Window Information
Returns the number of lines in a multi-line edit control.

Syntax

```
Count = EditLineCount( hCtrl )
```

See Also

ControlFind( ), CtrlSelText( ), CtrlText( )

Operation

This function returns the number of lines in the multi-line edit control whose window handle is hCtrl. The window handle can be obtained by using the EditFind( )function — one of the ControlFind( ) group of functions. This function can only be used on an edit control.

The function returns 0 if the window handle is invalid or the specified control is not an edit control.

Examples

```
; this example calculates the number of lines in Notepad's edit
; window and selects them all
Attach "Notepad MainWindow" ; attach to Notepad
hCtrl = FocusWindow( ) ; get handle of the edit window
ret=EditLineCount( hCtrl ) ; get the line count
Attach hCtrl ; attach to it
Type "{Control {Home}}" ; send cursor to top
count=0 ; initialize counter
while count != ret ; while not number of lines
Type "{Shift {Down}}" ; select line
count = count + 1 ; increment counter
endwhile ; until all selected
```

EditText( )

Dialog Control

Enters text into an edit control.

Syntax

ret = EditText("ControlId", "Text")

See Also

Button( ), CheckBox( ), ComboBox( ), ComboText( ), ListBox( ), RadioButton( ), ScrollBar( )

Operation

This function loads the text in the “Text” parameter into the edit control specified by the “ControlId” parameter within the currently attached dialog box.

The parameters are:

"ControlId" Specifies the label shown to the side of the edit control. If the ControlId is numeric, it represents the index value of the edit control: "~1" for the first edit control, "~2" for the second, etc.

"Text" The text to be entered into the edit control. The value of “Text” can be literal or the contents of a variable.

The function returns 1 if the control processed, and it returns 0 if it did not. When this command is generated by the Learn facility, the parentheses are omitted

Examples

Example 1:

```
; enter the text file name into the File Open dialog box
Attach "~N~KERNEL32.DLL~#32770~Open"
EditText "~1", "testps.txt"
Button "OK", 'SingleClick'
```

Example 2:

```
; open all text files in the specified directory
; declare an arrayed variable
var TextFile[]
; fill array with .TXT filenames
files = FillArray( TextFile, "c:\windows\*.txt" )
; return the number of array elements
FilesFound = ArraySize( TextFile )
; start at the beginning of the array
ElementNumber = 1
```

```
; construct the loop
While FilesFound > 0
; while files found is greater than zero
Attach "~N~KERNEL32.DLL~#32770~Open"
EditText "~1", TextFile[ElementNumber] ; enter value
Button "OK", 'SingleClick'
ElementNumber = ElementNumber+1 ; go to next element
; in array
FilesFound = FilesFound-1 ; decrease files by 1
Wend
```

Err

Miscellaneous
Reports the current error code.

Syntax

```
ErrorCode = Err
```

Variants

```
Err = Value
```

See Also

ErrFile, ErrFunc, ErrLine, ErrMsg, Error, Error Codes, On Error, Resume Next

Operation

Err contains the numeric error code of the last error. You can also assign a value to Err to invoke your own error handling routines.

Examples

```
Function Main
On Error Call Main_Error_Handler ; main error handler
< Instructions >
Attach "MyAppWindow" ; attach to a window
Call Test_Control ; check for a control
<Instructions >
End Function ; Main
Function Test_Control
On Error Call FindControlError ; find control error handler
hWnd = ButtonFind( "Cancel" ) ; get handle to a control
If hWnd = 0 ; if not there
Error 12345 ; raise own error code
Endif
End Function; Test_Control
Function FindControlError ; handler for test_control
If Err != 12345 ; if error isn't user defined
Error ; pass control to main handler
Else ; otherwise
< Handle own error > ; deal with it
Endif
End Function ; FindControlError
Function Main_Error_Handler
< Handle all other errors >
End Function ; Main_Error_Handler
```

ErrFile

Miscellaneous
Reports the name of the script file that generated an error.

Syntax

```
FileName = ErrFile
```

See Also

Err, ErrFunc, ErrLine, ErrMsg, Error, Error Codes, On Error, Resume Next

Operation

ErrFile contains the name of the script file that generated the current error.

Examples

```
Function Main
On Error Call Main_Error_Handler; main error handler
Run "First Test"
Run "Second Script"
...
End Function ; Main
Function Main_Error_Handler
Error_Message = " Error in line " + Str( ErrLine ) + " of File " +
ErrFile + ". Error is " + ErrMsg + " - " + Str( Err )
WriteLine( "c:\audit.log", Error_Message )
End Function ; Main_Error_Handler
```

ErrFunc

Miscellaneous

Reports the function that caused a runtime error.

Syntax

```
ErrType = ErrFunc
```

See Also

Err, ErrFile, ErrLine, ErrMsg, Error, Error Codes, On Error, Resume Next

Operation

This variable is used within an 'On Error' handler to determine the type of function which caused the error. The result is always in lowercase.

ErrFunc contains a null if the error was raised explicitly by the Error command.

Examples

```
Function MyErrorHandler ; error handler
If ErrFunc = "attach" ; if an Attach error
Call HandleAttach ; call routine handling Attach errors
Else
Error ; call previous error handler
Endif
End Function
```

ErrLine

Miscellaneous

Reports the line number of the script where an error was generated.

Syntax

```
LineNumber = ErrLine
```

See Also

Err, ErrFile, ErrFunc, Error, Error Codes, On Error, Resume Next

Operation

ErrLine contains the line number of the script where an error was generated.

Examples

```
Function Main
On Error Call Main_Error_Handler; main error handler
Run "First Test"
Run "Second Script"
...
End Function ; Main
Function Main_Error_Handler
Error_Message = "Error in line " + Str( ErrLine ) + " of File " +
ErrFile + ". Error is " + ErrMsg + " - " + Str( Err )
WriteLine( "c:\audit.log", Error_Message )
End Function ; Main_Error_Handler
```

ErrMsg

Miscellaneous
Reports a textual description of the current error.

Syntax

Reason = ErrMsg

See Also

Err, ErrFile, ErrFunc, ErrLine, Error, Error Codes, On Error, Resume Next

Operation

The ErrMsg contains a description of the current error. The descriptions and the corresponding error codes are described in the *EZ Test* online help.

Examples

```
Function Main
On Error Call Main_Error_Handler; main error handler
Run "First Test"
Run "Second Script"
...
End Function ; Main
Function Main_Error_Handler
Error_Message = "Error in line " + Str( ErrLine ) + " of File " +
ErrFile + ". Error is " + ErrMsg + " - " + Str( Err )
WriteLine( "c:\audit.log", Error_Message )
End Function ; Main_Error_Handler
```

Error

Program Flow
Aborts the current error handler and calls the previous one.

Syntax

Error

Variants

Error < ErrCode > , < ErrMsg >

See Also

Err, ErrFile, ErrFunc, ErrLine, ErrMsg, Error Codes, On Error, Resume Next

Operation

This command aborts the current error handler and calls the previous one in the chain. If there is no previous error handler, a runtime error is generated and the script stops.
This command should only be used within an error handling routine.
The variant Error < ErrCode > < ErrMsg > raises the error specified by ErrCode.
< ErrMsg > is an optional textual description of the error. If not specified, it defaults to "The statement 'error < ErrCode >' was executed."

Examples

```
Function Main
On Error Call Main_Error_Handler ; main error handler
< Instructions >
Attach "MyAppWindow" ; attach to a window
Call Test_Control ; check for a control
<Instructions >
End Function ; Main
Function Test_Control
On Error Call FindControlError ; find control error handler
hWnd = ButtonFind( "Cancel" ) ; get handle to a control
If hWnd = 0 ; if not there
Error 1234 , "No Button" ; raise a "No Button" error
Endif
End Function ; Test_Control
Function FindControlError ; handler for test_control
If Err != 1234 ; if error is not "No button"
Error ; pass control to main
```

```
handler
Else ; otherwise
< Handle "No Button" > ; deal with it
Endif
End Function ; FindControlError
Function Main_Error_Handler
< Handle all other errors >
End Function ; Main_Error_Handler
```

Event( )

Synchronization
Checks the status of an event.

Syntax

```
ret = Event( "EventId" )
```

See Also

MakeEvent( ), Wait( ), Whenever

Operation

This function returns the status of the event specified by the "EventId" parameter. The status of the event is reset after the call.
The Event( ) function can be used to determine if the event referred to in a Wait( ) function (or a Whenever) actually occurred or whether the Wait( ) function timed out.
This function returns 1 if the event has triggered since the last call to Event( ), or it returns 0 if the event has not occurred.

Examples

Example 1:

```
Exec "NOTEPAD.EXE" ; start application
Wait( 10, "", "NPExists" ) ; allow 10 secs for it to exist
If Event( "NPExists" ) = 1 ; if it does
MsgBox( "", "You may proceed" ) ; display message
Else
MsgBox( "Warning", "Notepad not responding" )
Endif
```

Example 2:

```
Wait( 30, "", "DTMove" ) ; wait up to 30 secs for a window
Wait( 10, "", "DTMin" ) ; to be moved, 10 secs to be
Wait( 30, "", "DTClose" ) ; minimized and 30 secs to close
Msgbox( "DTMove", Event( "DTMove" ) )
Msgbox( "DTMin", Event( "DTMin" ) ); show which actions occurred
Msgbox( "DTClose", Event( "DTClose" ) )
```

Example 3:

```
Function Main
;Function to return the name of the last event satisfied
Wait(30, "for any", "enterkey", "Escape", "F1")
if Event( "enterkey" )
MessageBox ( "", "enterkey" )
endif
if Event( "Escape" )
MessageBox( "","Escape" )
endif
if Event( "F1" )
MessageBox( "", "F1 hit" )
endif
```

Exec( )

Program Flow
Executes a program.

Syntax

```
ret = Exec( "filename", "options" )
```

Variants
Exec("filename")
Exec("filename", "options")
See Also
Chain( ), Run( ), IsRunning( )
Operation
This function executes the program specified by the "filename" parameter. If the filename extension is omitted, the default .EXE will be used. If the filename does not contain a path, the following directories are searched in order:
1. The current directory.
2. The Windows system directory.
3. The Windows directory.
4. Directories listed in the path.
The parameters for the function are as follows:
"filename" Specifies the program to execute. An optional command can be added.
"iconized" If used as an option, the program will be iconized when run.
"nowaitidle" Do not wait for the program to become idle. Continues execution of the script without waiting for the program to complete initialization.
The function returns 1 if the program runs successfully, and it returns 0 if it does not.
Examples
; open the system.ini file using Notepad
ret = Exec("notepad")
if ret = 0 ; check it worked
Fatal("Operation Failed") ; if not generate runtime error
endif

Exit( )

Program Flow
Exits the current script.
Syntax
Exit([ReturnValue])
See Also
ExitWindows( ), Fatal( ), Stop
Operation
This function causes the current script to terminate and return control to the parent script (if any). You may use this command to exit the script and issue a test return value. The parameters are as follows:
ReturnValue This is an optional parameter that returns a pass or fail status when the script is exited. A ReturnValue of 1 exits and passes the script. A ReturnValue of 2 exits and fails the script.
Examples
Public exitcode ; public variable for exitcode
<Instructions> ; script to logon to target
exitcode = "Invalid Password" ; save reason for exit
Exit( ) ; exit current script
End function

ExitWindows( )

Program Flow
Shuts down Windows.
Syntax
ExitWindows "Logoff|Reboot|Shutdown"
Variants
ExitWindows "Force Logoff|Reboot|Shutdown"

See Also
Exit( ), Fatal( ), Stop
Operation
This command shuts down Windows. One or more of the following options must be specified:
Logoff Closes all programs and logs on as a new user for Windows 2000 and NT systems.
Reboot Restarts the computer.
Shutdown Shuts down the computer.
The optional "Force" parameter forces all applications to terminate, even if they are not responding. Use this option with caution — you will lose changes to any unsaved documents.
Examples
ExitWindows "shutdown" ; close all applications and
; shuts down the computer.
; Windows prompts you to save
; changes
ExitWindows "logoff" ; close all applications and log
; on as a new user.
; Windows prompts you to save
; changes
ExitWindows "force reboot" ; close all applications,
; close Windows and reboot.
; Windows does not prompt
; you to save changes

Fatal( )

Program Flow
Generates a fatal runtime error and aborts the script.
Syntax
Fatal(message)
Variants
Fatal( )
See Also
Exit( ), ExitWindows( ), Stop
Operation
This function generates a fatal runtime error, and aborts the current script and all of its parents. The message parameter specifies the fatal error message generated. If message is not specified, a default error “Fatal error issued within the script” is generated.
Examples
Fatal( ) ; generates a " Fatal error issued
; within the script" error message
; and closes the script.
Fatal("Invalid Password"); generates an "Invalid Password"
; error and closes the script.

FileExists( )

File Access
Checks if a file exists.
Syntax
ret = FileExists("filename")
See Also
IsFile( ), Create( ), Read( ), Write( ), DeleteFile( )
Operation

This function is used to check a file's existence before processing it further. The function returns 1 if the file exists, and it 0 if it does not. If no path is specified as part of filename, the current directory is assumed.

Examples

```
ret = FileExists( "c:\config.sys" )
if ret = 1 ; if file exists
Copy( "c:\config.sys", "c:\config.bak" )
else ; if not
MessageBox( "Warning", "Config file missing" )
endif
```

FilePos( )

File Access

Returns or sets the position of the filepointer.

Syntax

```
ret = FilePos( "filename", position )
```

Variants

```
ret = FilePos( "filename" )
```

See Also

Open( ), Read( ), ReadIni( ), ReadLine( ), Write( ), WriteLine( )

Operation

This function returns the current position of the filepointer in the "filename" parameter. If position is specified, then the filepointer is set to this value. The FilePos( ) function is automatically updated after any Read( ) or Write( ) to the specified file. The parameters are as follows:

"filename" The file to report.

position Sets the FilePos( ) function to this position (optional).

The function returns a positive integer if the filepointer is read or set successfully, and it returns 0 if it is not.

Examples

Example 1:

```
Open( "c:\config.sys", "readwrite" )
ret = FilePos( "c:\config.sys" ) ; returns 1 (start of file)
```

Example 2:

```
; update a value in a fixed length record data file
filename = "c:\data\names.dat" ; a file containing names
Open( filename, "readwrite" ) ; open it for read/write access
Do ; start of loop
sav = FilePos( filename ) ; save the filepointer
ReadLine( filename, nextname ) ; read the next line
RTrimStr( nextname ) ; trim the trailing spaces
If nextname = "Miss Vera Jones" ; if it is the right name
FilePos( filename, sav ) ; move filepointer to
; start of line
newname = PadStr( "Mrs Vera West, 40 ); pad the new name
WriteLine( filename, newname ) ; update the record
EndIf
Loop While FileStatus( filename ) <> 2 ; stop at end of file
```

FileStatus( )

File Access

Returns the status of a previously opened file.

Syntax

```
ret = FileStatus( "filename" )
```

See Also

FilePos( ), Open( ), Read( ), ReadIni( ), ReadLine( ), WriteLine( )

Operation
This function returns the current status of the "filename" parameter, a file that must previously have been opened with the Open( ) function. The return values are as follows:
0 The file is not currently open.
1 The file is open and okay to use.
2 The end of file has been reached.
3 An error occurred when a Read( ) or Write( ) was attempted.

Examples
```
; read values from a file, with error handling
filename = "c:\data\data.dat" ; the data file
Do
ReadLine( filename, nextline ) ; read next line of file
If FileStatus( filename ) = 0 ; if not open
Open( filename ) ; open it
Continue ; and go back to the top
Elseif FileStatus( filename ) = 2 ; if at the end
Break ; exit the loop
Elseif FileStatus( filename ) = 3 ; if an error occurs,
Call "File_Error" ; call error handling routine
Else ; otherwise
< Instructions > ; process the data
Endif
Loop While 1 = 1 ; endless loop
```

FileTime( )

File Access
Gives the date and time a file was last modified.

Syntax
```
ret = FileTime( "filename", datetime )
```

See Also
FilePos( ), Open( ), Read( ), ReadIni( ), ReadLine( )

Operation
This function updates the datetime parameter with the date and time that the "filename" parameter was last modified. The datetime value can be used as a parameter to the Date( ) and Time( ) functions.
The parameters are as follows:
"filename" The file to report.
datetime The date and time value the file was last modified.
The function returns 1 if the date/time is retrieved successfully, and it returns 0 if it is not. If the function fails, datetime reflects 01/01/1970 at 00:00:00.

Examples
```
; update one file if it is older than another
FileTime( "file1", file1time ) ; get the date and time of file1
FileTime( "file2", file2time ) ; and the date and time of file2
If file1time < file2time ; if file1 is older than file2
Copy( "file2", "file1" ) ; update it
Endif
```

FillArray( )

File Access
Fills an array with filenames matching a filespec.

Syntax
```
ret = FillArray( arrayname, filespec, "filter", "format" )
```

Variants
```
ret = FillArray( arrayname, filespec )
ret = FillArray( arrayname, filespec, "filter" )
```

See Also
ArraySize( ), Delete ArrayName[Element], Dir( ), Var

Operation
This function fills an array with file names matching a given specification. The function returns the number of matching files.
The parameters are as follows:
arrayname The name of the array to fill.
filespec The search pattern that specifies the file names required. This can include a path name and wildcard characters.
filter Specifies the types of files to include. If omitted, the default is "fdr". See below for valid filters.
format Specifies the display format to use. The string can contain normal characters and embedded codes. See below for formatting options.
The "filter" options are:
"f" Include all normal files — exclude hidden and system files unless "h" or "s" are also specified. This is part of the default filter setting.
"d" Include directories. This is part of the default.
"h" Include hidden files.
"s" Include system files.
"r" Include read only files. This is part of the default.
The "format" options are:
"<b>" Expand to the base name — the file name including the extension.
"<n>" The file name without the extension.
"<A>" Display the access time in the format “yyyy-mm-dd hh:mm:ss”.
"<M>" Display the modified time.
"<C>" Display the creation time.
"<a>" Display the file attributes.
"<s>" Display the file size (expanded to 10 digits).
All options are case sensitive. If omitted or a null value is specified, the default format "b" is taken.

Examples
Example 1:
```
var target[] ; the array to fill
ret = FillArray( target, "*.exe" ) ; fill with .EXE filenames
MessageBox( "Result", ret ) ; display number found
```
Example 2:
```
; fill the array with .EXE files, their access times,
; file sizes, and full names
FillArray( target, "*.*", "", "<A><s><b>" )
MsgBox( "Result", target[3] ) ; display element 3 of the array
```
Example 3:
```
; fill the array with base names of all files
FillArray( target, "*.*", "", "Filename is <b>" )
MsgBox( "Result", target[3] ) ; display element 3 of the array
```

FindChar( )
String Manipulation
Scans a string for the first character that is not in a search list.

Syntax
FindChar(target, searchlist, "Match" | "Nonmatch", start)

Variants
FindChar(target, searchlist, "Nonmatch")
FindChar(target, searchlist, "Match")
FindChar(target, searchlist)

See Also
FindStr( ), IgnoreCase( ), InStr( )

Operation
This function scans the target string and, by default, returns the position of the first character that is not contained in the search list. The search is case-sensitive and is unaffected by the IgnoreCase ( ) flag. The parameters are as follows:
target The string to search. If this parameter is numeric, it is automatically converted to a string.
searchlist The list of search characters. If this parameter is numeric, it is automatically converted to a string.
"Nonmatch" Returns the position of the first character in target that is not contained in searchlist; this is the default.
"Match" Returns the position of the first character in target that is contained in searchlist.
start The position in target at which to start the search.

Examples
Example 1:
target = "the quick brown fox" ; the target string
searchlist = "abcde" ; the search list
ret = FindChar(target, searchlist) ; result is 1 ("t")
ret = FindChar(target, searchlist, "match"); result is 3 ("e")
Example 2:
ret = FindChar("abcd", "abcdefgh") ; returns 0 (all match)
Example 3:
ret = FindChar("abcd", "1234", "match") ; returns 0 (none match)
Example 4:
ret = FindChar("abcdefg", "aeiou", "match", 2); returns 5 ("e")
Example 5:
ret = FindChar("123456", 123) ; returns 4

FindStr( )
String Manipulation
Returns the position of one string within another.

Syntax
Ret = FindStr(target, searchstring, startpos)

Variants
ret = FindStr(target, searchstring)

See Also
FindChar( ), IgnoreCase( ), InStr( ), RfindStr( )

Operation
Searches target for the existence of searchstring and, if found, returns its position. If target contains more than one instance of searchstring, the position of the first instance is returned. If target does not contain searchstring, 0 is returned.
The parameters are as follows:
target The string to search. If this parameter is numeric, it is automatically converted to a string.
searchstring The value to search for. If this parameter is numeric, it is automatically converted to a string.
startpos Optional starting point within the target string. If omitted, the whole string is searched.

Examples
Example 1:
target = "the quick brown fox" ; target string
searchstring = "r" ; value to search for
ret = FindStr(target, searchstring)
MsgBox("Return is:", ret) ; result is 12
Example 2:
target = "the quick brown fox" ; target string
searchstring = "r" ; value to search for

ret = FindStr(target, searchstring, 13)
MsgBox("Return is:", ret) ; result is 0
Example 3:
target = "the quick brown fox" ; target string
ret = FindStr(target, "brown")
MsgBox("Return is:", ret) ; result is 11
Example 4:
target = "ALABAMA" ; target string
ret = FindStr(target, "A", 4)
MsgBox("Return is:", ret) ; result is 5

Fix( )

Number Manipulation
Removes the fractional part of a number.
Syntax
ret = Fix(value)
See Also
Clng( ), Int( )
Operation
This function removes the fractional part of value. No rounding takes place.
Examples
ret = Fix(10.12) ; returns 10
ret = Fix(10.95) ; returns 10
ret = Fix(-10.50) ; returns -10

Focus( )

System Information
Determines the application that has focus.
Syntax
ret = Focus("application name")
Variants
ret = Focus( )
See Also
IsRunning( )
Operation
This function has two uses. When used with the application name parameter, it returns a numeric value 1 if the application has focus, and it returns 0 if not.
When used without the "application name" parameter, it returns the name and extension of the program in focus. The string is always in uppercase.
Examples
Example 1:
ret = Focus("notepad.exe") ; returns 1 or 0, depending on focus
Example 2:
ret = Focus( )
; returns the name of the application in focus
Example 3:
while Focus( ) <> "NOTEPAD.EXE" ; while Notepad is not in
Pause 1 ; focus — pause one second
Wend ; and check again

FocusName( )

Window Information
Returns the name of the currently active window.
Syntax
ret = FocusName( )

See Also
ActiveWindow( ), TopWindow( ), MouseWindow( ), IsWindow( ), WinGetPos( ), FocusWindow( ), ActiveName( ), AttachName( )
Operation
This function returns the attach name of the parent or child window which currently has focus.
Examples
```
; check that the correct document window has focus
Attach "~N~WINWORD.EXE~OpusApp~Microsoft Word"
While result = 0 ; set up a loop
ret = FocusName( ) ; get focus window name
UpperCase( ret ) ; convert to uppercase
result = FindStr( ret, "ACTNAME.DOC" ) ; check for known text
If result = 0 ; if not there jump
Type "{Control {F6}}" ; to another window
Endif
Wend
```

FocusWindow( )

Window Information
Returns the handle of the window in focus.
Syntax
ret = FocusWindow( )
See Also
ActiveWindow( ), ActiveName( ), AttachName( ), FocusName( ), IsWindow( ), MouseWindow( ), TopWindow( ), WinGetPos( )
Operation
This function returns the handle of the window currently in focus.
Examples
```
a = 1 ; set up a counter
ret = FocusWindow( ) ; get active window handle
while a = 1 ; an eternal loop
if ret <> FocusWindow( ) ; if window changes focus
< instructions > ; carry out these instructions
endif
wend
```

For…Next

Program Flow
Repeats a series of instructions a number of times.
Syntax
For <variable> = <startvalue> to <endvalue> [step <stepvalue>]
<Instruction>
Next
See Also
Do…Loop While, Repeat…Until, While…Wend
Operation
This command executes the <Instructions> between the For … Next statements repeatedly while <variable> falls within the range <startvalue> to <endvalue>.
With each iteration, the value of <variable> is increased by the value of <stepvalue>.
The parameters are:
<variable> The counter used for the loop. This variable must not be the name of an array variable or an array element.
<startvalue> An expression used to initialize the counter. This expression is always converted to a numeric expression.

<endvalue> The loop expression. The loop continues to execute if the counter is <= <endvalue> and the <stepvalue> is positive (or zero) or if the counter is >= <endvalue> and the <stepvalue> is negative.
<stepvalue> Determines the amount to add or subtract from the counter on each iteration. If this value is not specified, then <stepvalue> = 1.
On exiting the loop, execution of the script continues on the statement following the Next.

Examples

```
For I = 1 to 5
Print I ; prints 1, 2, 3, 4, 5
Next
For I = 1 to 10 Step 2
Print I ; prints 1, 3, 5, 7, 9
Next
For I = 10 to 3 Step -1
Print I ; prints 10, 9, 8, 7, 6, 5, 4, 3
Next
```

FormatDate( )

Date/Time
Formats a date and time into a string.

Syntax

ret = FormatDate("FormatString", DateVal)

Variants

ret = FormatDate("FormatString")

See Also

DateVal( ), TimeVal( ), CurTime( )

Operation

This function formats a date and time into a string of the specified format. If no date/time value is given, the current system date and time is used.
The DateVal parameter specifies the date and time. This value can be derived from the DateVal( ), TimeVal( ) or CurTime( ) functions.
The parameters for "FormatString" are as follows:
LongDate Displays a long date using the format specified in the “Regional Settings” section of “Control Panel”.
ShortDate Displays a short date using the format specified in the “Regional Settings” section of “Control Panel”.
Time Displays the time using the format specified in the “Regional Settings” section of “Control Panel”.
d The day of the month without a leading zero (1-31).
dd The day of the month as a two digit number (01-31).
ddd The abbreviated name of the day of the week — “Sun”, “Mon”, etc.
dddd The full name of the day of the week.
w The day of the week as a number (1-7), where Sunday = 1, Monday = 2, etc.
m The month number without a leading zero (1-12).
mm The month as a two digit number (01-12).
mmm The abbreviated name of the month — “Jan”, Feb”, etc.
mmmm The full name of the month.
y The day of the year as a number (1-365).
yy The year as a two digit number (00-99).
yyyy The year as a four digit number (1900-).
h The hour without a leading zero (0-23).
hh The hour as a two digit number (00-23).
n The minute with no leading zero (0-59).
nn The minute as a two digit number (00-59).

m After "h" or "hh", the minute with no leading zero.
mm After "h" or "hh", the minute as a two digit number.
s The seconds without a leading zero (0-59).
ss The seconds as a two digit number (00-59).
AMPM Display “AM” if the time is before noon or “PM” if the time is after noon.
The following table shows symbols that have special meaning. All other symbols, including spaces, are displayed without any processing.
: Time separator. The actual character used depends on the value specified in the “Regional Settings” section of “Control Panel”.
/ Date separator. The actual character used depends on the value specified in the “Regional Settings” section of “Control Panel”.
"abc" A string within quotes is displayed without any processing.

Examples

formatdate("LongDate") ; "12 January 1996"
formatdate(""Time is" hh:mm:ss") ; "Time is 13:01:00"
formatdate(""Date is" mm/dd/yyyy") ; "Date is 01/12/1996"
formatdate("[hh:mm:ss] LongDate") ; "[13:01:00]
; 12 January 1996"
formatdate("ddd Shortdate",
DateVal(1996, 12, 01)) ; "Sun 01/12/96"

Function…End Function

Program Flow
Declares a user-defined function.

Syntax

Function <functionname>(argumentlist) rettype
< instructions >
End Function

Variants

Func <functionname>(argumentlist) rettype
< instructions >
End Func

See Also

Var

Operation

A function is a self-contained set of instructions that carries out a specific task and may, optionally, return a value. Functions allow commonly used code to be reused, making scripts modular, easier to read and to maintain.
A *EZ Test* script consists of a series of functions; script commands are functions with predefined meanings.
A Function…End Function allows you to define your own reusable block of code.
<functionname> Is the name of the Function; it must begin with an alpha character and can be up to 128 characters long (spaces are not permitted).
argumentlist Is a list of dummy constants, variables or expressions — separated by commas. Each item within the list has the following format:
[Ref] <VariableName>[]
Where:
Ref Indicates that the item is passed by reference; this means that the value of the item passed can be changed by the Function. If not specified, the item is passed by its value and is not changed by the Function.
<Variable Name> Is the name of the item.
rettype Defines an optional return type where:

:Var Indicates a return value.
:Var[] Indicates that the Function returns an array.
When a Function is called, a list of actual values (called arguments) is passed to replace the dummy values in the Function definition. The items in the passed argument list are checked for type compatibility with the items in the Function definition, according to the rules detailed in Table 4-1.
The Var command can be used inside the Function definition to declare local variables of string, numeric, or array types. This allows the same variable names (for counters etc.) to be used in different Functions.
To return a value from a Function, use the Return command.
A Function definition can appear before or after the reference to it.
When a Function is called, program flow is passed to the first command following the Function statement. Execution continues until an End Function or Return statement is reached. Functions can call other Functions.
Table 4-1. Type Compatibility Rules

Variable Variable Yes
Variable Array N
Variable Expression Yes
Array Variable No
Array Array No
Array Expression No
Ref Variable Variable Yes
Ref Variable Array No
Ref Variable Expression No
Ref Array Variable No
Ref Array Array Yes
Ref Array Expression No

Examples

Example 1:

```
Function Main ; main body of script
y = 2 ; set up a variable
ret = double( y ) ; call function to double its value
MsgBox( y, ret ) ; display results — y is not changed
End Function
Function double( x ) :var ; a function to double a number
x = x * 2 ; double the passed value
return x ; return the result
End Function
```

Example 2:

```
Function Main ; main body of script
y = 2 ; set up a variable
ret = double( y ) ; call function to double its value
MsgBox( y, ret ) ; display results — y is updated
End Function
Function double( ref x ) :var ; double a number passed by reference
x = x * 2 ; double the passed value
return x ; return the result
End Function
```

Example 3:

```
Function Main
ReadLine( "pw.dat", password ); read encrypted password file
decrypt( password ) ; call function to decrypt password
Attach "Logon Screen" ; attach to target application
Type decoded ; enter decrypted password
< further instructions > ; continue script
End Function
Function decrypt( password ) :var ; function to decrypt a password
len = Length( password ) ; calculate length of password
```

```
c = 1 ; initialize counter
decoded = "" ; and result string
Repeat ; repeat
nextchr = Mid( password, ; get next character
c, 1 )
ansi = Asc( nextchr ) ; calculate ANSI value
ansi = ansi — c ; top secret decoding algorithm
new = Chr( ansi ) ; convert back to a character
decoded = decoded + new ; add to result string
c = c + 1 ; increment counter
Until c > len ; until the end
Return decoded ; return decoded string
End Function
```

Example 4:

```
Function Main
Startup( ) ; Functions with no arguments
EnterData( )
AccessAboutBox( )
CloseDown( )
End Function
Function Startup( )
exec "NOTEPAD.EXE"
End Function
Function EnterData( )
Attach "~P~NOTEPAD.EXE~Edit~Untitled — Notepad"
Type "This has been entered by ENTERDATA( ){Return}"
End Function
Function AccessAboutBox( )
Attach "~N~NOTEPAD.EXE~Notepad~Untitled — Notepad"
MenuSelect "Help~About Notepad"
Attach "~N~NOTEPAD.EXE~#32770~About Notepad"
Button "OK" 'Left SingleClick'
End Function
Function CloseDown( )
Attach "~N~NOTEPAD.EXE~Notepad~Untitled — Notepad"
MenuSelect "File~Exit"
Attach "~N~NOTEPAD.EXE~#32770~Notepad"
Button "&No" 'Left SingleClick'
End Function
```

Example 5:

```
Var number[] ; declare global array
Function main
get_numbers( ) ; generate random numbers
show_array( ) ; show them
sort_numbers( ) ; sort them
show_array( ) ; show the result
End Function
Function get_numbers( ) ; function to generate numbers
c = 1 ; initialize counter
While c < 11 ; loop
number[c] = Random( ) ; read number into array
c = c + 1 ; increment counter
Endwhile ; end of loop
End Function
Function show_array( ) ; show contents of array
c = 1 ; initialize counter
While c < 11 ; loop
Print number[c] ; show values in Viewport window
c = c + 1 ; increment counter
Endwhile
Print "" ; print a blank line
Print "" ; and another
```

```
End Function
Function sort_numbers( ) ; function to sort numbers
done = 0 ; exit flag
While done = 0 ; while not done
done = 1 ; set exit flag
c = 1 ; initialize counter
While c < 11 ; loop
If number[c] > number[c+1] ; if this number >
; the next
hold = number[c] ; hold it
number[c] = number[c+1] ; replace number
; with next
number[c+1] = hold ; update next with
; held value
done = 0 ; flag as not done
Endif
c = c + 1 ; increment counter
Endwhile
Endwhile
End Function
```

Get4GLInfo( )

4GL Commands
Get browser name and version number.

Syntax

Get4GLInfo(name, release, major)

Operation

This function returns the browser name and version number when you attach a window from IE or Netscape. The first parameter is required. The last two are optional

Example

```
Attach "American Systems Home ChildWindow"
MouseClick 777,68,'Left SingleClick ;click in window
Get4GLInfo( name,release,major ) ;gets browser,number,version
MessageBox( name, release) ;gets browser, number
MessageBox( name, major) ; gets broswer, version
```

GetEnv( )

System Information
Get the value of an environment setting.

Syntax

ret = GetEnv("string")

See Also

SystemInfo( )

Operation

This function returns the current value of the environment setting specified by string.

Examples

```
ret = GetEnv( "path" ) ; returns the system path value
ret = GetEnv( "comspec" ) ; returns "C:\COMMAND.COM
```

GetProperty ( )

Miscellaneous
Retrieves a property from a Java control

Syntax

ret = GetProperty ("<ControlType>", "<ControlID>",
"<Property>")

Operation

This command retrieves a Property from the control specified by ControlType and ControlID parameters. This command can be used to retrieve basic properties from a Java control such as title, x, y, etc. This command can also be used to retrieve extended properties defined internally for all supported controls or user properties defined in Active Object Recognition (for example, text in an edit field).
To identify what properties are available for a control, follow these steps:
1. Create a new Form Check
2. Select Capture Extended Properties on the General Tab of the Form Check Dialog Box.
3. Identify the control
4. From the Properties tab, select the control in the tree.
5. Click Edit
6. The extended property appears in the Properties tab.
The parameters for this function are as follows:
<ControlType> The type of control. These include:
CheckBox
ComboBox
Edit
Grid
LabelCtrl
ListBox
PushButton
Radio
ScrollBar
Static
TabDialog
TreeView
UpDown
<ControlID> This is the control label, index, or internal name.
See “Control Labels” on page 4-55.
<Property> The property to retrieve from the control.

Standard Internal Properties

The following default properties can be retrieved for all supported types:
title Title/label of the control
class Class name of the control
module Module name of the control
object 4GL/internal name of the control
x X position of the control in relation to its parent
y Y position of the control in relation to its parent
height Height of the control in pixels
width Width of the control in pixels
enabled Enabled status of the control
The function returns the value of the property or generates an "Unable to perform command on this control" runtime error if a property can not be retrieved for a control.
Use the AOR utility to define custom properties.

Examples

```
Attach "SwingSet Mainwindow"
; retrieve the text from the Username edit field
ret = GetProperty ("Edit", Username", "text")
; retrieve the title/label from the first push button
ret = GetProperty ("PushButton", "!~1","title")
```

GetReadyState

Window Information
Returns the ready state of the browser window.

Syntax
GetReadyState("windowname")
Operation
Returns the ready state of the browser window from Microsoft Internet Explorer. The first parameter is required.
The returned state is numeric.
The ready states are as follows:
0 FAIL Window is not a browser window.
1 OK Browser is not busy.
2 BUSY Browser window is busy.
3 INTERACTIVE Browser State is interactive.
Examples
```
Attach "Program Manager PopupWindow"
ListViewCtrl "~1", "Internet Explorer",'Left SingleClick'
ListViewCtrl "~1", "Internet Explorer",'Left DoubleClick'
Attach "C:\testinggetreadystate.html-Microsoft Internet
Explorer ChildWindow~1"
AnchorSelect "Testing GetReadyState", 'Left SingleClick'
Attach "Enter Network Password PopupWindow"
Button "OK", 'Left SingleClick'
ret = GetReadyState( "CNN.com-Microsoft Internet Explorer
MainWindow" )
MsgBox( "Returning state of window", ret ) ; returns the
state of the browser window
```

Goto

Program Flow
Causes program execution to jump to a specified label.
Syntax
Goto <LabelName>
Operation
This command causes the program to jump to a location specified by LabelName. LabelName defines a location for a Goto command to jump to. LabelName is declared by specifying the name of the label followed by a colon. The label declaration may appear before or after the Goto command.
The Goto command causes the program execution to jump to the specified label declaration and continue execution from the line *following* the label.
Examples
```
; The following example executes "old_app" or "new_app" depending
; on the host operating system
Function Main
If WinVersion = 351 ; if Windows/NT Version 3.51
Goto nt_351
Else ; if it is not Windows/NT 3.52
GoTo win95
Endif
nt_351: ; declaration of label
Exec "old_app.exe"
Goto end_func ; after executing old_app, goto end_func
win95: ; declaration of label
Exec "new_app.exe"
end_func: ; declaration of label
End Function
```

HeaderCtrl( )

Dialog Control
Selects a column header control.
Syntax

Ret = HeaderCtrl("ControlId", "Item", "Options" [, x, y])

Operation

This function drives the column headings in a list view window (such as that found in Explorer). The parameters are as follows:
"ControlId" The index value of the header control.
"Item" The column to select. This value may be literal or variable, text or position. To select the first column use "@1" in place of a text value for "Item".
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"double" Double click the mouse button.
"click" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Used in conjunction with "control" and "shift".
x , y These optional parameters specify where on the control the mouse button will be clicked. If omitted, the button is clicked in the center.
The function returns 1 if the selection is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, the parentheses are required.

Examples

Example 1:

```
; sort files displayed in Explorer by size
Attach "~N~EXPLORER.EXE~ExploreWClass~Exploring - (C:)"
HeaderCtrl "~1", "Size", "Left SingleClick"
```

Example 2:

```
; sort by the second column
Attach "~N~EXPLORER.EXE~ExploreWClass~Exploring - (C:)"
HeaderCtrl "~1", "@2", "Left SingleClick"
```

Hotkey

Dialog Control
Simulates the pressing of a shortcut key.

Syntax

Hotkey id

See Also

Type( )

Operation

This command simulates the pressing of an application defined shortcut key (hotkey). Each hotkey is assigned a numeric ID by Windows. Use of a hotkey is learned as a HotKey( ) function, rather than as a normal keystroke.
This command has no return value.

Examples

```
; A hotkey (shortcut key) can be assigned to an icon on the
; desktop through the Properties dialog. Once defined, the
; hotkey can be used to launch the application
; associated with the icon.
; In this example, the {F12} key is defined as the hotkey
; to launch Version 2 of the EZ TESTDEMO.EXE program.
; When used, the following script is generated:
Attach "~U~EXPLORER.EXE~Shell_TrayWnd~"; attach to the desktop
```

HotKey 1 ; use hotkey number 1
Attach "~N~EZ TESTDEMO.EXE~Afx~EZ TESTDemo" ; attach to the
Size 700, 500 ; application started
Move 162, 120

HotspotCtrl( )

4GL Commands
Presses a UNIFACE hotspot control.

Syntax
ret = HotspotCtrl("ControlId", "Options", x, y)

Variant
HotspotCtrl("ControlId", "Options", x, y)
HotspotCtrl "ControlId", "Options", x, y

See Also
Button( ), LabelCtrl( ), PictureCtrl( )

Operation
This command processes a UNIFACE hotspot in the attached window. The control can have Windows Class of either UHotspot or UniHotspot. The action to be performed is specified with the Options parameter. The parameters are as follows:
"ControlId" Specifies the hotspot logical Object Name.
"Options" The Options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press-and-hold the mouse button down.
"up" Release the mouse button.
"singleclick" Single-click the mouse button.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with "control" and "shift".
x,y Specifies where on the control the mouse is clicked.
When the command is generated by Learn, the parentheses are omitted.

Examples
; If you click Application Model Manager in UNIFACE 6.1,
; the following is learned:
HotspotCtrl "@DUMMY.DUMMY.STANDARD Hotspot", 'Left SingleClick', 182, 115

Hours( )

Date/Time
Returns the specified hour.

Syntax
ret = Hours(timeval)

Variants
ret = Hours( )

See Also
TimeVal( ), CurTime( )

Operation
This function returns the hour of the day specified by the timeval parameter. The time value, timeval, can be derived from the TimeVal( ) or CurTime( ) functions. If the timeval parameter is not specified, the current system time is used.

Examples

Example 1:
n = TimeVal(15, 11, 30) ; returns 54690
Hour_of_Day = Hours(n) ; returns 15
Example 2:
Hour_of_Day = Hours( ) ; current hour of the day

If…Else…Endif

Program Flow
Allows the script to perform runtime decisions.

Syntax

If <Expression> [THEN]
<If_Instructions>
Endif

Variants

If <Expression> [THEN]
<If_Instructions>
Else
<Else_Instructions>
Endif
If <Expression> [THEN]
<If_Instructions>
Elseif <Expression> [THEN]
<Elseif_Instructions>
Else
<Else_Instructions>
Endif

Operation

When an If statement is executed, the script checks to see whether the <Expression> is true or false. The <Expression> can be an arithmetic or Boolean expression.
If the <Expression> is true, the <Instructions> following the If statement are executed up to the next Else or Endif statement.
If the <Expression> is false, the <Instructions> after the Else statement are executed. If there is no Else statement, execution of the script continues after the closing Endif statement.
The word THEN is present only for readability and can be omitted.

Examples

```
; run the backup software only if the day is Monday,
; Wednesday, or Friday
ret = WeekDay( ) ; get day of week
if ret = 1 ; if it's Monday
Exec( "backup" ) ; run the backup software
elseif ret = 3 ; if it's Wednesday
Exec( "backup" ) ; run the backup software
elseif ret = 5 ; if it's Friday
Exec( "backup" ) ; run the backup software
else ; otherwise display
following message
MessageBox( "Network","System will be backed up tomorrow" )
endif
```

IgnoreCase( )

String Manipulation
Sets case sensitivity for string comparisons and searches.

Syntax

ret = IgnoreCase(value)

See Also

Boolean Expressions, Compare( ), FindStr( )

Operation
Sets the ignore case flag for Boolean string comparisons (= = , > = , < = , etc.) and for the Compare( ) and FindStr( ) functions.
Value can be:
1 String comparisons are case insensitive (case is ignored).
0 String comparisons are case sensitive (case is not ignored).
The previous value of the flag is returned.
By default, the ignore case flag is set to 1 and this is the initial setting for every script. Child scripts do not inherit their parent script's ignore case setting.

Examples
Example 1:
```
oldval = IgnoreCase( 1 ) ; ignore case
MsgBox( "Previous ignore case setting ", oldval )
a = "Hello"
b = "hELLO"
if a = b
msgbox( "Ignore Case", "Strings are equivalent" )
else
msgbox( "Ignore Case", "Strings are not equivalent" )
endif
```
Example 2:
Parent Script
```
public a, b ; declare variables public
IgnoreCase( 0 ) ; do not ignore case in parent script
a = "Hello"
b = "hELLO"
if a = b
msgbox( "Parent Script", "Case Insensitive" )
else
msgbox( "Parent Script", "Case sensitive" )
endif
Run( "child" ) ; run child script - sets ignore case = 1
if a = b
msgbox( "Return to Parent", "Case Insensitive" )
else
msgbox( "Return to Parent", "Case sensitive" )
endif
```
Child Script
```
if a = b
msgbox( "Child Script", "Case Insensitive" )
else
msgbox( "Child Script", "Case Sensitive" )
endif
```
Example 3:
```
; direct assignment
IgnoreCase = 0 ; do not ignore case
```

ImageSelect( )
Dialog Control
Selects Web objects that are created using the "IMG" HTML tag.

Syntax
```
ret = ImageSelect( "ControlId", "Options", x, y )
```
Variants
```
ImageSelect( "ControlId", "Options" )
```
See Also
AnchorSelect( )

Operation
This function processes an HTML image in the currently attached dialog box. The action

is specified in "Options". The function parameters are as follows:
"ControlId" Specifies the text on the image label, such as “American Systems Corporation” (the text that appears in the drop-down box). If an image label is not available, the object name is used.
"Options" The options are as follows:
"left" Use the left mouse button to select the image.
"right" Use the right mouse button to select the image.
"middle" Use the middle mouse button to select the image.
Note
The ControlId parameter is the control’s text identification (i.e., the image’s label), not an actual number.
"down" Press the mouse button down to select the image.
"up" Release the mouse button to select the image.
"doubleclick" Double-click the button to select the image.
"singleclick" Click the button once to select the image.
"control" Press the control key before clicking the button.
"shift" Press the shift key before clicking the button.
"with" Use in conjunction with "control" and "shift".
x , y These optional parameters specify where on the image the mouse button will be clicked. If omitted, the image is clicked in the center. This option is useful if a single image contains multiple jumps.
The function returns 1 if the image is successfully selected, and it returns 0 if the image is not successfully selected.
When this command is generated by the Learn facility, the parentheses are omitted.

Examples

```
Function Main
Attach "Program Manager PopupWindow"
ListViewCtrl "~1", "Internet Explorer", 'Left SingleClick'
Attach "http://compuweb.American Systems.com/ - Microsoft Internet
Explorer MainWindow"
Attach "~P~IEXPLORE.EXE~Edit~Compuweb Home - Microsoft Internet
Explorer"
EditClick "~0", 'Left SingleClick', 218, 7
Attach "~P~IEXPLORE.EXE~ComboBox~Compuweb Home - Microsoft
Internet Explorer"
ComboText "~0", "www.American Systems.com"
Attach "~P~IEXPLORE.EXE~Edit~Compuweb Home - Microsoft Internet
Explorer"
TypeToControl "Edit", "~0", "{Return}"
Attach "American Systems Corporation Home Page - Microsoft Internet
Explorer ChildWindow~1"
ImageSelect "American Systems Alliances", 'Left SingleClick'
Attach "American Systems Alliances - Microsoft Internet Explorer Child-
Window~1"
AnchorSelect "index.htm~6", 'Left SingleClick'
End Function
```

Include

Miscellaneous
Adds another script file to the script during compilation.

Syntax

```
Include "scriptname"
```

See Also
Run( )
Operation
This command instructs the compiler to include the contents of the named script when compiling the current script. The file to be included is usually another *EZ Test* script containing information common to many scripts.
The "scriptname" parameter is the name of a script from the *EZ Test* database.
Examples
; include EZ Test script
Include "addrec"
<INSTRUCTIONS>

InsertStr( )

String Manipulation
Inserts a string into a target string.
Syntax
ret = InsertStr(target, "new", start, length)
Variants
InsertStr(target, "new", start)
See Also
Left( ), Right( ), Mid( )
Operation
Inserts a string into another string. The parameters are as follows:
target The string in which to make the insertion.
"new" The string to insert. This can be literal or the contents of a variable.
start The position in target to start the insertion.
length The number of characters from the new string to insert into the target. If omitted, the whole string is inserted.
If the string is 11characters long, the point of insertion must be between 1 and 11. You can not insert outside of this range. The function returns 1 if the insert is successful, and it returns 0 if it is not.
Examples
Example 1:
target = "Hello World" ; set up target variable
new = "There " ; text to insert
ret = InsertStr(target, new, 6) ; insert whole variable
; starting at position 6
; - target becomes
; "Hello There World"
Example 2:
target = "Hello World" ; set up target variable
new = " There" ; text to insert
ret = InsertStr(target, new, 6, 4)
; insert 4 characters from variable,
; starting insertion at position 6 -
; result is "Hello There World"

InStr( )

String Manipulation
Returns the position of one string within another.
Syntax
ret = InStr(target, searchstring)
See Also
FindStr( )
Operation

Searches target for the existence of searchstring and, if found, returns its position. If target contains more than one instance of searchstring, the function returns the position of the first instance. If target does not contain searchstring, the function returns -1.

Examples

target = "the quick brown fox"
ret = InStr(target, "fox") ; Result is 16
ret = InStr(target, "green") ; Result is -1

Int( )

Number Manipulation
Returns the integer part of a number.

Syntax

ret = Int(value)

See Also

Clng( ), Fix( )

Operation

This function returns the integer part of value, rounded down.

Examples

ret = Int(10.12) ; returns 10
ret = Int(10.95) ; returns 10
ret = Int(-10.50) ; returns -11

Note

InStr( ) is implemented to preserve compatibility with previous versions of *EZ Test*. In this function, counting of characters starts from 0. It is superseded by the FindStr( ) function, in which characters are counted from 1.

IPControl( )

Dialog Control
Sets the IPAddress value on a Windows IPAddress control.

Syntax

ret = IPControl ("ControlID", IPVal1, IPVal2, IPVal3, IPVal4)

Operation

This command sets the value of the Windows IPAddress control specified in the ControlId parameter. The IPVal1, IPVal2, IPVal3, and IPVal4 are required to specify the value of the control. The parameters are as follows:
"ControlId" The index value of the IPControl.
IPVal1 The first value of the IPAddress control. The value should be between (0–255).
IPVal2 The second value of the IPAddress control. The value should be between (0– 255).
IPVal3 The third value of the IPAddress control. The value should be between (0– 255).
IPVal4 The fourth value of the IPAddress control. The value should be between (0– 255).
ret A value of 1 is returned if the operation is successful. A value of 0 is returned for failure.

Examples

```
Function Main
Attach "Microsoft Control Spy - IP Address PopupWindow"
IPControl "~1”, 12, 22, 33, 12
IPControl "~1", 1, 252, 1, 1
IPControl "~1", 123, 123, 112, 124
IPControl "~1", 22, 22, 21, 27
IPControl "~1", 99, 100, 1, 4
End Function ; Main
```

Note

IPVal1, IPVal2, IPVal3, and IPVal4 typically represent a portion of the complete TCP/IP address. For example, a TCP/IP address of 172.222.22.23 would be indicated with the following: IPVal1 = 172, IPVal2 = 222, IPVal3 = 22, and IPVal4 = 23.

IsFile( )

File Access
Checks for file existence and attributes.

Syntax

ret = IsFile("filename", options)

See Also

FilePos( ), Open( ), Read( ), ReadIni( ), ReadLine( )

Operation

This function enables you to check for the existence and attributes of filename.
The options are as follows:
none Test for filename's existence.
"f" Test for filename's existence.
"r" Is filename read-only.
"h" Is filename a hidden file.
"s" Is filename a system file.
"d" Is filename a directory.
The function returns 1 if all the options are true, and it returns 0 if any test fails.

Examples

Example 1:

```
If IsFile( "oldfile.dat" ) ; if this file exists
Delete ( "oldfile.dat" ) ; delete it
Endif
```

Example 2:

```
ret = IsFile( "c:\io.sys" ) ; returns 1 (bootstrap
; file)
ret = IsFile( "c:\io.sys", "r" ) ; returns 1 (read only)
ret = IsFile( "c:\io.sys", "h" ) ; returns 1 (hidden file)
ret = IsFile( "c:\io.sys", "s" ) ; returns 1 (system file)
ret = IsFile( "c:\io.sys", "rhs" ) ; returns 1 (read-only,
; system hidden)
ret = IsFile( "c:\io.sys", "d" ) ; returns 0 (not a directory)
ret = IsFile( "c:\", "d" ) ; returns 1 (root directory)
ret = IsFile( "c:\", "f" ) ; returns 0 (not a file)
```

IsMenu( )

Menu Information
Returns the state of a specific menu item.

Syntax

ret = IsMenu("AttachName" , "MenuName" , "MenuItem" , ["Options"])

See Also

MenuCount( ), MenuItem( ), MenuFindItem( )

Operation

This function determines if the menu item specified by AttachName, MenuName, and MenuItem is in the state specified by options.
"AttachName" The object map name or raw attach name of the window that owns the top-level menu.
"MenuName" The path from the top-level menu to the sub-menu that contains the menu item. Mnemonic and accelerator key notation should not be included in the MenuName parameter (for example, "&File~&Exit" should be stated

as “File~Exit”).
The MenuName parameter can be preceded by the following prefixes:
Normal# Indicates the menu is a normal menu (default).
Popup# Indicates the menu is a popup menu. If the menu is a pop-up, the script must contain code to create the menu (i.e., a right mouseclick).
System# Indicates the menu is a system menu.
For example, to interrogate the popup menu on the *EZ Test* grid in the Event Map, use the following value for MenuName:
"Popup#Create"
The “#” sign is used to distinguish the prefix from the actual menu name. This prevents *EZ Test* from interpreting the prefix as part of the actual menu name.
"MenuItem" The actual menu item whose state is being returned. The MenuItem parameter can reference the menu’s text, the numeric control ID, or the menu item’s ordinal position. The “#” sign denotes ordinal position, for example: “#15”.
["Options"] The options are as follows:
enabled Checks to see if the menu item enabled.
exists Checks to see if the menu item exist.
checked Checks to see if the menu item is checked.
grayed Checks to see if the menu item grayed.
separator Checks to see if the menu item is a separator.
cascading Checks to see if the menu item is a sub or popup menu?
default Checks to see if the menu item is a default menu item. A sub menu can contain one default menu item. If a default exists, and the user selects the menu, the default is automatically selected.
This command can not be used to attach to toolbar menus that create pop-up menus (i.e., pop-up menus that are created using Internet Explorer’s toolbar buttons).
The function always returns 0.

Examples

Example1:

```
; Get the enabled status of EZ Test's Script Editor Menu item
; Insert~Check~Bitmap...
Ret = IsMenu ( "EZ Test" , "Insert~Check" , "Bitmap..." , "enabled" )
```

Example 2:

```
; Get the checked status of EZ Test's Script Editor Menu item
; On the Insert menu, with ID = 10, on the Normal menu
Ret = IsMenu ( "EZ Test" , "Insert" , "10" , "checked" )
```

Example 3:

```
; Is the menu item at ordinal position 10 a separator?
Ret = IsMenu( "EZ Test" , "Edit", "#10" , "separator" )
```

Example 4:

```
; Uses the desktop window (Explorer) as an example of calling
; up the menu and checking a popup menu.
; Remember popup menus do not exist until they are visible
Attach "desktop"
MouseClick 1, 1, 'Right SingleClick'
ret = IsMenu( "desktop" , "Popup#" , "Paste" , 'enabled' )
if ret = 1
logcomment "Paste is enabled"
```

```
else
logcomment "Paste is disabled"
endif
```

IsRunning( )

System Information
Determines if the specified application is running.

Syntax

```
ret = IsRunning( "app.exe" )
```

See Also

Focus( )

Operation

This function determines whether the specified application is running. It returns 1 if it is, and it returns 0 if it is not.

Examples

```
ret = IsRunning( "notepad.exe" )
if ret = 0 ; if not running
Exec "notepad.exe ; run it
endif
```

IsWindow( )

Window Information
Determines if a window is in a specified state.

Syntax

```
ret = IsWindow( "Attachname", "options" )
```

See Also

ActiveWindow( ), ActiveName( ), TopWindow( ), MouseWindow( ),WinGetPos( ), FocusWindow( ), FocusName( ), AttachName( )

Operation

This function determines if the window specified by Attachname is in one of the following states as specified by certain options:
"maximized" Is the window maximized.
"Minimized" Is the window minimized.
"Restored" Is the window in its restored state (neither maximized or minimized).
The options are:
"Iconized" Is the window minimized.
"Active" Is the window currently active.
"Focus" Does the window have focus.
"exists" Does the window exist.
"Enable" Is the window ready for input.
"Disable" Is the window unable to accept input.
"Hidden" Is the window hidden.
"Visible" Is the window visible.
"Checked" Is the window checked (for check boxes only).
Each option can be abbreviated to the single character shown in uppercase in the left column of the above list.
The function returns 1 if the window has all the specified options, and it returns 0 if it does not. Multiple options can be used.

Examples

```
; see if an application is running, execute it if not
ret = IsWindow( "~P~NOTEPAD.EXE~Edit~Untitled - Notepad", "exists" )
If ret = 0
Exec "Notepad.exe"
Endif
```

JulianDate( )

Date/Time
Returns the number of seconds since 12:00 a.m. December 31, 1899.

Syntax

ret = JulianDate(JulianDateVal, [yyyy])

See Also

Date( ), DateVal( ), JulianDateVal( )

Operation

The JulianDate( ) function returns the number of seconds elapsed from 12:00 a.m. December 30, 1899 to 12:00 a.m. on the date entered in the function. The parameters are as follows:
JulianDateVal Specifies a value from 1 - 366 that represents the number of days since the beginning of the year.
yyyy An optional value that specifies the year to use (1899 onward). If the yyyy option is not provided, the current year setting on the PC is used.
If an invalid value is supplied, the JulianDate( ) function returns a value of -1.

Examples

Function Main
ret = JulianDate(292) ; JulianDate(JulianDateVal, [yyyy])
MessageBox "292 - 1998", ret ; value returned is 3117916800
ret = JulianDate(1, 1900)
MessageBox "1 - 1900", ret ; value returned is 172800
Note
IsWindow does not work with rendered controls.
End Function Main

JulianDateVal( )

Date/Time
Returns the number of days since the beginning of the year (1 - 366).

Syntax

ret = JulianDateVal(yyyy, mm, dd)

See Also

Date( ), DateVal( ), JulianDate( )

Operation

The JulianDateVal( ) function returns the day of the year in Julian date format (1 - 366). The parameters are as follows:
yyyy Specifies the year to use (1899 onward).
mm Specifies the month to use (01 - 12).
dd Specifies the day to use (01 - 31).
If an invalid value is supplied, the JulianDateVal( ) function returns a value of -1.

Examples

Function Main
ret = JulianDateVal(1998, 10, 19); JulianDateVal (YYYY, MM, DD)
MessageBox "1998/10/19", ret ; value returned here is 292
ret = JulianDateVal(2000, 12, 31)
MessageBox "2000/12/31", ret ; value returned here is 366
; (29 days in February)
End Function Main

LabelCtrl( )

4GL Commands
Presses a UNIFACE or ND-DK label control.

Syntax

ret = LabelCtrl("ControlId", "Options")

Variant
LabelCtrl("ControlId", "Options")
LabelCtrl "ControlId", "Options"
See Also
Button( ), HotspotCtrl( ), PictureCtrl( )
Operation
This command processes a UNIFACE label control in the attached window. The control has a Windows Class of UniLabel. The action to be performed is specified with "Options". The parameters are as follows:
"ControlId" Specifies the label logical Object Name.
"Options" The Options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse down.
"up" Release the mouse button.
"singleclick" Single-click the mouse button.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with "control" and "shift".
When the command is generated by Learn, the parentheses are omitted.
Examples
; When a label is single clicked, the following is learned:
LabelCtrl "Entity Name", 'Left SingleClick'

LastKey( )

System Information
Returns the virtual keycode of the last key pressed.
Syntax
ret = LastKey( )
See Also
LastKeyStr( )
Operation
This function returns the virtual keycode of the last key pressed. The function takes the value of the unshifted key. For example, the dollar symbol "$" which, on a US keyboard, is produced by shifting the number 4 key, returns the same result as pressing the number 4 key on the QWERTY pad.
Extended keys (the grey keys to the right of the QWERTY pad) return a keycode greater than 256.
Examples
; process instructions until the user presses "Escape"
While LastKey <> 27
<Process Instructions>
Wend

LastKeyStr( )

System Information
Returns the keytop string of the last key pressed.
Syntax
ret = LastKeyStr( )
See Also
LastKey( )
Operation
This function returns the virtual keytop of the last key pressed. The function takes the

value of the unshifted key. For example, the dollar symbol “$” which, on a US keyboard, is the shifted number 4 key, returns the same result as pressing the number 4 key on the QWERTY pad.

Examples

```
; process instructions until the user presses "F1"
While LastKeyStr( ) <> "{F1}"
<Process Instructions>
Wend
```

Left( )

String Manipulation

Extracts the left-most characters from a string variable.

Syntax

ret = Left(source, count)

See Also

Length( ), Mid( ), Right( )

Operation

This function returns the first count characters contained in source. The parameters are:

source The string containing the required information.

count The number of characters required.

Examples

Example 1:

source = "This is a value" ; set up the variable

ret = Left(source, 7) ; take the first 7 characters

; result is "This is"

Example 2:

ret = Left("Hello World", 5) ; result is "Hello"

Example 3:

ret = Left("Hello World", 20) ; result is "Hello World"

Length( )

String Manipulation

Returns the length of a string.

Syntax

ret = Length(str1 [, str2, ...])

See Also

Left( ), Mid( ), Right( ), FindStr( )

Operation

This function returns the total length of the specified strings.

Examples

len = Length("hello world") ; returns 11

len = Length("hello", "world") ; returns 10

a = "The quick brown fox "

b = "jumps over "

len = Length(a, b, "the lazy dog"); returns 43

LinkCheck( )

Checks

Reports on the existence of a link.

Syntax

ret = LinkCheck("Name" , "URL" , ["InternetProfile"])

Operation

This function attempts to access the link specified as URL using the proxy settings identified using the InternetProfile parameter. *EZ Test* attempts to access the site reports on its existence. The parameters are as follows:

"Name" The name assigned to the check and reported in
the log.
"URL" The Internet address for the site to be checked.
Only the link specified as URL will be verified.
Any additional secondary links located on the
first-level URL will not be verified.
["InternetProfile"] The internet profile that will be used to attempt a
connection to the URL. If an InternetProfile
is not specified, the profile identified in *EZ Test*'s
run environment settings is used. If there is no
profile identified in the run environment settings,
the Default profile is used.
The function returns a value of 1 if the link exists and a value of 0 if the link does not exist, or if *EZ Test* was unable to access the link using the proxy settings.

Examples

```
Function Main
; Check the American Systems web site and report pass or fail in log
; under the check name WebSite.
Ret = LinkCheck("WebSite" , "http://www.American Systems.com")
If ret = 1
MessageBox( "pass" , "The Link Exists" , 'ok' )
ElseIf ret = 0
MessageBox( "fail" , "The Link Failed" , 'ok' )
EndIf
End Function
```

ListBox( )

Dialog Control
Selects a string from a listbox.

Syntax

```
ret = ListBox( "ControlId", "Item" [, "Options" ] [, x, y ] )
```

Variants

```
ListBox( "ControlId", "@ItemPosition" [, "Options" ] [, x, y ] )
```

See Also

Button( ), CheckBox( ), ComboBox( ), ComboText( ), EditText( ), RadioButton( ), ScrollBar( )

Operation

This function selects the item specified by the "Item" parameter from the listbox specified by the "ControlId" in the currently Attached dialog.
The parameters are:
"ControlId" Specifies the index value of the listbox: "~1" for the first listbox,
"~2" for the second, etc.
"Item" The item to select from the listbox. This value can be a literal or
variable, or a position in the list. To select the first item in the list,
use "@1" in place of a text value for "Item".
"Options" Determines how the item in the listbox is selected. This can be
either "SingleClick" or "DoubleClick".
x , y These optional parameters specify where on the item the mouse
button will be clicked. If omitted, the button is clicked in the
center.
The function returns 1 if the control is selected successfully, and it returns 0 if it is not.
When this command is generated by the Learn facility, the parentheses are omitted.

Examples

Example 1:

```
; select a directory from the second listbox in Notepad's
; File~Open dialog
```

```
Attach "~N~KERNEL32.DLL~#32770~File Open"
ListBox "~2", "C:\", "DoubleClick"
Attach "~N~KERNEL32.DLL~#32770~File Open"
ListBox "~2", "EZ TESTPT", "DoubleClick"
; select the file to open from the first listbox
Attach "~N~KERNEL32.DLL~#32770~File Open"
ListBox "~1", "ADDRESS.DB", "SingleClick"
Button "OK", "SingleClick"
```

Example 2:

```
; always select the first item from the first listbox
Attach "~N~KERNEL32.DLL~#32770~File Open"
ListBox "~1", "@1", "SingleClick"
Button "OK", "SingleClick"
```

ListCount( )

Window Information

Returns the number of items in a list control.

Syntax

```
ret = ListCount( hCtrl )
ret = ListCount( CtrlType, CtrlID)
```

See Also

ControlFind( ), ListItem( )

Operation

This function returns the number of items in the list control whose window handle is hCtrl, or whose CtrlType and CtrlID matches the control to which you attached.

The window handle can be obtained by using one of the ControlFind( ) group of functions.

The function can be used on any of the following list type controls:

- ComboBox
- Header
- ListBox
- ListView
- TabDialog
- ToolBar

A value of 0 is returned if the control does not support a list of items (an edit control or button, for example) or if the window handle specified is invalid.

Examples

Example 1:

```
Attach "Open PopupWindow" ; attach to File Open dialog
hCtrl = ListViewFind( "~1" ) ; get handle of list control
nItems = ListCount( hCtrl ) ; get number of items
count = 1
While count != nItems
Text = ListItem( hCtrl, count )
print Text
count = count + 1
Wend
```

Example 2:

```
Attach "Display Properties PopupWindow"
tCtrl = TabFind( "~1" ) ; get handle of tab control
nItems = ListCount( tCtrl ) ; get number of items
count = 1
While count != nItems
Text = ListItem( tCtrl, count )
print Text
count = count + 1
Wend
```

Example 3:

```
Attach "Display Properties PopupWindow"
nItems = ListCount( "ListBox","~1" ) ; get number of items
count = 1
While count != nItems
Text = ListItem( "ListBox","~1", count )
print Text
count = count + 1
Wend
```

ListFindItem( )

Window Information
Returns the position of an item in a list control.

Syntax

```
Pos = ListFindItem( hCtrl, "Text" )
Pos = ListFindItem( CtrlType, CtrlID)
```

Variants

```
Pos = ListFindItem( hCtrl, "Text", StartPos )
```

See Also

ControlFind( ), ListItem( )

Operation

This function returns the position of the "Text" item in the list control whose window handle is hCtrl, or whose CtrlType and CtrlID matches the control to which you attached.
. The window handle can be obtained by using one of the ControlFind( ) group of functions.
The optional StartPos parameter denotes the position from which the search should commence. If not specified, searching starts from the top of the list.
The function can be used on any of the following list type controls:

- ComboBox
- Header
- ListBox
- ListView
- TabDialog
- ToolBar

A value of 0 is returned if the window handle specified is invalid or the item is not found.

Examples

Example 1:

```
Attach "Open PopupWindow" ; attach to File Open dialog
hCtrl = ListViewFind( "~1" ) ; get handle of list control
Pos = ListFindItem( hCtrl, "MyDoc.doc" ) ; search for item
If Pos != 0 ; if found
ListViewCtrl( "~1", "@" + Str( Pos ),
'Left Double' ) ; select it
Else ; otherwise
MsgBox( "Error", "Document Not Found") ; show warning
Endif
```

Example 2:

```
Attach "Open PopupWindow" ; attach to File Open dialog
Pos = ListFindItem( "ListView", "~1","MyDoc.doc" ); search for
item
If Pos != 0 ; if found
ListViewCtrl( "~1", "@" + Str( Pos ),
'Left Double' ) ; select it
Else ; otherwise
MsgBox( "Error", "Document Not Found") ; show warning
Endif
```

ListFocus( )

Window Information

Returns the position of the selected item in a list control.

Syntax

Pos = ListFocus(hCtrl)
Pos = ListFocus(CtrlType,CtrolID)

See Also

ControlFind( ), ListItem( ), ListTopIndex( )

Operation

This function returns the position of the currently selected item in the list control whose window handle is hCtrl, or whose CtrlType and CtrlID matches the control to which you attached.

. The window handle can be obtained by using one of the ControlFind( ) group of functions.

This function can be used on the following list-type controls:

- ComboBox
- Header
- ListBox
- ListView
- TabDialog
- ToolBar

A 0 is returned if no item was selected.

Examples

Example 1:

```
Attach "Open PopupWindow" ; attach to File Open dialog
hCtrl = ListViewFind( "~1" ) ; get handle of list control
Pos = ListFocus( hCtrl ) ; get current selection's position
If Pos != 0 ; if found
Item = ListItem( hCtrl, Pos ) ; get the text
Else ; otherwise
ListViewCtrl( "~1", "@1, 'Left Single' ) ; select first item
Endif
```

Example 2:

```
Attach "Open PopupWindow" ; attach to File Open dialog
hPos = ListFocus( hCtrl ) ; get current selection's position
If Pos != 0 ; if found
Item = ListItem( "ListView", "~1") ; get the text
Else ; otherwise
ListViewCtrl( "~1", "@1, 'Left Single' ) ; select first item
Endif
```

ListItem( )

Window Information

Retrieves the text from an item in a list control.

Syntax

ret = ListItem(hCtrl, Position, [column])
ret = ListItem(CtrlType, CtrlID, [column])

See Also

ControlFind( ), ListItem( )

Operation

This function returns the text of the Position item in the list control with window handle hCtrl. The window handle can be obtained by using one of the ControlFind( ) group of functions.

This function can be used on the following list-type controls:

- ComboBox
- Header

- ListBox
- ListView
- TabDialog
- ToolBar

An empty string is returned if the position specified is invalid. A runtime error is generated if the control does not support a list of items (an Edit control or Button, for example) or the window handle is invalid. By default data is retrieved from the first column of the control. Add a third parameter to specify the column from which the data will be retrieved for the control.

For combos/lists drawn by the user, the list must be first drawn in order for *EZ Test* to capture/select the text. If the list is not drawn, the command will not return text.

Examples

Example 1:

```
Attach "Open PopupWindow" ; attach to File Open dialog
hCtrl = ListViewFind( "~1" ) ; get handle of list control
nItems = ListCount( hCtrl ) ; get number of items
count = 1
While count <= nItems
```

Note

In *EZ Test* 4.9.0 and later, the ID-based variant of the ListItem command supports returning text from Swing-based Java TreeView (JTree) controls.

```
Text = ListItem( hCtrl, count, 2) ; Get data from second column
print Text
count = count + 1
Wend
```

Example 2:

```
Attach "Open PopupWindow" ; attach to File Open dialog
nItems = ListCount( "ListView", "~1"); get number of items
count = 1
While count <= nItems
Text = ListItem( "ListView", "~1", count, 2) ; Get data from
second column
print Text
count = count + 1
Wend
```

ListTopIndex( )

Window Information

Returns the position of the first visible item in a list control.

Syntax

```
Pos = ListTopIndex( hCtrl )
Pos = ListTopIndex( CtrlType, CtrlID)
```

See Also

ControlFind( ), ListFindItem( ), ListItem( )

Operation

This function returns the position of the first visible item in the list control with window handle hCtrl, or whose CtrlType and CtrlID matches the control to which you attached.

. The window handle can be obtained by using one of the ControlFind( ) group of functions.

The function can be used on the following list-type controls:

- ComboBox
- Header
- ListBox
- ListView
- TabDialog
- ToolBar

A 0 is returned if the control does not support a list item (an Edit control or Button, for example) or if the window handle specified is invalid.

Examples

Example 1:

```
Attach "Open PopupWindow" ; attach to File Open dialog
hCtrl = ListViewFind( "~1" ) ; get handle of list control
GetOut = 0 ; initialize flag
Repeat ; start repeat loop
If ListTopIndex( hCtrl ) = 31 ; if first visible
; item is No 31
GetOut = 1 ; get out of loop
Else ; otherwise
Attach "~P~NOTEPAD.EXE~SysListView32~Open" ; attach to list
NCMouseClick 396, 129, 'Left SingleClick' ; click scrollbar
Pause 200, 'ms' ; allow time to process
Endif
Until GetOut = 1 ; until found
```

Example 2:

```
Attach "Open PopupWindow" ; attach to File Open dialog
GetOut = 0 ; initialize flag
Repeat ; start repeat loop
If ListTopIndex( "ListView", "~1") = 31; if first
visible
; item is No 31
GetOut = 1 ; get out of loop
Else ; otherwise
Attach "~P~NOTEPAD.EXE~SysListView32~Open" ; attach to list
NCMouseClick 396, 129, 'Left SingleClick' ; click scrollbar
Pause 200, 'ms' ; allow time to process
Endif
Until GetOut = 1 ; until found
```

ListViewCtrl( )

Dialog Control

Drives the file list area in a dialog box.

Syntax

```
Ret = ListViewCtrl( "ControlId", "Item", "Options" [, x, y] )
```

Operation

This function drives the file list area in a dialog box. The parameters are as follows:

"ControlId" The index value of the list view control.

"Item" The file or folder to select. This value can be literal or variable, text or position. To select the first item use "@1" in place of a text value for "Item".

"Options" The options are as follows:

"left" Use the left mouse button.

"right" Use the right mouse button.

"middle" Use the middle mouse button.

"down" Press the mouse button down.

"up" Release the mouse button.

"double" Double-click the mouse button.

"click" Click the mouse button once.

"control" Press the control key before the mouse button.

"shift" Press the shift key before the mouse button.

"with" Use in conjunction with "control" and "shift".

x , y These optional parameters specify where on the item the mouse button will be clicked. If omitted, the button is clicked in the center.

The function returns 1 if the selection is successful, and generates a runtime error if it is

not. See the On Error command for information on processing runtime errors in scripts. If this command is generated using the Learn facility, the parentheses are omitted. If a return value is required, you must use the parentheses.

Examples

Example 1:

```
; from the Browse dialog, change folders and run
; the Calculator program
Attach "~N~EXPLORER.EXE~#32770~Browse"
ListViewCtrl "~1", "(C:)", "Left Double"
ListViewCtrl "~1", "Windows", "Left Double"
ListViewCtrl "~1", "Calc", "Left Double"
```

Example 2:

```
; repeatedly select the third item
Attach "~N~EXPLORER.EXE~#32770~Browse"
ListViewCtrl "~1", "@3", "Left Double"
ListViewCtrl "~1", "@3", "Left Double"
ListViewCtrl "~1", "@3", "Left Double"
```

Log.Checks

Logging

Determines if checks are to be logged.

Syntax

Log.Checks = < ON | OFF >

See Also

Log.Comments, Log.Commands, Log.System, Log.DllCalls, Log.Enable

Operation

This system variable determines if checks will be recorded in the log. Setting the variable to 0 switches check logging off. The default value is 1 (checks are logged).

This system variable overrides the setting in the Logging area of the Run Environment Settings dialog box.

Examples

```
Log.Checks = 0 ; turn check logging off
Check "EZ TESTDemo Main Window" ; don't log this check
Log.Checks = 1 ; re-enable checks logging
```

Log.Commands

Logging

Determines if function calls are to be logged.

Syntax

Log.Commands = < ON | OFF >

See Also

Log.Comments, Log.Checks, Log.System, Log.DllCalls, Log.Enable

Operation

This system variable determines if function calls (commands) are to be recorded in the log. Setting the variable to 0 switches command logging off. The default value is 1 (commands are logged).

Examples

```
; turn function logging off
Log.Commands = 0
Type "This line is not logged"
; turn function logging on for this section
Log.Commands = 1
Type "This line is logged"
; turn function logging off again
Log.Commands = 0
Type "This line is not logged"
```

LogComment( )

Logging
Sends user comments to the log.

Syntax

LogComment("String to log")

See Also

Log.Comments

Operation

This function allows user-defined comments to be written to the log. This system variable overrides the setting in the Logging area of the Run Environment Settings dialog box.

Examples

Example 1:

```
; state why log entries will be turned off at this point
LogComment( "About to enter password details" )
LogComment( "Switching off logging for security reasons" )
Log.Enable = 0
Type Password
Log.Enable = 1
LogComment( "Password entered - logging resumed" )
```

Example 2:

```
; An example the includes an expression
ret = MessageBox( "Test" , "Select a button" , 'okcancel' )
LogComment( "User selected button " + ret )
```

Log.Comments

Logging
Determines if comments are to be logged.

Syntax

Log.Comments = value

See Also

LogComment( ), Log.Commands, Log.Checks, Log.System, Log.DllCalls, Log.Enable

Operation

This system variable determines if LogComment( ) commands are to be written to the log. Setting the value parameter to 0 switches comment logging off; setting value to 1 (the default value) switches comment logging on. This system variable overrides the setting in the Logging area of the Run Environment Settings dialog box.

Examples

```
; turn comment logging off
Log.Comments = 0
LogComment( "This comment is not logged" )"
; turn comment logging on
Log.Comments = 1
LogComment( "This comment is logged" )"
; turn comment logging off
Log.Comments = 0
LogComment( "And this one isn't" )"
```

Log.DLLCalls

Logging
Determines if DLL calls are to be logged.

Syntax

Log.DLLCalls = value

See Also

Log.Comments, Log.Checks, Log.System, Log.DllCalls, Log.Enable

Operation

This system variable determines if calls to DLL functions are to be recorded in the log. Setting the value parameter to 0 switches logging of DLL calls off, setting value to 1 (the default value) switches DLL call logging on.
This system variable overrides the setting in the Logging area of the Run Environment Settings dialog box.

Log.Enable

Logging
Turns logging on and off.
Syntax
Log.Enable = value
See Also
Log.Commands, Log.Comments, Log.System, Log.Checks, Log.DllCalls
Operation
This system variable is used to turn logging on and off. If value is set to 1, logging is enabled; if value is set to 0, logging is disabled.
This system variable overrides the setting in the Logging area of the Run Environment Settings dialog box.
Examples
; switch off logging when entering passwords and user ID
Attach "Logon Dialog"
Log.Enable = 0
Type ID ; contents of the ID variable
Type PassWord ; contents of the password variable
Log.Enable = 1 ; turn logging back on

Log.Name

Logging
Retrieves the name of the current log.
Syntax
CurLog = Log.Name
See Also
LogOpen( )
Operation
This read-only system variable contains the name of the current log. Use the LogOpen( ) function to set a new log.
Examples
OldLog = Log.Name ; save open log name
LogOpen("Audit", ; setup a status log
"AutoIncrement", "Status Report")
LogComment("Run started at " + Time( )) ; log status message
LogOpen("OldLog", "Append") ; reset previous log

LogOff( )

Logging
Disables the logging of specified functions.
Syntax
LogOff("FunctionType")
See Also
Log.Commands, Log.Enable, LogOn( )
Operation
This function disables the logging of functions specified by FunctionType. The parameters are as follows:
FunctionName The name of the function to be disabled, such as Attach, Type etc.

"*" An asterisk can be used to disable logging of all functions.
The LogOff( ) command only affects the logging of commands that are logged using the Log.Commands command. The Log.Commands command must be set to a value of 1 for the LogOff( ) command to affect the items being logged. If you wish to turn all logging off (checks, comments, DLL calls, etc.), use the Log.Enable command.

Examples

LogOff("Attach") ; single function
LogOff("Attach", "Type", "Pause") ; multiple functions
LogOff("*") ; all functions

LogOn( )

Logging
Enables the logging of specified functions.

Syntax

LogOn("FunctionType")

See Also

Log.Commands, Log.Enable, LogOff( )

Operation

This function enables the logging of functions specified by FunctionType. The parameters are as follows:
FunctionType The name of the function to be enabled, such as Attach, Type etc.
"*" An asterisk enables the logging of all functions (this is the default setting).
The LogOn( ) command only affects the logging of commands that are logged using the Log.Commands command. The Log.Commands command must be set to a value of 1 for the LogOn( ) command to affect the items being logged. If you wish to turn all logging off (checks, comments, DLL calls, etc.), use the Log.Enable command.

Examples

LogOn("Attach") ; single function
LogOn("Attach", "Type", "Pause") ; multiple functions
LogOn("*") ; all functions
;Only log checks
LogOff("*") ; turn off all function logging
LogOn("Checks") ; turn on check function logging

LogOpen( )

Logging
Sets the current Log.

Syntax

OldLog = LogOpen("NewLog" [, "Options", "Description"])

See Also

Log.Name

Operation

This function creates or opens the NewLog log and begins logging to it, overriding the setting in the Run Environment dialog box.
The "Options" are:
Append Appends new entries to the existing log.
AutoIncrement Creates a new log, incrementing the Run Number.
If no option is set, the existing log is erased. An optional "description" may be added; this is shown in the Description column of the Browse Logs grid.
This function returns the name of the previously opened log, or a null if no log was opened.

Examples

OldLog = LogOpen("Audit", ; setup a status log
"AutoIncrement", "Status Report")

```
LogComment( "Run started at " + Time( ) ) ; log status message
LogOpen( "OldLog", "Append" ) ; reset previous log
```

Log.System

Logging
Determines if system messages are to be logged.

Syntax

```
Log.System = value
```

See Also

Log.Enable, LogOff( )

Operation

This system variable determines if system messages, such as runtime errors, are to be recorded in the log. Setting value to 0 switches logging of system messages off. Setting value to 1 causes system messages to be logged.
This system variable overrides the setting in the Logging area of the Run Environment Settings dialog box.

Examples

```
Log.System = 0 ; disable logging of
; system messages
On Error Call Fix_Errors ; global error handler
Attach "~N~NOTEPAD.EXE~#32770~Open"
Button "&Open", 'Left SingleClick'
<further instructions>
```

LowerCase( )

String Manipulation
Converts a string to lowercase.

Syntax

```
ret = LowerCase( target )
```

Variants

```
ret = Lower( target )
```

See Also

UpperCase( )

Operation

Converts the contents of a string into lowercase characters.

Examples

```
target = "The Quick Brown Fox"
lctarget = LowerCase( target ) ; result = "the quick brown fox"
lc = LowerCase( "Hello World" ) ; result is "hello world"
```

LtrimStr( )

String Manipulation
Removes leading spaces from a string.

Syntax

```
ret = LtrimStr( target )
```

See Also

RtrimStr( )

Operation

This function removes leading spaces, tabs, carriage returns, and line feeds from a variable.

Examples

```
target = " Hello"
target = LtrimStr( target ) ; result is "Hello"
```

MakeCheck( )

Checks

Dynamically creates a new check using an existing check as a template.

Syntax

ret = MakeCheck("template" , "newcheckname" , "desc" ,
["attachname"])

See Also

Check( ), CheckExists( )

Operation

The MakeCheck( ) commands allows you to create checks at runtime. The check is created using all of the options (ignores, styles, pause 5 seconds, etc.) of the original check template. The attachname option can be used to specify an attach name other than that specified in the template check. Checks generated using the MakeCheck( ) command are treated just like any other check created manually. The parameters are as follows:

"template" The name of an existing check in the *EZ Test* database. The template check is used as a basis for the newcheckname when the check is generated at runtime. If the specified template check does not exist, the script will fail.

"newcheckname" The name that will be assigned to the new check. The check name is added to the *EZ Test* database once the check is run.

"desc" The description for the new check. The description appears in the MakeCheck( ) log entry and appears in the check's Description field in the *EZ Test* database.

"attachname" The optional attach name that will be used in the check. If no attachname is specified, the attach name identified in the template check will be used.

EZ Test evaluates attach name usage in the following order:

- The attachname specified in the MakeCheck function call.
- The attach name used in the template check.
- The current attach name if Replay.CheckCurrentAttach is set.
- Unable to create a check the function will return any empty string.

The function returns the name of the check that was created. If the check already exists, the function creates a numbered version (e.g., newcheckname0001), which is incremented each time a check is created using the same name. If the MakeCheck command is not successful, an empty string is returned.

Examples

Example 1:

```
; The first time, this script creates 9 checks & executes them
; When you run the script again, it checks the checks
Function Main
; Make the check for the first screen
MakeCheckAndWait()
; Setup the actionkeys we want
Replay.ActionKeys = "{F1}{F2}{F3}{F4}{F5}" +
"{F6}{F7}{F8}{F9}{F10}" +
"{F11}{F12}{Return}{Esc}"
; Start the whenever for the actionkeys
Whenever "actionkey" Call MakeCheckAndWait
; Automate testbed from the login screen
Attach "Testbed Connected"
Type "cw{Tab}pass{Return}"
Type "{F9}{Esc}{F1}{Esc}{Esc}"
Type "cw{Tab}pass{RShift}{Return}"
Type "{F9}{Esc}{F1}{F9}{Esc}{Esc}"
```

```
Type "{F4}{F9}{Esc}{Esc}{F5}{F9}"
Type "{Esc}{Esc}{Esc}"
End Function ; Main
;
Function MakeCheckAndWait()
Var ScreenID, ScreenTitle
; Wait for the system light to go off
Wait(30, "", "No X System")
; Get details of screen and attempt to create check
ScreenID = CaptureBox("Testbed Connect" , 563 , 47 , 76 , 11)
ScreenTitle= CaptureBox("Testbed Connect" , 137 , 45 , 393 , 12)
ScreenID = LTrimStr( ScreenID )
ScreenID = RTrimStr( ScreenID )
ScreenTitle = LTrimStr( ScreenTitle )
ScreenTitle = RTrimStr( ScreenTitle )
; If the template does not exist generate an error
If CheckExists( "MyTemplate" ) = 0
UserCheck("Temp Not Found" , 0 , "Cannot find MyTemplate")
return ;
endif
; If a check for this screen does not exist make one
If CheckExists( ScreenID ) = 0
; Use screen id for the name and make up a description using
; the screen id the title of the window
MakeCheck( "MyTemplate" , ScreenID , ScreenID + ", " +
ScreenTitle )
endif
; Now execute the check
Check( ScreenID )
End Function ; MakeCheckAndWait()
```

Example 2:

```
If CheckExists( "New" ) = 0 ; Check does NOT exist
; Create the check
MakeCheck( "Template" , "New" , "This is a new check" )
Endif
; Execute the check
Check( "New" )
```

MakeDir( )

File Access

Creates a new directory (folder).

Syntax

MakeDir(path)

Variants

MkDir(path)

See Also

RemoveDir( )

Operation

This function creates a new directory, or folder, at the path specified. The function returns a value of 1 if the operation is successful, and it returns 0 if it is not. The error handler must be enabled for command to be passed into the function successfully.

Examples

```
; create a working folder
MakeDir( "c:\Bob's Working Folder" )
```

MakeEvent( )

Synchronization

Defines a bitmap, date/time, keyboard, menu, mouse, window, screen, or window event

Syntax

Eventname = MakeEvent("Event type", "Options")

Variants

There are eight user-definable event types and each type has different options. The following section covers each variant.

See Also

Cancel( ), DestroyEvent( ), Event( ), Wait( ), Whenever

Operation

An event is a condition that occurs within the PC. For example:

- a key is struck
- a menu selection is made
- some text is displayed in a window
- a bitmap appears
- the internal clock reaches a particular time of day.

Events that are crucial to the successful execution of a script can be defined so that the script is made to wait for them to happen or to perform some action when they occur. The occurrence of a defined event can be determined by the Event( ) function.

The script language supports the definition of events within a script. However, you are strongly advised to define events using the Insert\Event facility from the script editor's menu. This defines the event within the event map — which has the following advantages over defining events within a script:

- It removes the event definition from the script — making it easier to read.
- It makes the defined event available to other scripts and other users — avoiding duplication of effort.
- It provides a single point of maintenance should the event definition need to be altered in future.

Should you wish to define an event within a script, use the syntax described below. Events created using the MakeEvent( ) command are maintained in memory until the script completes.

If you use MakeEvent( ) in a loop you will continue to create new instances of the same event, and not release the event from memory. As a general rule, after you are no longer expecting an event trigger, you should use the DestroyEvent( ) command to release the event from memory.

Bitmap Events:

Eventname = MakeEvent("Bitmap [event]", "Attachname", "BitmapCRC", "[notfound]", "[grabCRC]" , x, y, width, height)

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.

Bitmap Is the event type. This may be followed by the optional word, [Event].

"Attachname" Is the attach name (or object map name) of the window in which the event must occur (note that "anywindow" and "module" cannot be specified for bitmap events).

"BitmapCRC" Specifies the bitmap CRC number that triggers the event. The CRC is the number generated from *EZ Test*'s Cyclic Redundancy Check of the captured area.

"grabCRC" Forces *EZ Test* to recalculate the bitmap CRC during runtime, when the event is created (i.e., when the MakeEvent command is executed in the script).

The event will use the calculated CRC value when it is used in a Wait() or Whenever() statement. This option is typically use when you don't know what the bitmap will look like when the script is run, but you want to grab whatever is there.

"notfound" Specifies that the absence of "BitmapCRC" triggers the event. If the CRC values has changed from the expected value, the event will trigger.
x, y, width, height Defines a rectangle within which the Bitmap must be found for the event to trigger. The x,y values define the coordinates of the top-left corner of the rectangle, width and height define its size.
Database Schema Represents the *EZ Test* database schema. This parameter must be added for database schemas 127 and higher.

Date Events:

Eventname = MakeEvent("Date [event]", year, "month", "day", hh, mm, ss)
Eventname = MakeEvent("Date [event]", year, "month", "day", every_secs)
Where:
Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
Date Is the event type. This may be followed by the optional word, [Event].
year Is the year in which the event will trigger. A value of 0 represents any year.
month Is the month in which the event will trigger. This may be either a:
Numeric value (0-12) Where 1 = January, 2 = February, 3 = March, etc. and 0 represents any month.
String Representing Month Name "January", "February", "March", etc.
day Is the day on which the event will trigger. This may be either a:
Numeric value (0 - 31) Where 0 represents any day.
String Representing Day Name "Sunday", "Monday", "Tuesday", etc. This can be used to set up events that trigger every week.
hh Specifies the hour the event will trigger.
mm Specifies the minute the event will trigger.
ss Specifies the second the event will trigger.
every_secs Is used with a Whenever to specify an event that will trigger every <every_secs> seconds when the date part of the event has triggered.

Keyboard Events:

Eventname = MakeEvent("Keyboard [event throwaway]", "Attachname", "Keylist")
Where:
Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
Keyboard Is the event type. This may be followed by the optional word, [event].
throwaway Prevents the keys defined in "Keylist" from reaching the application.
"Attachname" Is one of the following forms:
"anywindow" Indicates that the event can be triggered in any application window.
"<AttachName>" The attach name (or object map name) of the window in which the event must occur.
"module <ModuleName>" Instructs the event to use the

application's EXE name. The
"<ModuleName>" specifies the module
name.
"Keylist" Defines the key(s) that trigger the event. When multiple keys are defined, for example "abcd", the event triggers on any one of the defined keys.

Menu Events:

Eventname = MakeEvent("Menu [event]", "Attachname", "MenuItem")

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
Menu Is the event type. This may be followed by the optional word, [event].
"Attachname" Is one of the following forms:
"anywindow" Indicates that the event can be triggered in any application window.
"<AttachName>" The attach name (or object map name) of the window in which the event must occur.
"module <ModuleName>"Instructs the event to use the application's EXE name. The "<ModuleName>" specifies the module name.
"MenuItem" Defines the menu selection that triggers the event. This can be a numeric or string value. ID has a prefix of an '@'.

Mouse Events:

Eventname = MakeEvent("Mouse [event throwaway]", "Attachname",
" button, action, [with]", x, y, width, height)

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
Mouse Is the event type. This may be followed by the optional word, [event].
throwaway Prevents the actions defined in action from reaching the application.
"Attachname" Is one of the following forms:
"anywindow" Indicates that the event can be triggered in any application window.
"<AttachName>" The attach name (or object map name) of the window in which the event must occur.
"module <ModuleName>"Instructs the event to use the application's EXE name. The "<ModuleName>" specifies the module name.
button Defines which button to set the event for. This can be:
left The left mouse button.
middle The middle mouse button.
right The right mouse button.
action Defines the mouse action. This can be:
up Releasing the mouse button.
down Pressing the mouse button down.
double Double-Clicking the mouse button.
with Specifies that a key must be held down. The options are:
with control While holding down the Ctrl key.
with shift While holding down a Shift key.

with control shift While holding down both Shift and Ctrl keys.
x, y, width, height Defines a rectangle within which the mouse action must occur for the event to trigger. The x and y values define the coordinates of the top left corner of the rectangle. The width and height values define its size.

Screen Events:

Eventname = MakeEvent("Screen [event]", "Attachname", "Text", "[notfound] [erase] [windowtext] ", x, y, width, height)

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
Screen Is the event type. This may be followed by the optional word, [Event].
"Attachname" Is the attach name (or object map name) of the window in which the event must occur (note that "anywindow" and "module" cannot be specified for screen events).
"Text" Specifies the text string that triggers the event.
"notfound" Specifies that the absence of "Text" triggers the event.
"erase" Forces the target application window to repaint. Use this option if a screen event does not trigger correctly due to the target application's reluctance to repaint (a screen flicker may be apparent).
"windowtext" Captures all text in the specified attach window, even if the complete text may not be visible. This option is useful for capturing status bar and title bar information.
x, y, width, height Defines a rectangle within which the Text must be found for the event to trigger. The x,y values define the coordinates of the top-left corner of the rectangle, width and height define its size.

Time Events:

Eventname = MakeEvent("Time [event]", hh, mm, ss)
Eventname = MakeEvent("Time [event]", every_secs)

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
hh Specifies the hour the event will trigger.
mm Specifies the minute the event will trigger.
ss specifies the second the event will trigger.
every_secs Is used with a Whenever to specify an event that will trigger every <every_secs> seconds when the date part of the event has triggered.

Time events are simplified Date/Time events that can be used when the Date information is irrelevant.

Window Events:

Eventname = MakeEvent("Window [event]", "Attachname" , "type")

Where:

Eventname Is the ID used to identify the event within event calls. This is updated with the result of the event following a call.
Window Is the event type. This may be followed by the optional word, [event].
"Attachname" Is one of the following forms:
"anywindow" Indicates that the event can be triggered in any application window.
"<AttachName>" The attach name (or object map name) of

the window in which the event must
occur.
"module <ModuleName>"Instructs the event to use the
application's EXE name. The
"<ModuleName>" specifies the module
name.
type Defines the type of window event. It can be any one of the
following forms:
minimize The window is minimized (iconized).
maximize The window is maximized.
restore The window is restored.
destroy The window is destroyed.
move The window is moved.
size The window is re-sized.
create The window is created.
focus The window is in focus.
killfocus The window loses focus.
exists The window exists.
notexists The window does not exist.
active The window is in focus.
deactive The window is not in focus.

Examples

Example 1 (Keyboard Event):

```
F9Key = MakeEvent( "keyboard throwaway", "anywindow", "{F9}" )
Whenever F9Key Call Display ; whenever the event happens,
; call the Display function
Suspend ; suspend, leave whenevers active
Function Display
MsgBox( Event( F9Key ), "You struck the {F9} key" )
End Function
```

Example 2 (Mouse Event):

```
ML2 = MakeEvent( "mouse event", "module Notepad.exe", "left
double", 0 0 30, 20 )
Whenever ML2 Call Search
Suspend
Function Search
Attach "Notepad"
MenuSelect "Search~Find..."
End Function
```

Example 3 (Window Event):

```
NPExists = MakeEvent( "window", "module NOTEPAD.EXE", "exists" )
Exec "Notepad.exe"
Wait (10 "for" NPExists )
If Event( NPExists ) = 1
MsgBox( "Notepad", "You may now continue" )
Else
MsgBox( "Notepad", Notepad is not responding" )
Endif
```

Example 4 (Menu Event):

```
NoPrint = MakeEvent( "Menu throwaway", "anywindow", "File~Print" )
Whenever NoPrint call NoPrint
Suspend
Function NoPrint
MsgBox( "Printing", "Printing Services are suspended" )
End Function
```

Example 5 (Date / Time Event):

```
; at 09:30:00 on December 25, 1996
DT = MakeEvent( "date", 1996, 12, 25, 09, 30, 00 )
Wait( 0, "", DT )
```

```
MsgBox( "", "Merry Christmas" )
```
Example 6 (Date / Time Event):
```
; every 10 minutes on December 25, 1996
DT = MakeEvent( "date", 1996, 12, 25, 600 )
Whenever DT Call Greetings
Function Greetings
MsgBox( "", "Merry Christmas" )
End Function
```
Example 7 (Date/Time Event):
```
; at 09:30:00 on December 25, 1996
DT = MakeEvent( "date", 1996, "December", 25, 09, 30, 00 )
```
Example 8 (Date/Time Event):
```
; at 09:30:00 every Saturday in December 1996
DT = MakeEvent( "date", 1996, "December", "Saturday", 09, 30, 00 )
Whenever DT Call Satjob
```
Example 9 (Date/Time Event):
```
; at 00:00:01 on the first day of each month
DT = MakeEvent( "date", 0, 0, 01, 00, 00, 01 )
Whenever DT Call NewMonth
```
Example 10 (Date/Time Event):
```
; every 5 minutes
DT = MakeEvent( "date", 0, 0, 0, 300 )
Whenever DT Call CheckMail
```
Example 11 (Time Event):
```
; at a quarter past two
TE = MakeEvent( "time", 14, 15, 0 )
Wait( 0, "", TE )
```
Example 12 (Time Event):
```
; every hour
TE = MakeEvent( "time", 3600 )
Whenever TE Call CheckMail
```
Example 13 (Bitmap Event):
```
Function Main
; Determine when the web browser has stopped processing
; requested information using busy indicator
Attach "CompuServe Mosaic - CompuServe/Web MainWindow"
Button "@Globie PushButton", 'Left SingleClick'
Compuserve = MakeEvent( "Bitmap event", ; Event Type
"module Globie PushButton", ; Attach
0x6572C065, ; Bitmap CRC value
"", ; CRC is the same
6, 4, 57, 54 ) ;(X,Y,Width,Height)
If Wait(30, "", Compuserve ) = 1
MsgBox( "Processing" , "Finished Processing Request" , 'ok' )
Else
MsgBox( "Timeout" , "Request Timed Out" , 'ok' )
EndIf
End Function
```

Max( )

Number Manipulation

Returns the maximum value from a list of numbers.

Syntax

```
ret = Max( val1, val2, [,...valn] )
```

See Also

Min( )

Operation

This function returns the maximum value from a list of numbers.

Examples

ret = Max(10, 20, 3, 5) ; returns 20

ret = Max(-10, -20, -3, -5) ; returns -3

Maximize( )

Window Control
Maximizes a window.

Syntax

ret = Maximize("Windowname")

Variants

Maximize(Windowhandle)
Maximize( )

See Also

Minimize( ), Move( ), Restore( ), SetFocus( ), Size( ), WinClose( )

Operation

This function maximizes the window specified by the "Windowname" parameter. If no parameter is specified, the currently attached window is maximized. In some applications, the attached window is not necessarily the top-most window. For example, *EZ Test* may attach to an edit control or some other child window. If this is the case the "Windowname" parameter must be used, or the function will attempt to maximize the edit control or other child window.
The function returns 1 if the window is maximized successfully, and it returns 0 if it does not.
When this command is generated by the Learn facility, the parentheses are omitted.

Examples

Example 1:

```
; execute the target application and maximize it
Exec( "C:\EZ TESTPT\ADDRESS" ) ; Run the Address Book Application
; attach to the parent window
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
Maximize ; maximize the application
```

Example 2:

```
ret = Activewindow( ) ; get active window handle
Maximize( ret ) ; and maximize it
```

MenuCount( )

Menu Information
Returns the number of items on the specified menu level.

Syntax

Ret = MenuCount("AttachName" , "MenuName" , ["Options"])

See Also

IsMenu( ), MenuFindItem( ), MenuItem( )

Operation

This function returns the number of items on the specified menu level.
"AttachName" The object map name or raw attach name of the parent window of the top-level menu.
"MenuName" The root menu name to look for the MenuItem in (e.g. "File" means search the File menu. "View~Grid~Sort by" means search at the "Sort by" menu level).
The MenuName parameter can be preceded by the following prefixes.
Normal# Indicates the menu is a normal menu (default).
Popup# Indicates the menu is a popup menu.
System# Indicates the menu is a system menu.
For example, to interrogate the popup menu on the *EZ Test* grid in Event Map, use the following value for

MenuName:
"Popup#Create"
The "#" sign is used to distinguish the prefix from the
actual menu name. This prevents *EZ Test* from
interpreting the prefix as part of the actual menu name.
"Options" The options are as follows:
"includesubmenus" Includes submenus (default).
"nosubmenus" Returns items from this menu level,
but don't include submenus.
This command can not be used to attach to toolbar menus that create pop-up menus (i.e., pop-up menus that are created using Internet Explorer's toolbar buttons).

Examples

Example 1:

```
; Get the number of items of the top-level menu
Ret = MenuCount( "Notepad" , "File" , "nosubmenus" )
; Returns 9
```

Example 2:

```
; Get the number of items of the Search menu
Ret = MenuCount( "Notepad" , "Search")
; Returns 3
```

MenuCtrl( )

Dialog Control
Processes a menu control on Web-based applications (for example, Oracle Web Forms).

Syntax

```
ret = MenuCtrl( "ControlId", "Options", x, y )
```

Variants

```
MenuCtrl( "ControlId", "Options" )
```

See Also

CheckBox( ), ComboBox( ), ComboText( ), EditText( ), ListBox( ), RadioButton( ), ScrollBar( )

Operation

This function processes a menu item in the menu bar in the currently attached window. The action is specified in "Options". The function parameters are as follows:
"ControlId" Specifies the menu label, such as "File", "Edit", "Help".
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the button.
"singleclick" Click the button once.
"control" Press the control key before clicking the
button.
"shift" Press the shift key before clicking the button.
"with" Use in conjunction with "control" and
"shift".
x , y These optional parameters specify where on the menu the mouse
button will be clicked. If omitted, the menu item is clicked in the
center.
The function returns 1 if the menu item is successfully selected, and it returns 0 if the menu item could not be selected.
When this command is generated by the Learn facility, the parentheses are omitted.

Examples

```
Attach "Puzzle Applet"
```

MenuCtrl "Game", 'Left SingleClick' 34 , 1

MenuFindItem( )

Menu Information
Returns the name or position of a specified menu item.

Syntax

Ret = MenuFindItem("AttachName" , "MenuName" , "MenuItem")

See Also

IsMenu( ), MenuCount( ), MenuItem( )

Operation

This command can be used to pass a menu item name and returns the position, or to pass a position and return the menu item name at that position. The parameters are as follows:
"AttachName" The object map name or raw attach name of the parent window of the top-level menu.
"MenuName" The root menu name in which to look for the MenuItem (e.g. “File” means search the File menu. “View~Grid~Sort by” means search at the “Sort by” menu level).
The MenuName parameter can be preceded by the following prefixes.
Normal# Indicates the menu is a normal menu (default).
Popup# Indicates the menu is a popup menu.
System# Indicates the menu is a system menu.
For example, to interrogate the popup menu of Notepad’s edit control, use the following value for MenuName:
"Popup#"
The “#” sign is used to distinguish the prefix from the actual menu name. This prevents *EZ Test* from interpreting the prefix as part of the actual menu name.
"MenuItem" The menu text to search for, or the menu position (e.g., “Open” or 4).

Return Values

If menu text is used, the position is returned. In contrast, if the position is used, the menu item text is returned. Position is always relative to the parent of the menu item

Notes

This command can not be used to attach to toolbar menus that create pop-up menus (i.e., pop-up menus that are created using Internet Explorer’s toolbar buttons).
This command returns a -1 or NOTFOUND if MenuItem is not found.

Examples

The examples will all use the the Windows' Notepad program, and an Object Map entry named "Untitled - Notepad MainWindow" which corresponds the Main Window of Notepad.

Example 1:

```
; Get the position of the Save item from the File menu
Ret = MenuFindItem( "Untitled - Notepad MainWindow" , "File"
, "Save" )
```

Example 2:

```
; Get the text on the 1st menu item on the Help menu
Ret = MenuFindItem( "Untitled - Notepad MainWindow", "Help",
1 )
```

Example 3:

```
; Get the text of the 3rd menu item on the popup menu displayed
when Notepad's edit field is Right Clicked
Attach "Untitled - Notepad MainWindow"
```

```
EditClick "~1", 'Right SingleClick', 1, 1
Ret = MenuFindItem( "Untitled - Notepad MainWindow" , "Popup#"
, 3 )
```
Example 4:
```
;Get the position of the Paste item on the popup menu
Attach "Untitled - Notepad MainWindow"
EditClick "~1", 'Right SingleClick', 1, 1
ret = MenuFindItem( "Untitled - Notepad MainWindow" , "Popup#"
, "Paste")
```

MenuItem( )

Menu Information
Returns the text of a specific menu item.

Syntax

ret = MenuItem("Attachname" , "MenuName" , Item , ["Options"])

See Also

IsMenu( ), MenuCount( ), MenuFindItem( )

Operation

The function returns the text of the menu item specified by the MenuName and MenuItem parameters. The parameters are as follows:
"AttachName" The object map name or raw attach name of the parent window of the top-level menu.
"MenuName" The top-level menu name to get the MenuItem from (for example, “File” means search the File menu and it’s sub menus).
The MenuName parameter can be preceded by the following prefixes:
Normal# Indicates the menu is a normal menu (default).
Popup# Indicates the menu is a popup menu.
System# Indicates the menu is a system menu.
For example, to interrogate the popup menu on the *EZ Test* grid in Event Map, use the following value for MenuName:
"Popup#Create"
The “#” sign is used to distinguish the prefix from the actual menu name. This prevents *EZ Test* from interpreting the prefix as part of the actual menu name.
Item Either the index number of the Item (for use with byindex), or the Menu ID (for use with byid). The options are:
"byindex" Gets the text of the item in the menu as specified in Item (default).
"byid" Gets the text of this menu ID.
This command can not be used to attach to toolbar menus that create pop-up menus (i.e., pop-up menus that are created using Internet Explorer’s toolbar buttons).

Examples

Example 1:
```
; Get the fourth item of the Normal Menu
Ret = MenuItem( "Notepad", "File", 4 ) ; returns Ret = "Save as"
```
Example 2:
```
; Get the text of the menu item with ID = 768
Ret = MenuItem( "Notepad", "Normal#Edit", 768, "byid" )
; returns Ret = "Cut"
```

MenuSelect( )

Menu Control
Selects an item from the menu of the currently attached window.

Syntax

ret = MenuSelect("MenuItem")

Variants

MenuSelect(NumericID)

See Also

SysmenuSelect( )

Operation

This function selects the menu item specified by the "MenuItem" parameter from the menu of the currently attached window. The parameter can be numeric or a string. The function returns 1 if the selection is successful, and it returns 0 if it is not.
Note: Menu selections which are "Learned" have the quick key and accelerator key removed. For example, learning File>**Open**… from the following menu: generates the script: MenuSelect "File~Open…". This makes the script insensitive to changes to quick keys and accelerators.
If this command is generated using the Learn facility, the parentheses are omitted. However, if a return value is required, you must use parentheses.

Examples

Example 1:

```
; put the target application Address Book into View mode
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
MenuSelect "&Address~&View"
```

Example 2:

```
; as above using numeric ID
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
MenuSelect 13
```

Example 3:

```
; return result to check it worked
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
ret = MenuSelect( "&Address~&View" )
MessageBox( "Result", ret )
```

MessageBox( )

Miscellaneous
Creates a message box with buttons, an icon, and message text.

Syntax

ret = MessageBox("Title", "Message", ["Options"])

Variants

ret = MsgBox("Title", "Message", ["Options"])

See Also

Dialog( ), PromptBox( )

Operation

This function creates a standard Windows message box. The "Title" is displayed in the title bar of the box, the "Message" is user-defined text that displays in the center of the box and the "Options" determine the controls contained in the box. If no "Options" are specified, the box contains an OK button only. The "Options" are:

Option	Displays:
"ok"	OK button (Default).
"okcancel"	OK and Cancel buttons.
"abortretryignore"	Abort, Retry, and Ignore buttons.
"yesnocancel"	Yes, No, and Cancel buttons.
"yesno"	Yes and No buttons.
"retrycancel"	Retry and Cancel buttons.
"default1"	Make the first button the default (this is

the default setting).
"default2" Make the second button the default.
"default3" Make the third button the default.
"hand" The Stop icon.
"question" The Question Mark icon.
"exclamation" The Triangle Exclamation Icon.
"asterisk" The Information Icon.
Each button has an associated return value that is used to determine the user's response and process the result. The values are:

Button type Return Value

OK 1
Cancel 2
Abort 3
Retry 4
Ignore 5
Yes 6
No 7

Examples

Example 1:

```
ret = MessageBox( "Error", "Network failure", "retrycancel hand" )
; generates the following MessageBox:
```

Example 2:

```
repeat
< Process instructions >
ret = MsgBox( "Process", "Repeat Process?", "yesno question" )
until ret <> 6
```

Mid( )

String Manipulation
Extracts a substring from the middle of another string.

Syntax

```
ret = Mid( source, startpos, count )
```

Variants

```
ret = Mid( source, startpos )
```

See Also

Length( ), Left( ), Right( )

Operation

This function extracts a substring from a given string. The parameters are as follows:
source The source string. This can be a literal or variable value.
startpos The position in source to start the extraction.
count An optional setting for the number of characters to extract. If not specified, the rest of the string from the startpos position is extracted.

Examples

```
astring = "The quick brown fox" ; the source string
ret = Mid( astring, 5 ) ; returns "quick brown fox"
ret = Mid( astring, 5, 5 ) ; returns "quick"
ret = Mid( astring, 24 ) ; returns ""
ret = Mid( astring, 0 ) ; returns ""
```

Min( )

Number Manipulation
Returns the minimum value from a list of numbers.

Syntax

```
ret = Min( val1, val2, [,...valn] )
```

See Also
Max( )
Operation
This function returns the minimum value from a list of numbers.
Examples
ret = Min(10, 20, 3, 5) ; returns 3
ret = Min(-10, -20, -3, -5) ; returns -20

Minimize( )

Window Control
Minimizes a window.
Syntax
ret = Minimize("Windowname")
Variants
Minimize(Windowhandle)
Minimize( )
Iconize("Windowname")
Iconize(Windowhandle)
Iconize( )
See Also
Maximize( ), Move( ), Restore( ), SetFocus( ), Size( ), WinClose( )
Operation
This function minimizes the window specified by the "Windowname" parameter. If no parameter is specified, the currently attached window is minimized. In some applications, the attached window is not necessarily the top-most window. For example, *EZ Test* may attach to an edit control or some other child window. In this case, the "Windowname" parameter must be used, or the function attempts to minimize the edit control or child window. When this command is generated by Learn , the parentheses are omitted.
The function returns 1 if the window is minimized successfully, and it returns 0 if it is not.
Examples
Example 1:
Attach "~N~EXPLORER.EXE~ExploreWClass~Exploring - Docs"
Minimize
Example 2:
; minimize all active applications before running test scripts
Ret1 = isWindow("~U~EXPLORER.EXE~Shell_TrayWnd~","Active"); is the
active WindowTray
While ret1 <> 1 ; loop while WindowTray is not active
Ret = activewindow( ) ; get currently active window handle
While ret <> 180 ; as long as it's not the desktop
Pause 1 ; pause one second
Minimize(ret) ; minimize the active window
Ret = activewindow( ) ; get the handle of the next one
Wend
Ret1 =
isWindow("~U~EXPLORER.EXE~Shell_TrayWnd~","Active"); is
the active WindowTray

Mins( )

Date/Time
Returns the specified minutes.
Syntax
ret = Mins(timeval)
Variants
ret = Mins( )
See Also

TimeVal( ), CurTime( )

Operation

This function returns the minutes of the hour specified by timeval. The timeval parameter is a value that can be derived from the TimeVal( ) or CurTime( ) functions. If timeval is not specified, the current system time is used.

Examples

Example 1:

n = TimeVal(15, 11, 30) ; returns 54690
Minute_of_Hour = Mins(n) ; returns 11
Example 2:
Currrent_Minutes = Mins( ) ; current minutes value

Month( )

Date/Time
Returns the month number.

Syntax

ret = Month(dateval)

Variants

ret = Month( )

See Also

DateVal( ), CurTime( )

Operation

This function returns the month number specified by the dateval parameter. The dateval is a date value that can be derived from the DateVal( ) or CurTime( ) functions. If dateval is not specified, the current system date is used.

Examples

Example 1:

n = DateVal(1995, 11, 15) ; returns 816393600
Month_of_Year = Month(n) ; returns 11
Example 2:
Month_of_Year = Month( ) ; current month of the year

MouseClick( )

Mouse Control
Simulates the clicking of a mouse button in the currently attached window.

Syntax

ret = MouseClick(x, y, "options")

See Also

MouseMove( )

Operation

This function simulates the clicking of a mouse button in the currently attached window. The function returns 1 if the mouse click is successful, and it returns 0 if it is not. If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, the parentheses are needed.
The parameters are as follows:
x The x-position relative to the top-left corner of the client area of the currently attached window.
y The y-position within the client area of the currently attached window.
The "options" are:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the mouse button.

"singleclick" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Used in conjunction with "control" and "shift".

Examples

Example 1:

```
; double-click on the desktop to launch an application
Attach "~P~EXPLORER.EXE~SysListView32~Program Manager"
MouseClick 38, 98, "Left Doubleclick" ; double-click left button
```

Example 2:

```
; select a block of files from Explorer
Attach "~P~EXPLORER.EXE~SysListView32~Exploring - DTL"
MouseClick 30, 43, "Left Down" ; click down
MouseClick 30, 43, "Left Up" ; release button
; using the Shift Key select a range
MouseClick 38, 162, "Left Down With Shift"
MouseClick 38, 162, "Left Up With Shift"
```

Example 3:

```
; deselect files from a range using the Control Key
Attach "~P~EXPLORER.EXE~SysListView32~Exploring - DTL"
MouseClick 31, 89, "Left Down With Control" ; first file
MouseClick 31, 89, "Left Up With Control"
MouseClick 40, 127, "Left Down With Control" ; second file
MouseClick 40, 127, "Left Up With Control"
```

Example 4:

```
; select a menu option from the desktop popup menu
Attach "~P~EXPLORER.EXE~SysListView32~Program Manager"
; click the right button once to bring up menu
MouseClick 728, 400, "Right Down"
MouseClick 728, 400, "Right Up"
; make a menu selection from the popup menu
Attach "~P~EXPLORER.EXE~SHELLDLL_DefView~Program Manager"
PopupMenuSelect "Lin&e up Icons"
```

MouseCursor( )

Window Information
Returns the shape or type of Windows cursor (arrow, ibeam, cross, etc.).

Syntax

```
ret = MouseCursor( )
```

Variants

```
ret = MouseCursor( "name" )
ret = MouseCursor( number )
```

Operation

This function returns the shape of the mouse-cursor of the attached window.
If a parameter is not specified, the function returns a number that indicates the cursor shape. If the cursor is not one of the standard windows cursors, the function returns a value of 0.
To extract the name of the cursor as a string, pass the "name" parameter.
Passing a numeric parameter between 1 and 15 causes the MouseCursor( ) command to check if the cursor number matches the current cursor. If it does, the function returns a value of 1, otherwise it returns 0.
The following is a table of cursor numbers and names of the cursors. These are the default values for the cursors. The name and number will always be the same.

MouseCursor( ) MouseCursor("name")
0 Unknown Cursor -
1 ARROW
2 IBEAM
3 WAIT

4 CROSS
5 UPARROW
6 SIZE
7 ICON -
8 SIZENWSE
9 SIZENESW
-

Examples

```
; This will wait until the cursor is not the WAIT cursor
;
Repeat
Pause 1 "ticks"
Until MouseCursor( ) <> 3
;This uses the name of the cursor to check for an IBEAM
;
If MouseCursor( "name" ) = "IBEAM"
Type "data into application"
EndIf
; This example passes the cursor number and checks
; whether the function returns 1 to indicate that the
; cursor is the same.
;
If MouseCursor( 3 ) = 1 ; WAIT Cursor is 3
TextPanel( 1, "Wait cursor is visible" )
EndIf
```

MouseHover( )

Mouse Control

Moves the mouse pointer to the control specified and "hovers" for the specified seconds.

Syntax

```
ret = MouseHover("ControlType" , "ControlID" ,[x, y] ,[ LengthOfTime ])
10 SIZEWE
11 SIZENS
12 SIZEALL
13 NO
14 APPSTARTING
15 HELP
MouseCursor( ) MouseCursor( "name" )
```

Operation

This command moves the mouse pointer over the control specified by ControlType and ControlID parameters. This command can be used when a control changes its state or appearance as the mouse is moved over it. For example, some Web pages include Anchors that, when hovered over, change color and then display a menu. This command allows the script to hover over the control for the specified period and then verify the result of the hover (new graphic, pop-up menu, change in color, etc.).

If the x and y parameters are also specified, the mouse is moved relative to the top-left coordinates within the control. If the LengthOfTime parameter is specified, this option causes the script to pause for the specified number of seconds. If the LengthOfTime parameter is not specified, the script pauses for a default time specified by Mouse Hover Delay replay setting. The parameters are as follows:

"ControlType" The type of control (for example, Anchor, Image, etc.).

"ControlID" This is the control label or the internal name (if Learn by 4GL option was used).

[x, y] Optional parameter specifying mouse position relative to the top-left corner of the control.

[LengthOfTime] Optional parameter that causes the script to pause for the specified number of seconds. If not

specified, the value indicated in the script's replay
options will be used.
When this command is generated using the Learn facility, the parentheses are omitted. All mouse movements are ignored unless a special event occurs.
The function returns 1 if the mouse hover is successful, and it returns a 0 if it is not.

Examples

Attach "MSDN Online Web Workshop ChildWindow"
MouseHover "Anchor", "Community"
MouseHover "Anchor", "Essential", 1, 2, 3

MouseMove( )

Mouse Control
Moves the mouse pointer to the position specified.

Syntax

ret = MouseMove(x, y)

Variants

MouseMove x, y

See Also

MouseClick( )

Operation

This function moves the mouse pointer to the position specified by the x, y value. The position is relative to the top-left corner of the currently attached window and is specified in pixels.
When this command is generated using the Learn facility, the parentheses are omitted. All mouse movements are ignored unless a button is held down.
The function returns 1 if the mouse move is successful, and it returns a 0 if it is not.

Examples

; draw a line in Paint
Attach "~P~MSPAINT.EXE~Afx~untitled - Paint"
MouseClick 49, 59, "Left Down" ; hold left button down
MouseMove 120, 67 ; move mouse
MouseMove 208, 93
MouseMove 254, 137
MouseClick 254, 137, "Left Up" ; release button

MouseWindow( )

Window Information
Returns the attach name of the window under the mouse pointer.

Syntax

ret = MouseWindow( )

See Also

ActiveWindow( ), ActiveName( ), TopWindow( ), IsWindow( ), WinGetPos( ), FocusWindow( ), FocusName( ), AttachName( )

Operation

This function returns the name of the window directly under the mouse pointer. The name returned is a valid attach name and can be used to Attach to the found window. This function can be used to determine the name of pop-up menus — which disappear as soon as you click on another area.

Examples

; get the attachname of the desktop pop up menu
; attach to the desktop
Attach "~P~EXPLORER.EXE~SysListView32~Program Manager"
MouseClick 400, 400, "Right Down" ; click right mouse button
MouseClick 400, 400, "Right Up" ; release mouse button
MouseMove 425, 425 ; move mouse pointer over popup

```
; menu
pause 1 ; wait a second before doing
; function
ret = MouseWindow( ) ; get Attach name
MsgBox( "Result", ret ) ; display it in a MessageBox
```

MouseX( )

Mouse Information

Returns the X-position of the mouse pointer, in pixels, relative to the left of the screen.

Syntax

```
ret = MouseX( )
```

See Also

MouseY( ), AttachMouseX( ), AttachMouseY( ), MouseClick( ), MouseMove( )

Operation

This function returns the absolute position of the mouse pointer, in pixels, from the left side of the screen.

Examples

```
; click the mouse in the center of a SVGA (800*600) screen
; using both MouseX( ) and MouseY( ) functions
x = MouseX( ) ; get horizontal position of
; mouse pointer
y = MouseY( ) ; get vertical position of
; mouse pointer
If x <> 400 and y <> 300 ; if mouse is not centered
; attach to the desktop
Attach "~P~EXPLORER.EXE~SysListView32~Program Manager"
MouseMove( 400, 300 ) ; move mouse to center of
; screen
x = MouseX( ) ; get X position again
y = MouseY( ) ; get Y position again
MouseClick( x, y, "Right Down" ) ; click right button down
MouseClick( x, y, "Right Up" ) ; release button
Endif
```

MouseY( )

Mouse Information

Returns the Y-position of the mouse pointer, in pixels, relative to the top of the screen.

Syntax

```
ret = MouseY( )
```

See Also

MouseX( ), AttachMouseX( ), AttachMouseY( ), MouseClick( ), MouseMove( )

Operation

This function returns the absolute position of the mouse pointer, in pixels, from the top of the screen.

Examples

```
; click the mouse in the center of a SVGA (800*600) screen
; using both MouseX( ) and MouseY( ) functions
x = MouseX( ) ; get horizontal position of
; mouse pointer
y = MouseY( ) ; get vertical position of mouse
; pointer
If x <> 400 and y <> 300 ; if the mouse is not centered
; attach to the desktop
Attach "~P~EXPLORER.EXE~SysListView32~Program Manager"
MouseMove( 400, 300 ) ; move mouse to center of screen
x = MouseX( ) ; get X position again
y = MouseY( ) ; get Y position again
```

```
MouseClick( x, y, "Right Down" ); click right button down
MouseClick( x, y, "Right Up" ) ; release button
Endif
```

Move( )

Window Control
Moves the currently attached window to the specified position.

Syntax

ret = Move(x, y)

See Also

Maximize( ), Minimize( ), Size( ), WinClose( ), SetFocus( )

Operation

This function moves the currently attached window to the location specified by the x, y position. The x,y positions are relative to the top-left corner of the screen.
When this command is generated using the Learn facility, the parentheses are omitted.
The function returns 1 if the move is successful, and it returns 0 if it is not.

Examples

Example 1:

```
; attach to the target application
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
Move 195, 32 ; move the window
```

Example 2:

```
; attach to the target application
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
ret = Move( 195, 32 ) ; move the window
```

NCMouseClick( )

Mouse Control
Executes a mouse click in a non-client window.

Syntax

NCMouseClick(x, y, "options")

See Also

MouseClick( )

Operation

This function simulates the clicking of a mouse button in a non-client area of a window. A non-client area is a part of a window that would not normally receive keyboard or mouse input. Examples are:

- Window's title bar
- Scrollbar window when no "bar" is displayed
- Area behind a client window.

The parameters are as follows:
x The x-position relative to the top-left corner of the client area of the currently attached window.
y The y-position within the client area of the currently attached window.
The options are:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick"Double-click the mouse button.
"singleclick" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.

"with" Use in conjunction with “control” and “shift”.
The function returns 1 if the mouse click is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted.
However, if the return value is required, you must use parentheses.

Examples

Attach "~N~NOTEPAD.EXE~Notepad~Untitled - Notepad"
NCMouseClick 125, -31, 'Right Down' ; right mouse click on
MouseClick 125, -31, 'Right Up' ; Notepad's title bar
Attach "~P~NOTEPAD.EXE~Edit~Untitled - Notepad"
NCMouseClick 24, 251, 'Left Down' ; left mouse click on
NCMouseClick 24, 251, 'Left Up' ; an inactive scrollbar
ScrollBarWindow 0, 'Set Horz'
Attach "~P~NOTEPAD.EXE~Edit~Untitled - Notepad"
NCMouseClick 30, 479, 'Right Down With Control'; right mouse
NCMouseClick 30, 479, 'Right Up With Control' ; with Ctrl key
NCMouseClick 30, 479, 'Right DoubleClick With Control'

NotifyEvent( )

Performance Monitoring
Generates an event that can be monitored by the American Systems ClientVantage application, to time round-trip transactions.

Syntax

ret = NotifyEvent("strEventText")

Operation

This function fires an event to an external performance monitor passing the text specified by the “strEventText” parameter.
The function returns 1 if the call to the performance monitor was successful, or 0 otherwise.

Examples

NotifyEvent(“Performance Checkpoint 1”)
Attach "Untitled - Notepad MainWindow"
MenuSelect "Help~About Notepad"
Attach "About Notepad PopupWindow"
Button "OK", 'Left SingleClick'
NotifyEvent(“Performance Checkpoint 2”

On Error

Program Flow
Handles runtime errors in scripts.

Syntax

On Error Call <Error Handling Routine>

Variants

On Error End,

See Also

Err, ErrFile, ErrFunc, ErrLine, ErrMsg, Error, Error Codes, Resume Next

Operation

The scripting language supports the capturing and handling of errors that can occur during the execution of a script. These errors are usually critical events that require immediate attention before the script can continue executing. For example, an Attach( ) that fails because the specified attach window is not visible or an Open( ) that attempts to open a non-existent file. Errors are trapped using the command:
On Error Call <Error Handling Routine>
where:
<Error Handling Routine> Is a user-defined function that takes no parameters. When an error occurs, the error handling routine is called.

The error handler can determine the error code by examining the Err system variable. The system variables ErrMsg, ErrFile, and ErrLine contain a description of the error, the name of the script generating the error, and the line number within that script.
Each function may have its own error handling routine. Error routines activated in subfunctions are nested. When an error occurs, the most active routine is called first. If this does not handle the error, the previous error routine is called.
The variant On Error End disables the currently active error handling routine; the previous error routine becomes the active routine.
The error routine remains active until the function terminates or On Error End is processed within the script. An error handling routine can perform one of the following operations:

- Correct the error and resume the script at the next instruction (Resume Next).
- Pass the error to the previous error routine (Error).
- Retry the operation that failed (Resume 0).

Examples

Example 1:

```
Function main
On Error Call MyErrorRoutine ; set error handler
Run "abc1"
On Error End ; disable error handler
Run "abc2"
End Function ; main
; disable runtime errors in script "abc1", but not in abc2.
Function MyErrorRoutine
Resume Next
End Function ; MyErrorRoutine
```

Example 2:

```
Function Main
MsgBox( "", "Starting Function Main" )
On Error Call Main_Error_Trap ; set error handler
Call Function_A ; call another function
MsgBox( "", “Ending Function Main" )
End Function ; Main
Function Function_A
MsgBox( "A", "Entering Function A" )
On Error Call Function_A_Errors ; set error handler
Call Function_B ; call another function
Attach "Not_there" ; generate error
MsgBox( "A", "Leaving Function A" )
End Function ; Function_A
Function Function_B
MsgBox( "B", "Entering Function B" )
On Error Call Function_B_Errors ; set error handler
ReadLine( "c:\notthere.dat" , ret ) ; generate error
MsgBox( "B", "Leaving Function B" )
End Function ; Function_B
Function Main_Error_Trap ; error handler for Main
MsgBox( "Main_Error_Trap", (Str(Err) + " " + ErrMsg) )
End Function ; Main_Error_Trap
Function Function_A_Errors ; error handler for Function_A
MsgBox( "Function_A_Errors", (Str(Err) + " " + ErrMsg) )
End Function ; Function_A_Errors
Function Function_B_Errors ; error handler for Function_B
MsgBox( "Function_B_Errors", (Str(Err) + " " + ErrMsg) )
End Function ; Function_B_Errors
```

Example 3:

```
Function Global_Error_Trap ; error handler for all
errors
MyErrCode = Err ; save error code
Resume Next ; and return to script
```

```
End Function ; Global_Error_Trap
Function Main
On Error Call Global_Error_Trap ; set error handler
< Instructions >
ReadLine "c:\names.dat" ret ; read line from file
If MyErrCode = 10504 ; if read error
CopyFile "c:\names.bak" "c:\names.dat"; use backup
Endif
< Instructions >
End Function ; Main
```

Example 4:

```
Function Error_Trap ; error handler
Resume 0 ; retry failed operation
End Function ; Error_Trap
Function Main
On Error Call Error_Trap ; set error handler
< Instructions >
Attach "MyAppWindow" ; retry until successful
< Instructions >
End Function ; Main
```

Open( )

File Access

Opens a file for reading and/or writing.

Syntax

```
ret = Open( "filename", "options" )
```

See Also

Read( ), Write( )

Operation

This function opens file for processing. If the file does not exist, it can be created using the create or write parameters.

The options are:

read Opens the file for reading. Other processes can also read from and write to the file.

write Opens the file for writing. The file is created if it does not exist. Other processes can also read and write to the file.

readwrite Opens the file for reading and writing. In this mode the FilePos( ) function must be executed between reading and writing.

create Forces creation of the file and opens it for writing. If the file exists, its current contents are erased.

shared Allows other processes to open the file for reading/writing (default).

lockread Prevents other processes reading the file.

lockwrite Prevents other processes writing to the file.

lockreadwrite Prevents other processes reading from or writing to the file.

Binary Opens a file in read-only binary format.

If two or more options are specified, they are acted on in first to last order — one option in the list may, therefore, be overridden by another option that comes later in the list.

The function returns 1 if the file is opened successfully, and it returns 0 if it is not.

Examples

Example 1:

```
; open a file for reading
Open( "c:\autoexec.bat", "read" )
readline( "c:\autoexec.bat", line, )
```

Example 2:

```
; open a file for writing
Open( "c:\audit\progress.dat", "write" )
writeline( "c:\audit\progress.dat", "File written successfully" )
```

Example 3:
; erase contents of a file
Open("c:\audit\progress.dat", "create")
Example 4:
; open file for exclusive reading and writing - allow
; other processes write access
Open("c:\users.dat", "write lockreadwrite")
Example 5:
; open file for exclusive writing - allow other processes
; read access
Open("c:\users.dat", "read lockwrite")
Example 6:
; share mode overridden by exclusive read mode
ret = Open("c:\users.dat", "shared lockread")

OpenCom( )

Serial Communications
Opens a specified COM port on a PC.

Syntax

ret = OpenCom(Port, Baud, DatBits, Parity, StopBits)

See Also

CloseCom( ), PurgeCom( ), ReadCom( ), WriteCom( )

Operation

This function opens the specified COM port on the PC using the determined communications settings. Once open, data can be read from the COM port using the ReadCom command. Data can also be written to the COM port using the WriteCom command. After using the OpenCom command in the script, the CloseCom command should be used to close the COM port before the script finishes.
The options are:
Port A number from 1 - 255.
Baud The signalling rate of the communication channel. Acceptable values are 300, 600, 900, 1200, 2400, 4800, 9600, 2920.
DataBits 7 or 8.
Parity The extra bit added to a byte or word to reveal errors in storage or transmission. Acceptable values are 0 (none), 1 (odd), or 2 (even).
StopBits The extra "1" bits that follows the data and any parity bit. They mark the end of a unit of transmission . Acceptable values are 1, 2, or 3 (1.5 stop bits).
The function has the following return values:
1 Success
0 Failure
-1 Bad Port
-2 Bad baud
-3 Bad data bits
-4 Bad parity
-5 Bad stop bits

Examples

Var y[ ]
Var x[ ]
y[1] = 41
y[2] = 41
y[3] = 41
y[4] = 41
y[5] = 41
z = OpenCom(4, 9600, 8, 0, 1) ;open up COM port 4

```
z = PurgeCom ( 4 ) ;purge data in COM 4
z = WriteCom( 4, y, 5 ) ;writes 5 bytes of data to COM 4
z = ReadCom ( 4, x, 5, 1 ) ;reads back 5 bytes of data
z = CloseCom( 4 ) ;Close COM port 4
Print x[1]
Print x[2]
Print x[3]
Print x[4]
Print x[5]
```

Operators

Language
Performs operations on expressions.

Syntax

expression1 operator expression2

See Also

Boolean Expressions

Operation

Operators (which are usually represented by single symbols) perform operations on two or more expressions. The action of the operator depends on the type of expression it is operating on. The operators are:

Operator Description

+ Add expression1 to expression2. If expression1 and expression2 are strings, their values are concatenated. If both expressions are numeric, they are added together. If one expression is numeric and the other is a string, the result is determined by the type of expression on the left.

- Subtract expression2 from expression1. Both sides of the expression are cast to numeric values and the subtraction is performed. The result is always a numeric value. The "-" operator can also negate an expression.

* Multiply expression1 by expression2. Both sides of the expression are cast to numeric values and then multiplied. The result is always a numeric value.

/ Divides expression1 by expression2. Both sides of the expression are cast to numeric values and then divided. The result is always a numeric value

% The remainder after dividing the integer part of expression1 by the integer part of expression2. Both sides of the expression are cast to numeric values and then divided. The result is always an integer value

& Performs a bit-wise AND of expression1 and expression2. Both sides of the expression are cast to 32-bit unsigned integers before the operation and the result is a 32-bit unsigned integer.

| Performs a bit-wise OR of expression1 and expression2. Both sides of the expression are cast to 32-bit unsigned integers before the operation and the result is a 32-bit unsigned integer.

^ Performs a bit-wise NOT of expression1 and expression2. Both sides of the expression are cast to 32-bit unsigned integers before the operation and the result is a 32-bit unsigned integer.

~ Performs a bit-wise NOT of an expression. The expression is cast to a 32-bit unsigned integer before the operation. The result is a 32-bit unsigned integer.

<< Performs a bit-wise left shift. The bits in expression1 are shifted left by the numeric value of expression2. Both sides of the expression are cast to 32-bit unsigned integers before the operation and the result is a 32-bit unsigned integer.
>> Performs a bit-wise right shift. The bits in expression1 are shifted right by the numeric value of expression2. Both sides of the expression are cast to 32-bit unsigned integers before the operation and the result is a 32-bit unsigned integer.
And Returns True if both expression1 and expression2 are true, otherwise returns False.
Or Returns True if either expression1 or expression2 are true, otherwise returns False.
Not Negates an expression. Returns true if the expression is false and false if the expression is true.

Examples

"Hello " + "World" ; result is "Hello World"
3 + 7 ; result is 10
3 + “7” ; result is 10
“3” + 7 ; result is "37"
6 - 1 ; result is 5
12 - "8" ; result is 4
"10" - "3" ; result is 7
- "12.00" ; result is -12
-(-4) ; result is 4
6 * 2 ; result is 12
3 * "2" ; result is 6
"10" * "3" ; result is 30
6 / 2 ; result is 3
3 / "2" ; result is 1.5
"12" / "1.5" ; result is 8
12 % 5 ; result is 2
"12.5" % 5.5 ; result is 2 (12 % 5)
"101" % "12.4" ; result is 5 (101 % 12)
7 & 3 ; result is 3 (0111 & 0011 = 0011)
7 & "4" ; result is 4 (0111 & 0100 = 0100)
21 & 10.5 ; result is 0 (10101 & 01010 = 00000)
7 | 3 ; result is 7 (0111 | 0011 = 0111)
7 | "4" ; result is 7 (0111 | 0100 = 0111)
21 | 10.5 ; result is 31 (10101 | 01010 = 11111)
7 ^ 3 ; result is 4 (0111 | 0011 = 0100)
7 ^ "4" ; result is 3 (0111 | 0100 = 0011)
21 ^ 10.5 ; result is 31 (10101 | 01010 = 11111)
~0xFFFF0000 ; result is 0xFFFF
~0x0000ABCD ; result is 0xFFFF5432
2 << 2 ; result is 8 (0010 << 2 = 1000)
1 << 8 ; result is 256 (1 << 8 = 100000000)
8 >> 1 ; result is 4 (1000 >> 1 = 0100)
260 >> 7 ; result is 2 (100000100 >> 7 = 10)
true And true ; returns true
true And false ; returns false
false And true ; returns false
false And false ; returns false
true Or true ; returns true
true Or false ; returns true
false Or true ; returns true
false Or false ; returns false
Not true ; returns false
Not false ; returns true

OverlayStr( )

String Manipulation
Overlays one string onto another at a given position.

Syntax
OverlayStr(target, newval, start)

Variants
OverlayStr(target, newval)

See Also
ReplaceStr( ), InsertStr( )

Operation
This function overlays characters contained in the target variable with characters contained in the newval variable.
The parameters are:
target The string variable containing the characters to be overlaid.
newval The characters to overlay; these can be literal or contained in another variable.
start Where to start the overlay; if omitted, the default is the first character.
The function returns 1 if the overlay is successful, and it returns 0 if it is not.

Examples
target = "the quick brown fox"
OverlayStr(target, "green", 11) ; Result "the quick green fox"
target = "the quick brown fox"
OverlayStr(target, "red", 11) ; Result "the quick redwn fox"
target = "the quick brown fox"
OverlayStr(target, "red") ; Result "red quick brown fox"

PadStr( )

String Manipulation
Pads a string with spaces or a specific character.

Syntax
ret = PadStr(string, length, char, options)

Variants
ret = PadStr(string, length)
ret = PadString(string, length, char, options)
ret = PadString(string, length)

Operation
This function is used to make string the length specified by length by padding it with the characters specified by char. By default, string is padded with spaces but other characters can be used.
The parameters are as follows:
string The variable to pad.
length The length to pad the string to.
char The character to pad the string with (default is space).
left Align the string to the left (default).
right Align the string to the right.
center Center the string in the field.
The position "options" can be abbreviated to their first letter only (l, r, or c).

Examples
target = "Hello"
ret = PadStr(target, 11) ; Result is "Hello "
ret = PadStr(target, 11, "-") ; Result is "Hello------"
ret = PadStr(target, 11, "-", "r") ; Result is "------Hello"
ret = PadStr(target, 11, "-", "c") ; Result is "---Hello---"

Pause( )

Synchronization
Pauses the current script for a specified length of time.

Syntax

Pause(LengthOfTime, "Units")

See Also

Replay.PauseMode, Sleep( )

Operation

This function causes the script to pause for the period of time specified by the LengthOfTime parameter. Units may be specified in seconds (the default), ticks (10ths of a second), or ms (milliseconds).
Pause statements are automatically inserted into the script during Learn if the Pause Threshold in Options\Configure**Learn** is set to a value of 1 or more.
Pause statements in the script can be ignored during replay by setting the system variable Replay.PauseMode = 0.

Examples

```
; pause the script for ten seconds
Pause 10 "Seconds"
Pause 10 "Secs"
Pause 10
; pause the script for half a second
Pause 5 "Ticks"
; pause the script for half a second
Pause 500, 'ms'
```

PictureCtrl( )

4GL Commands
Presses a UNIFACE or NS-DK picture control.

Syntax

ret = PictureCtrl("ControlId", "Options")

Variant

PictureCtrl("ControlId", "Options")
PictureCtrl "ControlId", "Options"

See Also

Button( ), HotspotCtrl( ), LabelCtrl( )

Operation

This command processes a UNIFACE picture control in the attached window. The control has Windows Class of UniPict. The action to be performed is specified in "Options".
The parameters are as follows:
"ControlId" Specifies the picture logical Object Name.
"Options" The "options" are as follows:
"Left" Use the left mouse button.
"Right" Use the right mouse button.
"Middle" Use the middle mouse button.
"Down" Press the mouse down.
"Up" Release the mouse button.
"singleclick" Single-click the mouse button.
"doubleclick" Double-click the mouse button.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with "control" and "shift".
When the command is generated by Learn, the parentheses are omitted.

Examples
; If the picture at the bottom-left corner of the
; Define Entity screen is pressed in UNIFACE 6.1,
; the following is learned:
PictureCtrl "WKB.BUTBX.STANDARD", 'Left SingleClick'

PopUpMenuSelect( )

Menu Control
Selects a menu item from a pop-up menu.
Syntax
ret = PopUpMenuSelect("MenuItem")
Variants
PopUpMenuSelect("MenuText")
PopUpMenuSelect(NumericID)
See Also
SysMenuSelect( ), MenuSelect( )
Operation
This function selects an item from the currently open pop-up menu. The selection can be a string or numeric ID.
The function returns 1 if the selection is successful, and it returns 0 if it is not.
Examples
Example 1:
; select Line Up Icons from the desktop pop up menu
Attach "~P~EXPLORER.EXE~SHELLDLL_DefView~Program Manager"
PopupMenuSelect "Lin&e up Icons"
Example 2:
Attach "~P~EXPLORER.EXE~SHELLDLL_DefView~Program Manager"
PopupMenuSelect 0x7032

Print( )

Miscellaneous
Sends output to the ViewPort window
Syntax
Print("Item1", [Item2…ItemN]")
See Also
ViewPortClear( )
Operation
This command sends Item1…ItemN to the ViewPort window. Each Print( ) starts on a new line.
You must first open the ViewPort window. The ViewPort window is opened by selecting View**Output** from the Editor's menu. The ViewPort window can be cleared using the ViewPortClear( ) command.
This function has no return value.
Examples
Example 1:
For I=1 to 100
Print(I)
If I % 10 = 0
ViewPortClear( )
Pause 1
Endif
Next
Example 2:
Print "Name = ", Name, "Date of Birth = " , dob

PromptBox( )

Miscellaneous
Defines a dialog box that requires user input.

Syntax

ret = PromptBox("Title", "Prompt", Value)

Variants

ret = PromptBox("Title", "Prompt", Value, x, y)

See Also

MessageBox( ), Dialog( )

Operation

This function generates a simple dialog box containing an edit control. The "Title" parameter sets the title of the box, "Prompt" defines static text that is displayed to the left of the edit control, and "Value" takes the value entered by the user. This can be set to a default value prior to the PromptBox definition.
The x, y position specifies the X- and Y-coordinates of the top-left corner of the box. If not specified, the prompt box appears in the center of the screen.
The prompt box also contains two buttons that have the following return values:

Button Selected Return Value

OK 1
Cancel 0

Examples

```
; set up default value
value = "NONE"
; display PromptBox
ret = PromptBox( "Sales Discount", "Enter Code", value )
; generates the following PromptBox:
; process the button selection
if ret = 0 ; Cancel button selected
<Instructions>
else ; OK button selected
; Process user input
if value = "NONE" ; default value
<Instructions>
else ; new value
<Instructions>
endif
endif
```

Public

Language
Declares public variables.

Syntax

Public Variable1 [, Variable2, …, VariableN]

Variants

Public Variable1[ ] [, Variable2[ ], …, VariableN[ ]]

See Also

Arrays, Const, Var

Operation

Numeric and string variables can be public, private, or local. The Public statement is used to declare variables as public. Variables declared as public can be accessed by all child scripts executed using the Run( ) function. For a child script to see the public variable, it must also declare the variable as public. Public variables cannot be declared within function definitions.
All string variables are initially assigned to the null string (""), and all numeric variables are initially 0 (zero). The maximum number of public variables is 4096.

The maximum length of a string variable is only limited by available memory.

Examples

Example 1:

```
Public a, ret, c ; declare public variables
Function Main
Setup ; call setup process
MessageBox( "a is" a ) ; a is initialized here
call "Child" ; run child script
End Function
Function Setup
a = 10 ; initializes a
End Function
; script "Child"
Public a, ret, c ; declare public variables
Function Main
MessageBox( "a is" a ) ; a is initialized here too
End Function
```

Example 2:

```
; "master" script
Public globala[ ], globalb[ ] ; declaration public arrays
Function Main
FillArray( globala, "c:\*.bat" )
Run "Child" ; run child script
End Function
; "child" script
Public globala[ ], globalb[ ] ; declaration public arrays
Function Main
c = 1
While c < ArraySize( globala )
MessageBox( "", globala[c] ) ; display values here
c=c+1
EndWhile
End Function
```

PurgeCom( )

Serial Communications

Purges any existing inbound or outbound date that is currently queued to the PC's specified COM port.

Syntax

```
ret = PurgeCom( Port )
```

See Also

CloseCom( ), OpenCom( ), ReadCom( ), WriteCom( )

Operation

This function purges any existing inbound or outbound data that is queued to the specified COM port on the PC.

The options are:

Port A number from 1 - 255.

The function has the following return values:

1 Success

0 Failure

-1 Bad Port

Examples

```
Var y[ ]
Var x[ ]
y[1] = 41
y[2] = 41
y[3] = 41
y[4] = 41
```

```
y[5] = 41
z = OpenCom( 4, 9600, 8, 0, 1 ) ;open up COM port 4
z = PurgeCom ( 4 ) ;purge data in COM 4
z = WriteCom( 4, y, 5 ) ;writes 5 bytes of data to COM 4
z = ReadCom ( 4, x, 5, 1 ) ;reads back 5 bytes of data
z = CloseCom( 4 ) ;Close COM port 4
Print x[1]
Print x[2]
Print x[3]
Print x[4]
Print x[5]
```

RadioButton( )

Dialog Control
Processes a radio button control.

Syntax

```
ret = RadioButton("ControlId", "Options" [, x, y ] )
```

See Also

Button( ), CheckBox( ), ComboBox( ), ComboText( ), EditText( ), ListBox( ), ScrollBar( )

Operation

This function processes a radio button contained within the currently attached dialog box. The action taken is determined by the "Options" parameter.
The parameters are:
"ControlId" Specifies the control label shown to the side of the radio button or the index value of the control (such as RadioButton "~1").
"Options" The "options" are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the radiobutton.
"singleclick" Click the radiobutton once.
"control" Press the control key before the radiobutton.
"shift" Press the shift key before the radiobutton.
"with" Use in conjunction with "control" and "shift".
x , y These optional parameters specify where on the control the mouse button will be clicked.
When this command is generated by the Learn facility, the parentheses are omitted.
The function returns 1 if the button is processed successfully, and it returns 0 if it does not.

Examples

```
; from the communications settings dialog, select the correct
; baud rate for the modem
Attach "~N~KERNEL32.DLL~#32770~Configure Session"
RadioButton "19200", "SingleClick"
```

Random( )

Number Manipulation
Generates a random number between two values.

Syntax

```
ret = Random( minval, maxval )
```

Variants

```
ret = Random( maxval )
ret = Random( )
```

See Also
RandomSeed( )
Operation
This function returns a random integer value between minval and maxval (minval and maxval must be in the range -32767 to +32768). If a single value is specified, it is assumed to be the maxval, and a number between 0 and maxval is returned. If neither minval nor maxval are specified, a number between 0 and 32768 is returned.
Examples
ret = Random(100, 200) ; a random integer between 100 and 200
ret = Random(-100, 200); a random integer >= -100 and <= 200
ret = Random(123.456) ; a random integer between 0 and 123
ret = Random( ) ; a random integer between 0 and 32767

RandomSeed( )

Number Manipulation
Seeds the random number generation function.
Syntax
RandomSeed(value)
See Also
Random( )
Operation
This function instigates one of 65535 repeatable sequences of pseudo-random numbers from the Random( ) function. If RandomSeed( ) is not reset each time a program is run, Random( ) returns the same sequence of random numbers.
Examples
RandomSeed(1) ; set seed to 1
ret = Random( ) ; generates 41
RandomSeed(secs( )) ; seed between 0 and 59
ret = Random( ) ; generates one of 60 possible numbers

Read( )

File Access
Reads a number of characters from a file.
Syntax
ret = Read("filename", "string", "len")
See Also
FilePos( ), Open( ), ReadIni( ), ReadLine( ), Write( ), WriteLine( )
Operation
This function reads "len" characters from "filename" (starting from FilePos( )) into "string". If the end of file is encountered before len characters are read, the characters up to the end of file are returned. If filename has not been opened with the Open( ) function, the Read( ) function opens it for reading. The value of FilePos( ) is updated following the Read. The parameters are as follows:
filename The file to read from.
string The string to read into.
len The number of characters to read.
The function returns the number of bytes read. This may be less than len if the end of file is reached.
Examples
Example 1:
; read five fixed length records from a file
var name[] ; declare array for names
c = 1 ; initialize a counter
repeat
Read("c:\data\names.dat", name[c], 9)

```
c = c+1
until c = 6
```
Example 2:
```
num = Read( "c:\myfile.txt", ret, 1000000 )
print "Number of bytes in file: ", num
```

ReadCom( )

Serial Communications
Reads a specific number of bytes from the PC's open COM port directly into a data array.

Syntax
```
ret = ReadCom( Port, DataArray, NumberOfBytes, Timeout )
```
See Also
CloseCom( ), OpenCom( ), PurgeCom( ), WriteCom( )
Operation
This function reads the specified number of bytes from the PC's open COM port into the specified data array. If the specified number of bytes do not arrive at the COM port within the assigned timeout period, then the function returns the number of bytes actually read.
The options are:
Port A number from 1 - 255.
DataArray The name of the data array where to bytes are read to.
Number of BytesThe number of bytes to be read.
TimeOut The amount of time in seconds for the bytes to arrive at the COM port.
The function has the following return values:
n Number of bytes read
0 Failure
-1 Bad Port
Examples
```
Var y[ ]
Var x[ ]
y[1] = 41
y[2] = 41
y[3] = 41
y[4] = 41
y[5] = 41
z = OpenCom( 4, 9600, 8, 0, 1 ) ;open up COM port 4
z = PurgeCom ( 4 ) ;purge data in COM 4
z = WriteCom( 4, y, 5 ) ;writes 5 bytes of data to COM 4
z = ReadCom ( 4, x, 5, 1 ) ;reads back 5 bytes of data
z = CloseCom( 4 ) ;Close COM port 4
Print x[1]
Print x[2]
Print x[3]
Print x[4]
Print x[5]
```

Readini( )

File Access
Returns a value from an INI file.

Syntax
```
ret = Readini( "inifile", "section", "key", "defaultval" )
```
See Also
WriteIni( )
Operation
This function gets the value of a specified item from an .INI file.
The parameters are as follows:

inifile The .INI file to read from.
section The section in the .INI file where the information is located.
key The item to read.
defaultval The default value of the key being read.

Examples

```
; read the [boot] section of system.ini
; return the value of the screen saver item
ret = Readini( "c:\windows\system.ini", "boot", "SCRNSAVE.EXE", "" )
MsgBox( "Result", ret ) ; display the result
```

ReadLine( )

File Access
Reads a line from a file.

Syntax

```
ret = ReadLine( "filename", string, "delimiter" )
```

Variants

```
ret = ReadLine( "filename", string )
```

See Also

FilePos( ), Open( ), Read( ), ReadIni( ), WriteLine( )

Note

When reading from a Unicode file in Windows NT4, there must be a comment at the beginning of the ini file. If there isn't one, one must be added, as in the following example:

; Do not delete this comment

Operation

This function reads the next line from "filename" into string. A line is defined as all characters up to, but not including, "delimiter". The default value for “delimiter” is Carriage Return/Line Feed.
Following the read, the file pointer is positioned at the character following "delimiter" and FilePos( ) is updated accordingly.
If filename has not been opened with the Open( ) function, the ReadLine( ) function opens it for reading.
The parameters are as follows:
"filename" The file to read from.
string The string to hold the information read.
"delimiter" The character(s) to read to (default is Carriage Return/Line Feed).
The function returns 1 if the file is read successfully, and it returns 0 if it is not.

Examples

Example 1:

```
; read the config.sys file a line at a time
Do
ret = ReadLine( "c:\config.sys", nextline )
MsgBox( "Next Line", nextline ) ; display the result
Loop While ret <> 0
```

Example 2:

```
; read values from a comma separated variable (CSV) file
filename = "c:\data\data.csv" ; file containing names & addresses
Do
ret = ReadLine( filename, lastname, "," ) ; read last name
if ret = 0 ; if end of file
Break ; exit the loop
else ; otherwise
ReadLine( filename, initial, "," ) ; read initial
ReadLine( filename, address1, "," ) ; read first address line
ReadLine( filename, address2, "," ) ; read second address line
ReadLine( filename, address3, "," ) ; read third address line
ReadLine( filename, address4 ) ; read last address line
```

< Instructions > ; process the data
Endif
Loop While 1 = 1 ; endless loop

RemoveDir( )

File Access
Removes a directory or folder at the specified path.

Syntax

RemoveDir("path")

Variants

RmDir("path")

Note

Block comments are not displayed in the color defined for Comments within the editor.

See Also

MakeDir( )

Operation

This function removes a directory (folder) at the path specified. The function returns 1 if the function is successful, and 0 if it is not.
A directory must be empty before it can be removed.

Examples

; remove a directory (folder)
RemoveDir("c:\Bob's Working Folder")

RenameFile( )

File Access
Renames a file.

Syntax

ret = RenameFile("oldfilename", "newfilename")

Variants

ret = Rename("oldfilename", "newfilename")

See Also

DeleteFile( ), IsFile( ), FileExists( ), Create( )

Operation

This function renames "oldfilename" to "newfilename". Either parameter can be a literal or variable string value. Wildcard characters are accepted.
The function returns 1 if the operation is successful, and it returns 0 if it is not.

Examples

Example 1:

; set up string contents
source = "c:\data\newdata.dat"
target = "c:\backup\daily.sav"
; check that the source file exists
if FileExists(source) = 1
; check for the target file also
if FileExists(target) = 1
; and delete it
deleteFile(target)
endif
; rename the source file to the target file
RenameFile(source, target)
endif

Example 2:

Rename("c:*.dat", "c:*.bak") ; back-up all data files

RepeatStr( )

String Manipulation
Creates a string consisting of another repeated string.

Syntax

ret = RepeatStr(source, count)

Operation

This function returns a string consisting of the source string repeated count times. Source can be a variable or literal value.

Examples

source = "Hello "
ret = RepeatStr(source, 3) ; Result "Hello Hello Hello "
ret = RepeatStr("New York ", 2) ; Result "New York New York "

Repeat...Until

Program Flow
Repeats a series of instructions until a condition is true.

Syntax

Repeat
<Instructions>
Until <Boolean Expression>

See Also

Break, Continue, Do...Loop While, While...Wend

Operation

This command executes the instructions between the Repeat and Until statements repeatedly until <Boolean Expression> is true. Execution of the script then continues on the statement following the Until. The <Boolean Expression> can contain literals or variables, including return values from functions.
The command is similar to the Do...Loop While structure, the difference being that Do...Loop While exits the loop when <Boolean Expression> is false, while Repeat...Until exits the loop when <Boolean Expression> is true.
Because <Boolean Expression> is evaluated after <Instructions> are executed, the loop always executes at least once.

Examples

Example 1:

```
i = 1
Repeat
MsgBox( "i is now", i )
i = i+1
Until i>5
```

Example 2:

```
Repeat
MsgBox( "Random Number", Random( ) )
Until MsgBox( "Run Again?", "Pick another number?", "yesno") <> 6
```

Example 3:

```
Repeat
text = Capture( "~P~KERNEL32.DLL~ReportWnd~PARTS.LST" )
ScrollBarWindow 1, "Page Vert"
Until FindStr( text, "More...") = 0
```

ReplaceStr( )

String Manipulation
Replaces characters within a string.

Syntax

ret = ReplaceStr(target, newval, start, length)

Variants

ret = ReplaceStr(target, "text", "findstr", count)

See Also

InsertStr( ), OverlayStr( )

Operation

This function replaces characters in a string. This function has two forms depending on the third parameter:
If the third parameter is numeric, it denotes the start position of the characters to be replaced. It can be followed by the number of characters to replace.
If the third parameter is a string, it can be followed by a number specifying how many occurrences of the string are to be replaced.
The parameters are as follows:
target The string variable containing the characters to be replaced.
newval The variable containing the replacement string.
start Number of characters from the beginning of the string in target to start replacement.
length How much of the string to replace. If omitted, the remainder of the string from the start position is replaced.
The parameters for the variant are as follows:
target The string variable containing the characters to be replaced.
"text" The replacement text. This can also be a variable.
"findstr" The text to be replaced.
count The number of instances matching “text” in the target to be replaced. If omitted, every occurrence is replaced.
The function returns the number of instances that have been changed.

Examples

target = "the quick brown fox"
ReplaceStr(target, "red", 11, 5) ; Result "the quick red fox"
target = "the quick brown fox"
ReplaceStr(target, "duck", 11) ; Result "the quick duck"

Variant Example

target = "the quick brown fox. the lazy fox. the foxy fox."
ReplaceStr(target, "duck", "fox", 2) ; Result "the quick brown duck. the lazy duck. the foxy fox"

Restore( )

Window Control
Restores the currently Attached window.

Syntax

ret = Restore("Attachname")

Variants

Restore( )

See Also

Maximize( ), Minimize( ), Move( ), Size( ), SetFocus( ), WinClose( )

Operation

This function restores the window specified by "Attachname" to its former size. If no parameter is specified, the currently attached window is restored.
With some applications, the attached window is not necessarily the parent. Often an edit control or some other child window is the attached window. In this case, the "Attachname" of the parent window must be used or the function will attempt to restore the edit control or other child window.
The function returns 1 if the window is restored successfully, and it returns 0 if it does not.

Examples

```
; attach to the top, parent, window
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
maximize( ) ; maximize the window
<Instructions> ; process further instructions
```

; restore the parent window, not the currently attached window
Restore("~N~KERNEL32.DLL~ThunderMDIForm~Address Book ")

Resume

Program Flow
Restarts execution of a suspended script.

Syntax

Resume

Variants

Resume

See Also

Cancel( ), MakeEvent( ), Suspend, Whenever

Operation

The Resume function restarts execution of a script that has been suspended with a Suspend. Resume can only be called from a Whenever...End Whenever.
Resume has no return value.

Examples

function begin
Resume ; resume script
end function
ret = MakeEvent("keyboard anywindow", "", "{F9}")
; key event
whenever ret call begin ; whenever key event, call begin
Suspend ; suspend script here
MsgBox("", "Script resumed..."); show script has resumed

Resume Next

Program Flow
Resumes execution of a script following an error.

Syntax

Resume Next

Variants

Resume 0

See Also

On Error

Operation

The Resume Next command returns from the current error handling routine and resumes execution of the script at the instruction following the point that caused the error.
Resume 0 returns from the current error handling routine and retries the instruction.

Examples

Example 1:
Function Global_Error_Trap ; error handler for all errors
Resume Next ; return to the script
End Function ; Global_Error_Trap
Function Main
On Error Call Global_Error_Trap ; set error handler
< Instructions >
ReadLine "c:\names.dat" ret ; read line from file
If Err = 10504 ; if read error
CopyFile "c:\names.bak" "c:\names.dat"; use backup
Endif
< Instructions >
End Function ; Main
Example 2:
Function OnErrrorRoutine ; error handling routine
If retry != 3 ; if not tried 3 times

```
Sleep 1000 ; pause a second
retry += 1 ; increment tries count
Resume 0 ; try again
Endif
End Function
```

Return

Program Flow
Returns from a function with an optional return value.

Syntax

```
Return "value"
```

Variants

Return

See Also

Function…End Function

Operation

This command returns from a user-defined function and sets its return value to "value".

Examples

```
userpw = Enter_Password( ) ; call the password function
If userpw = "Invalid Password" ; test its return value
Stop ; stop on error
Else ; otherwise
Type value ; type the password
Endif
Function Enter_Password( ):var ; define the password function
ret = PromptBox( "Logon", ; prompt for password
"Enter Password", value )
If ret = 1 ; if OK button clicked
Return value ; return the password entered
Else ; if Cancel button clicked
Return "Invalid Password"; return an error value
Endif
EndFunction
```

Reverse( )

String Manipulation
Reverses a string.

Syntax

```
Reverse( string_var )
```

Operation

This function reverses a string of characters. The original string variable is updated.

Examples

```
a = "0123456789"
Reverse( a ) ; a becomes "9876543210"
a = "The quick brown fox"
Reverse( a ) ; a becomes "xof nworb kciuq ehT"
```

RfindStr( )

String Manipulation
Returns the position of the last occurrence of one string within another.

Syntax

```
RfindStr( target, searchlist, [ startpos ] )
```

Variants

```
RfindStr( target, searchlist )
```

See Also
FindChar( ), FindStr( )
Operation
This function searches the target for occurrences of searchlist and returns the start position of the last occurrence found. The parameters are as follows:
target The string to search.
searchlist The value to search for.
startpos Starting point within the string. If omitted, the entire string is searched.
Examples
Example 1:
target = "aabbccddeeffaa" ; string to search
searchlist = "a" ; value to search for
ret = RfindStr(target, searchlist) ; result is 14 (the last "a")
Example 2:
target = "The Lord of The Rings"
searchword = "The"
ret = RfindStr(target, searchword) ; result is 13

Right( )

String Manipulation
Extracts a number of characters from the end of a string.
Syntax
Ret = Right(source, count)
See Also
Length( ), Mid( ), Left( )
Operation
This function returns the last count characters contained in source. The parameters are:
source The source string.
count The number of characters to extract from the end of source.
Examples
source = "This is a value" ; set up the variable
ret = Right(source, 10) ; take the last 10 characters
; returns "is a value"
ret = Right("Hello World", 5) ; returns "World"

RtrimStr( )

String Manipulation
Removes trailing spaces from a string.
Syntax
ret = RtrimStr(target)
See Also
LtrimStr( )
Operation
This function removes trailing spaces, tabs, carriage returns, and line feeds from a string. If a return value is not specified, the string is updated. If a return value is specified, the target string is unchanged.
Examples
Example 1:
target = "Hello "
target = RtrimStr(target) ; target becomes "Hello"
Example 2:
target = "Hello "
ret = RtrimStr(target) ; ret is "Hello"
; target remains "Hello "

Run( )

Program Flow
Runs another script from this script. This script is suspended until the other finishes.

Syntax
Run("scriptname" [, “parameters’])

Variants
ret = Run("scriptname")

See Also
Chain( ), CmdLine( )

Operation
This function runs the script specified by "scriptname". The calling script is suspended until "scriptname" has finished processing. Then it resumes from the line following the Run command.
All WhenEver statements declared in the calling script remain active. All variables and arrays declared as public in the calling script can be accessed by the subscript.
Files opened by the calling script are not inherited by the subscript. The "parameters" may be retrieved by the receiving script using the CmdLine( ) function.

Examples
<Instructions> ; process these instructions
<Instructions> ; process these instructions
<Instructions> ; process these instructions
Run("Account Update") ; suspend parent, run this
; child script
<Instructions> ; resume parent script here
<Instructions> ; process these instruction
Run("Invoice Create") ; suspend parent, run this
; child script
<Instructions> ; resume parent script etc.

ScrollBar( )

Dialog Control
Drives the slider controls of the currently attached window.

Syntax
ret = ScrollBar("ControlId", Position, "Options")

See Also
Control Labels, ScrollBarWindow( )

Operation
This function moves the slider control specified by "ControlId" to the position specified by Position, using the options specified by "Options".
The options are as follows:
Line Move the slider control "position" lines.
Page Move the slider control "position" pages.
Set Move the slider control to the position specified by "Position".
Top Move the slider control to the top (or the left on a horizontal slide control); the "position" parameter is ignored.
Bottom Move the slider control to the bottom (or the right on a horizontal slide control); the "position" parameter is ignored.
The function returns 1 if the operation is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, you must use the parentheses.

Examples
Attach "Mouse Properties Dialog - Buttons"

ScrollBar "Fast", 513, 'Set'
Attach "Mouse Properties Dialog - Motion"
ScrollBar "Fast", 4, 'Bottom'

ScrollBarPos( )

Window Information
Retrieves the position of a slider control.

Syntax

Pos = ScrollBarPos(hCtrl)

See Also

CtrlEnabled( ), ControlFind( ), CtrlFocus( ), IsWindow( )

Operation

This function retrieves the position of the slider on the track bar control with window handle hCtrl.
You can determine hCtrl from the ScrollBarFind( ) function.

Examples

```
Function SetScreenRes
Attach "Display Properties PopupWindow" ; attach to display
; properties dialog
hCtrl = ScrollBarFind( "&Desktop area" ); get handle of control
pos = ScrollBarPos( hCtrl ) ; position of slider
If pos <> 3 ; if wrong resolution
ScrollBar "&Desktop area", 3, ‘Set’ ; adjust setting
Endif
End Function ; SetScreenRes
```

ScrollBarWindow( )

Dialog Control
Drives the scrollbars of the currently attached window.

Syntax

ret = ScrollBarWindow(Position, "Options" Irange)

See Also

ScrollBar( )

Operation

This function moves the scrollbars of the currently attached window to the position specified by Position, using the options specified by "Options". There is also an optional parameter, Irange, that you can use if you will be conducting testing that requires cross-browser support.
"Options" The options are as follows:
Line Horz Move the horizontal scrollbar one or more lines at a time.
Line Vert Move the vertical scrollbar one or more lines at a time.
Page Horz Move the horizontal scrollbar one or more pages at a time.
Page Vert Move the vertical scrollbar one or more pages at a time.
Set Horz Move the horizontal scrollbar to the exact position specified by "Position".
Set Vert Move the vertical scrollbar to exact position specified by "Position".
Top Horz Move the scrollbar slider control to the left on a horizontal slide control; the "position" parameter is ignored.
Top Vert Move the scrollbar slider control to the top on a vertical

slide control; the "position" parameter is ignored.
Bottom Horz Move the scrollbar slider control to the right on a horizontal slide control; the "position" parameter is ignored.
Bottom Vert Move the scrollbar slider control to the bottom on a vertical slide control; the "position" parameter is ignored.
IRange Allows *EZ Test* to replay the same scrollbar position in supported versions of both Internet Explorer and Netscape. This is useful if you will be conducting cross-browser testing.
The function returns 1 if the operation is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, you must use the parentheses.

Examples

Example 1:

```
Attach "~P~EXPLORER.EXE~SysListView32~Browse"; attach to
; the Browse
ScrollBarWindow 1, "Line Vert"; dialog and move the
ScrollBarWindow 1, "Line Vert"; vertical scrollbar one
ScrollBarWindow 1, "Line Vert"; line at a time
```

Example 2:

```
Attach "~P~EXPLORER.EXE~SysListView32~Browse"; ; attach to the
; Browse
ScrollBarWindow 1, "Page Vert"; dialog and move the
ScrollBarWindow 1, "Page Vert"; vertical scrollbar one
ScrollBarWindow 1, "Page Vert"; page at a time
```

Example 3:

```
Attach "~P~EXPLORER.EXE~SysListView32~Browse"; scroll to 107th
; line
ScrollBarWindow 107, "Set Vert" ; in this list and then
ScrollBarWindow 1, "Set Vert" ; the first in the list
```

Example 4:

```
Attach "~P~EXPLORER.EXE~SysListView32~Browse"; move the vertical
; scroll
ScrollBarWindow -1, "Page Vert" ; bar back one page
```

Example 5:

```
Attach "demo.txt - Notepad MultiLineEdit~1" ; move the horizontal
ScrollBarWindow 78, 'Set Horz' ; scrollbar
ScrollBarWindow 1, 'Set Horz'
```

Example 6:

```
Attach "Customer Invoice ChildWindow~1"
ScrollBarWindow 27, 'Set Vert' ; move vertical and
ScrollBarWindow 78, 'Set Horz' ; horizontal bars
```

Secs( )

Date/Time
Returns the specified seconds.

Syntax

```
ret = Secs( timeval )
```

Variants

```
ret = Secs( )
```

See Also

TimeVal( ), CurTime( )

Operation

This function returns the seconds value specified by timeval.
The timeval is a time value that can be derived from the TimeVal( ) or CurTime( ) functions. If timeval is not specified, the current system time is used.

Examples

Example 1:
n = TimeVal(15, 11, 30) ; returns 54690
Seconds_of_Minute = Secs(n) ; returns 30
Example 2:
Seconds_of_Minute = Secs( ) ; current seconds

SendToEditor( )

Miscellaneous
Pastes text into a script in the editor.

Syntax

SendToEditor("text", "scriptname")

Operation

This function pastes "text" into the script "scriptname" at the position of the cursor.
If the specified script is not currently loaded, it is opened, and "text" is pasted at the top of the script. If "scriptname" does not exist, there is no action.
The SendToEditor( ) function is used to enable a running script to paste (often on a keyboard Whenever) common code or information about the target system into the current script, while learning.
The function has no return value.

Examples

```
; this example captures the screen title from "MyApp" and pastes a
; MakeEvent statement into current script ("MyScript") each time
; the developer presses {F12}
Function Main
Whenever "Paste" Call Paste
End Function ; Main
Function Paste
Title = CaptureBox( "MyApp", 0,140,1000,10 )
EventName = Left( Title, 10 )
MyPaste = EventName + ‘ = MakeEvent( "Screen", "MyApp", "‘ +
Title + ‘", "0, 140, 1000, 10" )’
SendToEditor( MyPaste, "MyScript")
SendToEditor( chr(13)+chr(10), "MyScript" )
SendToEditor( 'Wait(30, "', "' + EventName + '")', "MyScript" )
SendToEditor( chr(13)+chr(10), "MyScript" )
End Function ; Paste
```

SetDate( )

Date/Time
Sets the PC date.

Syntax

ret = SetDate(year , month , day)

See Also

CreateDate( ), SetTime( ), Replay.TodaysDate

Operation

This function uses year, month, and day values to set the current PC date to the values specified.
The function returns 1 if the date was set correctly, or it returns 0 if an invalid date was passed.

Examples

```
Function Main
; Set the date to the 1st of January 2000
ret = SetDate( 2000 , 1 , 1 )
; Set the date to the next year, current date
ret = SetDate( year() + 1 , month() , day )
```

```
; Attempts to set an invalid date
ret = SetDate( 1999 , 2 , 31 )
if ret = 0
msgbox( "Error" , "Invalid date supplied to SetDate()" )
endif
End Function ; main
```

SetFocus( )

Window Control
Sets focus to the specified window.

Syntax

```
ret = SetFocus( "AttachName" )
```

Variants

```
SetFocus( AttachID )
```

See Also

Attach( )

Operation

This function makes the window specified by "AttachName" the focus window. The "AttachName" parameter can be the window name or its window handle (AttachID). The function returns 1 if the set focus is successful, and it returns 0 if it is not.

Examples

Example 1:

```
appname= FocusName( ) ; check the application in focus
if appname<> "Address Book" ; if not Address Book, force SetFocus
SetFocus( "~N~KERNEL32.DLL~ThmdrFrm~Address Book Version 1.0" )
endif
```

Example 2:

```
ret = ActiveWindow( ) ; find the handle of window in focus
if ret <> 2036 ; if not the required application
SetFocus( 2036 ) ; set focus using a window handle
endif
```

SetStrLen( )

String Manipulation
Prepares a string to use in a DLL function.

Syntax

```
SetStrLen( string, length )
```

See Also

DLLFunc, PadStr( )

Operation

This function guarantees that a string has a buffer of a given size. It is used to allocate a buffer of a required size for calls to DLL functions that modify string parameters. The parameters are:

string The string to use.
length The minimum length of the string buffer.

The function has no return value.

Examples

```
; declare a DLLFunc to get the name of the computer
Declare DllFunc "int GetComputerNameA( str, ulong* ) kernel32" GetName
Function Main
; prepare a string to receive the name
len=100
SetStrLen( Name, len )
; call the DLL function
GetName( Name, len )
; display the result
```

```
MsgBox( "Computer Name", Name )
End Function ; Main
```

SetTime( )

Date/Time
Sets the PC's internal time

Syntax

```
ret = SetTime( hours , minutes , seconds )
```

Variants

```
ret = SetTime( hours , minutes , seconds , milliseconds )
```

See Also

CreateDate( ), SetDate( ), Replay.TodaysDate

Operation

This function uses hours, minutes, and seconds values to set the current PC time to the values specified.
The function returns 1 if the time was set correctly or 0 if an invalid time was passed.

Examples

```
; Set the time to midnight
ret = SetTime( 0 , 0 , 0 )
; Set the time to noon
ret = SetTime( 12 , 0 , 0 )
; Set the time with milliseconds (nearly 6 o'clock)
ret = SetTime( 17 , 59 , 59 , 500 )
; Attempts to set an invalid time
ret = SetTime( 25 , 64 , 44 )
if ret = 0
msgbox( "Error" , "Invalid time supplied to SetTime()" )
endif
```

Size( )

Window Control
Sizes the currently attached window.

Syntax

```
ret = Size( Width, Height )
```

See Also

Maximize( ), Minimize( ), Move( ), WinClose( ), SetFocus( )

Operation

This function sizes the currently attached window to the values specified by Width and Height. The Size is specified in pixels.
Negative numbers are converted to positive. Values greater than the screen size are converted to the maximum allowable value. When this command is generated using the Learn facility, the parentheses are omitted.
The function returns 1 if the window is sized successfully, and it returns 0 if it is not.

Examples

```
Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
; attach to the target application
Size 360, 441 ; and size the window
```

Sleep( )

Synchronization
Pauses the current script for a specified length of time.

Syntax

```
Sleep( LengthOfTime, "Units" )
```

See Also

Pause( ), Replay.PauseMode

Operation
This function causes the script to pause for the period of time specified by the LengthOfTime value. "Units" may be specified in seconds (the default), ticks (10ths of a second), or ms (milliseconds).
The command operates in the same way as Pause( ), except that a Sleep( ) is not ignored during replay if the system variable Replay.PauseMode = 0.

Examples
Example 1:
Sleep 10 "Seconds" ; pause the script for ten seconds
Sleep 5 "Ticks" ; pause the script for half a second
Pause 500, 'ms' ; pause the script for half a second
Example 2:
Replay.PauseMode = 0 ; ignore pauses in the script
Beep
Pause 5 ; ignore this pause
Beep
Sleep 5 ; but do not ignore this pause
Beep

SplitPath( )
String Manipulation
Returns part of a path string.

Syntax
ret = SplitPath(pathstring, options)

Operation
Returns part of the path, depending on the option set. The options are as follows:
path Returns the path name, including a trailing "\", but excluding the filename.
file Returns the filename without the path name.
base Returns the filename without the path name or file extension.
ext Returns only the extension part of the filename.
noext Returns the path and filename without the extension.
nobase Returns the path name without trailing "\" or filename.

Examples
filename = "C:\windows\system.ini"
SplitPath(filename, "path") ; result "C:\windows\"
SplitPath(filename, "file") ; result "system.ini"
SplitPath(filename, "base") ; result "system"
SplitPath(filename, "ext") ; result "ini"
SplitPath(filename, "noext") ; result "C:\windows\system"
SplitPath(filename, "nobase") ; result "C:\windows"

Sqr( )
Number Manipulation
Returns the square root of a number.

Syntax
ret = Sqr(value)

Operation
This function returns the square root of value. If value is negative, the function returns -1.

Examples
ret = Sqr(9) ; returns 3
ret = Sqr(123.456) ; returns 11.11107555549867
ret = Sqr(-123.456) ; error, returns -1

Stop
Program Flow

Stops the current script and all its parents.

Syntax

Stop

See Also

Chain( ), Exit( ), ExitWindows( ), Fatal( ), Run( )

Operation

This function causes the current script and all its parent scripts (if any) to stop. Scripts that have been chained are not stopped.

Examples

Stop ; stop this script and all its parents

Str( )

String Manipulation

Converts a number into its string equivalent.

Syntax

ret = Str(num)

See Also

Val( )

Operation

This function converts a number into a string of characters.

Examples

x = Str(123) ; returns "123"
Value = Str(17/6) ; returns "2.833333"
y = -12.64
y = Str(y) ; returns "-12.64"

StrCat( )

String Manipulation

Concatenates strings (with an optional separator).

Syntax

ret = StrCat(separator, target1, target2 [, ...targetN])

Operation

This function concatenates two or more variables. The separator parameter is used to specify how the strings are separated.

Examples

t1 = "The"
t2 = "quick"
t3 = "brown"
t4 = "fox"
ret = StrCat(" ", t1, t2, t3, t4); result "The quick brown fox"
ret = StrCat("-", t1, t2, t3, t4); result "The-quick-brown-fox"

SubStr( )

String Manipulation

Returns part of a string.

Syntax

ret = SubStr(target, startpos, endpos)

Variants

ret = SubStr(target, startpos)

See Also

Mid( )

Operation

This function returns a portion of the target string, starting from startpos and ending at endpos. If endpos is not specified, all the characters from startpos to the end of

target are returned.
The parameters are as follows:
target The variable containing the characters to be extracted. This can be a literal or variable value.
Startpos The position of the first character to extract (offset from 0).
endpos The position of the last character to extract (offset from 0). If omitted, the end of string is assumed.

Examples

target = "the quick brown fox"
ret = SubStr(target, 4, 8) ; result "quick
ret = SubStr(target, 4) ; result "quick brown fox"

Note

SubStr( ) is implemented to preserve compatibility with previous versions of *EZ Test*. In this function, counting of characters starts from 0. It is superseded by the Mid( ) function, in which characters are counted from 1.

Suspend

Program Flow
Suspends the current script, leaving Whenevers active.

Syntax

Suspend

Variants

Suspend

See Also

Exit( ), ExitWindows( ), Fatal( ), Resume, Stop

Operation

This function causes the current script to suspend. “Whenevers” remain active. A suspended script can be restarted with the Resume function.

Examples

Suspend ; suspend this script

Switch...End Switch

Program Flow
Creates a Case statement to switch on a value.

Syntax

Switch <Variable>
Case <Value 1>
<Instructions>
Case <Value n>
<Instructions>
Default
<Default Instructions>
End Switch

See Also

If...Else...EndIf

Operation

This command performs a set of <Instructions> based upon the <Value> of <Variable>. The <Variable> can be a string or a numeric value. The <Value> can be literal or variable. If there is no match for <Variable>, the instructions following the Default case are processed. The Default case is mandatory.
Each Switch statement must end with an End Switch statement.

Examples

; run a different routine depending on the time of day.
; hours 2, 3 and 4 PM have special processing - all others are
; processed by DEFAULT
Switch hours( )

Case 14 ; if it is 2PM
<Process 2PM instructions>
Case 15 ; if it is 3PM
<Process 3PM instructions>
Case 16 ; if it is 4PM
<Process 4PM instructions>
Default
<Otherwise process these instructions>
end switch

SysMenuSelect( )

Menu Control
Selects an item from the system menu of the currently attached window.

Syntax

ret = SysMenuSelect("SysMenuItem")

Variants

SysMenuSelect(NumericID)

See Also

MenuSelect( )

Operation

This function selects the menu item specified by "SysMenuItem" from the system menu of the currently attached window. The "SysMenuItem" parameter can be numeric or a string.
The function returns 1 if the menu selection is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is needed, you must use parentheses.

Examples

Example 1:

Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
; attach to target application
SysMenuSelect "&Close Alt+F4" ; and close it

Example 2:

Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
; as above, but
SysMenuSelect -4000 ; using numeric ID

Example 3:

Attach "~N~KERNEL32.DLL~ThunderMDIForm~Address Book"
ret = SysMenuSelect("&Close Alt+F4") ; return result to
MessageBox("Result", ret) ; check if it worked

SystemInfo( )

System Information
Retrieves system information.

Syntax

ret = SystemInfo("option")

See Also

WinVersion( )

Operation

This function retrieves the system information specified in the options parameter.
The "options" are:
"MemoryLoad" Returns the memory load as a percentage.
"MemoryFree" Returns the amount of available memory.
"MemoryTotalPhysical" Returns the total memory of the system.
"os" Returns a string value indicating which operating system is in use:
"Win32s" Windows 32s on 3.xx

"95" Windows 95
"98" Windows 98
"NT" Windows NT
"Win2K" Windows 2000
"WinXP" Windows XP
"Server 2003" Windows Server 2003
"Unknown Version" Unknown Version
Note that Windows 32s on 3.xx and Windows 95 are no longer supported.

Examples

ret = SystemInfo("MemoryLoad"); percentage memory load (approx.)
ret = SystemInfo("MemoryFree"); numeric value - memory available
ret = SystemInfo("os") ; string value such as "nt"

TabCtrl( )

Dialog Control
Selects a tab control in a dialog box.

Syntax

Ret = TabCtrl("ControlId", "Item", "Options" [, x, y])

Variants

TabCtrl("ControlId", "Item", "Options")

Operation

This function drives the tab options on a dialog box. The parameters are as follows:
"ControlId" The index value of the tab control.
"Item" The tab to select on the dialog box. This value can be literal or variable and selection can be by text or by position. To select the first tab use "@1" in place of the text value for "Item".
The "Options" are:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"double" Double-click the mouse button.
"click" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with "control" and "shift".
x The x-position relative to the top-left corner of the client area of the currently attached window.
y The y-position within the client area of currently attached window.
The function returns 1 if the selection is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, you must use parentheses.

Examples

Example 1:

```
; select the Settings tab from the Display Properties dialog
Attach "~N~RUNDLL32.EXE~#32770~Display Properties"
TabCtrl "~1", "Settings", "Left SingleClick"
; change the color resolution and font size
Attach "~N~RUNDLL32.EXE~#32770~Settings"
ComboBox "~1", "High Color (16 bit)", "Left SingleClick"
ComboBox "~2", "Large Fonts", "Left SingleClick"
```

Example 2:

```
; select the Screen Saver tab on the Display Properties dialog
Attach "~N~RUNDLL32.EXE~#32770~Display Properties"
TabCtrl "~1", "@2", "Left SingleClick"
```

TableColumns( )

Window Information
Returns the number of columns in a PowerBuilder or Java application's table.

Syntax

ret = TableColumns("Attachname")

See Also

TableRows( ), TableItem( ), TableSelect( )

Operation

This function returns the number of columns in a table control identified by the attachname.
If there are no columns in the table the function returns 0.
Note that this function only operates on supported tables, see the Release Notes for a list of supported tables.

Examples

```
; Count the number of columns and log it.
ret = TableColumns( "~P~VIEWER.EXE~Grid~Reports" )
if ret = 0
logComment "No columns in the table"
else
logcomment "The table has " + ret + " columns"
endif
```

TableItem( )

Window Information
Returns data from a cell within a PowerBuilder or Java application's table.

Syntax

ret = TableItem("Attachname" , row , column)

See Also

TableColumns( ), TableRows( ), TableSelect( )

Operation

This function returns the text of a cell within a table control identified by the attachname. The row and column parameters specify the cell within the table to get the data from. If the row and column numbers exceed the size of the table a null is returned.
Note that this function only operates on supported tables, see the Release Notes for a list of supported tables.

Examples

Example 1:

```
; Get the first item in the table
ret = TableItem( "~P~VIEWER.EXE~Grid~Reports" , 1 , 1 )
logcomment "The first item is " + ret
```

Example 2:

```
; Gets every item in the table
; Define the tables attach name
table = "~P~VIEWER.EXE~Grid~Reports"
; Get the size of the table
maxcols = TableColumns( table )
maxrows = TableRows( table )
; loop on the rows
for row = 1 to maxrows
; loop on the columns
for column = 1 to maxcols
; Get cell item
text = TableItem( table , row , column )
; Log to the log file
string = "R=" + row + ",C=" + column + ", Text=[" + text + "]"
logcomment string
next
```

next

TableRows( )

Window Information
Returns the number of rows in a PowerBuilder or Java application's table.

Syntax

ret = TableRows("Attachname")

See Also

TableColumns( ), TableItem( ), TableSelect( )

Operation

This function returns the number of rows in a table control identified by the attachname. If there are no rows in the table the function returns 0.

Examples

```
; Select the first item on the last row
ret = TableRows( "~P~VIEWER.EXE~Grid~Reports" )
if ret = 0
logcomment "No rows in the table"
else
;
; Select the last item in the table
;
TableSelect "~1" , ret , 1 , 'Left SingleClick'
endif
```

TableSelect( )

Dialog Control
Selects an item in a Java application's table control.

Syntax

ret = TableSelect("ControlId", row , column , "Options")

See Also

TableColumns( ), TableRows( ), TableItem( )

Operation

This function selects an item from a table control using the row and column specified.
"ControlId" Specifies the control id of the table control, e.g. "~1"
row and column These values can be alpha or numeric values. Depending on the way the item is to be selected the values can be specified in different ways.

Row and Column as numeric: When both are specified as numbers (greater than 0), the item selected refers to the absolute row and column number in the table. For example:

```
; Select cell row 23 column 7
TableSelect "~1" , 23 , 7 , 'Left SingleClick'
```

Row Numeric, Column Alpha: With this syntax, the row is searched for the text in the column parameter and the appropriate cell selected. For example:

```
; Select cell row 17 column with text "Housing"
TableSelect "~1" , 17 , "Housing" ,
'Left SingleClick'
```

Row Alpha, Column Numeric: With this syntax, the column is searched for the text in the row parameter and the appropriate cell selected. For example,

```
; Select cell row with text "ETA" on column 4
TableSelect "~1" , "ETA" , 4 , 'Left SingleClick'
```

Row Alpha, Column 0: With this syntax, the entire table is searched for the text in the row field starting at row 1 column 1, left to right, top to bottom. For example:

; Select cell with text "Car"
TableSelect "~1" , "Car" , 0 , 'Left SingleClick'
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the button.
"singleclick" Click the button once.
"control" Press the control key before clicking the button.
"shift" Press the shift key before clicking the button.
"with" Use in conjunction with "control" and
"shift".
If the TableSelect( ) command is used on a table that does not support cell selection, the command is generated with the row containing the text in the first column and the column number set to 1. This is regardless of the position where the table was clicked on during Learn.
The function returns 1 if the table item is successfully selected, and it returns 0 if the item is not successfully selected.
When this command is generated by the Learn facility, the parentheses are omitted.
This function only operates on supported tables, see the Release Notes for a list of supported tables.

TerminateApp( )

Synchronization
Terminates an application.

Syntax

ret = TerminateApp("appname.exe")

See Also

WinClose( )

Operation

This function terminates the application specified by "appname.exe". Use this function cautiously or data may be lost. It's best used to terminate applications that fail to respond. A 1 is returned if the application is terminated successfully, and 0 if the application could not be closed or was not running.

Examples

; close notepad
Attach "~N~NOTEPAD.EXE~Notepad~Untitled - Notepad"
WinClose
; check if it closed
ret = IsWindow("~P~NOTEPAD.EXE~Edit~Untitled - Notepad", "exists")
if ret = 1 ; if it's still running
TerminateApp("notepad.exe") ; force it to close
endif

TestData( )

TestData Handling

Note

When using the Command Wizard, all information entered for the row and column fields is entered WYSIWYG (this is what is meant by "Raw Data" in the Command Wizard). To depict a string (such as "ETA" in the Row Alpha, Column Numeric example), you need to enter ETA in quotes to specify that it is a string. To enter a variable, enter the variable name. No quotes are added.
Sets the current testdata file.

Syntax

TestData(TDFileName, FieldDelim, RecordDelim)

Variants

TestData = TDFileName
CurTDFile = TestData( )

See Also

TestDataIndex( ), TestDataField( )

Operation

This function opens the testdata file specified by TDFileName and positions the file pointer just before the first field in the first record — nominally at Record 0, Field 0. This enables the first record and field to be accessed using the {+.+} testdata expression. If no parameter is passed to the function, the name of the current testdata file is returned, and no change is made. The parameters are as follows:

FieldDelim Identifies the field delimiter used in the .csv file. If no FieldDelim is specified, commas are assumed.

RecordDelim Identifies the record delimiter used in the .csv file. If no RecordDelim is specified, a carriage return and line feed is assumed.

A testdata file is a comma separated variable (CSV) file in which each line is a record. Each record contains fields that are separated by commas (use the FieldDelim parameter to identify fields that are separated by single-character deliminators other than commas; for example, tabs). For example:

Example 1 (A Testdata File Containing 3 Records, Each with 5 Fields):

Tom,Jones,24,Software Development,4227
Dick,Tracy,36,Quality Assurance,1044
Harry,Hawk,52,Product Planning,2128

Example 2 (A Testdata File Containing 'm' Records, Each with 'n' Fields):

R1F1,R1F2,R1F3,…….R1Fn
R2F1,R2F2,R2F3,…….R2Fn
…
RmF1,RmF2,RmF3,…….RmFn

Fields containing commas may be included within a testdata file if each field is enclosed in double quotes:

American Systems Corp.,31440 Northwestern Highway,"Farmington Hills, MI"
American Systems Europe,"551, London Road","Isleworth, TW7 4DS"

Testdata files are indexed by *EZ Test* to ensure quick location of individual fields. If the testdata file does not have an index file, or the existing index file is older than the testdata source file, a new index file is created.

Testdata files should be “rectangular” — that is, each record should contain the same number of fields. Records that contain fewer data fields should be padded with blank fields or indexation will fail (the indexing process assumes that all records contain the same number of fields as the first record).

If no path is specified, the testdata file is assumed to be located in the directory containing the current *EZ Test* database.

Examples

Example 1:

```
TestData "MyData.csv" ; use the "MyData.csv" comma
; separated file in the EZ Test
; database directory.
TestData( "y:\data\invoice.dat" ) ; use "invoice.dat" CSV file in
; the "data" directory on "y:"
; drive
```

Example 2:

```
; Extracts the first field and record from a .csv file
; that uses a comma as the field delimiter.
;
Function Main
fielddelim = chr(44) ; identifies comma field delim
recorddelim = chr(13) + chr(10) ; carriage return and line feed
```

```
testdata ( "Customer.csv" , fielddelim, recorddelim )
ret=TestDataField( 1, 1 ) ; extract first field from
; the first record
msgbox ( "", ret ) ; display it in a message box
End Function ;Main
```

TestDataClose

TestData Handling

Note

There should be no spaces between the fields and the commas.

Closes the current testdata file and releases the handle of the corresponding index file.

Syntax

TestDataClose

See Also

TestData( )

Operation

This function closes the currently open testdata file. When the testdata file is closed, *EZ Test* releases the handle of both the testdata file and its corresponding index file.

Examples

```
Function Main
TestData( "CustChar.csv" )
Repeat
Attach "Testbed for Windows V1.00 Connected MainWindow"
Type "a{Tab}{+.1}"
Type "{F2}"
Repeat
Type "{Tab}"
Type "{=.+}"
Until TestDataCurField = TestDataFieldCount
Type "{F1}"
Wait(0, "", "RecordUpdated")
Type "{Escape}"
Until TestDataCurRecord = TestDataRecordCount
TestDataClose
End Function ; Main
```

TestDataCurField( )

TestData Handling

Sets the current field number in the testdata file.

Syntax

OldFieldNo = TestDataCurField(NewFieldNo)

Variants

TestDataCurField(NewFieldNo)
TestDataCurField( ) = NewFieldNo
CurFieldNo = TestDataCurField( )

See Also

TestData( ), TestDataCurRecord( ), TestDataField( ), TestDataFieldCount( )

Operation

This function sets the current field number in the current testdata file. The function returns the previously current field.

Passing a parameter to the function sets the current field to the value of the parameter. If no parameter is passed to the function, the current field number is returned and no change is made.

Examples

```
OldField = TestDataCurField( 100 ) ; set the current field to 100
; and save the previous value
Type "{1.=}" ; type record 1, field 100
```

TestDataCurField(OldField) ; restore to previous field

TestDataCurRecord( )

TestData Handling
Sets the current record number in the testdata file.

Syntax

OldRecNo = TestDataCurRecord(NewRecNo)

Variants

TestDataCurRecord(NewRecNo)
TestDataCurRecord( ) = NewRecNo
CurRecNo = TestDataCurRecord( )

See Also

TestData( ), TestDataCurField( ), TestDataField( ), TestDataRecordCount( )

Operation

This function sets the current record number in the current testdata file. The function returns the previously current record.
Passing a parameter to the function sets the current record to the value of the parameter. If no parameter is passed to the function, the current record is returned and no change is made.

Examples

OldRec = TestDataCurRecord(100) ; set the current record to 100
; and save the previous value
Type "{=.1}" ; type record 100, field 1
TestDataCurRecord(OldRec) ; restore to previous record

TestData Expressions

Language
Handling of testdata files.

Syntax

TDVal = TestDataFunction(TestData Expression)

See Also

TestData( ), TestDataCurField( ), TestDataCurRecord( ), TestDataField( ), TestDataFieldCount( ), TestDataIndex( ), TestDataRecordCount( ), TestDataTransform( ), Type( )

Operation

Testdata files provide an efficient way for scripts to access external data. The use of testdata files enables the logic of a script to be separated from its data. For example, to input 500 new entries into a database application, you need only script a single entry. The 500 sets of input data can be read by the script from an external testdata file at runtime.
A testdata file is a comma separated variable (CSV) file in which each line constitutes a record. Each record contains a number of fields that are separated by commas:

Example 1 (A Testdata File Containing 3 Records, Each with 5 Fields):

Tom,Jones,24,Software Development,4227
Dick,Tracy,36,Quality Assurance,1044
Harry,Hawk,52,Product Planning,2128

Example 2 (A Testdata File Containing 'm' Records, Each with 'n' Fields):

R1F1,R1F2,R1F3,…….R1Fn
R2F1,R2F2,R2F3,…….R2Fn
…
RmF1,RmF2,RmF3, RmFn

Testdata files can be created with a text editor, or they can be produced from any spreadsheet or database program that can export or save files in CSV format.
Fields containing commas may be included within a testdata file if they are enclosed in double quotes. For example:

American Systems Corp,31440 Northwestern Highway,"Farmington Hills, MI"
American Systems Ltd,"163, Bath Road","Slough, SL1 4AA"

Testdata files are indexed by *EZ Test* to ensure quick location of individual fields. If the testdata file does not have an index file, or the existing index file is older than the testdata source file, a new index file is created automatically.
Testdata files should be "rectangular" — that is, each record should contain the same number of fields. Records that contain fewer data fields should be padded with blank fields or indexation will fail (the indexing process assumes that all records contain the same number of fields as the first record).
There are a number of functions that allow access to testdata files. These functions interpret a string containing testdata expressions and return the corresponding field values from the currently open testdata file. Each testdata string expression is of the form:
"{<R>.<F>}"
Where <R> can be:
<N> Number specifying the index of a record.
= Retrieve from the current record.
+ Retrieve from the next record.
- Retrieve from the previous record.
* Retrieve from a record selected at random.
Where <F> can be:
<N> Number specifying the index of a field.
= Retrieve from the current field.
+ Retrieve from the next field.
- Retrieve from the previous field.
* Retrieve from a field selected at random.
If no path is specified, a testdata file is assumed to be located in the directory containing the current *EZ Test* database.

Note

There should be no spaces between the fields and the commas.

Examples

```
; This example demonstrates access to a testdata file, typing all
; the fields into Notepad. Note that this script monitors its own
; progress using the TestDataCurRecord / Field and TestDataField /
; RecordCount functions. The script will work on any TestData file,
; regardless of format
Attach "Untitled - Notepad MultiLineEdit~1" ; attach to target app
Repeat ; start "record" loop
Type "{+.1}" ; next record, first field
Type "{Tab}" ; type a "field separator"
Repeat ; start "field" loop
Type "{=.+}" ; same record, next field
Until TestDataCurField = TestDataFieldCount
; end of "field" loop
Type "{Return}" ; type a "record separator"
Until TestDataCurRecord = TestDataRecordCount
; end of "record" loop
```

TestDataField( )

TestData Handling
Retrieves a field from the currently open testdata file.

Syntax

```
TDField = TestDataField( RecNum, FieldNum )
```

See Also

TestData( ), TestDataFieldCount( ), TestDataRecordCount( ), TestDataTransform( )

Operation

This function returns the value of the FieldNum field in the RecNum record of the currently open testdata file.
An empty string is returned if no testdata file is open or if the record or field parameters are invalid. Valid values for the record and field parameters can be obtained from the

TestDataRecordCount( ) and TestDataFieldCount( ) functions.

Examples

```
TestData( "MyData.csv" ) ; use the "mydata.csv" TestData
; file
ret=TestDataField( 10, 1 ) ; extract first field from the
; tenth record
```

TestDataFieldCount( )

TestData Handling

Returns the maximum number of fields per record in the current testdata file.

Syntax

```
NumRecs = TestDataFieldCount( )
```

See Also

TestData( ), TestDataCurField( ), TestDataField( )

Operation

This function returns the maximum number of fields per record in the currently open testdata file.

A 0 is returned if no testdata file is open.

Although it is possible for a testdata file to have differing numbers of fields per record, we recommend avoiding this practice. Making each record contain the same number of fields, by adding empty fields where necessary, makes program loops using testdata expressions easier to conduct.

Examples

```
; this example reads all records in a testdata file and types them
; to the target application
TestData( "c:\data\names.dat" ) ; open the testdata file
Attach "MyApplication" ; attach to application
Repeat ; start record loop
Type "{+.1}" ; next record, first field
Repeat ; start field loop
Type "{=.+}" ; same record, next field
Until TestDataCurField( ) = TestDataFieldCount( )
Until TestDataCurRecord( ) = TestDataRecordCount( )
```

TestDataIndex( )

TestData Handling

Creates an index to a testdata file.

Syntax

```
TestDataIndex( TDFileName )
```

See Also

TestData( ), TestDataField( )

Operation

This function creates an index for the testdata file TDFileName. The resulting index file has the same name as the testdata source file and the extension ".INX". If no path is specified, the testdata file is assumed to be located in the directory containing the current EZ TESTCenter database.

This function has no return value.

Examples

```
TestDataIndex( "MyData.csv" ) ; index the "mydata.csv" TestData
; file
TestDataIndex "y:\data\invoice.dat"
```

TestDataRecordCount( )

TestData Handling

Returns the number of records in the current testdata file.

Syntax
NumRecs = TestDataRecordCount( )
See Also
TestData( ), TestDataCurRecord( ), TestDataField( )
Operation
This function returns the number of records in the currently open testdata file.
A 0 is returned if no testdata file is open.
Examples
Example 1:
TestData("c:\data\names.dat") ; open the testdata file
NumRecs = TestDataRecordCount( ) ; number of records
Example 2:
; this example reads all records in a testdata file and types them
; to the target application
TestData("c:\data\names.dat") ; open the testdata file
Attach "MyApplication" ; attach to application
Repeat ; start record loop
Type "{+.1}" ; next record, first field
Repeat ; start field loop
Type "{=.+}" ; same record, next field
Until TestDataCurField( ) = TestDataFieldCount( )
Until TestDataCurRecord( ) = TestDataRecordCount( )

TestDataTransform( )

TestData Handling
Transforms testdata expressions and retrieves the corresponding field value.
Syntax
FVal = TestDataTransform("{<R>.<F>}")
See Also
TestData( ), TestDataField( )
Operation
This function interprets a string containing testdata expressions and retrieves the corresponding field values from the currently open testdata file. Each testdata string expression is of the form:
"{<R>.<F>}"
Where <R> can be:
<N> Number specifying the index of a record.
= Retrieve from the current record.
+ Retrieve from the next record.
- Retrieve from the previous record.
* Retrieve from a record selected at random.
Where <F> can be:
<N> Number specifying the index of a field.
= Retrieve from the current field.
+ Retrieve from the next field.
- Retrieve from the previous field.
* Retrieve from a field selected at random.
Text that is not a testdata expression is unchanged. The function returns the expanded string.
Both TestDataCurRecord( ) and TestDataCurField( ) are updated when a TestDataTransform( ) expression is evaluated.
This function is used to extract values from a testdata file, either to use in controls that cannot be “typed” in (such as Edit controls) or to enable processing of the value to take place before passing it on to the target application.
TestDataTransform( ) is more flexible than the TestDataField( ) function because it is able to process testdata expressions. On the other hand, TestDataField( ) is easier to use

in loops with numeric counters.

Examples

```
; the example assumes processing of the following TestData file:
Ives,Copland,Gershwin,Bernstein,Joplin,Berlin
Satie,Milhaud,Faure,Saint-Saens,Debussy,Ravel
Elgar,Britten,Vaughan-Williams,Walton,Tippett,Delius
Shostakovich,Tchaikovsky,Prokofiev,Stravinsky,Rimsky-Korsakov,Glinka
Bach,Handel,Mozart,Schubert,Brahms,Hindemith
TestData( "Composer.dat" ) ; open TestData file
Composer = TestDataTransform( "{3.4}" ) ; returns Walton
ListViewCtrl "@Choose Composer", "Composer", 'Left DoubleClick'
Composer = TestDataTransform( "{+.-}" ) ; returns Prokofiev
EditText "@Enter Composer Name:", Composer
German = TestDataTransform( "{5.*}" ) ; a German composer
; selected at random
ComboBox "@Composer ComboBox:", "German", 'Left SingleClick'
French = TestDataTransform( "{5.*} is a French Composer" )
; with
Type French ; text
Type UpperCase(TestDataTransform("{2.*} is a French composer"))
```

TestValue

Checks

Assigns a value to set the current test status of a script.

Syntax

TestValue = Value

See Also

Exit( ), Err, ErrFile, ErrFunc, ErrLine, ErrMsg

Operation

The TestValue command allow you to assign a value to set the current test status. The parameters are as follows:

Value The following are acceptable values:

1 = Pass

0 = Fail

-2 = Runtime error occurred in the script

-1 = Check failed

The script's log will only record a test run as "passed" if the TestValue is set to a value of 1.

Examples

```
Function Main
; Script name test.awl
retval = 0
ret = CaptureBox( "TaskBarTime" , 10 , 1 , 43 , 18 )
ret = ltrimstr( rtrimstr( ret ) )
if ret = "3:19 PM" then
UserCheck( "CheckData", 1, "pass" )
TestValue = 1
else
UserCheck( "CheckData", 0, "fail" )
TestValue = 0
endif
msgbox "" , TestValue
Exit
; logview will show the script either passing or failing
; or
; exit retval
End Function ; Main
```

Calling Script:

```
Function Main
```

```
; Script that calls Test.awl
If Run( "Test" ) = 1
MsgBox "" "All Ok"
Else
MsgBox "" "Script Failed"
EndIf
End Function ; Main
```

TextPanel( )

Miscellaneous
Creates a panel with message text.

Syntax

ret = TextPanel(id, "text" [, x, y, width, height])

Variants

TextPanelClose(id)

See Also

MessageBox( ), PromptBox( )

Operation

This function creates a simple display panel containing a message.
The parameters are:
id The textpanel ID. Numbers 1 to 30 are valid.
"text" The message displayed in the panel (392-character limit).
x The horizontal position of the textpanel's top-left corner.
y The vertical position of the textpanel's top-left corner.
width The panel width in pixels.
height The panel height in pixels.
If the panel position is not specified, it is displayed in the top-left corner of the screen. If the panel dimensions are not specified, it is automatically sized to show the "text".
To remove a textpanel from the screen, use the TextPanelClose variant.

Examples

Example 1:

```
TextPanel 1 "Software Testing Software", 100, 200
; displays this panel
```

Example 2:

```
Function Main
Whenever "manual control" Call manualcontrol
Whenever "auto control" Call autocontrol
< Instructions >
End Function ; Main
Function manualcontrol
;
; Function to handle the keyboard event 'manual control'
;
Message = "You now have manual control of the system"
Message = Message + Chr(13)+Chr(10)
Message = Message + "Press {Alt {ScrollLock}} to return to auto"
TextPanel 20, message
Suspend
End Function ; manualcontrol
Function autocontrol
;
; Function to handle the keyboard event 'auto control'
;
TextPanelClose 20
Resume
End Function ; autocontrol
```

TextSelect( )

Dialog Control
Clicks the mouse on a string of text.

Syntax

ret = TextSelect("text", "option")

Operation

This command moves the mouse pointer to the center of "text" within the currently attached window and performs the action specified by "option".
The "option" parameters can be any combination of the actions supported by the MouseClick( ) command:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"doubleclick" Double-click the mouse button.
"singleclick" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with "control" and "shift".
The function returns 1 if the text is found; Otherwise, an error message is generated or, if used, the script's error handler is called.
TextSelects are generated automatically by the Learn facility if the **TextSelects** option is selected in the Script Editor's Configure Learn Settings dialog box.

Examples

Example 1:

```
Attach "~U~EXPLORER.EXE~SysTabControl32~" ; restore an application
TextSelect "MyApp", 'Left SingleClick'; by clicking the TaskBar
```

Example 2:

```
Attach "~N~MYAPP.EXE~EmulWin~Host123~1" ; attach to host system
TextSelect tr_no 'Left DoubleClick' ; select transaction no.
MenuSelect "Edit~Copy" ; copy to clipboard
```

Example 3:

```
Function Main
On Error Call Test
Attach "~p~notepad.exe~edit~untitled - Notepad"
Ret = 0
Ret = Textselect( "Help" , 'left Singleclick' )
Test( )
End Function
Function Test
Retval = Str( Ret )
If Retval = ""
MessageBox( "Value Of Ret" , "Retval = " + Retval , 'ok' )
Endif
End Function
```

Time( )

Date/Time
Converts a time value into a string.

Syntax

ret = time(timeval)

Variants

ret = time( )

See Also

TimeVal( ), CurTime( )

Operation
This function returns a time value as a string in "hh:mm:ss" format. The timeval is the number of seconds elapsed since midnight and can be obtained from the TimeVal( ) or CurTime( ) functions.
If timeval is not specified, the current time is returned. If timeval is invalid, the string "InvalidTime" is returned.

Examples
time_str = Time(0) ; returns "00:00:00"
time_str = Time(60) ; returns "00:01:00"
The_Time = (11*60*60)+(42*60)+(23)
time_str = Time(The_Time) ; returns "11:42:23"
The_Time = TimeVal(11, 42, 23)
time_str = Time(The_Time) ; returns "11:42:23"
time_str = Time( ) ; returns current time
time_str = Time(-1234) ; returns "InvalidTime"

TimeVal( )

Date/Time
Converts a time into a numerical representation.

Syntax
ret = TimeVal(hh, mm, ss)

See Also
Time( ), CurTime( ), FormatDate( )

Operation
This function returns the time in number format. The value returned is the number of seconds elapsed since midnight.
The parameters are:
hh The hour (0 - 23).
mm The minutes (0 - 59).
ss The seconds (0 - 59).
This function can be used to calculate a future time. The result can be used by the FormatDate( ) and Time( ) functions to produce the time as a string. It can be combined with the value returned by DateVal( ) to make a date/time value.
The function returns -1 if the time format is invalid.

Examples
Example 1:
n = TimeVal(12, 10, 20) ; returns 43820
Example 2:
Start_Time = TimeVal(12, 10, 20) ; returns 43820
End_Time = Start_Time + (18*60*60) ; end in 18 hours
End_Time = Time(End_Time) ; returns "06:10:20"
Example 3:
n = TimeVal(25, 70, 0) ; returns -1 (time invalid)

ToolBarCtrl( )

Dialog Control
Selects options from a toolbar.

Syntax
Ret = ToolBarCtrl("ControlId", "Item", "Options" [, x, y])

Operation
This function is used to make a selection from a standard Windows 95/NT 4 toolbar — such as that found in Explorer. The parameters are as follows:
"ControlId" The index value of the tool bar control.
"Item" The button to select from the tool bar. This value can be literal or variable, by text or by the id of the button.

"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"double" Double-click the mouse button.
"click" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Used in conjunction with "control" and "shift".
x , y These optional parameters specify where on the control the mouse button will be clicked. If omitted, the button is clicked in the center.
The function returns 1 if the selection is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, you must use parentheses.

Examples

Example 1:

```
; In Explorer, change Icon display size and show the Properties of
; the highlighted file
Attach "~N~EXPLORER.EXE~ExploreWClass~Exploring - Documents"
ToolbarCtrl "~1", "Large Icons", "Left SingleClick"
ToolbarCtrl "~1", "Properties", "Left SingleClick"
```

Example 2:

```
; select the third toolbar item
Attach "~N~EXPLORER.EXE~ExploreWClass~Exploring - Documents"
ToolbarCtrl "~1", "@3", "Left SingleClick"
```

TopWindow( )

Window Information
Returns the attach name of the top-most window.

Syntax

```
ret = TopWindow( )
```

Variants

```
ret = TopWindow( "ignoretopmost" )
```

See Also

ActiveName( ), MouseWindow( ), FocusWindow( ), FocusName( ), AttachName( ), AttachWindow( ), Replay.AttachExact

Operation

This function returns the attach name of the top-most window or pop-up menu. In Windows 95/NT 4, the taskbar is always the top-most window. Use the ignoretopmost option to ignore the taskbar and report the top-most application window.
Note that Windows 95 is no longer supported.

Examples

Example 1:

```
Attach TopWindow( "ignoretopmost" ) ; ignore Windows 95 TaskBar
```

Example 2:

```
repeat ; repeat
pause 1 ; wait a while
ret = TopWindow( ) ; check the top window
until ret = "Customer Details" ; until this one is displayed
```

Transpose( )

String Manipulation
Performs actions on characters in a string.

Syntax
Transpose(target, "actions", set1, set2)
See Also
LtrimStr( ), LowerCase( ), RtrimStr( ), TrSet( ), UpperCase( )
Operation
This function performs actions on the set1 characters in the target string. The set2 character is used only if the action specified is replace. The characters in set1 are mapped one-to-one onto the characters in set2; the first character in set1 is replaced with the first character in set2, the second character in set1 is replaced with the second in set2, etc. If set1 is longer than set2, the characters that cannot be mapped are replaced with the last character in set2. This function has no return value; the target string is updated. The actions supported are as follows:
replace Replace the characters contained in set1 with those in set2.
squeeze Delete multiple occurrences of characters contained in set1 from target.
delete Delete the characters specified in set1 from target.
leading Perform the action on leading characters only.
trailing Perform the action on trailing characters only.
both Perform the action on both leading and trailing characters.
all Perform the action on the entire string; this is the default.
Examples
Example 1:
; delete leading spaces from a string
target = " abc123"
Transpose(target, "delete leading", " "); target is "abc123"
Example 2:
; squeeze multiple spaces into single spaces
target = "The quick brown fox"
Transpose(target, "squeeze", " ") ; "The quick brown fox"
Example 3:
; map characters in one string onto those in another
target = "The Quick Brown Fox"
Transpose(target, "replace", "o", "abc"); "The Quick Brawn Fax"
Example 4:
; map characters in one string onto those in another
target = "The Quick Brown Fox"
Transpose(target, "replace", "Fox", "xy") ; "The Quick Brywn xyy"
Example 5:
; convert a string to uppercase
target = "The Quick Brown Fox"
Transpose(target, "replace", TrSet("[a-z]"), TrSet("[A-Z]"))
Example 6:
; delete specific characters from the string
target = "The Quick Brown Fox"
Transpose(target, "delete", "Brown") ; "The Quick Fx"

TreeViewCtrl( )

Dialog Control
Drives the directory list area in a dialog box.
Syntax
Ret = TreeViewCtrl("ControlId", "Item", "Options" [, x, y])
See Also
ListViewCtrl( )
Operation
This function drives the directory list area in a dialog box. The parameters are as follows:
"ControlId" The index value of the tree view control.
"Item" The file or folder to select. This value can be literal or variable,

text or position. To select the first item use "@1" in place of a text value for "Item".
"Options" The options are as follows:
"left" Use the left mouse button.
"right" Use the right mouse button.
"middle" Use the middle mouse button.
"down" Press the mouse button down.
"up" Release the mouse button.
"double" Double-click the mouse button.
"click" Click the mouse button once.
"control" Press the control key before the mouse button.
"shift" Press the shift key before the mouse button.
"with" Use in conjunction with "control" and "shift".
"Label" Click on the text of a directory list item.
"Icon" Click on the directory list item's picture (usually a file folder).
"Button" Click on the directory list item's button to expand or collapse the directory list.
x , y These optional parameters specify where on the item the mouse button will be clicked. If omitted, the button is clicked in the center.
The function returns 1 if the selection is successful, and generates a runtime error if it is not. See the On Error command for information on processing runtime errors in scripts. If this command is generated using the Learn facility, the parentheses are omitted. If a return value is required, you must use the parentheses.

Examples

Attach "Exploring - Directed MainWindow"
TreeViewCtrl "\Desktop\My Computer\Iexplore", 'Left Down', 'Label'

TypeToControl

Dialog Control
Learns typing actions on known controls without requiring repeated Attach statements.

Syntax

TypeToControl "<ControlType>", "<Label>", "<KeyList>"

Variants

TypeCtrl "<ControlType>.<Label>", "<KeyList>"

See Also

Type( )

Operation

This command is used to Learn typing actions in known controls. Unlike the Type( ) command, TypeToControl works on a control label that is passed as a parameter. Consequently, the command does not need an attach statement beforehand. This increases the script readability. The Learn option, Learn TypeToControl, must be selected to Learn TypeToControl commands. Otherwise, the Type command will be learned.
The parameters are as follows:
<ControlType> The following are valid ControlType options:
CheckBox
ComboBox
ComboLBox
DataWindow
Dialog
Edit
Grid
GroupBox
Header
HotspotCtrl

LabelCtrl
ListBox
ListView
PictureCtrl
PushButton
Radio
ScrollBar
Static
StatusBar
TabDialog
ToolBar
TreeView
UpDown
<Label> This is the control label
<KeyList> The list of keystrokes typed to the control.
When the Learn option Learn Tab Key is switched on, *EZ Test* Learns TypeToControl commands. If a {Tab} key is used to exit a control, this is learned as part of the original TypeToControl.
When the Learn option Learn all keys in controls is turned on, *EZ Test* Learns all keystrokes that cause a control to loose focus.

Examples

```
; This example was learned with Learn tab key option enabled
Attach "Find PopupWindow"
EditText "Fi&nd what:", "the quick "
TypeToControl "Edit", "Fi&nd what:", "{F2}"
EditText "Fi&nd what:", "the quick brown"
TypeToControl "Edit", "Fi&nd what:", "{F3}{Tab}"
Checkbox "Match &case", 'Left SingleClick'
TypeToControl "CheckBox", "Match &case", "{Tab}"
RadioButton "&Up", 'Left SingleClick'
RadioButton "&Down", 'Left SingleClick'
TypeToControl "Radio", "&Down", "{Tab}"
TypeToControl "PushButton", "&Find Next", "{F2}{F3}{F4}{Tab}"
TypeToControl "PushButton", "Cancel", "{Tab}"
```

Trset( )

String Manipulation
Expands a string containing a range of characters.

Syntax

ret = Trset([start-end])

See Also

Transpose( )

Operation

This function returns a string containing characters between two values. Ranges are specified [start-end] where start is the first character or number in the range and end is the last. The function evaluates the ASCII values of the start and end characters and expands the string to include all intermediate characters.

Note
There must be no spaces between start, the "-", and end.

Examples

ret = Trset("abcd[g-k]lm") ; result "abcdghijklm"
ret = Trset("[A-Z]") ; result "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
ret = Trset("[a-z]") ; result "abcdefghijklmnopqrstuvwxyz"
ret = Trset("[0-9]") ; result "0123456789"
ret = Trset("[z-a]") ; result "zyxwvutsrqponmlkjihgfedcba"
ret = Trset("a[g-k]2[6-0]"); result "aghijk26543210"
ret = Trset("[!-+]") ; result "!"#%&'( )*+"
ret = Trset("[A - Z]") ; returns "[A - Z]" (space between

; start, - and end

Type( )

Dialog Control
Types a string of keys to the currently attached window.

Syntax

ret = Type("typestr")

Variants

Type(TestData Expression)

See Also

TestData( ), TestDataTransform( ), Replay.TypeDelay

Operation

This command sends the keystrokes in "typestr" to the currently attached window. The characters in "typestr" are most easily (and accurately) generated by using the Learn facility.
Most keys that produce printable characters are specified as seen. Exceptions are the double quote character " (represented by a double-double quote ""), the open brace { (represented by an open brace within single quotes within braces {'{'}), and the close brace } (represented by a close brace within single quotes within braces {'}'}).
Non-printing keystrokes are represented by keynames within braces, for example:
{F1}..{F12} The function keys.
{Escape} The {Escape} key.
{Return} The carriage return key.
{Enter} The {Enter} key on the numeric keypad.
{End} The {End} key on the numeric keypad.
{ExtEnd} The "gray" {End} key (between the alpha and numeric pads).
Auto-repeating keys that are held down are learned as a single key followed by the number of repetitions. For example:

```
Type "{f:28}" ; hold down the "f" key to generate 28 characters
```

The Type function also supports testdata expressions within strings, enabling a field from the currently open testdata file to be entered directly into an application. A testdata expression is of the form:
"{<R>.<F>}"
Where <R> can be:
<N> Number specifying the index of a record.
= Retrieve from the current record.
+ Retrieve from the next record.
- Retrieve from the previous record.
* Retrieve from a record selected at random.
Where <F> can be:
<N> Number specifying the index of a field.
= Retrieve from the current field.
+ Retrieve from the next field.
- Retrieve from the previous field.
* Retrieve from a field selected at random.
Text within the type string that is not a testdata expression is unchanged.
Before a Type is executed the script must attach to a window, else there is no recipient for the keystrokes.
The function returns 1 if Type was successful, and returns 0 if it was not.

Examples

```
Attach "~P~NOTEPAD.EXE~Edit~Untitled - Notepad"
Type "The quick brown fox{Return}"
Type "EZ Test - ""Software Testing Software""{Return}"
Type "Hit the {'{'}Return{'}'} key now{Return}"
Type "{-:24}"
TestData( "names.dat" ) ; open a TestData file
```

Type "{2.3}" ; type third field of second record
Type "His name was {+.*}" ; type a field selected at random
; from the next record

UpDownCtrl( )

Dialog Control
Drives the up and down spin control found on some dialogs.

Syntax

Ret = UpDownCtrl("ControlId", Pos, "Set")

Operation

This function drives up or down the spin control in the currently attached dialog. This type of control is used to increment or decrement a value, such as hours and minutes on the clock. The parameters are as follows:
"ControlId" The index value of the spin control.
Pos The position to set the control to.
"Set" Sets the new value.
The function returns 1 if the selection is successful, and it returns 0 if it is not.
If this command is generated using the Learn facility, the parentheses are omitted. However, if the return value is required, you must use parentheses.

Examples

; change the seconds value on the Date and Time Settings dialog
Attach "~N~RUNDLL32.EXE~#32770~Date/Time Properties"
UpDownCtrl "~2", 58, "Set"
UpDownCtrl "~2", 57, "Set"
UpDownCtrl "~2", 56, "Set"
UpDownCtrl "~2", 55, "Set"

UpDownPos( )

Window Information
Retrieves the value of a spin control.

Syntax

Pos = UpDownPos(hCtrl)

See Also

CtrlEnabled( ), ControlFind( ), CtrlFocus( ), IsWindow( )

Operation

This function retrieves the value of the up-down (spin) control with window handle hCtrl. The hCtrl value can be determined from the UpDownFind( ) function.

Examples

Function SetYear
Attach "Date/Time Properties PopupWindow" ; date/time settings
hCtrl = UpDownFind("~1") ; get handle of "year"
pos = UpDownPos(hCtrl) ; get current setting
If pos <> 1996 ; if not 1996
UpDownCtrl "~1", 1996, 'Set' ; reset year
Endif
End Function ; SetYear

UpperCase( )

String Manipulation
Converts a string into uppercase characters.

Syntax

ret = UpperCase(target)

Variants

ret = Upper(target)

See Also

LowerCase( )

Operation

This function returns the contents of target, converted to uppercase characters.

Examples

target = "the quick brown fox"
target = UpperCase(target) ; returns "THE QUICK BROWN FOX"
ucname = UpperCase("John Smith") ; returns "JOHN SMITH"

UserCheck( )

Checks

Sends a user-defined check entry to the log.

Syntax

UserCheck("checkname", result, "comment")

See Also

LogComment( )

Operation

This function allows the result of a user-defined check to be written to the log. The entry appears within the log as if it were a standard check, with the appropriate color coding. A comment, describing the check, can be logged with the result. The parameters are:
"checkname" The check name to send to the log.
result If set to 1, a "Pass" is logged; if set to 0, a "Fail" is logged.
"comment" A comment associated with the check.
Note that, if result is set to 0 and Log.CheckExit is set to 1, a runtime error message displays as if this was a standard check.

Examples

Exec "target.exe" ; run the target application
; and check that it is maximized
If IsWindow("~N~TARGET.EXE~Target~New Session", "maximized")
UserCheck("InitialState", 1, "Started Maximized"); pass
Else
UserCheck("InitialState", 0, "Not Maximized") ; fail
Endif

Var

Language

Declares private or local variables.

Syntax

Var Variable1 [, Variable2, …, VariableN]

Variants

Var Variable = value
Var Variable1[] [, Variable2[], …, VariableN[]]

See Also

Arrays, Const, Public

Operation

Numeric and string variables and arrays can be public, private or local. Variables or arrays declared as public can be accessed by all child scripts executed using the Run( ) function. The Var statement is used to declare variables or arrays as private or local. Variables or arrays declared outside of functions are private to the current script. The Var statement can be used to declare private variables, but this is not mandatory (in other words, variables are, by default, private).
Variables or arrays declared inside function definitions are local to that function. All string variables are initially assigned to the null string ("") and all numeric variables are initially 0 (zero).
The maximum number of private variables is 4096. The maximum number of local variables is limited by the remaining stack space. The maximum length of a string

variable is only limited by available memory.
All arrays are initialized with no elements

Examples

Example 1:

```
Var a, ret, c ; this line is optional in
Function Main
Setup
MessageBox( "a is" a ) ; a has a value here
End Function
Function Setup ; declaration of private
variables
a = 10
End Function
```

Example 2:

```
Function Setup
var a ; a is local to this function
a = 10
End Function
Function Main
Setup
MessageBox( "a is" a ) ; a is uninitialized in function
End Function
```

Example 3:

```
Var privateA[ ], privateB[ ] ; declaration of private arrays
Function Main
FillArray( privateA, "*.exe" ); fill privatea with .EXE
; filenames
ret=ArraySize(privateA) ; get size of array
Call Show
End Function
Function Show
MsgBox( "", privateA[ret - 1] ); value shown here
End Function
```

Example 4:

```
Function Main
Var locala[ ], localb[ ] ; declaration of local arrays
FillArray( locala, "*.exe" ) ; fill locala with .EXE filenames
ret=ArraySize(locala) ; get size of array
MsgBox( "", locala[ret - 1] ) ; local value shown here
Call Show
End Function
Function Show
Var locala[ ], localb[ ] ; declaration of local arrays
MsgBox( "", locala[ret - 1] ) ; no value shown here
End Function
```

ViewPortClear( )

Miscellaneous
Clears the ViewPort window

Syntax

ViewPortClear( )

See Also

Print( )

Operation

This command clears the ViewPort output window. The ViewPort window is opened by selecting **View>Output** from the Editor's menu. Output is sent to the ViewPort window using the Print( ) command.
This function has no return value.

Examples

```
For I=1 to 100
Print( I )
If I % 10 = 0
ViewPortClear( )
Pause 1
Endif
Next
```

Val( )

String Manipulation
Converts a string into its numeric equivalent.

Syntax

ret = Val(string)

See Also

Str( )

Operation

This function converts a string of numeric characters into a number.

Examples

```
x = Val( "123" ) ; returns 123
y = "-12.64"
y = Val( y ) ; returns -12.64
y = Val("Hello World" ) ; returns 0
y = "551 London Road"
y = Val( y ) ; returns 551
```

Wait( )

Synchronization
Pauses the script until an event occurs.

Syntax

ret = Wait(timeout, "options", eventlist)

See Also

Pause( ), Event( ), Replay.WaitTimeout

Operation

This function causes the script to pause for the time specified by timeout or until the events in the eventlist occur.
Waiting for events to occur is essential for reliable and efficient replay of scripts, particularly when the target application has variable response times. Waits allow the script to run at the maximum speed that the target allows, but ensure that it never runs ahead of the target.
The parameters are:
timeout The maximum time to wait. This is specified in seconds unless the "ms" option is used. If timeout = 0, the timeout period is determined by the Replay.WaitTimeout system variable.
options Use one of the following options:
"ms" Specifies the timeout in milliseconds.
"all" Wait for all of the events in the event list to become true (this is the default case).
"any" Wait for any one of the events in the event list to become true.
"for" Optional word to improve readability.
"until" Optional word to improve readability.
eventlist A list of the event IDs to wait upon.
If there is more than one event in the eventlist and the "any" option is not specified, the script will wait for all the events to trigger before continuing.
The function returns 1 if the Wait( ) function terminated because an event became true

(you can use the Event( ) function to determine which events have occurred). The function returns 0 if the Wait( ) function timed out. — that is, no events occurred within the timeout period.

Examples

Example 1:

```
; allows you to process multiple events
; wait for a keyboard response from the user
Wait 5 "for any" enterkey, helpkey, escapekey
If Event( "enterkey" ) ; if enter key was pressed
<enterkey processing>
Endif
If Event( "helpkey" ) ; if help key was pressed
<helpkey processing>
Endif
If Event ( "escapekey" ) ; if escape key was pressed
<escapekey processing>
Endif
```

Example 2:

```
; Process the first event in the list that becomes true
Wait 5 "for any" enterkey, helpkey, escapekey
If Event( "enterkey" )
<enterkey processing>
Else
If Event( "helpkey" )
<helpkey processing>
Else
If Event ( "escapekey" )
<escapekey processing>
Endif
Endif
Endif
```

Example 3:

```
Exec( "c:\host\emulator.exe" ) ; run the terminal emulator
Attach "Emulator Main Window" ; attach to the emulator and
MenuSelect "&Session~&Connect" ; connect to the host
; define a screen event to confirm display of the logon screen
LogOnScreen = MakeEvent( "screen", "Emulator Main Window", "USERID" )
Wait( 10 "for" LogOnScreen ) ; wait for it to appear
type "Logon DTL" ; before typing in user id
```

Example 4:

```
Function Main
;Function to return the name of the last event satisfied
Wait(30, "for any", "enterkey", "Escape", "F1")
if Event( "enterkey" )
MessageBox ( "", "enterkey" )
endif
if Event( "Escape" )
MessageBox( "","Escape" )
endif
if Event( "F1" )
MessageBox( "", "F1 hit" )
endif
```

WeekDay( )

Date/Time

Returns the day of the week.

Syntax

```
ret = WeekDay( dateval )
```

Variants

```
ret = WeekDay( )
```

See Also
DateVal( ), CurTime( )

Operation
This function returns the day number specified by dateval, where:
0 = Sunday
1 = Monday
...
6 = Saturday
The dateval is a date value that can be derived from the DateVal( ) or CurTime( ) functions.
If dateval is not specified, the current system date is used.

Examples
n = DateVal(1995, 11, 15) ; returns 816393600
Day_of_Week = WeekDay(n) ; returns 3 (Wednesday)
Day_of_Week = WeekDay( ) ; returns current day number

Whenever

Program Flow
Executes a function whenever an event occurs.

Syntax
Whenever "<EventID>" Call <FuncName>

Variants
Whenever <EventID> Call <FuncName>
Whenever "ActionKey" Call <FuncName>

See Also
Cancel( ), Chain( ), Function…End Function, MakeEvent( ), Replay.ActionKeys, Replay.AutoWait, Run( ), Suspend, Wait( )

Operation
This command executes the function named <FuncName> whenever the event specified by <EventID> occurs.
<EventID> Refers to an event defined within the event map or by MakeEvent( ). ActionKey is a reserved EventID that refers to the keys in the Replay.ActionKeys list. When you are using an event defined in the Event Map or using ActionKey, you must enclose the <EventID> in quotes. If you are using an event defined with the MakeEvent( ) command, the quotes should not be present.
<FuncName> Is a user-defined function that is called whenever the specified event becomes true.
This command allows a script to check for an event continually while executing normal instructions. Whenever the event occurs, program flow is diverted to the function <FuncName>. On completion of the instructions in this function, control is returned to the command immediately following the point that the script was interrupted by the event. An event must become false and then true again before a Whenever triggers a second time. This avoids continual calling of the function while the event remains true.
Whenever ActionKey is used to call a function every time *EZ Test* Types any of the action keys specified in the Replay.ActionKeys list. This is especially useful to synchronize with a host-based system accessed via a Windows terminal emulator. The ActionKey whenever is invoked after the action key has been typed and the Replay.AutoWait delay has expired. Only one active ActionKey Whenever is permitted — though it can be redefined with a different function.
Whenevers work across multiple scripts. A Whenever defined in one script remains active during the running of all child scripts.
The command should be used to handle ActionKeys and events when occurrence is unpredictable. Such events would include system or network messages and user-defined

"hot-keys."
A Whenever can interrupt another Whenever — and it can interrupt itself! Whenevers are only active while the program is executing; a script can be made dormant — with Whenevers active — by using the Suspend command. Whenevers can be deactivated using the Cancel command.

Examples

Example 1:

```
F1Key = MakeEvent( "keyboard throwaway",; set up keyboard event in
"Beta Release", "{F1}" ) ; the Beta Release window
Whenever F1Key Call F1KeyPanel ; set up whenever
Suspend ; suspend - leaving
; whenevers active
Function F1KeyPanel
MsgBox( "Help System", ; display help panel
"Help System not yet implemented" )
End Function
```

Example 2:

```
Replay.ActionKeys="{Return}{Enter}" ; set up action keys
Whenever "ActionKey" call Sync ; on any action key, call
; synchronization routine
Function Sync ; synchronization
```

Note

A Whenever causes the specified event to be added to the Whenever Event List and to be continuously monitored by *EZ Test*. This behavior is different to a Wait event — which is discarded from the Wait Event List as soon as the event has occurred or timed out. Large numbers of Whenevers will have a deleterious effect on system performance.

```
; function
Wait( 0, "", Ready )
End Function
```

While...Wend

Program Flow

Repeats a series of instructions while a condition is true.

Syntax

```
While <Boolean Expression>
<Instructions>
Wend
```

Variants

```
While <Boolean Expression> Do
<Instructions>
Wend
```

See Also

Break, Continue, Do...Loop While, Repeat...Until

Operation

This command executes the <Instructions> between While and Wend repeatedly until <Boolean Expression> is false. Execution of the script then continues on the statement following the Wend. The <Boolean Expression> can contain literal values, variables, or return values from functions.

The command is similar to the Do...Loop While structure, except that <Boolean Expression> is evaluated before <Instructions> are executed in a While...Wend structure and after <Instructions> are executed in a Do...Loop While structure.

Because <Boolean Expression> is evaluated at the top of the loop, <Instructions> may not necessarily be executed.

Examples

Example 1:

```
; display a MessageBox six times
i = 1
```

```
While i < 6
MsgBox( "i is now", i )
i = i+1
Wend
```
Example 2:
```
; display random numbers until the user selects No
While MsgBox( "Random Number", "Pick a number?", "yesno" ) = 6
MsgBox( "Random Number", Random( ) )
Wend
```
Example 3:
```
; scroll the page down until it no longer displays "More..."
While FindStr( text, "More..." ) <> 0
text = Capture( "~P~KERNEL32.DLL~ReportWnd~PARTS.LST" )
ScrollBarWindow 1, "Page Vert"
Wend
```

WinClose( )

Window Control
Closes the specified or currently attached window.

Syntax
```
ret = WinClose( "Windowname" )
```
Variants
```
WinClose( )
```
See Also
Maximize( ), Minimize( ), Restore( ), Size( ), Move( ), SetFocus( )

Operation
This function closes the window specified by "Windowname". If no parameter is specified, the currently attached window is closed. In some applications, the attached window is not necessarily the top-most one. For example, *EZ Test* may attach to an edit control or some other child window. If this is the case, the "Windowname" parameter must be used or else the function attempts to close the edit control or other child window. The function returns 1 if the window is closed successfully, and returns 0 if it does not. When this command is generated by the Learn facility, the parentheses are omitted.

Examples
```
; prevent the file "Bootlog.txt" being opened
a = 1 ; set up a counter
while a = 1 ; eternal loop
ret = ActiveName( ) ; get active window name
result = FindStr(ret, "Bootlog.txt"); search for filename
if result <> 0 ; if found in attach name
Attach ret ; attach to the window
WinClose( ) ; and close it
endif
wend
```

WindowText( )

Window Information
Retrieves text from a window.

Syntax
```
ret = WindowText( "WindowName", maxlen )
```
Variants
```
ret = WindowText( )
```
See Also
ActiveWindow( ), ActiveName( ), TopWindow( ), IsWindow( ), WinGetPos( ), FocusWindow( ), FocusName( ), AttachName( )

Operation

This function retrieves the text from the window specified by WindowName. The maxlen is the maximum number of characters to retrieve. If not specified, maxlen defaults to 2000. For most windows, the text returned is the title. For window controls, the text returned is that displayed by the control as follows:
Listboxes The text of the currently selected item.
Comboboxes The text within the edit control of the combo box.
Editboxes The text contained within the edit box.
ListViews The text of the current or last selected item.
Tab Controls The currently selected tab (only 32-bit tab controls are supported).
If a WindowName is not specified, the currently attached window is used.

Examples

```
; move the mouse to these coordinates
MouseMove 475, 475
; get the contents of the dynamic help panel
ret= WindowText( "~P~KERNEL32.DLL~ThunderSSPanel~4" )
; if the response is correct, add a record
if ret= "Add a record"
Button "~1", 'SingleClick'
endif
```

WinGetPos( )

Window Information
Retrieves the position and size of a window.

Syntax

WinGetPos(x, y, width, height, "Windowname")

See Also

Maximize( ), Minimize( ), Restore( ), Size( ), Move( )

Operation

This function returns the size and location of the window specified by "Windowname". If a window name is not specified, the currently attached window is used. The window handle can be used instead of the window name. The return values are as follows (all values are in pixels):
"x" The x-coordinate of the window.
"y" The y-coordinate of the window.
"width" The width of the window.
"height" The height of the window.
If the window does not exist, all returned values are null.

Examples

```
; check if a window has been re-sized and moved
; if it has, restore it to its original state
Attach "~N~NOTEPAD.EXE~Notepad~Untitled - Notepad"
WinGetPos( x, y, w, h, "~N~NOTEPAD.EXE~Notepad~Untitled - Notepad" )
if x <> 100 and y <> 100
move 100, 100
if w <> 400 and h <> 400
size 400, 400
endif
endif
```

WinVersion( )

System Information
Returns the Windows version as a numeric value.

Syntax

ret = WinVersion( )

See Also

SystemInfo( )

Operation
This function returns the version number of Windows as a numeric value. A number of 400 is broken down as 4 for the major release number and 00 for the minor.
Examples
ret = WinVersion( ) ; get the Windows version
If ret = 351 ; if Windows/NT Version 3.51
Exec("old_app.exe") ; run old software
ElseIf ret = 400 ; if Windows 95 or NT 4
Exec("new_app.exe") ; run new software
EndIf
Note that Windows 95 is not supported.

WndAtPoint( )

Window Information
Retrieves the handle of a window at a point on the screen.
Syntax
hWnd = WndAtPoint(x, y)
Variants
hWnd = WndAtPoint( )
See Also
AttachWindow( ), MouseWindow( )
Operation
This function returns the handle of the window at position x, y. If no screen coordinates are specified, the function returns the handle of the window beneath the mouse pointer.
Examples
Function Main
Attach "Open PopupWindow" ; attach to dialog
TextSelect "Open", 'Right SingleClick' ; textselect a control
MsgBox("", WndAtPoint( )) ; show its handle
End Function ; Main

Word( )

String Manipulation
Extracts words from a string.
Syntax
ret = Word(target, startwordnumber, number of words)
Variants
ret = Word(target, startwordnumber)
See Also
Words( )
Operation
This function extracts a word or a group of words from a string.
The parameters are as follows:
target The variable or string containing the words to be extracted.
startwordnumber The first word to extract.
number of words The number of words to extract from the string. If omitted, only the startwordnumber is extracted.
Examples
target = "the quick brown fox"
ret = Word(target, 2, 3) ; result is "quick brown fox"
ret = Word(target, 2) ; result is "quick"

Words( )

String Manipulation

Returns the number of words in a string.

Syntax

ret = Words(target)

See Also

Word( )

Operation

This function returns the number of words in a string. Words are delimited by spaces, tabs, new lines, or carriage returns.

Examples

target = "the quick brown fox"
ret = Words(target) ; result is 4
target = "the quick" ; tab delimited string
ret = Words(target) ; result is 2

Write( )

File Access
Writes a string to a file.

Syntax

ret = Write("filename", string, [number] , [DelimiterString])

Variants

ret = Write("filename", string)

See Also

FilePos( ), Open( ), Read( ), ReadIni( ), ReadLine( ), WriteLine( )

Operation

This function writes string to filename. If filename has not been opened with the Open( ) function, Write( ) opens it for writing at the end of the file. If filename does not exist, it is created. If filename has been opened with Open( ), the write occurs at FilePos( ).
The optional number parameter specifies the maximum number of characters to write. Following the write, the filepointer is positioned at the character following string and FilePos( ) is updated accordingly.
The parameters are as follows:
"filename" The file to write to.
string The string to be written.
number An optional parameter that specifies the maximum number of characters to write.
DelimiterString An optional parameter that specifies a string that indicates the delimiter. The DelimiterString parameter can be up to 99 characters. To write records with carriage returns and line feeds as the delimiter, use chr(13) + chr(10) as the delimiter.
The function returns 1 if the file is written successfully, and returns 0 if it is not.

Examples

Example 1:

; add a line to the end of the config.sys file
Write("c:\config.sys", "Files = 100")

Example 2:

; create a comma separated variable (CSV) data file
filename= "c:\data\names.csv" ; a file for names
Open(filename, "readwrite") ; open it for read/write access
Write(filename, name+",") ; write name + a comma
Write(filename, address1+",") ; address line 1 plus comma
Write(filename, address2+",") ; address line 2 plus comma
Write(filename, address3+chr(13)+ chr(10))
; end with a CR / LF

WriteCom( )

Serial Communications
Writes a specific number of bytes to the PC's open COM port from a data array.

Syntax

ret = WriteCom(Port, DataArray, NumberOfBytes)

See Also

CloseCom( ), OpenCom( ), PurgeCom( ), ReadCom( )

Operation

This function writes the specified number of bytes to the PC's open COM port from the specified data array.
The options are:
Port A number from 1 - 255.
DataArray The name of the data array where to bytes are read to.
Number of BytesThe number of bytes to be read.
The function has the following return values:
n Number of bytes read
0 Failure
-1 Bad Port

Examples

```
Var y[ ]
Var x[ ]
y[1] = 41
y[2] = 41
y[3] = 41
y[4] = 41
y[5] = 41
z = OpenCom( 4, 9600, 8, 0, 1 ) ;open up COM port 4
z = PurgeCom ( 4 ) ;purge data in COM 4
z = WriteCom( 4, y, 5 ) ;writes 5 bytes of data to COM 4
z = ReadCom ( 4, x, 5, 1 ) ;reads back 5 bytes of data
z = CloseCom( 4 ) ;Close COM port 4
Print x[1]
Print x[2]
Print x[3]
Print x[4]
Print x[5]
```

Writeini( )

File Access
Writes a value to an .INI file.

Syntax

ret = Writeini("inifile", "section", "key", "value")

See Also

ReadIni( )

Operation

This function writes a value to an .INI file.
The parameters are as follows:
inifile The .INI file to write to.
section The name of the section in the INI file where information is to be written.
key The command line to change.
value The value to set the key to.
The function returns 1 if the write is successful, and returns 0 if it not.

Examples

```
; set the value of the "Filesharing" entry in the "Network"
; section of "system.ini" to "No"
ret = Writeini( "c:\windows\system.ini", "Network", "Filesharing",
```

"No")
MsgBox("Result", ret) ; result is 1 if successful 0 if not

WriteLine( )

File Access
Writes a line to a file.

Syntax

ret = WriteLine("filename", string, number)

Variants

ret = WriteLine("filename", string)

See Also

FilePos( ), Open( ), Read( ), ReadIni( ), ReadLine( )

Operation

This function writes string to filename, followed by a carriage return/line feed. If filename has not been opened with the Open( ) function, WriteLine( ) opens it for writing at the end of the file. If filename does not exist, it is created.
If filename has been opened with Open( ), the write occurs at FilePos( ). The optional number parameter specifies the maximum number of characters to write.
Following the write, the filepointer is positioned at the character following string CR/LF and FilePos( ) is updated accordingly.
The parameters are as follows:
"filename" The file to write to.
string The string to be written.
number The maximum number of characters to write.
The function returns 1 if the file is written successfully, and returns 0 if it is not.

Examples

Example 1:

```
; add a line to the end of the config.sys file
WriteLine( "c:\config.sys", "Files = 100" )
```

Example 2:

```
; update a value in a fixed length record data file
filename= "c:\data\names.dat" ; a file containing names
Open( filename, "readwrite" ) ; open it for read/write access
Do ; start of loop
sav = FilePos( filename) ; save the filepointer
ReadLine( filename, nextname) ; read the next line
RTrimStr( nextname) ; trim the trailing spaces
If nextname= "Miss Vera Jones" ; if it is the right name
FilePos( filename, sav ) ; move filepointer to start
; of line
newname= PadStr( "Mrs Vera Westwood, 40 )
; pad new name
WriteLine( filename, newname) ; update the record
EndIf
Loop While FileStatus( filename) <> 2 ; stop at end of file
```

Year( )

Date/Time
Returns the year.

Syntax

ret = Year(dateval)

Variants

ret = Year( )

See Also

DateVal( ), CurTime( )

Operation

This function returns the year specified by dateval. The dateval is a date value that can be derived from the DateVal( ) or CurTime( ) functions. If dateval is not specified, the current system date is used.

Examples

n = DateVal(1996, 12, 25) ; returns 851472000
The_Year = Year(n) ; returns 1996
The_Year = Year( ) ; returns the current year

Index