Tell me about using Delrina Basic

What is Scripting?

Scripting is an easy way to customize how you use WinComm. By creating scripts you can create custom sessions and set up how they interact with a remote system. Scripts are written using the Delrina Basic language.

A script can initially be recorded automatically or it can be custom-written.

Regardless of how you create the script, you can then edit it, adding code according to your own needs.

About the Delrina Basic Language

The Delrina Basic language has commands that you use to access other servers and control online sessions with these servers. It is based on Summit BasicScript 2.1 with WinComm specific extensions.

In addition to the extensive and detailed information available in the WinComm online help, note the following about the Delrina Basic commands:

- They are not case sensitive. For example, the command DlgControlId can also be entered as DlgcontrolID.
- Script commands that talk directly to the modem (for example, to set the baud rate or the parity) work only with Standard COM port modems - modems that communicate directly with the COM port, rather than through Windows.

  Examples of these commands are DialingMethod, BaudRate and BitsPerCharacter.

Tell me about writing scripts

With a WinComm script, you can:

- Automate a repetitive task, such as logging on to a server and checking your mail.
- Give flexibility to a standard task. For example, you can create a script to perform an online activity like download a file, even though commands to perform this task may vary between sessions.
- Eliminate the need to manually enter session information, such as user IDs and passwords. You can create a script that automatically sends this information to the remote system.

As you create your script, you can:

- design a specific dialog
- design a pop-up information box
- send specific information to a remote system
- manipulate information before it is used by the remote system
- manipulate information from the server before it is displayed to the WinComm user.

Start the script using one of these methods:

- record a logon script.
- create the script using the New command on the Script menu.

Creating a new script

1. On the Script menu, point to New and click File. The New Script Source File dialog appears.
2. In the File name field, type the name of the script you want to create.
3. Click Open. The Script Editor dialog appears.
4. Enter the Delrina Basic commands as required.
5. On the File menu, click Save to save the script.

Editing an existing script

1. On the Script menu, click Edit. The Select Script dialog appears.
2. In the File name field, type the name and path of the script.
3. Click Open. The Script Editor dialog appears.
4. Modify the script as required.
5. On the File menu, click Save to save the script.

Tell me about script projects

With scripts both the source files and compiled versions are limited to a size of 64K. For this reason, as well as to facilitate modular programming, scripts are usually grouped into projects.

In a project, include only one script that contains a Main

procedure one that starts with the command:

Sub Main()

All scripts within the project may contain subroutines that call one another. When you run a project, the Main subroutine runs first. The other subroutines, even if they exist in different scripts, run as they are called from within the project.

Creating a new project

1. On the Script menu, point to New and click Project. The New Script Project dialog appears.
2. In the File name field, type the name and path of the project.
3. Click Open. The Script Project Editor dialog appears.
4. To add a script to the project, click Add. The Select Script dialog appears, where you choose the name of the script you want to add.
5. Select the script you want to add to the project.

Editing an existing project

1. On the Script menu, click Edit. The Select Script dialog appears.
2. In the File name field, type the name of the project.
3. Click Open. The Script Project Editor dialog appears.
4. To add a script to the project, click Add. The Select Script dialog appears, where you choose the name of the script you want to add.
5. Select the script and click Edit. The Script Editor dialog appears.
6. Modify the script as required.

Tell me about compiling scripts

Normally a script is interpreted every time you run the source.

You can also compile the script using the Compile command on the Script menu. The compiler finds any syntax errors that may exist, so that you can correct them before running the script. Since a compiled script does not have to be interpreted each time it is run, it will execute faster.

Compiling a script

1. On the Script menu, click Compile. The Select Script dialog appears.
2. In the File name field, type the name of the script to compile.
3. Click Open. The script is compiled and, when there are no syntax errors, the following message appears.

Compiling all the scripts in a project

1. On the Script menu, click Compile. The Select Script dialog appears.
2. In the File name field, type the name and path of the project.
3. Click Open. The scripts listed in the project are compiled.

Tell me about runtime data strings

Twenty runtime data strings are available to use in your script. A data string is a variable normally entered by the user before running the script. It could contain information such as the name of the newsgroup that the user wants to access in the current session, or a file to download. By storing these variables as data strings, there is less interaction with the user required to complete the online task.

Entering a data string

1. In a session, click Runtime Data Strings on the Script menu. The Runtime Data Strings dialog appears.
2. Enter the data string in the appropriate field.

Note

- Data strings 1 to 10 are cleared each time you start a session. Data strings 11 to 20 are stored with the session, available the next time this session is run.

Moving the insertion point to a specified line

1. On the Edit menu, click Goto Line. The Goto Line dialog appears.
2. Enter the number of the line in your script to which you want to move the insertion point.
3. Click OK.

Note

- The insertion point cannot be moved so far below the end of a script as to scroll the script entirely off the display. When the last line of your script becomes the first line on your screen, the script will stop scrolling, and you will be unable to move the insertion point below the bottom of that screen.

Adding a full-line comment

1. Position the cursor where you want to add the comment.
2. Type an apostrophe (') at the start of the line.
3. Type your comment following the apostrophe. When your script is run, the presence of the apostrophe at the start of the line will cause the entire line to be ignored.

Adding a comment at the end of a line

1. Position the insertion point in the empty space beyond the end of the line of code.
2. Type an apostrophe (').
3. Type your comment following the apostrophe. When your script is run, the code on the first portion of the line will be run, but the presence of the apostrophe at the start of the comment will cause the remainder of the line to be ignored.

Replacing specified text

1. Move the insertion point to where you want to start the replacement operation. (To start at the beginning of your script, press Ctrl+Home.)
2. Choose the Replace command from the Search menu. Script Editor displays the Replace dialog box:
3. In the Find What field, specify the text you want to replace.
4. In the Replace With field, specify the replacement text.
5. Select the Match Case check box if you want the replacement operation to be case-sensitive. Otherwise, it will be case-insensitive.
6. To replace all instances of the specified text, click the Replace All button.
7. To replace selected instances of the specified text, click the Find Next button.
8. If the specified text has been found, either click the Replace button to replace that instance of it or click the Find Next button to highlight the next instance (if any).

Inserting a new dialog template into scripts

1. Place the insertion point where you want the new dialog box template to appear in your script.
2. From the Edit menu, choose the Insert New Dialog command. Script Editor's application window is temporarily disabled, and Dialog Editor appears, displaying a new dialog box in its application window.
3. Use Dialog Editor to create your dialog box.
4. Exit from Dialog Editor and return to Script Editor.

Editing an existing dialog template into a script

1. Select the Delrina Basic code for the entire dialog box template.
2. From the Edit menu, choose the Edit Dialog command. Script Editor's application window is temporarily disabled, and Dialog Editor appears, displaying in its application window a dialog box created from the code you selected.
3. Use Dialog Editor to modify your dialog box.
4. Exit from Dialog Editor and return to Script Editor.

Running a script

- To run your script on the Run menu, click Start. The script is compiled (if it has not already been compiled), the focus is switched to the parent window, and the script is run.

Note

- During script execution, Script Editor's application window is available only in a limited manner. Some of the menu commands may be disabled, and the toolbar tools may be inoperative.

Pausing an executing script

- To pause an executing script on the Run menu, click Pause. Execution of the script is suspended, and the instruction pointer (a gray highlight) appears on the line of code where the script stopped executing.

Note

- The instruction pointer designates the line of code that will be run next if you resume running your script.

Stopping an executing script

- To stop an executing script, on the Run menu click End.

Note

- Many of the functions of Script Editor's application window may be unavailable while you are running a script. If you want to stop your script but find that the toolbar is currently inoperative, press Ctrl+Break to pause your script, then click the End tool.

Stepping through a script

1. To take a single step through your script, click Single Step on the Debug menu.
2. To continue tracing the execution of your script line by line, repeat step 1. When you finish tracing the execution of your script, either click the Start tool on the toolbar (or press F5) to run the balance of the script at full speed or click the End tool to halt execution of the script.

Note

- When you initiate execution of your script with any of these methods, the script will first be compiled, if necessary. Therefore, there may be a slight pause before execution actually begins. If your script contains any compile errors, it will not be run. To debug your script, first correct any compile errors, then initiate execution again.

Starting debugging partway through a script

1. Place the insertion point in the line where you want to start debugging.
2. To set a breakpoint on that line, click the Toggle Breakpoint tool on the toolbar. The line on which you set the breakpoint now appears in contrasting type.
3. Click the Start tool on the toolbar.

Continuing debugging at a line outside the current subroutine

1. Place the insertion point in the line where you want to continue debugging.
2. To set a breakpoint on that line, press F9.
3. To run your script, click the Start tool on the toolbar or press F5.

Debugging selected portions of a script

1. Place a breakpoint at the start of each portion of your script that you want to debug.
2. To run the script, click the Start tool on the toolbar or press F5. The script runs at full speed until it reaches the line containing the first breakpoint and then pauses with the instruction pointer on that line.
3. Step through as much of the code as you need to.
4. To resume running your script, click the Start tool on the toolbar or press F5. The script runs at full speed until it reaches the line containing the second breakpoint and then pauses with the instruction pointer on that line.
5. Repeat steps 3 and 4 until you have finished debugging the selected portions of your script.

Note

- Up to 255 lines in your script can contain breakpoints.

Removing a single breakpoint

1. Place the insertion point on the line containing the breakpoint that you want to remove.
2. Click the Toggle Breakpoint tool on the toolbar.

Removing all breakpoints

- On the Debug menu, click Clear All Breakpoints.

Adding watch variables

1. Click the Add Watch tool on the toolbar.
2. Use the controls in the Context box to specify where the variable is defined (locally, publicly, or privately) and, if it is defined locally, in which routine it is defined.
3. In the Variable Name field, enter the name of the variable you want to add to the watch variable list.

   You can only watch variables of fundamental data types, such as Integer, Long, Variant, and so on; you cannot watch complex variables such as structures or arrays. You can, however, watch individual elements of arrays or structure members using the following syntax:

   [variable [(index,...)] [.member [(index,...)]]...]

   Where variable is the name of the structure or array variable, index is a literal number, and member is the name of a structure member.

   For example, the following are valid watch expressions:

Watch Variable	Description
a(1)	Element 1 of array a
person.age	Member age of structure person
company(10,23).person.age	Member age of structure person that is at element 10,23 within the array of structures called company

Notes

- If you are executing the script, you can display the names of all the variables that are "in scope," or defined within the current function or subroutine, on the drop-down Variable Name list and select the variable you want from that list.
- Although you can add as many watch variables to the list as you want, the watch pane only expands until it fills half of Script Editor's application window. If your list of watch variables becomes longer than that, you can use the watch pane's scroll bars to bring hidden portions of the list into view.

Selecting variables on the watch list

- Place the mouse pointer on the variable you want to select and click the left mouse button.

Note

- Pressing F6 again returns the insertion point to its previous position in the edit pane.

Modifying the value of variables on the watch variable list

1. Place the mouse pointer on the name of the variable whose value you want to modify and double-click the left mouse button.
2. Enter the new value for your variable in the Value field.
3. Click OK.

Note

- The name of the variable you selected on the watch variable list appears in the Name field. If you want to change another variable, you can either enter a different variable in the Name field or select a different variable from the Variables list box, which shows the names of the variables that are defined within the current function or subroutine.
- When you use the Modify Variable dialog box to change the value of a variable, you don't have to specify the context. Script Editor first searches locally for the definition of that variable, then privately, then publicly.

Finding specified text

1. Move the insertion point to where you want to start your search. (To start at the beginning of your script, press Ctrl+Home.)
2. Press Ctrl+F. Script Editor displays the Find dialog box:
3. In the Find What field, specify the text you want to find.
4. Select the Match Case check box if you want the search to be case-sensitive. Otherwise, the search will be case-insensitive.
5. Click the Find Next button or press Enter. The Find dialog box remains displayed, and Script Editor either highlights the first instance of the specified text or indicates that it cannot be found.
6. If the specified text has been found, repeat step 5 to search for the next instance of it.

Note

- If the Find dialog box blocks your view of an instance of the specified text, you can move the dialog box out of your way and continue with your search. You can also click the Cancel button, which removes the Find dialog box while maintaining the established search criteria, and then press F3 to find successive occurrences of the specified text. (If you press F3 when you have not previously specified text for which you want to search, Script Editor displays the Find dialog box so you can specify the desired text.)

Examples

The following are examples of Delrina Basic. These examples use WinComm Exensions, and may be customized to work with different remote systems. Different examples may be run together, which enables complex interactions with different systems.

Capture to File

Connection Status

User Information

Set Session Parity

Session Port Name

Run Script

Size Session Window

Connect and Wait for Prompt

Capture to File

The following example script sets file to capture text to, and turns capture to file on.

```
Sub Main()
    Dim Sess As New Session
    rem Dim Return_Value As Integer
    Dim Status As Integer
    Dim Sess_Name$ As String
    Dim Cap_File$ As String
    Cap_File$ = "C:\WinComm\CAPTURE.TXT"
    Sess_Name$ = "C:\WinComm\DELRINA.WCS"
    MsgBox "Opening Delrina Session for text capture.", , "Capture to File
    Example"
    Sess.OpenSession Sess_Name$
    Sess.Connect DC_CNCT_STANDARD
    Status = Sess.connectionStatus
    MsgBox "Connection Status = " & Status, ,"Capture to File Example"
    Sess.captureFileName = Cap_File$
    Sess.CaptureToFileBegin DC_C_LINES, FALSE
End Sub
```

Connection Status

The following example script displays the connection status for a Delrina BBS session.

```
Sub Main()
    Dim Sess As New Session
    Dim Status As Integer
    MsgBox "Opening Delrina Session.", , "Connection Status Example"
    Sess.OpenSession "C:\WinComm\DELRINA.WCS"
    Sess.Connect DC_CNCT_STANDARD
    Status = Sess.connectionStatus
    MsgBox Status, , "Connection Status Example"
End Sub
```

User Information

The following example script gets the current User Name, User ID and Password for the Delrina BBS session.

```
Sub Main ()
    Dim Sess As New Session
    Dim InputStr$ As String
    Dim RunTimeString$ As String
    MsgBox "Opening Delrina Session ", , "Get User Information Example"
    Sess.OpenSession "C:\WinComm\DELRINA.WCS"
    RunTimeString$ = Sess.GetRunTimeValue(DC_RV_USERNAME,FALSE)
    MsgBox "User Name Returned was " + RunTimeString$, , "Get User
    Information Example "
    RunTimeString$ = Sess.GetRunTimeValue(DC_RV_USERID,FALSE)
    MsgBox "The User Id Returned was " + RunTimeString$, , "Get User
    Information Example "
    RunTimeString$ = Sess.GetRunTimeValue(DC_RV_PASSWORD,FALSE)
    MsgBox "The Password Returned was " + RunTimeString$, , "Get User
    Information Example "
End Sub
```

Set Session Parity

The following example script sets the parity for the Delrina BBS session.

```
Sub Main ()
    Dim Sess As New Session
    Dim X As Integer
    MsgBox "Opening Delrina Session for Testing", ,"Get / Set Parity Example"
    Sess.OpenSession "C:\WinComm\DELRINA.WCS"
    MsgBox "Setting Parity to Even", ,"Get / Set Parity QA Test"
    Sess.parity = DC_M_E_PRTY
    X = Sess.parity
    MsgBox " X has the value: " & X, ," Get / Set Parity Example "
    If X = 4 then
    MsgBox "Parity Set to Even", ," Get / Set Parity Example "
    end if
    MsgBox "Setting Parity to Odd", ," Get / Set Parity Example "
    Sess.parity = DC_M_O_PRTY
    X = Sess.parity
    MsgBox " X has the value: " & X, ," Get / Set Parity Example "
    if X = 8 then
    MsgBox "Parity Set to Odd", ," Get / Set Parity Example "
    end if
    MsgBox "Setting Parity to None", ," Get / Set Parity Example "
    Sess.parity = DC_M_N_PRTY
    X = Sess.parity
    MsgBox " X has the value: " & X, ," Get / Set Parity Example "
    if X = 16 then
    MsgBox "Parity Set to None", ," Get / Set Parity Example "
    end if
    MsgBox "Setting Parity to Mark", ," Get / Set Parity Example "
    Sess.parity = DC_M_M_PRTY
    X = Sess.parity
    MsgBox " X has the value: " & X, ," Get / Set Parity Example "
    if X = 32 then
    MsgBox "Parity Set to Mark", ," Get / Set Parity Example "
    end if
    MsgBox "Setting Parity to Space", ," Get / Set Parity Example "
```

```
    Sess.parity = DC_M_S_PRTY
    X = Sess.parity
    MsgBox " X has the value: " & X, ,"Get / Set Parity QA Test"
    if X = 64 then
    MsgBox "Parity Set to Space", ,"Get / Set Parity QA Test"
    end if
End Sub
```

Session Port Name

The following example script returns the current port type for the Delrina BBS session.

```
Sub Main()

    Dim Sess As New Session

    Dim PortName$ As String

    Dim StoredValue$ As String

    MsgBox "Opening the Delrina Session for Testing", , "Get / Set Port Name
    Example"

    Sess.OpenSession "C:\WinComm\DELRINA.WCS"

    PortName$ = InputBox$ ("Enter a Valid Port Name (COM1 - COM4): ","Get /
    Set Port Name Example")

    Sess.portName = PortName$

    StoredValue$ = Sess.portName

    MsgBox "The Port Name is set to " + StoredValue$, , "Get / Set Port Name
    Example"

    Sess.Connect DC_CNCT_STANDARD

End Sub
```

Run Script

The following example script calls and runs another script. This example is useful for combining two or more scripts into one larger script.

This example opens the Delrina BBS session and runs the script: EXAMPLE1.

```
Sub Main()
   Dim Sess As New Session
   Dim Return_Value As Integer
   MsgBox "Opening Delrina Session for testing.", , "Run Script Example"
   Sess.OpenSession "C:\WinComm\DELRINA.WCS"
   MsgBox "Running Script C:\WinComm\EXAMPLE1.dbc", ,"Run Script Example"
   RunScript "C:\WinComm\EXAMPLE1.dbp"
   Return_Value = Sess.Error
   MsgBox "The Return Value was ", Return_Value, "Run Script Example"
End Sub
```

Size Session Window

The following example script opens the Delrina BBS session, and automatically sizes the session window to fit the window.

```
Sub Main()
    Dim Sess As New Session
    MsgBox "Opening Delrina Session " , , "Session Size Example"
    Sess.OpenSession "C:\WinComm\DELRINA.WCS"
    MsgBox "Starting to minimize Session Window", , "Size Session Example"
    sessionSize DC_S_MIN
    MsgBox "Starting to maximize Session Window", , "Size Session Example"
    sessionSize DC_S_MAX
    MsgBox "Starting to Restore Session Window", , "Size Session Example"
    sessionSize DC_S_RSTR
End Sub
```

Connect and Wait for Prompt

The following example script connects to the Delrina BBS, logs on using the current user information, and waits for a prompt.

```
Sub Main()
    REM Declared variables which may be used in rest of program.
    Dim ReturnValue As Integer
    Dim RuntimeValue As String
    Dim Sess_Name$ As String
    Dim Status As Integer
    REM Used to Open New Session to stablish a link between this script
    REM program and Delrina WinComm PRO.
    Dim Sess As New Session
    REM ************************************************************
    REM * Delrina WinComm PRO Delrina Basic calls that automate       *
    REM * your interaction with the remote system.                  *
    REM ************************************************************
    REM
    Sess_Name$ = "C:\Windows\Desktop\DELRINA.WCS"
    MsgBox "Opening Delrina Session for testing.", , "Capture to File Control
    QA Test"
    Sess.OpenSession Sess_Name$
    Sess.Connect DC_CNCT_DO_NOT_LOGIN
    Status = Sess.connectionStatus
    ReturnValue = Sess.WaitForPrompt( "Do you want graphics       (Enter)=no?
    "+Chr$( 0 ), 300, 100000 )
    Sess.TypeText "y"+Chr$( 13 )+""
    ReturnValue = Sess.WaitForPrompt( "What is your first name? "+Chr$( 27 )
    +"[0m"+Chr$( 0 ), 300, 100000 )
    Sess.TypeText "John"+Chr$( 13 )+""
    ReturnValue = Sess.WaitForPrompt( "What is your last name? "+Chr$( 27 )
    +"[0m"+Chr$( 0 ), 300, 100000 )
    Sess.TypeText "Doe"+Chr$( 13 )+""
    ReturnValue = Sess.WaitForPrompt( "[13D"+Chr$( 27 )+"[0m"+Chr$( 0 ), 300,
    100000 )
    REM ReturnValue = Sess.WaitForPrompt( "Password (Dots will echo)? "+Chr$(
    27 )+"[0m"+Chr$( 0 ), 300, 100000 )
    Sess.TypeText "password"+Chr$( 13 )+""
    ReturnValue = Sess.WaitForPrompt( "More? "+Chr$( 27 )+"[0m"+Chr$( 0 ),
    300, 100000 )
```

```
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "More? "+Chr$( 27 )+"[0m"+Chr$( 0 ),
   300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "More? "+Chr$( 27 )+"[0m"+Chr$( 0 ),
   300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "More? "+Chr$( 27 )+"[0m"+Chr$( 0 ),
   300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "[0m"+Chr$( 0 ), 300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "m"+Chr$( 0 ), 300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "Continue? "+Chr$( 27 )+"[0m"+Chr$
   ( 0 ), 300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   ReturnValue = Sess.WaitForPrompt( "More? "+Chr$( 27 )+"[0m"+Chr$( 0 ),
   300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
    ReturnValue = Sess.WaitForPrompt( "More? "+Chr$( 27 )+"[0m"+Chr$( 0 ),
   300, 100000 )
   Sess.TypeText ""+Chr$( 13 )+""
   REM Terminate link between Delrina WinComm PRO and script program.
   Exit Sub
End Sub
```

Command Reference

Click the first letter of the word you want defined.

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

& (operator)
' (keyword)
() (keyword)
* (operator)
+ (operator)
- (operator)
. (keyword)
/ (operator)
< (operator)
<= (operator)
<> (operator)
= (statement)
= (operator)
> (operator)
>= (operator)
\ (operator)
^ (operator)
_ (keyword)

A

ActivateControl (statement)
AddToSendList
Abs (function)
AddToSendList
AnswerBox (function)
And (operator)
AppActivate (statement)
Any(data type)
application
AppFilename$ (function)
AppClose (statement)
AppGetActive$ (function)
AppFind$ (function)
AppGetState (function)
AppGetPosition (statement)
AppList (statement)
AppHide (statement)
AppMinimize (statement)
AppMaximize (statement)
AppRestore (statement)
AppMove (statement)
AppShow (statement)
AppSetState (statement)
AppType (function)
AppSize (statement)
Arrays (topic)

ArrayDims (function)
Asc (function)
ArraySort (statement)
ascii7BitReceive
asciiAppendLFReceive
asciiAppendLFSend
asciiCharDelayValue
asciiLineDelayValue
asciiExpandBlankToSpace
asciiInputTabValue
asciiInputWaitChar
asciiLocalEcho
asciiOutputTabValue
asciiRemoteEcho
asciiShowHex
asciiTabExpandSend
asciiWaitForLineEnd
asciiWrapLines
AskPassword$ (function)
AskBox$ (function)
Atn (function)

B

Basic.Capability (method)
Basic.Eoln$ (property)
Basic.FreeMemory (property)
Basic.HomeDir$ (property)
Basic.Version$ (property)
Basic.PathSeparator$ (property)
baudRate
Begin Dialog (statement)
Beep (statement)
bitsPerCharacter
BlockRemoteInput
Boolean (data type)
ButtonEnabled (function)
ButtonExists (function)
ByRef (keyword)
ByVal (keyword)

C

Call (statement)
CancelButton (statement)
CaptureFileName
CaptureToFileBegin
CaptureToFileControl
CaptureToPrinterBegin
CaptureToPrinterControl
CBool (function)
CCur (function)
CDate, CVDate (functions)
CDbl (function)
ChDir (statement)
CheckBox (statement)

ChDrive (statement)
CheckBoxExists (function)
CheckBoxEnabled (function)
Chr, Chr$ (functions)
Choose (function)
ClearDropList
ClearOutputBuffer
ClearSendList
Close
Clipboard$ (function)
CInt (function)
Clipboard.Clear (method)
Clipboard$ (statement)
Clipboard.GetText (method)
Clipboard.GetFormat (method)
CLng (function)
Close (statement)
Clipboard.SetText (method)
Connect
ConnectAndDial
connectionStatus
connectionTime
ComboBox (statement)
ComboBoxExists (function)
ComboBoxEnabled (function)
ComDriverSpecial
Comments (topic)
Command, Command$ (functions)
Const (statement)
Comparison Operators (topic)
Cos (function)
Constants (topic)
Cross-Platform Scripting (topic)
CreateObject (function)
CStr (function)
CSng (function)
Currency (data type)
CurDir, CurDir$ (functions)
CVErr (function)
CVar (function)

D

Date, Date$ (functions)
Date (data type)
DateAdd (function)
Date, Date$ (statements)
DatePart (function)
DateDiff (function)
DateValue (function)
DateSerial (function)
Day (function)
DDB (function)
DDEInitiate (function)
DDERun (statement)
DDERequest, DDERequest$ (functions)

DDEPoke (statement)
DDETerminate (statement)
DDESend (statement)
DDETimeout (statement)
DDETerminateAll (statement)
DefType (statement)
Declare (statement)
Desktop.Cascade (method)
Desktop.ArrangeIcons (method)
Desktop.SetWallpaper (method)
Desktop.SetColors (method)
Desktop.Tile (method)
Desktop.Snapshot (method)
dialingMethod
Disable7BitStrip
DisableEmulatorDisplay
DisableLocalEcho
DisableRemoteEcho
Disconnect
DropSend
Dialog (statement)
Dialog (function)
Dir, Dir$ (functions)
Dim (statement)
DiskFree (function)
DiskDrives (statement)
DlgEnable (function)
DlgControlId (function)
DlgFocus (function)
DlgEnable (statement)
DlgListBoxArray (function)
DlgFocus (statement)
DlgProc (function)
DlgListBoxArray (statement)
DlgText (statement)
DlgSetPicture (statement)
DlgValue (function)
DlgText$ (function)
DlgVisible (function)
DlgValue (statement)
Do...Loop (statement)
DlgVisible (statement)
DoEvents (statement)
DoEvents (function)
Double (data type)
DoKeys (statement)
DropListBox (statement)

E

ebAbort (constant)
ebAbortRetryIgnore (constant)
ebApplicationModal (constant)
ebArchive (constant)
ebBold (constant)
ebBoldItalic (constant)

ebBoolean (constant)
ebCancel (constant)
ebCritical (constant)
ebCurrency (constant)
ebDataObject (constant)
ebError (constant)
ebDate (constant)
ebDefaultButton1 (constant)
ebDefaultButton2 (constant)
ebDefaultButton3 (constant)
ebDirectory (constant)
ebDos (constant)
ebDouble (constant)
ebEmpty (constant)
ebExclamation (constant)
ebHidden (constant)
ebIgnore (constant)
ebInformation (constant)
ebInteger (constant)
ebItalic (constant)
ebLandscape (constant)
ebLeftButton (constant)
ebLong (constant)
ebMaximized (constant)
ebMinimized (constant)
ebNo (constant)
ebNone (constant)
ebNormal (constant)
ebNull (constant)
ebObject (constant)
ebOK (constant)
ebOKCancel (constant)
ebOKOnly (constant)
ebPortrait (constant)
ebQuestion (constant)
ebReadOnly (constant)
ebRegular (constant)
ebRestored (constant)
ebRetry (constant)
ebRetryCancel (constant)
ebRightButton (constant)
ebSingle (constant)
ebString (constant)
ebSystem (constant)
ebSystemModal (constant)
ebVariant (constant)
ebVolume (constant)
ebWindows (constant)
ebYes (constant)
ebYesNo (constant)
ebYesNoCancel (constant)
EditEnabled (function)
EditExists (function)
Empty (constant)
End (statement)
Environ, Environ$ (functions)

EOF (function)
Eqv (operator)
emulator
Enable7BitStrip
EnableLocalEcho
EnableEmulatorDisplay
EnableRemoteEcho
Erase (statement)
Erl (function)
Err (function)
Err (statement)
Error (statement)
Error Handling (topic)
Error, Error$ (functions)
Exit Do (statement)
Exit For (statement)
Exit Function (statement)
Exit Sub (statement)
Exp (function)
Expression Evaluation (topic)

F

False (constant)
FileAttr (function)
FileCopy (statement)
FileDateTime (function)
FileDirs (statement)
FileExists (function)
FileLen (function)
FileList (statement)
FileParse$ (function)
FileType (function)
finalStatus
Fix (function)
For...Next (statement)
Format, Format$ (functions)
frameState
FreeFile (function)
Function...End Function (statement)
Fv (function)

G

Get (statement)
GetAttr (function)
GetCheckBox (function)
GetComboBoxItem$ (function)
GetComboBoxItemCount (function)
GetDataString
GetDropList
GetEditText$ (function)
GetEntry
GetInput
GetListBoxItem$ (function)
GetListBoxItemCount (function)

GetObject (function)
GetOption (function)
GetRuntimeValue
GetSelectedEntry
GetTextFromScreen
Global (statement)
GoSub (statement)
Goto (statement)
GroupBox (statement)

H

Hex, Hex$ (functions)
HLine (statement)
Hour (function)
HPage (statement)
HScroll (statement)
HWND (object)
HWND.Value (property)

I

If...Then...Else (statement)
IIf (function)
Imp (operator)
Inline (statement)
Input# (statement)
Input, Input$ (functions)
InputBox, InputBox$ (functions)
InStr (function)
Int (function)
Integer (data type)
IPmt (function)
IRR (function)
Is (operator)
IsDate (function)
IsEmpty (function)
IsError (function)
IsMissing (function)
IsNull (function)
IsNumeric (function)
IsObject (function)
Item$ (function)
ItemCount (function)

K

Keywords (topic)
Kill (statement)

L

Left, Left$ (functions)
LBound (function)
LCase, LCase$ (functions)
Let (statement)

Len (function)
Line Input# (statement)
Like (operator)
Line$ (function)
Line Numbers (topic)
ListBox (statement)
LineCount (function)
ListBoxExists (function)
ListBoxEnabled (function)
Literals (topic)
Loc (function)
logFileName
logonTask
Lof (function)
Lock (statement)
Long (data type)
Log (function)
LTrim, LTrim$ (functions)
LSet (statement)

M

Main (statement)
Mci (function)
MenuString
messageTimer
MenuItemChecked (function)
Menu (statement)
MenuItemExists (function)
MenuItemEnabled (function)
Mid, Mid$ (statements)
Mid, Mid$ (functions)
MIRR (function)
Minute (function)
Mod (operator)
MkDir (statement)
MsgBox (function)
Month (function)
MsgBox (statement)

N

nameAddress
Name (statement)
Net.Browse$ (method)
Net.AddCon (method)
Net.Dialog (method)
Net.CancelCon (method)
Net.GetCon$ (method)
Net.GetCaps (method)
New (keyword)
Net.User$ (property)
Nothing (constant)
Not (operator)
NPer (function)
Now (function)

Null (constant)
numberReceived
Npv (function)

O

Objects (topic)
Object (data type)
Oct, Oct$ (functions)
OKButton (statement)
OpenSession
Open (statement)
Operator Precedence (topic)
On Error (statement)
Option Base (statement)
OpenFilename$ (function)
OpenSession
Option CStrings (statement)
Operator Precision (topic)
OptionEnabled (function)
Option Compare (statement)
OptionGroup (statement)
OptionButton (statement)
OptionExists (function)
Or (operator)

P

parameters
parity
percentComplete
phoneBookView
Pi (constant)
Picture (statement)
PictureButton (statement)
portDevice
portDeviceName
portName
portType
PopupMenu (function)
Print (statement)
Pmt (function)
PrinterGetOrientation (function)
PPmt (function)
PrintFile (function)
Print# (statement)
Public (statement)
PrinterSetOrientation (statement)
Private (statement)
Put (statement)
PushButton (statement)
Pv (function)

Q

QueKeyDn (statement)

QueEmpty (statement)
QueKeyUp (statement)
QueFlush (statement)
QueMouseDblClk (statement)
QueKeys (statement)
QueMouseDn (statement)
QueMouseClick (statement)
QueMouseMoveBatch (statement)
QueMouseDblDn (statement)
QueSetRelativeWindow (statement)
QueMouseMove (statement)
QueMouseUp (statement)

R

Randomize (statement)
Random (function)
Rate (function)
Receive
receiveProtocol
receivingDirectory
ReleaseRemoteInput
ReloadPhonebook
ReadIni$ (function)
Redim (statement)
ReadIniSection (statement)
Reset (statement)
Rem (statement)
Return (statement)
Resume (statement)
ringsForAnswer
Right, Right$ (functions)
RmDir (statement)
Rnd (function)
RSet (statement)
RTrim, RTrim$ (functions)
RunScript

S

SavePhonebook
SaveFilename$ (function)
Screen.TwipsPerPixelX (property)
Screen.DlgBaseUnitsX (property)
Screen.Width (property)
Screen.Height (property)
Seek (function)
Screen.TwipsPerPixelY (property)
Screen.DlgBaseUnitsY (property)
Send
SendFromList
SendBatch
sendProtocol
sendingDirectory
sessionSize
serialNumber

Second (function)
Seek (statement)
SelectButton (statement)
Select...Case (statement)
SelectListBoxItem (statement)
SelectBox (function)
SelectComboBoxItem (statement)
SendKeys (statement)
Set (statement)
SetAttr (statement)
SetCheckBox (statement)
SetPhoneNumber
SetOption (statement)
SetRuntimeValue
Shell (function)
SetEditText (statement)
SetNameAddress
SetDataString
SetSerialNumber
SizeWinComm
sizePhonebook
Single (data type)
Sgn (function)
SkipConnection
Sln (function)
Sin (function)
Sleep (statement)
Spc (function)
SQLClose (function)
Space, Space$ (functions)
SQLExecQuery (function)
SQLBind (function)
SQLOpen (function)
SQLError (function)
SQLRetrieve (function)
SQLGetSchema (function)
Sqr (function)
SQLRequest (function)
SQLRetrieveToFile (function)
stopBits
String (data type)
Str, Str$ (functions)
Stop (statement)
StrComp (function)
String, String$ (functions)
Sub...End Sub (statement)
Switch (function)
SYD (function)
System.FreeMemory (property)
System.MouseTrails (method)
System.Exit (method)
System.TotalMemory (property)
System.FreeResources (property)
System.WindowsVersion$ (property)
System.Restart (method)
System.WindowsDirectory$ (property)

T

TextSend
TextBox (statement)
Text (statement)
Time, Time$ (statements)
Time, Time$ (functions)
TimeSerial (function)
Timer (function)
Trim, Trim$ (functions)
True (constant)
Transfer
TypeLocalText
Type (statement)
TypeText

U

UBound (function)
UCase, UCase$ (functions)
Unlock (statement)
User-Defined Types (topic)

V

Val (function)
Variant (data type)
VarType (function)
ViewportClear (statement)
ViewportClose (statement)
ViewportOpen (statement)
VLine (statement)
VPage (statement)
VScroll (statement)

W

WaitForConnection
WaitForActivity
WaitForOutputDone
WaitForLines
WaitForPrompt
WaitForLull
WaitForTransfer
WaitForString
Weekday (function)
While...Wend (statement)
winCommVersion
WinCommSleep
Width# (statement)
WinActivate (statement)
WinClose (statement)
WinFind (function)
WinList (statement)
WinMaximize (statement)

WinMinimize (statement)
WinMove (statement)
WinRestore (statement)
WinSize (statement)
Word$ (function)
WordCount (function)
Write# (statement)
WriteLogEntry
WriteIni (statement)

X

Xor (operator)

Y

Year (function)

Constants

ebAbort
ebAbortRetryIgnore
ebApplicationModal
ebArchive
ebBold
ebBoldItalic
ebBoolean
ebCancel
ebCritical
ebCurrency
ebDataObject
ebError
ebDate
ebDefaultButton1
ebDefaultButton2
ebDefaultButton3
ebDirectory
ebDos
ebDouble
ebEmpty
ebExclamation
ebHidden
ebIgnore
ebInformation
ebInteger
ebItalic
ebLandscape
ebLeftButton
ebLong
ebMaximized
ebMinimized
ebNo
ebNone
ebNormal
ebNull
ebObject
ebOK
ebOKCancel
ebOKOnly
ebPortrait
ebQuestion
ebReadOnly
ebRegular
ebRestored
ebRetry
ebRetryCancel
ebRightButton
ebSingle
ebString
ebSystem
ebSystemModal
ebVariant
ebVolume
ebWindows

ebYes
ebYesNo
ebYesNoCancel
Empty
False
Nothing
Null
Pi
True

Keywords

'
-
()
=
ByRef
ByVal
New

Properties

Basic.Eoln$ -
Basic.FreeMemory -
Basic.HomeDir$ -
Basic.Version$ -
Basic.PathSeparator$ -
Net.User$ -
HWND.Value -
Screen.DlgBaseUnitsY -
Screen.DlgBaseUnitsX -
Screen.TwipsPerPixelX -
Screen.Height -
Screen.Width -
Screen.TwipsPerPixelY -
System.FreeResources -
System.FreeMemory -
System.WindowsDirectory$ -
System.TotalMemory -
System.WindowsVersion$ -

Statements

=
ActivateControl
AppActivate
AppClose
AppGetPosition
AppHide
AppList
AppMaximize
AppMinimize
AppMove
AppRestore
AppSetState
AppShow
AppSize
ArraySort
Beep
Begin Dialog
Call
CancelButton
ChDir
ChDrive
CheckBox
Clipboard$
Close
ComboBox
Const
DDERun
DDEPoke
DDESend
DDETerminate
DDETerminateAll
DDETimeout
Declare
DefType
Dialog
Dim
DiskDrives
DlgEnable
DlgFocus
DlgListBoxArray
DlgSetPicture
DlgText
DlgValue
DlgVisible
Do...Loop
DoEvents
DoKeys
DropListBox
End
Erase
Err
Error
Exit Do

Exit For
Exit Function
Exit Sub
FileCopy
FileDirs
FileList
For...Next
Function...End Function
Get
Global
GoSub
Goto
GroupBox
HLine
HPage
HScroll
If...Then...Else
Inline
Input#
Kill
Let
Line Input#
ListBox
Lock
LSet
Main
Menu
MkDir
MsgBox
Name
OKButton
On Error
Open
Option Base
Option Compare
Option CStrings
OptionButton
OptionGroup
Picture
PictureButton
Print
Print#
PrinterSetOrientation
Private
Public
PushButton
Put
QueEmpty
QueFlush
QueKeyDn
QueKeys
QueKeyUp
QueMouseClick
QueMouseDblClk
QueMouseDblDn
QueMouseDn

QueMouseMove
QueMouseMoveBatch
QueMouseUp
QueSetRelativeWindow
Randomize
ReadIniSection
Redim
Rem
Reset
Resume
Return
RmDir
RSet
Seek
Select...Case
SelectButton
SelectComboBoxItem
SelectListBoxItem
SendKeys
Set
SetAttr
SetCheckBox
SetEditText
SetOption
Sleep
Stop
Sub...End Sub
Text
TextBox
Type
Unlock
ViewportClear
ViewportClose
ViewportOpen
VLine
VPage
VScroll
While...Wend
Width#
WinActivate
WinClose
WinList
WinMaximize
WinMinimize
WinMove
WinRestore
WinSize
Write#
WriteIni

Data Types

Any
Boolean
Currency
Date
Double
Integer
Long
Object
Single
String
Variant

Functions

Abs
AnswerBox
AppFilename$
AppFind$
AppGetActive$
AppGetState
AppType
ArrayDims
Asc
AskBox$
AskPassword$
Atn
ButtonEnabled
ButtonExists
CBool
CCur
CDbl
CheckBoxEnabled
CheckBoxExists
Choose
CInt
Clipboard$
CLng
ComboBoxEnabled
ComboBoxExists
Cos
CreateObject
CSng
CStr
CVar
CVErr
DateAdd
DateDiff
DatePart
DateSerial
DateValue
Day
DDB
DDEInitiate
Dialog
DiskFree
DlgControlId
DlgEnable
DlgFocus
DlgListBoxArray
DlgProc
DlgText$
DlgValue
DlgVisible
DoEvents
EditEnabled
EditExists
EOF

Erl
Err
Exp
FileAttr
FileDateTime
FileExists
FileLen
FileParse$
FileType
Fix
FreeFile
Fv
GetAttr
GetCheckBox
GetComboBoxItem$
GetComboBoxItemCount
GetEditText$
GetListBoxItem$
GetListBoxItemCount
GetObject
GetOption
Hour
Iif
InStr
Int
IPmt
IRR
IsDate
IsEmpty
IsError
IsMissing
IsNull
IsNumeric
IsObject
Item$
ItemCount
LBound
Len
Line$
LineCount
ListBoxEnabled
ListBoxExists
Loc
Lof
Log
Mci
MenuItemChecked
MenuItemEnabled
MenuItemExists
Minute
MIRR
Month
MsgBox
Now
NPer
Npv

OpenFilename$
OptionEnabled
OptionExists
Pmt
PopupMenu
PPmt
PrinterGetOrientation
PrintFile
Pv
Random
Rate
ReadIni$
Rnd
SaveFilename$
Second
Seek
SelectBox
Sgn
Shell
Sin
Sln
Spc
SQLBind
SQLClose
SQLError
SQLExecQuery
SQLGetSchema
SQLOpen
SQLRequest
SQLRetrieve
SQLRetrieveToFile
Sqr
StrComp
Switch
SYD
Tab
Tan
Timer
TimeSerial
TimeValue
UBound
Val
VarType
Weekday
WinFind
Word$
WordCount

Operators

&
*
=
+
=
-
=
/
=
<
=
<=
=
<>
=
=
=
>
=
>=
=
\
=
^
=
And
Eqv
Imp
Is
Like
Mod
Not
Or

Methods

Basic.Capability
Clipboard.Clear
Clipboard.GetFormat
Clipboard.GetText
Clipboard.SetText
Desktop.ArrangeIcons
Desktop.Cascade
Desktop.SetColors
Desktop.SetWallpaper
Desktop.Snapshot
Desktop.Tile
Net.AddCon
Net.Browse$
Net.CancelCon
Net.Dialog
Net.GetCaps
Net.GetCon$
System.Exit
System.MouseTrails
System.Restart

Topics

Arrays
Comments
Comparison Operators
Constants
CrossPlatform Scripting
Error Handling
Expression Evaluation
Keywords
Line Numbers
Literals
Objects
Operator Precedence
Operator Precision
UserDefined Types

Objects

HWND

WinComm Extensions

AddToSendList
application
ascii7BitReceive
asciiAppendLFReceive
asciiAppendLFSend
asciiCharDelayValue
asciiLineDelayValue
asciiExpandBlankToSpace
asciiInputTabValue
asciiInputWaitChar
asciiLocalEcho
asciiOutputTabValue
asciiRemoteEcho
asciiShowHex
asciiTabExpandSend
asciiWaitForLineEnd
asciiWrapLines
baudRate
bitsPerCharacter
BlockRemoteInput
CaptureToFileBegin
CaptureToFileControl
CaptureToPrinterBegin
CaptureToPrinterControl
ClearDropList
ClearOutputBuffer
ClearSendList
Close
Connect
ConnectAndDial
connectionStatus
connectionTime
dialingMethod
Disable7BitStrip
DisableEmulatorDisplay
DisableLocalEcho
DisableRemoteEcho
Disconnect
DropSend
emulator
Enable7BitStrip
EnableEmulatorDisplay
EnableLocalEcho
EnableRemoteEcho
finalStatus
frameState
GetDataString
GetDropList
GetEntry
GetInput
GetRuntimeValue
GetSelectedEntry
GetTextFromScreen

logFileName
logonTask
MenuString
messageTimer
nameAddress
numberReceived
OpenSession
parameters
parity
percentComplete
phoneBookView
portDevice
portDeviceName
portName
portType
Receive
receiveProtocol
receivingDirectory
ReleaseRemoteInput
ReloadPhonebook
ringsForAnswer
RunScript
SavePhonebook
Send
SendBatch
SendFromList
sendingDirectory
sendProtocol
serialNumber
sessionSize
SetDataString
SetNameAddress
SetPhoneNumber
SetRuntimeValue
SetSerialNumber
sizePhonebook
SizeWinComm
SkipConnection
stopBits
TextSend
Transfer
TypeLocalText
WaitForActivity
WaitForConnection
WaitForLines
WaitForOutputDone
WaitForLull
WaitForPrompt
WaitForString
WaitForTransfer
WinCommSleep
winCommVersion
WriteLogEntry

operator

&

Syntax

```
expression1 & expression2
```

Description

Returns the concatenation of expression1 and expression2.

Comments

- If both expressions are strings, then the type of the result is String. Otherwise, the type of the result is a String variant.
- When nonstring expressions are encountered, each expression is converted to a String variant. If both expressions are Null, then a Null variant is returned. If only one expression is Null, then it is treated as a zero-length string. Empty variants are also treated as zero-length strings.
- In many instances, the plus (+) operator can be used in place of &. The difference is that + attempts addition when used with at least one numeric expression, whereas & always concatenates.

Example

```
'This example assigns a concatenated string to variable s$ and a string to
's2$, then concatenates the two variables and displays the result in a
'dialog box.

Sub Main()
    s$ = "This string" & " is concatenated"
    s2$ = " with the '&' operator."
    MsgBox s$ & s2$
End Sub
```

See Also

+ (operator); Operator Precedence (topic).

Keyword
'

Syntax

```
'text
```

Description

Causes the compiler to skip all characters between this character and the end of the current line.

Comments

- This is very useful for commenting your code to make it more readable.

Example

```
Sub Main()
    'This whole line is treated as a comment.
    i$ = "Strings"        'This is a valid assignment with a comment.
    This line will cause an error (the apostrophe is missing).
End Sub
```

See Also

Rem (statement); Comments (topic).

Keyword

()

Syntax 1

```
...(expression)...
```

Syntax 2

```
...,(parameter),...
```

Description

Forces parts of an expression to be evaluated before others or forces a parameter to be passed by value.

Comments

Parentheses within Expressions

Parentheses override the normal precedence order of Delrina Basic operators, forcing a subexpression to be evaluated before other parts of the expression. For example, the use of parentheses in the following expressions causes different results:

```
i = 1 + 2 * 3        'Assigns 7.
i = (1 + 2) * 3      'Assigns 9.
```

Use of parentheses can make your code easier to read, removing any ambiguity in complicated expressions.

Parentheses Used in Parameter Passing

Parentheses can also be used when passing parameters to functions or subroutines to force a given parameter to be passed by value, as shown below:

```
ShowForm i           'Pass i by reference.
ShowForm (i)         'Pass i by value.
```

Enclosing parameters within parentheses can be misleading. For example, the following statement appears to be calling a function called ShowForm without assigning the result:

```
ShowForm(i)
```

The above statement actually calls a subroutine called ShowForm, passing it the variable i by value. It may be clearer to use the ByVal keyword in this case, which accomplishes the same thing:

```
ShowForm ByVal i
```

Note: The result of an expression is always passed by value.

Example

```
'This example uses parentheses to clarify an expression.

Sub Main()
    bill = False
    dave = True
    jim = True
```

```
    If (dave And bill) Or (jim And bill) Then
        Msgbox "The required parties for the meeting are here."
    Else
        MsgBox "Someone is late for the meeting!"
    End If
End Sub
```

See Also

ByVal (keyword); Operator Precedence (topic).

operator

*

Syntax

```
expression1 * expression2
```

Description

Returns the product of expression1 and expression2.

Comments

- The result is the same type as the most precise expression, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
Single	Long	Double
Boolean	Boolean	Integer
Date	Date	Double

- When the * operator is used with variants, the following additional rules apply:
- Empty is treated as 0.
- If the type of the result is an Integer variant that overflows, then the result is automatically promoted to a Long variant.
- If the type of the result is a Single, Long, or Date variant that overflows, then the result is automatically promoted to a Double variant.
- If expression1 is Null and expression2 is Boolean, then the result is Empty. Otherwise, If either expression is Null, then the result is Null.

Example

```
'This example assigns values to two variables and their product to
'a third variable, then displays the product of s# * t#.

Sub Main()
    s# = 123.55
    t# = 2.55
    u# = s# * t#
    MsgBox s# & " * " & t# & " = " & s# * t#
End Sub
```

See Also

Operator Precedence (topic).

operator

+

Syntax

```
expression1 + expression2
```

Description

Adds or concatenates two expressions.

Comments

- Addition operates differently depending on the type of the two expressions:

If one expression is	and the other expression is	then
Numeric	Numeric	Perform a numeric add (see below).
String	String	Concatenate, returning a string.
Numeric	String	A runtime error is generated.
Variant	String	Concatenate, returning a String variant.
Variant	Numeric	Perform a variant add (see below).
Empty variant	Empty variant	Return an Integer variant, value 0.
Empty variant	Boolean variant	Return an Integer variant (value 0 or -1)
Empty variant	Any data type	Return the non-Empty expression unchanged.
Null variant	Any data type	Return Null.
Variant	Variant	If either is numeric, add; otherwise, concatenate.

- When using + to concatenate two variants, the result depends on the types of each variant at runtime. You can remove any ambiguity by using the & operator.

Numeric Add

A numeric add is performed when both expressions are numeric (i.e., not variant or string). The result is the same type as the most precise expression, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
Single	Long	Double
Boolean	Boolean	Integer

A runtime error is generated if the result overflows its legal range.

Variant Add

If both expressions are variants, or one expression is numeric and the other expression is Variant, then a variant add is performed. The rules for variant add are the same as those for normal numeric add, with the following exceptions:

If the type of the result is an Integer variant that overflows, then the result is a Long variant.

If the type of the result is a Long, Single, or Date variant that overflows, then the result is a Double variant.

Example

```
'This example assigns string and numeric variable values and
'then uses the + operator to concatenate the strings and form the
'sums of numeric variables.

Sub Main()
      i$ = "concatenate " + "strings!"
      j% = 95 + 5          'Addition of numeric literals
      k# = j% + j%         'Addition of numeric variable
      MsgBox "You can " + i$
      MsgBox "You can add literals or variables:" + Str(j%) + ", " + Str(k#)
End Sub
```

See Also

& (operator); Operator Precedence (topic).

operator

-

Syntax 1

```
expression1 - expression2
```

Syntax 2

```
-expression
```

Description

Returns the difference between expression1 and expression2 or, in the second syntax, returns the negation of expression.

Comments

Syntax 1

- The type of the result is the same as that of the most precise expression, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
Long	Single	Double
Boolean	Boolean	Integer

- A runtime error is generated if the result overflows its legal range.
- When either or both expressions are Variant, then the following additional rules apply:
- If expression1 is Null and expression2 is Boolean, then the result is Empty. Otherwise, if either expression is Null, then the result is Null.
- Empty is treated as an Integer of value 0.
- If the type of the result is an Integer variant that overflows, then the result is a Long variant.
- If the type of the result is a Long, Single, or Date variant that overflows, then the result is a Double variant.

Syntax 2

- If expression is numeric, then the type of the result is the same type as expression, with the following exception:
- If expression is Boolean, then the result is Integer.

Note: In 2's compliment arithmetic, unary minus may result in an overflow with Integer and Long variables when the value of expression is the largest negative number representable for that data type. For example, the following generates an overflow error:

```
Sub Main()
    Dim a As Integer
    a = -32768
    a = -a                  'Generates overflow here.
End Sub
```

When negating variants, overflow will never occur because the result will be automatically promoted: integers to longs and longs to doubles.

Example

```
'This example assigns values to two numeric variables and their
'difference to a third variable, then displays the result.

Sub Main()
    i% = 100
    j# = 22.55
    k# = i% - j#
    MsgBox "The difference is: " & k#
End Sub
```

See Also

Operator Precedence (topic).

Keyword

.

Syntax 1

```
object.property
```

Syntax 2

```
structure.member
```

Description

Separates an object from a property or a structure from a structure member.

Examples

'This example uses the period to separate an object from a property.

```
Sub Main()
    MsgBox "The clipboard text is: " & Clipboard.GetText()
End Sub
```

'This example uses the period to separate a structure from a member.

```
Type Rect
    left As Integer
    top As Integer
    right As Integer
    bottom As Integer
End Type

Sub Main()
    Dim r As Rect
    r.left = 10
    r.right = 12
    Msgbox "r.left = " & r.left & ", r.right = " & r.right
End Sub
```

See Also

Objects (topic).

operator

/

Syntax

```
expression1 / expression2
```

Description

Returns the quotient of expression1 and expression2.

Comments

- The type of the result is Double, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
Integer	Integer	Single
Single	Single	Single
Boolean	Boolean	Single

- A runtime error is generated if the result overflows its legal range.
- When either or both expressions is Variant, then the following additional rules apply:
- If expression1 is Null and expression2 is Boolean, then the result is Empty. Otherwise, if either expression is Null, then the result is Null.
- Empty is treated as an Integer of value 0.
- If both expressions are either Integer or Single variants and the result overflows, then the result is automatically promoted to a Double variant.

Example

```
'This example assigns values to two variables and their quotient to a
'third variable, then displays the result.

Sub Main()
    i% = 100
    j# = 22.55
    k# = i% / j#
    MsgBox "The quotient of i/j is: " & k#
End Sub
```

See Also

\ (operator); Operator Precedence (topic).

operator

<

See Comparison Operators (topic).

operator

<=

See Comparison Operators (topic).

operator

<>

See Comparison Operators (topic).

statement

=

Syntax

```
variable = expression
```

Description

Assigns the result of an expression to a variable.

Comments

When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This occurs when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:

```
Dim amount As Long
Dim quantity As Integer

amount = 400123             'Assign a value out of range for int.
quantity = amount       'Attempt to assign to Integer.
```

When performing an automatic data conversion, underflow is not an error.

The assignment operator (=) cannot be used to assign objects. Use the Set statement instead.

Example

```
Sub Main()
    a$ = "This is a string"
    b% = 100
    c# = 1213.3443
    MsgBox a$ & "," & b% & "," & c#
End Sub
```

See Also

Let (statement); Operator Precedence (topic); Set (statement); Expression Evaluation (topic).

operator

=

See Comparison Operators (topic).

operator

>

See Comparison Operators (topic).

operator

>=

See Comparison Operators (topic).

operator

\

Syntax

```
expression1 \ expression2
```

Description

Returns the integer division of expression1 and expression2.

Comments

- Before the integer division is performed, each expression is converted to the data type of the most precise expression. If the type of the expressions is either Single, Double, Date, or Currency, then each is rounded to Long.
- If either expression is a Variant, then the following additional rules apply:
- If either expression is Null, then the result is Null.
- Empty is treated as an Integer of value 0.

Example

```
'This example assigns the quotient of two literals to a variable and
'displays the result.

Sub Main()
    s% = 100.99 \ 2.6
    MsgBox "Integer division of 100.99\2.6 is: " & s%
End Sub
```

See Also

/ (operator); Operator Precedence (topic).

operator
^

Syntax

```
expression1 ^ expression2
```

Description

Returns expression1 raised to the power specified in expression2.

Comments

- The following are special cases:

Special Case	Value
n^0	1
0^-n	Undefined
0^+n	0
1^n	1

- The type of the result is always Double, except with Boolean expressions, in which case the result is Boolean. Fractional and negative exponents are allowed.
- If either expression is a Variant containing Null, then the result is Null.
- It is important to note that raising a number to a negative exponent produces a fractional result.

Example

```
Sub Main()
    s# = 2 ^ 5              'Returns 2 to the 5th power.
    r# = 16 ^ .5            'Returns the square root of 16.
    MsgBox "2 to the 5th power is: " & s#
    MsgBox "The square root of 16 is: " & r#
End Sub
```

See Also

Operator Precedence (topic).

keyword

_

Syntax

```
s$ = "This is a very long line that I want to split " & _
     "onto two lines"
```

Description

Line-continuation character, which lets you to split a single Delrina Basic statement onto more than one line.

Comments

- The line-continuation character cannot be used within strings and must be preceded by white space (either a space or a tab).
- The line-continuation character can be followed by a comment, as shown below:

```
i = 5 + 6 & _        'Continue on the next line.
"Hello"
```

Example

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    'The line-continuation operator is useful when concatenating
    'long strings.

    msg = "This line is a line of text that" & crlf & "extends beyond " _
          & "the borders of the editor" & crlf & "so it is split into " _
          & "multiple lines"

    'It is also useful for separating and continuing long calculation lines.

    b# = .124
    a# = .223
    s# = ( (((Sin(b#) ^ 2) + (Cos(a#) ^ 2)) ^ .5) / _
           (((Sin(a#) ^ 2) + (Cos(b#) ^ 2)) ^ .5) ) * 2.00
    MsgBox msg & crlf & crlf & "The value of s# is: " & s#
End Sub
```

function

Abs

Syntax

```
Abs(expression)
```

Description

Returns the absolute value of expression.

Comments

- If expression is Null, then Null is returned. Empty is treated as 0.
- The type of the result is the same as that of expression, with the following exceptions:
- If expression is an Integer that overflows its legal range, then the result is returned as a Long. This only occurs with the largest negative Integer:

```
Dim a As Variant
Dim i As Integer
i = -32768
a = Abs(i)          'Result is a Long.
i = Abs(i)          'Overflow!
```

- If expression is a Long that overflows its legal range, then the result is returned as a Double. This only occurs with the largest negative Long:

```
Dim a As Variant
Dim l As Long
l = -2147483648
a = Abs(l)          'Result is a Double.
l = Abs(l)          'Overflow!
```

If expression is a Currency value that overflows its legal range, an overflow error is generated.

Example

```
'This example assigns absolute values to variables of four types and
'displays the result.

Sub Main()
    s1% = Abs(-10.55)
    s2& = Abs(-10.55)
    s3! = Abs(-10.55)
    s4# = Abs(-10.55)
    MsgBox "The absolute values are: " & s1% & "," & s2& & "," & s3! & "," & s4#
End Sub
```

See Also

Sgn (function).

statement

ActivateControl

Syntax

```
ActivateControl control
```

Description

Sets the focus to the control with the specified name or ID.

Comments

The control parameter specifies either the name or the ID of the control to be activated, as shown in the following table:

If control is	Then
String	A control associated with that name is activated. For push buttons, option buttons, or check boxes, the control with this name is activated. For list boxes, combo boxes, and text boxes, the control that immediately follows the text control with this name is activated.
Numeric	A control with this ID is activated. The ID is first converted to an Integer.

The ActivateControl statement generates a runtime error if the dialog control referenced by control cannot be found.

You can use the ActivateControl statement to set the focus to a custom control within a dialog box. First, set the focus to the control that immediately precedes the custom control, then simulate a Tab keypress, as in the following example:

```
ActivateControl "Portrait"
DoKeys "{TAB}"
```

Note: The ActivateControl statement is used to activate a control in another application's dialog box. Use the DlgFocus statement to activate a control in a dynamic dialog box.

Example

```
'This example runs Notepad using Program Manager's Run command. It uses the
'ActivateControl command to switch focus between the different controls of
'the Run dialog box.

Sub Main()
    If AppFind$("Program Manager") = "" Then Exit Sub
    AppActivate "Program Manager"
    Menu "File.Run"
    SendKeys "Notepad"
    ActivateControl "Run minimized"
    SendKeys " "
    ActivateControl "OK"
    SendKeys "{Enter}"
End Sub
```

See Also

DlgFocus (statement).

operator

And

Syntax

```
expression1 And expression2
```

Description

Performs a logical or binary conjunction on two expressions.

Comments

If both expressions are either Boolean, Boolean variants, or Null variants, then a logical conjunction is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	Null
Null	True	Null
Null	False	False
Null	Null	Null

Binary Conjunction

If the two expressions are Integer, then a binary conjunction is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long, and a binary conjunction is then performed, returning a Long result.

Binary conjunction forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

1	And	1	=	1	Example:
0	And	1	=	0	5 00001001
1	And	0	=	0	6 00001010
0	And	0	=	0	And 00001000

Example

```
Sub Main()
    n1 = 1001
    n2 = 1000
    b1 = True
    b2 = False
    'This example performs a numeric bitwise And operation and stores the
    'result in N3.
    n3 = n1 And n2
```

```
    'This example performs a logical And comparing b1 and b2 and displays the
    'result.
    If b1 And b2 Then
        MsgBox "b1 And b2 are True; n3 is: " & n3
    Else
        MsgBox "b1 And b2 are False; n3 is: " & n3
    End If
End Sub
```

See Also

Operator Precedence (topic); Or (operator); Xor (operator); Eqv (operator); Imp (operator).

function

AnswerBox

Syntax

```
AnswerBox(prompt [,[button1] [,[button2] [,button3]]]])
```

Description

Displays a dialog box prompting the user for a response and returns an Integer indicating which button was clicked (1 for the first button, 2 for the second, and so on).

Comments

The AnswerBox function takes the following parameters:

Parameter	Description
prompt	Text to be displayed above the text box. The prompt parameter can be any expression convertible to a String. Delrina BasicScript resizes the dialog box to hold the entire contents of prompt, up to a maximum width of 5/8 of the width of the screen and a maximum height of 5/8 of the height of the screen. Delrina Basic word-wraps any lines too long to fit within the dialog box and truncates all lines beyond the maximum number of lines that fit in the dialog box. You can insert a carriage-return/line-feed character in a string to cause a line break in your message. A runtime error is generated if this parameter is Null.
button1	Text for the first button. If omitted, then "OK" and "Cancel" are used. A runtime error is generated if this parameter is Null.
button2	Text for the second button. A runtime error is generated if this parameter is Null.
button3	Text for the third button. A runtime error is generated if this parameter is Null.

The width of each button is determined by the width of the widest button.

The AnswerBox function returns 0 if the user selects Cancel.

```
r% = AnswerBox("Copy files?")
r% = AnswerBox("Copy files?","Save","Restore","Cancel")
```

Example

```
'This example displays a dialog box containing three buttons. It displays
'an additional message based on which of the three buttons is selected.

Sub Main()
    r% = AnswerBox("Temporary File Operation?","Save","Remove","Cancel")
    Select Case r%
        Case 1
            MsgBox "Files will be saved."
        Case 2
            MsgBox "Files will be removed."
        Case Else
            MsgBox "Operation canceled."
    End Select
End Sub
```

See Also

MsgBox (statement); AskBox$ (function); AskPassword$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SaveFilename$ (function); SelectBox (function).

Note

- AnswerBox displays all text in its dialog box in 8-point MS Sans Serif.

data type

Any

Description

Used with the Declare statement to indicate that type checking is not to be performed with a given argument.

Comments

Given the following declaration:

```
Declare Sub Foo Lib "FOO.DLL" (a As Any)
```

the following calls are valid:

```
Foo 10
Foo "Hello, world."
```

Example

```
'The following example calls the FindWindow to determine if Program Manager
'is running.

'This example uses the Any keyword to pass a NULL pointer, which is accepted
'by the FindWindow function.

Declare Function FindWindow16 Lib "user" Alias "FindWindow" (ByVal Class _
    As Any,ByVal Title As Any) As Integer

Declare Function FindWindow32 Lib "user32" Alias "FindWindowA" (ByVal Class _
    As Any,ByVal Title As Any) As Long

Sub Main()
    Dim hWnd As Variant

    If Basic.Os = ebWin16 Then
        hWnd = FindWindow16("PROGMAN",0&)
    ElseIf Basic.Os = ebWin32 Then
        hWnd = FindWindow32("PROGMAN",0&)
    Else
        hWnd = 0
    End If

    If hWnd <> 0 Then
        MsgBox "Program manager is running, window handle is " & hWnd
    End If
End Sub
```

See Also

Declare (statement).

statement

AppActivate

Syntax

```
AppActivate name$ | taskID
```

Description

Activates an application given its name or task ID.

Comments

- The AppActivate statement takes the following parameters:

Parameter	Description
name$	String containing the name of the application to be activated.
taskID	Number specifying the task ID of the application to be activated. Acceptable task IDs are returned by the Shell function.

Note: When activating applications using the task ID, it is important to declare the variable used to hold the task ID as a Variant. The type of the ID depends on the platform on which Delrina Basic is running.

Examples

```
'This example activates Program Manager.

Sub Main()
    AppActivate "Program Manager"
End Sub



'This example runs another application, activates it, and maximizes it.

Sub Main()
    Dim id as variant
    id = Shell("notepad.exe")          'Run Notepad minimized.
    AppActivate id                         'Now activate Notepad.
    AppMaximize
End Sub
```

See Also

Shell (function); SendKeys (statement); WinActivate (statement).

Notes

- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- Minimized applications are not restored before activation. Thus, activating a minimized DOS application will not restore it; rather, it will highlight its icon.
- A runtime error results if the window being activated is not enabled, as is the case if that application is currently displaying a modal dialog box.

statement

AppClose

Syntax

```
AppClose [name$]
```

Description

Closes the named application.

Comments

- The name$ parameter is a String containing the name of the application. If the name$ parameter is absent, then the AppClose statement closes the active application.

Example

```
'This example activates Excel, then closes it.

Sub Main()
    If AppFind$("Microsoft Excel") = "" Then 'Make sure Excel is there.
        MsgBox "Excel is not running."
        Exit Sub
    End If
    AppActivate "Microsoft Excel"                'Activate it (unnecessary).
    AppClose "Microsoft Excel"                       'Close it.
End Sub
```

See Also

AppMaximize (statement); AppMinimize (statement); AppRestore (statement); AppMove (statement); AppSize (statement).

Notes

- A runtime error results if the application being closed is not enabled, as is the case if that application is currently displaying a modal dialog box.
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.

function

AppFilename$

Syntax

```
AppFilename$[([name$])]
```

Description

Returns the filename of the named application.

Comments

- The name$ parameter is a String containing the name of the desired application. If the name$ parameter is omitted, then the AppFilename$ function returns the filename of the active application.

Example

```
'This example switches the focus to Excel, then changes the current directory
'to be the same as that of Excel.

Sub Main()
    If AppFind$("Microsoft Excel") = "" Then 'Make sure Excel is there.
        MsgBox "Excel is not running."
        Exit Sub
    End If
    AppActivate "Microsoft Excel"   'Activate Excel.
    s$ = AppFilename$                   'Find where the Excel executable is.
    d$ = FileParse$(s$,2)               'Get the path portion of the filename.
    MsgBox d$                               'Display directory name.
End Sub
```

See Also

AppFind$ (function).

Notes

- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.

function

AppFind$

Syntax

```
AppFind$(partial_name$)
```

Description

Returns a String containing the full name of the application matching the partial_name$.

Comments

- The partial_name$ parameter specifies the title of the application to find. If there is no exact match, Delrina Basic will find an application whose title begins with partial_name$.
- AppFind$ returns a zero-length string if the specified application cannot be found.
- AppFind$ is generally used to determine whether a given application is running. The following expression returns True if Microsoft Word is running:

  AppFind$("Microsoft Word"**)**

Example

```
'This example checks to see whether Excel is running before activating it.

Sub Main()
    If AppFind$("Microsoft Excel") <> "" Then
        AppActivate "Microsoft Excel"
    Else
        MsgBox "Excel is not running."
    End If
End Sub
```

See Also

AppFileName$ (function).

Note

- This function returns a String containing the exact text appearing in the title bar of the active application's main window.

function

AppGetActive$

Syntax

AppGetActive$()

Description

Returns a String containing the name of the application.

Comments

- If no application is active, the AppGetActive$ function returns a zero-length string.
- You can use AppGetActive$ to retrieve the name of the active application. You can then use this name in calls to routines that require an application name.

Example

```
Sub Main()
    n$ = AppGetActive$()
    AppMinimize n$
End Sub
```

See Also

AppActivate (statement); WinFind (function).

Note

- This function returns a String containing the exact text appearing in the title bar of the active application's main window.

statement

AppGetPosition

Syntax

```
AppGetPosition X,Y,width,height [,name$]
```

Description

Retrieves the position of the named application.

Comments

- The AppGetPosition statement takes the following parameters:

Parameter	Description
X, Y	Names of Integer variables to receive the position of the application's window.
width, height	Names of Integer variables to receive the size of the application's window.
name$	String containing the name of the application. If the name$ parameter is omitted, then the active application is used.

- The x, y, width, and height variables are filled with the position and size of the application's window. If an argument is not a variable, then the argument is ignored, as in the following example, which only retrieves the x and y parameters and ignores the width and height parameters:

```
Dim x As Integer,y As Integer
AppGetPosition x,y,0,0,"Program Manager"
```

Example

```
Sub Main()
    Dim x As Integer,y As Integer
    Dim cx As Integer,cy As Integer
    AppGetPosition x,y,cx,cy,"Program Manager"
End Sub
```

See Also

AppMove (statement); AppSize (statement).

Note

- The position and size of the window are returned in twips.
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.

function

AppGetState

Syntax

```
AppGetState[([name$])]
```

Description

Returns an Integer specifying the state of the top-level window.

Comments

- The AppGetState function returns any of the following values:

If the window is	then AppGetState returns
Maximized	ebMaximized
Minimized	ebMinimized
Restored	ebRestored

- The name$ parameter is a String containing the name of the desired application. If it is omitted, then the AppGetState function returns the name of the active application.

Examples

```
'This example saves the state of Program Manager, changes it, then restores
'it to its original setting.

Sub Main()
    If AppFind$("Program Manager") = "" Then
        MsgBox "Can't find Program Manager."
        Exit Sub
    End If
    AppActivate "Program Manager"              'Activate Program Manager.
    state = AppGetState                        'Save its state.
    AppMinimize                                'Minimize it.
    MsgBox "Program Manager is now minimized. Select OK to restore it."
    AppActivate "Program Manager"
    AppSetState state                              'Restore it.
End Sub
```

See Also

AppMaximize (statement); AppMinimize (statement); AppRestore (statement).

Note

- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.

statement

AppHide

Syntax

```
AppHide [name$]
```

Description

Hides the named application.

Comments

- If the named application is already hidden, the AppHide statement will have no effect.
- The name$ parameter is a String containing the name of the desired application. If it is omitted, then the AppHide statement hides the active application.
- AppHide generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

Example

```
'This example hides Program Manager.

Sub Main()
    'See whether Program Manager is running.
    If AppFind$("Program Manager") = "" Then Exit Sub
    AppHide "Program Manager"
    MsgBox "Program Manager is now hidden. Press OK to show it once again."
    AppShow "Program Manager"
End Sub
```

See Also

AppShow (statement).

Note

- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.

statement

AppList

Syntax

```
AppList AppNames$()
```

Description

Fills an array with the names of all open applications.

Comments

- The AppNames$ parameter must specify either a zero- or one-dimensioned dynamic String array or a one-dimensional fixed String array. If the array is dynamic, then it will be redimensioned to match the number of open applications. For fixed arrays, AppList first erases each array element, then begins assigning application names to the elements in the array. If there are fewer elements than will fit in the array, then the remaining elements are unused. Delrina Basic returns a runtime error if the array is too small to hold the new elements.
- After calling this function, you can use LBound and UBound to determine the new size of the array.

Example

```
'This example minimizes all applications on the desktop.

Sub Main()
    Dim apps$()
    AppList apps

    'Check to see whether any applications were found.
    If ArrayDims(apps) = 0 Then Exit Sub

    For i = LBound(apps) To UBound(apps)
        AppMinimize apps(i)
    Next i
End Sub
```

See Also

WinList (statement).

Note

- The name of an application is considered to be the exact text that appears in the title bar of the application's main window.

statement

AppMaximize

Syntax

```
AppMaximize [name$]
```

Description

Maximizes the named application.

Comments

- The name$ parameter is a String containing the name of the desired application. If it is omitted, then the AppMaximize function maximizes the active application.

Example

```
Sub Main()
    AppMaximize "Program Manager"       'Maximize Program Manager.

    If AppFind$("NotePad") <> "" Then
        AppActivate "NotePad"           'Set the focus to NotePad.
        AppMaximize                         'Maximize it.
    End If
End Sub
```

See Also

AppMinimize (statement); AppRestore (statement); AppMove (statement); AppSize (statement); AppClose (statement).

Notes

- If the named application is maximized or hidden, the AppMaximize statement will have no effect.
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- AppMaximize generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

statement

AppMinimize

Syntax

```
AppMinimize [name$]
```

Description

Minimizes the named application.

Comments

- The name$ parameter is a String containing the name of the desired application. If it is omitted, then the AppMinimize function minimizes the active application.

Example

```
Sub Main()
    AppMinimize "Program Manager"        'Maximize Program Manager.

    If AppFind$("NotePad") <> "" Then
        AppActivate "NotePad"            'Set the focus to NotePad.
        AppMinimize                          'Maximize it.
    End If
End Sub
```

See Also

AppMaximize (statement); AppRestore (statement); AppMove (statement); AppSize (statement); AppClose (statement).

Notes

- If the named application is minimized or hidden, the AppMinimize statement will have no effect.
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- AppMinimize generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

statement

AppMove

Syntax

```
AppMove X, Y [,name$]
```

Description

Sets the upper left corner of the named application to a given location.

Comments

- The AppMove statement takes the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the upper left corner of the new location of the application, relative to the upper left corner of the display.
name$	String containing the name of the application to move. If this parameter is omitted, then the active application is moved.

Example

```
'This example activates Program Manager, then moves it 10 pixels to the
'right.

Sub Main()
    Dim x%,y%
    AppActivate "Program Manager"              'Activate Program Manager.
    AppGetPosition x%,y%,0,0                   'Retrieve its position.
    x% = x% + Screen.TwipsPerPixelX * 10       'Add 10 pixels.
    AppMove x% + 10,y%                         'Nudge it 10 pixels to the right.
End Sub
```

See Also

AppMaximize (statement); AppMinimize (statement); AppRestore (statement); AppSize (statement); AppClose (statement).

Notes

- If the named application is maximized or hidden, the AppMove statement will have no effect.
- The X and Y parameters are specified in twips.
- AppMove will accept X and Y parameters that are off the screen.
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- AppMove generates a runtime error if the named application is not enabled, as is the case if that application is currently displaying a modal dialog box.

statement

AppRestore

Syntax

```
AppRestore [name$]
```

Description

Restores the named application.

Comments

- The name$ parameter is a String containing the name of the application to restore. If this parameter is omitted, then the active application is restored.

Example

```
'This example minimizes Program Manager, then restores it.

Sub Main()
    If AppFind$("Program Manager") = "" Then Exit Sub
    AppActivate "Program Manager"
    AppMinimize "Program Manager"
    MsgBox "Program Manager is now minimized. Press OK to restore it."
    AppRestore "Program Manager"
End Sub
```

See Also

AppMaximize (statement); AppMinimize (statement); AppMove (statement); AppSize (statement); AppClose (statement).

Notes

- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- AppRestore will have an effect only if the main window of the named application is either maximized or minimized.
- AppRestore will have no effect if the named window is hidden.
- AppRestore generates a runtime error if the named application is not enabled, as is the case if that application is currently displaying a modal dialog box.

statement

AppSetState

Syntax

```
AppSetState newstate [,name$]
```

Description

Maximizes, minimizes, or restores the named application, depending on the value of newstate.

Comments

- The AppSetState statement takes the following parameters:

Parameter	Description
newstate	Integer specifying the new state of the window. It can be any of the following values:

Value	Description
ebMaximized	The named application is maximized.
ebMinimized	The named application is minimized.
ebRestored	The named application is restored.
name$	String containing the name of the application to change. If this parameter is omitted, then the active application is used.

Example

```
'This example saves the state of Program Manager, changes it, then restores
'it to its original setting.

Sub Main()
    If AppFind$("Program Manager") = "" Then
        MsgBox "Can't find Program Manager."
        Exit Sub
    End If
    AppActivate "Program Manager"           'Activate Program Manager.
    state = AppGetState                     'Save its state.
    AppMinimize                             'Minimize it.
    MsgBox "Program Manager is now minimized. Select OK to restore it."
    AppActivate "Program Manager"
    AppSetState state                           'Restore it.
End Sub
```

See Also

AppGetState (function); AppMinimize (statement); AppMaximize (statement); AppRestore (statement).

Notes

The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.

statement

AppShow

Syntax

```
AppShow [name$]
```

Description

Makes the named application visible.

Comments

- The name$ parameter is a String containing the name of the application to show. If this parameter is omitted, then the active application is shown.

Example

```
'This example hides Program Manager.

Sub Main()
    'See whether Program Manager is running.
    If AppFind$("Program Manager") = "" Then Exit Sub
    AppHide "Program Manager"
    MsgBox "Program Manager is now hidden. Press OK to show it once again."
    AppShow "Program Manager"
End Sub
```

See Also

AppHide (statement).

Notes

- If the named application is already visible, AppShow will have no effect.
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- AppShow generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

statement

AppSize

Syntax

```
AppSize width,height [,name$]
```

Description

Sets the width and height of the named application.

Comments

- The AppSize statement takes the following parameters:

Parameter	Description
width, height	Integer coordinates specifying the new size of the application.
name$	String containing the name of the application to resize. If this parameter is omitted, then the active application is used.

Example

```
'This example enlarges the active application by 10 pixels in both the
'vertical and horizontal directions.

Sub Main()
    Dim w%,h%
    AppGetPosition 0,0,w%,h%                    'Get current width/height.
    x% = x% + Screen.TwipsPerPixelX * 10  'Add 10 pixels.
    y% = y% + Screen.TwipsPerPixelY * 10  'Add 10 pixels.
    AppSize w%,h%                               'Change to new size.
End Sub
```

See Also

AppMaximize (statement); AppMinimize (statement); AppRestore (statement); AppMove (statement); AppClose (statement).

Notes

- The width and height parameters are specified in twips.
- This statement will only work if the named application is restored (i.e., not minimized or maximized).
- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that matches name$, then the first application encountered is used.
- A runtime error results if the application being resized is not enabled, which is the case if that application is displaying a modal dialog box when an AppSize statement is run.

function

AppType

Syntax

```
AppType [(name$)]
```

Description

Returns an Integer indicating the executable file type of the named application:

ebDosDOS executable

ebWindows Windows executable

Comments

- The name$ parameter is a String containing the name of the application. If this parameter is omitted, then the active application is used.

Example

```
'This example creates an array of strings containing the names of all the
'running Windows applications. It uses the AppType command to determine
'whether an application is a Windows application or a DOS application.

Sub Main()
    Dim apps$(),wapps$()

    AppList apps        'Retrieve a list of all Windows and DOS apps.
    If ArrayDims(apps) = 0 Then
        MsgBox "There are no running applications."
        Exit Sub
    End If

    'Create an array to hold only the Windows apps.
    ReDim wapps$(UBound(apps))
    n = 0       'Copy the Windows apps from one array to the target array.
    For i = LBound(apps) to UBound(apps)
        If AppType(apps(i)) = ebWindows Then
            wapps(n) = apps(i)
            n = n + 1
        End If
    Next i

    If n = 0 Then 'Make sure at least one Windows app was found.
        MsgBox "There are no running Windows applications."
        Exit Sub
    End If

    ReDim Preserve wapps(n - 1)   'Resize to hold the exact number.
    'Let the user pick one.
    index% = SelectBox("Windows Applications","Select a Windows application:",wapps)
End Sub
```

See Also

AppFilename$ (function).

Notes

- The name$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name$, then a second search is performed for applications whose title string begins with name$. If more than one application is found that

matches name$, then the first application encountered is used.

function

ArrayDims

Syntax

```
ArrayDims(arrayvariable)
```

Description

Returns an Integer containing the number of dimensions of a given array.

Comments

- This function can be used to determine whether a given array contains any elements or if the array is initially created with no dimensions and then redimensioned by another function, such as the FileList function, as shown in the following example.

Example

```
'This example allocates an empty (null-dimensioned) array; fills the array
'with a list of filenames, which resizes the array; then tests the array
'dimension and displays an appropriate message.

Sub Main()
    Dim f$()
    FileList f$,"c:\*.bat"
    If ArrayDims(f$) = 0 Then
        MsgBox "The array is empty."
    Else
        MsgBox "The array size is: " & (UBound(f$) - UBound(f$) + 1)
    End If
End Sub
```

See Also

LBound (function); UBound (function); Arrays (topic).

topic

Arrays

Arrays in Delrina Basic are declared using any of the following statements:

```
Dim
Public
Private
```

For example:

```
Dim a(10) As Integer
Public LastNames(1 to 5,-2 to 7) As Variant
Private
```

Arrays of any data type can be created, including Integer, Long, Single, Double, Boolean, Date, Variant, Object, user-defined structures, and data objects.

The lower and upper bounds of each array dimension must be within the following range:

-32768 <= `bound` <= 32767

Arrays can have up to 60 dimensions.

Arrays can be declared as either fixed or dynamic, as described below.

Fixed Arrays

The dimensions of fixed arrays cannot be adjusted at execution time. Once declared, a fixed array will always require the same amount of storage. Fixed arrays can be declared with the Dim, Private, or Public statement by supplying explicit dimensions. The following example declares a fixed array of ten strings:

```
Dim a(10) As String
```

Fixed arrays can be used as members of user-defined data types. The following example shows a structure containing fixed-length arrays:

```
Type Foo
    rect(4) As Integer
    colors(10) As Integer
End Type
```

Only fixed arrays can appear within structures.

Dynamic Arrays

Dynamic arrays are declared without explicit dimensions, as shown below:

```
Public Ages() As Integer
```

Dynamic arrays can be resized at execution time using the Redim statement:

```
Redim Ages$(100)
```

Subsequent to their initial declaration, dynamic arrays can be redimensioned any number of times. When redimensioning an array, the old array is first erased unless you use the Preserve keyword, as shown below:

```
Redim Preserve Ages$(100)
```

Dynamic arrays cannot be members of user-defined data types.

Passing Arrays

Arrays are always passed by reference.

Querying Arrays

The following table describes the functions used to retrieve information about arrays.

Use this function	to
LBound	Retrieve the lower bound of an array. A runtime error is generated if the array has no dimensions.
UBound	Retrieve the upper bound of an array. A runtime error is generated if the array has no dimensions.
ArrayDims	Retrieve the number of dimensions of an array. This function returns 0 if the array has no dimensions.

Operations on Arrays

The following table describes the function that operate on arrays:

Use this command	to
ArraySort	Sort an array of integers, longs, singles, doubles, currency, Booleans, dates, or variants.
FileList	Fill an array with a list of files in a given directory.
DiskDrives	Fill an array with a list of valid drive letters.
AppList	Fill an array with a list of running applications.
WinList	Fill an array with a list of top-level windows.
SelectBox	Display the contents of an array in a list box.
PopupMenu	Display the contents of an array in a pop-up menu.
ReadIniSection	Fill an array with the item names from a section in an ini file.
FileDirs	Fill an array with a list of subdirectories.
Erase	Erase all the elements of an array.
ReDim	Establish the bounds and dimensions of an array.
Dim	Declare an array.

statement

ArraySort

Syntax

```
ArraySort array()
```

Description

Sorts a single-dimensioned array in ascending order.

Comments

- If a string array is specified, then the routine sorts alphabetically in ascending order using case-sensitive string comparisons. If a numeric array is specified, the ArraySort statement sorts smaller numbers to the lowest array index locations.
- Delrina Basic generates a runtime error if you specify an array with more than one dimension.
- When sorting an array of variants, the following rules apply:
- A runtime error is generated if any element of the array is an object.
- String is greater than any numeric type.
- Null is less than String and all numeric types.
- Empty is treated as a number with the value 0.
- String comparison is case-sensitive (this function is not affected by the Option Compare setting).

Example

```
'This example dimensions an array and fills it with filenames using FileList,
'then sorts the array and displays it in a select box.

Sub Main()
    Dim f$()
    FileList f$,"c:\*.*"
    ArraySort f$
    r% = SelectBox("Files","Choose one:",f$)
End Sub
```

See Also

ArrayDims (function); LBound (function); UBound (function).

function

Asc

Syntax

```
Asc(text$)
```

Description

Returns an Integer containing the numeric code for the first character of text$.

Comments

- The return value is an integer between 0 and 255.

Example

```
'This example fills an array with the ASCII values of the string s components
'and displays the result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    s$ = InputBox("Please enter a string.","Enter String")
    If s$ = "" Then End          'Exit if no string entered.
    For i = 1 To Len(s$)
        msg = msg & Asc(Mid(s$,i,1)) & crlf
    Next i
    MsgBox "The Asc values of the string are:" & msg
End Sub
```

See Also

Chr, Chr$ (functions).

function

AskBox$

Syntax

```
AskBox$(prompt$ [,default$])
```

Description

Displays a dialog box requesting input from the user and returns that input as a String.

Comments

- The AskBox$ function takes the following parameters:

Parameter	Description
prompt$	String containing the text to be displayed above the text box. The dialog box is sized to the appropriate width depending on the width of prompt$. A runtime error is generated if prompt$ is Null.
default$	String containing the initial content of the text box. The user can return the default by immediately selecting OK. A runtime error is generated if default$ is Null.

- The AskBox$ function returns a String containing the input typed by the user in the text box. A zero-length string is returned if the user selects Cancel.
- When the dialog box is displayed, the text box has the focus.
- The user can type a maximum of 255 characters into the text box displayed by AskBox$.

s$ = **AskBox$(**"Type in the filename:"**)**

s$ = **AskBox$(**"Type in the filename:"**,**"filename.txt"**)**

Example

```
'This example asks the user to enter a filename and then displays what
'he or she has typed.

Sub Main()
    s$ = AskBox$("Type in the filename:")
    MsgBox "The filename was: " & s$
End Sub
```

See Also

MsgBox (statement); AskPassword$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SaveFilename$ (function); SelectBox (function).

Note

- The text in the dialog box is displayed in 8-point MS Sans Serif.

function

AskPassword$

Syntax

```
AskPassword$(prompt$)
```

Description

Returns a String containing the text that the user typed.

Comments

- Unlike the AskBox$ function, the user sees asterisks in place of the characters that are actually typed. This lets the hidden input of passwords.
- The prompt$ parameter is a String containing the text to appear above the text box. The dialog box is sized to the appropriate width depending on the width of prompt$.
- When the dialog box is displayed, the text box has the focus.
- A maximum of 255 characters can be typed into the text box.
- A zero-length string is returned if the user selects Cancel.

Example

```
Sub Main()
    s$ = AskPassword$("Type in the password:")
    MsgBox "The password entered is: " & s$
End Sub
```

See Also

MsgBox (statement); AskBox$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SaveFilename$ (function); SelectBox (function); AnswerBox (function).

Note

The text in the dialog box is displayed in 8-point MS Sans Serif.

function

Atn

Syntax

```
Atn(number)
```

Description

Returns the angle (in radians) whose tangent is number.

Comments

Some helpful conversions:

- Pi (3.1415926536) radians = 180 degrees.
- 1 radian = 57.2957795131 degrees.
- 1 degree = .0174532925 radians.

Example

```
'This example finds the angle whose tangent is 1 (45 degrees) and displays
'the result.

Sub Main()
    a# = Atn(1.00)
    MsgBox "1.00 is the tangent of " & a# & " radians (45 degrees)."
End Sub
```

See Also

Tan (function); Sin (function); Cos (function).

method

Basic.Capability

Syntax

```
Basic.Capability(which)
```

Description

Returns True if the specified capability exists on the current platform; returns False otherwise.

Comments

- The which parameter is an Integer specifying the capability for which to test. It can be any of the following values:

Value	Returns True If the Platform Supports
1	Disk drives
2	System file attribute (ebSystem)
3	Hidden file attribute (ebHidden)
4	Volume label file attribute (ebVolume)
5	Archive file attribute (ebArchive)
6	Denormalized floating-point math
7	File locking (i.e., the Lock and Unlock statements)
8	Big endian byte ordering

Example

```
'This example tests to see whether your current platform supports disk
'drives and hidden file attributes and displays the result.

Sub Main()
    msg = "This operating system "

    If Basic.Capability(1) Then
        msg = msg & "supports disk drives."
    Else
        msg = msg & "does not support disk drives."
    End If

    MsgBox msg
End Sub
```

property

Basic.Eoln$

Syntax

Basic.Eoln$

Description

Returns a String containing the end-of-line character sequence appropriate to the current platform.

Comments

- This string will be either a carriage return, a carriage return/line feed, or a line feed.

Example

'This example writes two lines of text in a message box.

```
Sub Main()
    MsgBox "This is the first line of text." & Basic.Eoln$ & "This is the second line of text."
End Sub
```

See Also

Cross-Platform Scripting (topic); Basic.PathSeparator$ (property).

property

Basic.FreeMemory

Syntax

Basic.FreeMemory

Description

Returns a Long representing the number of bytes of free memory in Delrina Basic's data space.

Comments

- This function returns the size of the largest free block in Delrina Basic's data space. Before this number is returned, the data space is compacted, consolidating free space into a single contiguous free block.
- Delrina Basic's data space contains strings and dynamic arrays.

Example

```
'This example displays free memory in a dialog box.

Sub Main()
    MsgBox "The largest free memory block is: " & Basic.FreeMemory
End Sub
```

See Also

System.TotalMemory (property); System.FreeMemory (property); System.FreeResources (property); Basic.FreeMemory (property).

property

Basic.HomeDir$

Syntax

Basic.HomeDir$

Description

Returns a String specifying the directory containing Delrina Basic.

Comments

- This method is used to find the directory in which the Delrina Basic files are located.

Example

```
'This example assigns the home directory to HD and displays it.

Sub Main()
    hd$ = Basic.HomeDir$
    MsgBox "The Delrina Basic home directory is: " & hd$
End Sub
```

See Also

System.WindowsDirectory$ (property).

property

Basic.PathSeparator$

Syntax

Basic.PathSeparator$

Description

Returns a String containing the path separator appropriate for the current platform.

Comments

- The returned string is any one of the following characters: / (slash), \ (back slash), : (colon)

Example

```
Sub Main()
    MsgBox "The path separator for this platform is: " & Basic.PathSeparator$
End Sub
```

See Also

Basic.Eoln$ (property); Cross-Platform Scripting (topic).

property

Basic.Version$

Syntax

Basic.Version$

Description

Returns a String containing the version of Delrina Basic.

Comments

- This function returns the major and minor version numbers in the format major.minor.BuildNumber, as in "2.00.30."

Example

```
'This example displays the current version of Delrina Basic.

Sub Main()
    MsgBox "Version " & Basic.Version$ & " of Delrina Basic is running"
End Sub
```

statement

Beep

Syntax

Beep

Description

Makes a single system beep.

Example

```
'This example causes the system to beep five times and displays a reminder
'message.

Sub Main()
    For i = 1 To 5
        Beep
        Sleep 200
    Next i
    MsgBox "You have an upcoming appointment!"
End Sub
```

See Also

Mci (function).

statement

Begin Dialog

Syntax

```
Begin Dialog DialogName [x],[y],width,height,title$ [,[.DlgProc] [,
[PicName$] [,style]]]
     Dialog Statements
End Dialog
```

Description

Defines a dialog box template for use with the Dialog statement and function.

Comments

- A dialog box template is constructed by placing any of the following statements between the Begin Dialog and End Dialog statements (no other statements besides comments can appear within a dialog box template):

Picture	OptionButton	OptionGroup
CancelButton	Text	TextBox
GroupBox	DropListBox	ListBox
ComboBox	CheckBox	PictureButton
PushButton	OKButton	

- The Begin Dialog statement requires the following parameters:

Parameter	Description
x, y	Integer coordinates specifying the position of the upper left corner of the dialog box relative to the parent window. These coordinates are in dialog units.
	styles for the dialog. It can be any of the following values:

Value	Meaning
0	Dialog does not contain a title or close box.
1	Dialog contains a title and no close box.
2 (or omitted)	Dialog contains both the title and close box.

- Delrina Basic generates an error if the dialog box template contains no controls.
- A dialog box template must have at least one PushButton, OKButton, or CancelButton statement. Otherwise, there will be no way to close the dialog box.
- Dialog units are defined as 1/4 the width of the font in the horizontal direction and 1/8 the height of the font in the vertical direction.
- Any number of user dialog boxes can be created, but each one must be created using a different name as the DialogName. Only one user dialog box may be invoked at any time.

Expression Evaluation within the Dialog Box Template

The Begin Dialog statement creates the template for the dialog box. Any expression or variable name that appears within any of the statements in the dialog box template is not evaluated until a variable is dimensioned of type DialogName. The following example shows this behavior:

```
Sub Main()
    MyTitle$ = "Hello, World"
    Begin Dialog MyTemplate 16,32,116,64,MyTitle$
        OKButton 12,40,40,14
    End Dialog
    MyTitle$ = "Sample Dialog"
    Dim dummy As MyTemplate
    rc% = Dialog(dummy)
End Sub
```

The above example creates a dialog box with the title "Sample Dialog".

Expressions within dialog box templates cannot reference external subroutines or functions.

All controls within a dialog box use the same font. The fonts used for text and text box control can be changed explicitly by setting the font parameters in the Text and TextBox statements. A maximum of 128 fonts can be used within a single dialog, although the practical limitation may be less.

Example

```
'This example creates an exit dialog box.

Sub Main()
    Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit"
        Text 4,8,108,8,"Are you sure you want to exit?"
        CheckBox 32,24,63,8,"Save Changes",.SaveChanges
        OKButton 12,40,40,14
        CancelButton 60,40,40,14
    End Dialog
    Dim QuitDialog As QuitDialogTemplate
    rc% = Dialog(QuitDialog)
    Select Case rc%
        Case -1
            MsgBox "OK was pressed!"
        Case 1
            MsgBox "Cancel was pressed!"
    End Select
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); DlgProc (function).

Note

Within user dialog boxes, the default font is 8-point MS Sans Serif.

data type

Boolean

Syntax

Boolean

Description

A data type capable of representing the logical values True and False.

Comments

- Boolean variables are used to hold a binary value-either True or False. Variables can be declared as Boolean using the Dim, Public, or Private statement.
- Variants can hold Boolean values when assigned the results of comparisons or the constants True or False.
- Internally, a Boolean variable is a 2-byte value holding -1 (for True) or 0 (for False).
- Any type of data can be assigned to Boolean variables. When assigning, non-0 values are converted to True, and 0 values are converted to False.
- When appearing as a structure member, Boolean members require 2 bytes of storage.
- When used within binary or random files, 2 bytes of storage are required.
- When passed to external routines, Boolean values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack.
- There is no type-declaration character for Boolean variables.
- Boolean variables that have not yet been assigned are given an initial value of False.

See Also

Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); DefType (statement); CBool (function); True (constant); False (constant).

function

ButtonEnabled

Syntax

```
ButtonEnabled(name$ | id)
```

Description

Returns True if the specified button within the current window is enabled; returns False otherwise.

Comments

- The ButtonEnabled function takes the following parameters:

Parameter	Description
name$	String containing the name of the push button.
id	Integer specifying the ID of the push button.

- When a button is enabled, it can be clicked using the SelectButton statement.

Note: The ButtonEnabled function is used to determine whether a push button is enabled in another application's dialog box. Use the DlgEnable function to retrieve the enabled state of a push button in a dynamic dialog box.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

ButtonExists (function); SelectButton (statement).

function

ButtonExists

Syntax

```
ButtonExists(name$ | id)
```

Description

Returns True if the specified button exists within the current window; returns False otherwise.

Comments

- The ButtonExists function takes the following parameters:

Parameter	Description
name$	String containing the name of the push button.
id	Integer specifying the ID of the push button.

Note: The ButtonExists function is used to determine whether a push button exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

ButtonEnabled (function); SelectButton (statement).

keyword

ByRef

Syntax

```
...,ByRef parameter,...
```

Description

Used within the Sub...End Sub, Function...End Function, or Declare statement to specify that a given parameter can be modified by the called routine.

Comments

- Passing a parameter by reference means that the caller can modify that variable's value.
- Unlike the ByVal keyword, the ByRef keyword cannot be used when passing a parameter. The absence of the ByVal keyword is sufficient to force a parameter to be passed by reference:

```
MySub ByVal i        'Pass i by value.
MySub ByRef i        'Illegal (will not compile).
MySub i              'Pass i by reference.
```

Example

```
Sub Test(ByRef a As Variant)
    a = 14
End Sub

Sub Main()
    b = 12
    Test b
    MsgBox "The ByRef value is: " & b          'Displays 14.
End Sub
```

See Also

() (keyword), ByVal (keyword).

keyword

ByVal

Syntax

```
...ByVal parameter...
```

Description

Forces a parameter to be passed by value rather than by reference.

Comments

- The ByVal keyword can appear before any parameter passed to any function, statement, or method to force that parameter to be passed by value. Passing a parameter by value means that the caller cannot modify that variable's value.
- Enclosing a variable within parentheses has the same effect as the ByVal keyword:

```
Foo ByVal i          'Forces i to be passed by value.
Foo(i)               'Forces i to be passed by value.
```

- When calling external statements and functions (i.e., routines defined using the Declare statement), the ByVal keyword forces the parameter to be passed by value regardless of the declaration of that parameter in the Declare statement. The following example shows the effect of the ByVal keyword used to passed an Integer to an external routine:

```
Declare Sub Foo Lib "MyLib" (ByRef i As Integer)
i% = 6
Foo ByVal i%         'Pass a 2-byte Integer.
Foo i%               'Pass a 4-byte pointer to an Integer.
```

Since the Foo routine expects to receive a pointer to an Integer, the first call to Foo will have unpredictable results.

Example

```
'This example demonstrates the use of the ByVal keyword.

Sub Foo(a As Integer)
    a = a + 1
End Sub

Sub Main()
    Dim i As Integer
    i = 10
    Foo i
    MsgBox "The ByVal value is: " & i          'Displays 11 (Foo changed the value).
    Foo ByVal i
    MsgBox "The ByVal value is still: " & i          'Displays 11 (Foo did not change the value).
End Sub
```

See Also

() (keyword), ByRef (keyword).

statement

Call

Syntax

```
Call subroutine_name [(arguments)]
```

Description

Transfers control to the given subroutine, optionally passing the specified arguments.

Comments

- Using this statement is equivalent to:
  ```
  subroutine_name [arguments]
  ```
- Use of the Call statement is optional. The Call statement can only be used to run subroutines; functions cannot be run with this statement. The subroutine to which control is transferred by the Call statement must be declared outside of the Main procedure, as shown in the following example.

Example

```
'This example demonstrates the use of the Call statement to pass control to
'another function.

Sub Example_Call(s$)
    'This subroutine is declared externally to Main and displays the text
    'passed in the parameter s$.
    MsgBox "Call: " & s$
End Sub

Sub Main()
    'This example assigns a string variable to display, then calls subroutine
    'Example_Call, passing parameter S$ to be displayed in a message box
    'within the subroutine.
    s$ = "DAVE"
    Example_Call s$
    Call Example_Call("SUSAN")
End Sub
```

See Also

Goto (statement); GoSub (statement); Declare (statement).

statement

CancelButton

Syntax

```
CancelButton X, Y, width, height [,.Identifier]
```

Description

Defines a Cancel button that appears within a dialog box template.

Comments

- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- Selecting the Cancel button (or pressing Esc) dismisses the user dialog box, causing the Dialog function to return 0. (Note: A dialog function can redefine this behavior.) Pressing the Esc key or double-clicking the close box will have no effect if a dialog box does not contain a CancelButton statement.
- The CancelButton statement requires the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
.Identifier	Optional parameter specifying the name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the word Cancel is used.

- A dialog box must contain at least one OKButton, CancelButton, or PushButton statement; otherwise, the dialog box cannot be dismissed.

Example

```
'This example creates a sample dialog box with OK and Cancel buttons.

Sub Main()
    Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit"
        Text 4,8,108,8,"Are you sure you want to exit?"
        CheckBox 32,24,63,8,"Save Changes",.SaveChanges
        OKButton 12,40,40,14
        CancelButton 60,40,40,14
    End Dialog
    Dim QuitDialog As QuitDialogTemplate
    rc% = Dialog(QuitDialog)
    Select Case rc%
        Case -1
            MsgBox "OK was pressed!"
        Case 1
            MsgBox "Cancel was pressed!"
    End Select
End Sub
```

See Also

CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

function

CBool

Syntax

```
CBool(expression)
```

Description

Converts expression to True or False, returning a Boolean value.

Comments

- The expression parameter is any expression that can be converted to a Boolean. A runtime error is generated if expression is Null.
- All numeric data types are convertible to Boolean. If expression is zero, then the CBool returns False; otherwise, CBool returns True. Empty is treated as False.
- If expression is a String, then CBool first attempts to convert it to a number, then converts the number to a Boolean. A runtime error is generated if expression cannot be converted to a number.
- A runtime error is generated if expression cannot be converted to a Boolean.

Example

```
'This example uses CBool to determine whether a string is numeric
'or just plain text.

Sub Main()
    Dim IsNumericOrDate As Boolean
    s$ = "34224.54"
    IsNumeric = CBool(IsNumeric(s$))
    If IsNumeric = True Then
        MsgBox s$ & " is either a valid number!"
    Else
        MsgBox s$ & " is not a valid number!"
    End If
End Sub
```

See Also

CCur (function); CDate, CVDate (functions); CDbl (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVErr (function); Boolean (data type).

function

CCur

Syntax

```
CCur(expression)
```

Description

Converts any expression to a Currency.

Comments

- This function accepts any expression convertible to a Currency, including strings. A runtime error is generated if expression is Null or a String not convertible to a number. Empty is treated as 0.
- When passed a numeric expression, this function has the same effect as assigning the numeric expression number to a Currency.
- When used with variants, this function guarantees that the variant will be assigned a Currency (VarType 6).

Example

'This example displays the value of a String converted into a Currency value.

```
Sub Main()
    i$ = "100.44"
    MsgBox "The currency value is: " & CCur(i$)
End Sub
```

See Also

CBool (function); CDate, CVDate (functions); CDbl (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVErr (function); Currency (data type).

function

CDate, CVDate

Syntax

```
CDate(expression)

CVDate(expression)
```

Description

Converts expression to a date, returning a Date value.

Comments

- The expression parameter is any expression that can be converted to a Date. A runtime error is generated if expression is Null.
- If expression is a String, an attempt is made to convert it to a Date using the current country settings. If expression does not represent a valid date, then an attempt is made to convert expression to a number. A runtime error is generated if expression cannot be represented as a date.
- These functions are sensitive to the date and time formats of your computer.
- The CDate and CVDate functions are identical.

Example

```
'This example takes two dates and computes the difference between them.

Sub Main()
    Dim date1 As Date
    Dim date2 As Date
    Dim diff As Date

    date1 = CDate(#1/1/1994#)
    date2 = CDate("February 1, 1994")
    diff = DateDiff("d",date1,date2)

    MsgBox "The date difference is " & CInt(diff) & " days."
End Sub
```

See Also

CCur (function); CBool (function); CDbl (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVErr (function); Date (data type).

function

CDbl

Syntax

```
CDbl(expression)
```

Description

Converts any expression to a Double.

Comments

- This function accepts any expression convertible to a Double, including strings. A runtime error is generated if expression is Null. Empty is treated as 0.0.
- When passed a numeric expression, this function has the same effect as assigning the numeric expression number to a Double.
- When used with variants, this function guarantees that the variant will be assigned a Double (VarType 5).

Example

```
'This example displays the result of two numbers as a Double.

Sub Main()
    i% = 100
    j! = 123.44
    MsgBox "The double value is: " & CDbl(i% * j!)
End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVErr (function); Double (data type).

statement

ChDir

Syntax

```
ChDir newdir$
```

Description

Changes the current directory of the specified drive to newdir$.

This routine will not change the current drive. (See ChDrive [statement].)

Example

```
'This example saves the current directory, then changes to the root
'directory, displays the old and new directories, restores the old directory,
'and displays it.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    save$ = CurDir$
    ChDir(Basic.PathSeparator$)
    MsgBox "Old directory: " & save$ & crlf & "New directory: " & CurDir$
    ChDir(save$)
    MsgBox "Directory restored to: " & CurDir$
End Sub
```

See Also

ChDrive (statement); CurDir, CurDir$ (functions); Dir, Dir$ (functions); MkDir (statement); RmDir (statement); DirList (statement).

statement

ChDrive

Syntax

```
ChDrive DriveLetter$
```

Description

Changes the default drive to the specified drive.

Comments

- Only the first character of DriveLetter$ is used.
- DriveLetter$ is not case-sensitive.
- If DriveLetter$ is empty, then the current drive is not changed.

Example

```
'This example lets the user to select a new current drive and uses
'ChDrive to make their choice the new current drive.

Const crlf$ = Chr$(13) + Chr$(10)

Sub Main()
    Dim d()
    old$ = FileParse$(CurDir,1)
    DiskDrives d
Again:
    r = SelectBox("Available Drives","Select new current drive:",d)
    On Error Goto Error_Trap
    If r <> -1 Then ChDrive d(r)
    MsgBox "Old Current Drive: " & old$ & crlf & "New Current Drive: " & CurDir
    End
Error_Trap:
    MsgBox Error(err)
    Resume Again
End Sub
```

See Also

ChDir (statement); CurDir, CurDir$ (functions); Dir, Dir$ (functions); MkDir (statement); RmDir (statement); DiskDrives (statement).

statement

CheckBox

Syntax

```
CheckBox X, Y, width, height, title$, .Identifier
```

Description

Defines a check box within a dialog box template.

Comments

- Check box controls are either on or off, depending on the value of .Identifier.
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The CheckBox statement requires the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
title$	String containing the text that appears within the check box. This text may contain an ampersand character to denote an accelerator letter, such as "&Font" for Font (indicating that the Font control may be selected by pressing the F accelerator key).
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the state of the check box (1 = checked; 0 = unchecked). This variable can be accessed using the syntax:
DialogVariable.Identifier.	

When the dialog box is first created, the value referenced by .Identifier is used to set the initial state of the check box. When the dialog box is dismissed, the final state of the check box is placed into this variable. By default, the .Identifier variable contains 0, meaning that the check box is unchecked.

Example

```
'This example displays a dialog box with two check boxes in different states.

Sub Main()
    Begin Dialog SaveOptionsTemplate 36,32,151,52,"Save"
        GroupBox 4,4,84,40,"GroupBox"
        CheckBox 12,16,67,8,"Include heading",.IncludeHeading
        CheckBox 12,28,73,8,"Expand keywords",.ExpandKeywords
        OKButton 104,8,40,14,.OK
        CancelButton 104,28,40,14,.Cancel
    End Dialog
    Dim SaveOptions As SaveOptionsTemplate
    SaveOptions.IncludeHeading = 1          'Check box initially on.
    SaveOptions.ExpandKeywords = 0          'Check box initially off.
    r% = Dialog(SaveOptions)
    If r% = -1 Then
        MsgBox "OK was pressed."
    End If
End Sub
```

See Also

CancelButton (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox

(statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Note

- Accelerators are underlined, and the accelerator combination Alt+letter is used.

function

CheckBoxEnabled

Syntax

```
CheckBoxEnabled(name$ | id)
```

Description

Returns True if the specified check box within the current window is enabled; returns False otherwise.

Comments

- The CheckBoxEnabled function takes the following parameters:

Parameter	Description
name$	String containing the name of the check box.
id	Integer specifying the ID of the check box.

- When a check box is enabled, its state can be set using the SetCheckBox statement.

Note: The CheckBoxEnabled function is used to determine whether a check box is enabled in another application's dialog box. Use the DlgEnable function within dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

CheckBoxExists (function); GetCheckBox (function); SetCheckBox (statement).

function

CheckBoxExists

Syntax

```
CheckBoxExists(name$ | id)
```

Description

Returns True if the specified check box exists within the current window; returns False otherwise.

Comments

- The CheckBoxExists function takes the following parameters:

Parameter	Description
name$	String containing the name of the check box.
id	Integer specifying the ID of the check box.

Note: The CheckBoxExists function is used to determine whether a check box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                    'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

CheckBoxEnabled (function); GetCheckBox (function); SetCheckBox (statement).

function

Choose

Syntax

```
Choose(index,expression1,expression2,...,expression13)
```

Description

Returns the expression at the specified index position.

Comments

- The index parameter specifies which expression is to be returned. If index is 1, then expression1 is returned; if index is 2, then expression2 is returned, and so on. If index is less than 1 or greater than the number of supplied expressions, then Null is returned.
- The Choose function returns the expression without converting its type. Each expression is evaluated before returning the selected one.

Example

'This example assigns a variable of indeterminate type to a.

```
Sub Main()
    Dim a As Variant
    Dim c As Integer
    c% = 2
    a = Choose(c%,"Hello, world",#1/1/94#,5.5,False)
    MsgBox "Item " & c% & " is '" & a & "'"          'Displays the date passed as parameter 2.
End Sub
```

See Also

Switch (function); IIf (function); If...Then...Else (statement); Select...Case (statement).

function

Chr, Chr$

Syntax

```
Chr[$](Code)
```

Description

Returns the character whose value is Code.

Comments

- Code must be an Integer between 0 and 255.
- Chr$ returns a string, whereas Chr returns a String variant.
- The Chr$ function can be used within constant declarations, as in the following example:

```
Const crlf = Chr$(13) + Chr$(10)
```

- Some common uses of this function are:

Chr$(9)	Tab
Chr$(13) + Chr$(10)	End-of-line (carriage return, linefeed)
Chr$(26)	End-of-file
Chr$(0)	Null

Example

```
Sub Main()
    'Concatenates carriage return (13) and linefeed (10) to CRLF$,
    'then displays a multiple-line message using CRLF$ to separate lines.
    crlf$ = Chr$(13) + Chr$(10)
    MsgBox "First line." & crlf$ & "Second line."

    'Fills an array with the ASCII characters for ABC and displays their
    'corresponding characters.
    Dim a%(2)
    For i = 0 To 2
        a%(i) = (65 + i)
    Next i
    MsgBox "The first three elements of the array are: " & Chr$(a%(0)) & Chr$(a%(1)) & Chr$(a%(2))
End Sub
```

See Also

Asc (function); Str, Str$ (functions).

function

CInt

Syntax

```
CInt(expression)
```

Description

Converts expression to an Integer.

Comments

- This function accepts any expression convertible to an Integer, including strings. A runtime error is generated if expression is Null. Empty is treated as 0.
- The passed numeric expression must be within the valid range for integers:
  `-32768 <= expression <= 32767`
- A runtime error results if the passed expression is not within the above range.
- When passed a numeric expression, this function has the same effect as assigning a numeric expression to an Integer. Note that integer variables are rounded before conversion.
- When used with variants, this function guarantees that the expression is converted to an Integer variant (VarType 2).

Example

```
'This example demonstrates the various results of integer manipulation
'with CInt.

Sub Main()

    '(1) Assigns i# to 100.55 and displays its integer representation (101).
    i# = 100.55
    MsgBox "The value of CInt(i) = " & CInt(i#)

    '(2) Sets j# to 100.22 and displays the CInt representation (100).
    j# = 100.22
    MsgBox "The value of CInt(j) = " & CInt(j#)

    '(3) Assigns k% (integer) to the CInt sum of j# and k% and displays k% (201).
    k% = CInt(i# + j#)
    MsgBox "The integer sum of 100.55 and 100.22 is: " & k%

    '(4) Reassigns i# to 50.35 and recalculates k%, then displays the result
    '(note rounding).
    i# = 50.35
    k% = CInt(i# + j#)
    MsgBox "The integer sum of 50.35 and 100.22 is: " & k%

End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CDbl (function); CLng (function); CSng (function); CStr (function); CVar (function); CVErr (function); Integer (data type).

function

Clipboard$

Syntax

Clipboard$[()]

Description

Returns a String containing the contents of the Clipboard.

Comments

- If the Clipboard doesn't contain text or the Clipboard is empty, then a zero-length string is returned.

Example

```
'This example puts text on the Clipboard, displays it, clears the Clipboard,
'and displays the Clipboard again.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Clipboard$ "Hello out there!"
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
    Clipboard.Clear
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
End Sub
```

See Also

Clipboard$ (statement); Clipboard.GetText (method); Clipboard.SetText (method).

statement

Clipboard$

Syntax

```
Clipboard$ NewContent$
```

Description

Copies NewContent$ into the Clipboard.

Example

```
'This example puts text on the Clipboard, displays it, clears the Clipboard,
'and displays the Clipboard again.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Clipboard$ "Hello out there!"
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
    Clipboard.Clear
    MsgBox "The text in the Clipboard is now:" & crlf & Clipboard$
End Sub
```

See Also

Clipboard$ (function); Clipboard.GetText (method); Clipboard.SetText (method).

method

Clipboard.Clear

Syntax

Clipboard.Clear

Description

This method clears the Clipboard by removing any content.

Example

```
'This example puts text on the Clipboard, displays it, clears the Clipboard,
'and displays the Clipboard again.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Clipboard$ "Hello out there!"
    MsgBox "The text in the Clipboard before clearing:" & crlf & Clipboard$
    Clipboard.Clear
    MsgBox "The text in the Clipboard after clearing:" & crlf & Clipboard$
End Sub
```

method

Clipboard.GetFormat

Syntax

```
WhichFormat = Clipboard.GetFormat(format)
```

Description

Returns True if data of the specified format is available in the Clipboard; returns False otherwise.

Comments

- This method is used to determine whether the data in the Clipboard is of a particular format. The format parameter is an Integer representing the format to be queried:

Format	Description
1	Text
2	Bitmap
3	Metafile
8	Device-independent bitmap (DIB)
9	Color palette

Example

```
'This example checks to see whether there is any text on the
'Clipboard, if so, it searches the text for a string matching what
'the user entered.

Option Compare Text

Sub Main()
    r$ = InputBox("Enter a word to search for:","Scan Clipboard")

    If Clipboard.GetFormat(1) Then
        If Instr(Clipboard.GetText(1),r) = 0 Then
            MsgBox """" & r & """" & " was not found in the clipboard."
        Else
            MsgBox """" & r & """" & " is definitely in the clipboard."
        End If
    Else
        MsgBox "The Clipboard does not contain any text."
    End If
End Sub
```

See Also

Clipboard$ (function); Clipboard$ (statement).

method

Clipboard.GetText

Syntax

```
text$ = Clipboard.GetText([format])
```

Description

Returns the text contained in the Clipboard.

Comments

- The format parameter, if specified, must be 1.

Example

```
'This example checks to see whether there is any text on the
'Clipboard, if so, it searches the text for a string matching what
'the user entered.

Option Compare Text

Sub Main()
    r$ = InputBox("Enter a word to search for:","Scan Clipboard")

    If Clipboard.GetFormat(1) Then
        If Instr(Clipboard.GetText(1),r) = 0 Then
            MsgBox """" & r & """" & " was not found in the clipboard."
        Else
            MsgBox """" & r & """" & " is definitely in the clipboard."
        End If
    Else
        MsgBox "The Clipboard does not contain any text."
    End If
End Sub
```

See Also

Clipboard$ (statement); Clipboard$ (function); Clipboard.SetText (method).

method

Clipboard.SetText

Syntax

```
Clipboard.SetText data$ [,format]
```

Description

Copies the specified text string to the Clipboard.

Comments

- The data$ parameter specifies the text to be copied to the Clipboard. The format parameter, if specified, must be 1.

Example

```
'This example gets the contents of the Clipboard and uppercases it.

Sub Main()
    If Not Clipboard.GetFormat(1) Then Exit Sub
    Clipboard.SetText UCase(Clipboard.GetText(1)),1
End Sub
```

See Also

Clipboard$ (statement); Clipboard.GetText (method); Clipboard$ (function).

function

CLng

Syntax

```
CLng(expression)
```

Description

Converts expression to a Long.

Comments

- This function accepts any expression convertible to a Long, including strings. A runtime error is generated if expression is Null. Empty is treated as 0.
- The passed expression must be within the following range:

  ```
  -2147483648 <= expression <= 2147483647
  ```

- A runtime error results if the passed expression is not within the above range.
- When passed a numeric expression, this function has the same effect as assigning the numeric expression to a Long. Note that long variables are rounded before conversion.
- When used with variants, this function guarantees that the expression is converted to a Long variant (VarType 3).

Example

```
'This example displays the results for various conversions of i and j (note
'rounding).

Sub Main()
    i% = 100
    j& = 123.666
    MsgBox "The result of i * j is: " & CLng(i% * j&)          'Displays 12367.
    MsgBox "The new variant type of i is: " & Vartype(CLng(i%))
End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CDbl (function); CInt (function); CSng (function); CStr (function); CVar (function); CVErr (function); Long (data type).

statement

Close

Syntax

```
Close [[#] filenumber [,[#] filenumber]...]
```

Description

Closes the specified files.

Comments

- If no arguments are specified, then all files are closed.

Example

```
'This example opens four files and closes them in various combinations.

Sub Main()
    Open "test1" For Output As #1
    Open "test2" For Output As #2
    Open "test3" For Random As #3
    Open "test4" For Binary As #4
    MsgBox "The next available file number is: " & FreeFile()
    Close #1                    'Closes file 1 only.
    Close #2,#3                 'Closes files 2 and 3.
    Close                       'Closes all remaining files(4).
    MsgBox "The next available file number is: " & FreeFile()
End Sub
```

See Also

Open (statement); Reset (statement); End (statement).

statement

ComboBox

Syntax

```
ComboBox X,Y,width,height,ArrayVariable,.Identifier
```

Description

This statement defines a combo box within a dialog box template.

Comments

- When the dialog box is invoked, the combo box will be filled with the elements from the specified array variable.
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The ComboBox statement requires the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
ArrayVariable	Single-dimensioned array used to initialize the elements of the combo box. If this array has no dimensions, then the combo box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates a string variable whose value corresponds to the content of the edit field of the combo box. This variable can be accessed using the syntax: DialogVariable.Identifier.

- When the dialog box is invoked, the elements from ArrayVariable are placed into the combo box. The .Identifier variable defines the initial content of the edit field of the combo box. When the dialog box is dismissed, the .Identifier variable is updated to contain the current value of the edit field.

Example

```
'This example creates a dialog box that lets the user to select a day of
'the week.
```

```
Sub Main()
    Dim days$(6)
    days$(0) = "Monday"
    days$(1) = "Tuesday"
    days$(2) = "Wednesday"
    days$(3) = "Thursday"
    days$(4) = "Friday"
    days$(5) = "Saturday"
    days$(6) = "Sunday"

    Begin Dialog DaysDialogTemplate 16,32,124,96,"Days"
        OKButton 76,8,40,14,.OK
        Text 8,10,39,8,"&Weekdays:"
        ComboBox 8,20,60,72,days$,.Days
    End Dialog
    Dim DaysDialog As DaysDialogTemplate
    DaysDialog.Days = Format(Now,"dddd") 'Set to today.
    r% = Dialog(DaysDialog)
    MsgBox "You selected: " & DaysDialog.Days
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

function

ComboBoxEnabled

Syntax

```
ComboBoxEnabled(name$ | id)
```

Description

Returns True if the specified combo box is enabled within the current window or dialog box; returns False otherwise.

Comments

- The ComboBoxEnabled function takes the following parameters:

Parameter	Description
name$	String containing the name of the combo box. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
id	Integer specifying the ID of the combo box. A runtime error is generated if the specified combo box does not exist.

Note: The ComboBoxEnabled function is used to determine whether a combo box is enabled in another application's dialog box. Use the DlgEnable function in dynamic dialog boxes.

Example

```
'This code fragment checks to ensure that a combo box exists and is enabled
'before selecting the last item.

    If ComboBoxExists("Filename:") Then
        If ComboBoxEnabled("Filename:") Then
            NumItems = GetComboBoxItemCount("Filename:")
            SelectComboBoxItem "Filename:",NumItems
        End If
    End If
```

See Also

ComboBoxExists (function); GetComboBoxItem$ (function); GetComboBoxItemCount (function); SelectComboBoxItem (statement).

function

ComboBoxExists

Syntax

```
ComboBoxExists(name$ | id)
```

Description

Returns True if the specified combo box exists within the current window or dialog box; returns False otherwise.

Comments

- The ComboBoxExists function takes the following parameters:

Parameter	Description
name$	`String` containing the name of the combo box. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
id	Integer specifying the ID of the combo box.

Note: The ComboBoxExists function is used to determine whether a combo box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This code fragment checks to ensure that a combo box exists and is enabled
'before selecting the last item.

    If ComboBoxExists("Filename:") Then
        If ComboBoxEnabled("Filename:") Then
            NumItems = GetComboBoxItemCount("Filename:")
            SelectComboBoxItem "Filename:",NumItems
        End If
    End If
```

See Also

ComboBoxEnabled (function); GetComboBoxItem$ (function); GetComboBoxItemCount (function); SelectComboBoxItem (statement).

function

Command, Command$

Syntax

Command[$][()]

Description

Returns the argument from the command line used to start the application.

Comments

- Command$ returns a string, whereas Command returns a String variant.

Example

```
'This example checks to see if any command line parameters were used.
'If parameters were used they are displayed and a check is made to
'see if the user used the "/s" switch.

Sub Main()
    cmd$ = Command

    If cmd$ <> "" Then
        If (InStr(cmd$,"/s")) <> 0 Then
            MsgBox "Safety Mode On!"
        Else
            MsgBox "Safety Mode Off!"
        End If

        MsgBox "The command line startup options were: " & cmd$
    Else
        MsgBox "No command line startup options were used!"
    End If
End Sub
```

See Also

Environ, Environ$ (functions).

topic

Comments

Comments can be added to Delrina Basic code in the following manner:

All text between a single quotation mark and the end of the line is ignored:

```
MsgBox "Hello"              'Displays a message box.
```

The REM statement causes the compiler to ignore the entire line:

```
REM This is a comment.
```

Delrina Basic supports C-style multiline comment blocks /*...*/, as shown in the following example:

```
MsgBox "Before comment"
/* This stuff is all commented out.
This line, too, will be ignored.
This is the last line of the comment. */
MsgBox "After comment"
```

Note

- C-style comments can be nested.

topic

Comparison Operators

Syntax

```
expression1 [< | > | <= | >= | <> | =] expression2
```

Description

Comparison operators return True or False depending on the operator.

Comments

- The comparison operators are listed in the following table:

Operator	Returns True If
>	expression1 is greater than expression2
<	expression1 is less than expression2
<=	expression1 is less than or equal to expression2
>=	expression1 is greater than or equal to expression2
<>	expression1 is not equal to expression2
=	expression1 is equal to expression2

- This operator behaves differently depending on the types of the expressions, as shown in the following table:

If one expression is	and the other expression is	then
Numeric	Numeric	A numeric comparison is performed (see below).
String	String	A string comparison is performed (see below).
Numeric	String	A compile error is generated.
Variant	String	A string comparison is performed (see below).
Variant	Numeric	A variant comparison is performed (see below).
Null variant	Any data type	Returns Null.
Variant	Variant	A variant comparison is performed (see below).

String Comparisons

If the two expressions are strings, then the operator performs a text comparison between the two string expressions, returning True if expression1 is less than expression2. The text comparison is case-sensitive if Option Compare is Binary; otherwise, the comparison is case-insensitive.

When comparing letters with regard to case, lowercase characters in a string sort greater than uppercase characters, so a comparison of "a" and "A" would indicate that "a" is greater than "A".

Numeric Comparisons

When comparing two numeric expressions, the less precise expression is converted to be the same type as the more precise expression.

Dates are compared as doubles. This may produce unexpected results as it is possible to have two dates that, when viewed as text, display as the same date when, in fact, they are different. This can be seen in the following example:

```
Sub Main()
    Dim date1 As Date
    Dim date2 As Date

    date1 = Now
    date2 = date1 + 0.000001  'Adds a fraction of a second.

    MsgBox date2 = date1          'Prints False (the dates are different).
    MsgBox date1 & "," & date2        'Prints two dates that are the same.
End Sub
```

Variant Comparisons

When comparing variants, the actual operation performed is determined at execution time according to the following table:

If one variant is	and the other variant is	then
Numeric	Numeric	The variants are compared as numbers.
String	String	The variants are compared as text.
Numeric	String	The number is less than the string.
Null	Any other data type	Null.
Numeric	Empty	The number is compared with 0.
String	Empty	The string is compared with a zero-length string.

Example

```
Sub Main()
    'Tests two literals and displays the result.
    If 5 < 2 Then
        MsgBox "5 is less than 2."
    Else
        MsgBox "5 is not less than 2."
    End If

    'Tests two strings and displays the result.
    If "This" < "That" Then
        MsgBox "'This' is less than 'That'."
    Else
        MsgBox "'That' is less than 'This'."
    End If
End Sub
```

See Also

Operator Precedence (topic); Is (operator); Like (operator); Option Compare (statement).

statement

Const

Syntax

```
Const name [As type] = expression [,name [As type] = expression]...
```

Description

Declares a constant for use within the current script.

Comments

- The name is only valid within the current Delrina Basic. Constant names must follow these rules:
  1. Must begin with a letter.
  2. May contain only letters, digits, and the underscore character.
  3. Must not exceed 80 characters in length.
  4. Cannot be a reserved word.
- Constant names are not case-sensitive.
- The expression must be assembled from literals or other constants. Calls to functions are not allowed except calls to the Chr$ function, as shown below:

```
Const s$ = "Hello, there" + Chr(44)
```

- Constants can be given an explicit type by declaring the name with a type-declaration character, as shown below:

```
Const a% = 5            'Constant Integer whose value is 5
Const b# = 5            'Constant Double whose value is 5.0
Const c$ = "5"          'Constant String whose value is "5"
Const d! = 5            'Constant Single whose value is 5.0
Const e& = 5            'Constant Long whose value is 5
```

- The type can also be given by specifying the As type clause:

```
Const a As Integer = 5      'Constant Integer whose value is 5
Const b As Double = 5       'Constant Double whose value is 5.0
Const c As String = "5"     'Constant String whose value is "5"
Const d As Single = 5       'Constant Single whose value is 5.0
Const e As Long = 5         'Constant Long whose value is 5
```

- You cannot specify both a type-declaration character and the type:

```
Const a% As Integer = 5     'THIS IS ILLEGAL.
```

- If an explicit type is not given, then Delrina Basic will choose the most imprecise type that completely represents the data, as shown below:

```
Const a = 5                 'Integer constant
Const b = 5.5               'Single constant
Const c = 5.5E200           'Double constant
```

- Constants defined within a Sub or Function are local to that subroutine or function. Constants defined outside of all subroutines and function can be used anywhere within that script. The following example demonstrates the scoping of constants:

```
Const DefFile = "default.txt"

Sub Test1
    Const DefFile = "foobar.txt"
    MsgBox DefFile              'Displays "foobar.txt".
End Sub
    Sub Test2
        MsgBox DefFile                          'Displays "default.txt".
    End Sub
```

Example

'This example displays the declared constants in a dialog box (crlf produces
'a new line in the dialog box).

```
Const crlf = Chr$(13) + Chr$(10)
Const greeting As String = "Hello, "
Const question1 As String = "How are you today?"

Sub Main()
    r = InputBox("Please enter your name","Enter Name")
    MsgBox greeting & r & crlf & crlf & question1
End Sub
```

See Also

DefType (statement); Let (statement); = (statement); Constants (topic).

topic

Constants

Constants are variables that cannot change value during script execution. The following constants are predefined by Delrina Basic:

True	False	Empty
Pi	ebRightButton	ebLeftButton
ebPortrait	ebLandscape	ebDOS
ebWindows	ebMaximized	ebMinimized
ebRestored	ebNormal	ebReadOnly
ebHidden	ebSystem	ebVolume
ebDirectory	ebArchive	ebNone
ebOKOnly	ebOKCancel	ebAbortRetryIgnore
ebYesNoCancel	ebYesNo	ebRetryCancel
ebCritical	ebQuestion	ebExclamation
ebInformation	ebApplicationModal	ebDefaultButton1
ebDefaultButton2	ebDefaultButton3	ebSystemModal
ebOK	ebCancel	ebAbort
ebRetry	ebIgnore	ebYes
ebNo	ebWin16	ebWin32
ebDOS16	ebSunOS	ebSolaris
ebHPUX	ebUltrix	ebIrix
ebAIX	ebNetWare	ebMacintosh
ebOS2	ebEmpty	ebNull
ebInteger	ebLong	ebSingle
ebDouble	ebDate	ebBoolean
ebObject	ebDataObject	ebVariant
ebDOS32	ebCurrency	

You can define your own constants using the Const statement.

function

Cos

Syntax

```
Cos(angle)
```

Description

Returns a Double representing the cosine of angle.

Comments

- The angle parameter is a Double specifying an angle in radians.

Example

```
'This example assigns the cosine of pi/4 radians (45 degrees) to C# and
'displays its value.

Sub Main()
    c# = Cos(3.14159 / 4)
    MsgBox "The cosine of 45 degrees is: " & c#
End Sub
```

See Also

Tan (function); Sin (function); Atn (function).

function

CreateObject

Syntax

```
CreateObject(class$)
```

Description

Creates an OLE automation object and returns a reference to that object.

Comments

- The class$ parameter specifies the application used to create the object and the type of object being created. It uses the following syntax:

  "application.class",

- where application is the application used to create the object and class is the type of the object to create.
- At runtime, CreateObject looks for the given application and runs that application if found. Once the object is created, its properties and methods can be accessed using the dot syntax (e.g., object.property = value).
- There may be a slight delay when an automation server is loaded (this depends on the speed with which a server can be loaded from disk). This delay is reduced if an instance of the automation server is already loaded.

Examples

'This first example instantiates Microsoft Excel. It then uses the 'resulting object to make Excel visible and then close Excel.

```
Sub Main()
    Dim Excel As Object

    On Error GoTo Trap1                          'Set error trap.
    Set Excel = CreateObject("excel.application")   'Instantiate object.
    Excel.Visible = True                         'Make Excel visible.
    Sleep 5000                                   'Wait 5 seconds.
    Excel.Quit                                   'Close Excel.

    Exit Sub                                     'Exit before error trap.

Trap1:
    MsgBox "Can't create Excel object."          'Display error message.
    Exit Sub                                     'Reset error handler.
End Sub
```

'This second example uses CreateObject to instantiate a Visio object. It
'then uses the resulting object to create a new document.

```
Sub Main()
    Dim Visio As Object
    Dim doc As Object
    Dim page As Object
    Dim shape As Object

    On Error Goto NO_VISIO
    Set Visio = CreateObject("visio.application") 'Create Visio object.
    On Error Goto 0
```

```
    Set doc = Visio.Documents.Add("")                   'Create a new document.
    Set page = doc.Pages(1)                                 'Get first page.
    Set shape = page.DrawRectangle(1,1,4,4)             'Create a new shape.
    shape.text = "Hello, world."                        'Set text within shape.
    End
NO_VISIO:
    MsgBox "'Visio' cannot be found!",ebExclamation
End Sub
```

See Also

GetObject (function); Object (data type).

topic

Cross-Platform Scripting

This section discusses different techniques that can be used to ensure that a given script runs on all platforms that support Delrina Basic.

Querying the Platform

A script can query the platform in order to take appropriate actions for that platform. This is done using the Basic.OS property. The following example uses this method to display a message to the user:

```
Sub Main()
    If Basic.OS = ebWindows Then
        MsgBox "This is a message."
    Else
        Print "This is a message."
    End If
End Sub
```

Querying the Capabilities of a Platform

Some capabilities of the current platform can be determined using the Basic.Capability method. This method takes a number indicating which capability is being queried and returns either True or False depending on whether that capability is or is not supported on the current platform. The following example uses this technique to read hidden files:

```
Sub Main()
    If Basic.Capability(3) Then
        f$ = Dir("*",ebHidden)  'This platform supports hidden files.
    Else
        f$ = Dir("*")                   'This platform doesn't support hidden files.
    End If

    'Print all the files.
    While f$ <> ""
        x = x + 1
        Msgbox "Matching file " & x & " is: " & f$
        f$ = Dir
    WEnd
End Sub
```

Byte Ordering with Files

One of the main differences between platforms is byte ordering. On some platforms, the processor requires that the bytes that make up a given data item be reversed from their expected ordering.

Byte ordering becomes problematic if binary data is transferred from one platform to another. This can only occur when writing data to files. For this reason, it is strongly recommended that files that are to be transported to a different platform with different byte ordering be sequential (i.e., do not use Binary and Random files).

If a Binary or Random file needs to be transported to another platform, you will have to take into consideration the following:

1. You must either decide on a byte ordering for your file or write information to the file indicating its byte ordering so that it can be queried by the script that is to read the file.
2. When reading a file on a platform in which the byte ordering matches, nothing further needs to be done. If the byte ordering is different, then the bytes of each data item read from a file need to be

reversed. This is a difficult proposition.

Byte Ordering with Structures

Due to byte ordering differences between platforms, structure copying using the LSet statement produces different results. Consider the following example:

```
Type TwoInts
    first As Integer
    second As Integer
End Type

Type OneLong
    first As Long
End Type

Sub Main()
    Dim l As OneLong
    Dim i As TwoInts
    l.First = 4
    LSet i = l
    MsgBox "First integer: " & i.first
    MsgBox "Second integer: " & i.second
End Sub
```

Bytes are stored in memory with the most significant byte first (known as little-endian format). Thus, the above example displays two dialog boxes, the first one displaying the number 4 and the second displaying the number 0.

Script that rely on binary images of data must take the byte ordering of the current platform into account.

Reading Text Files and Writing to Them

Different platforms use different characters to represent end-of-line in a file. For example, under Windows, a carriage-return/line-feed pair is used.

Delrina Basic takes this into account when reading text files. The following combinations are recognized and interpreted as end-of-line:

Carriage return	Chr(13)
Carriage return/line feed	Chr(13) + Chr(10)
Line feed	Chr(10)

When writing to text files, Delrina Basic uses the end-of-line appropriate to that platform. You can retrieve the same end-of-line used by Delrina Basic using the Basic.Eoln$ property:

```
crlf = Basic.Eoln$
Print #1,"Line 1." & crlf & "Line 2."
```

Alignment

A major difference between platforms supported by Delrina Basic is the forced alignment of data. Delrina Basic handles most alignment issues itself.

Portability of Compiled Code

Scripts compiled under Delrina Basic can be run without recompilation on any platform supported by

Delrina Basic.

Unsupported Language Elements

A compiled Delrina Basic is portable to any platform on which Delrina Basic runs. Because of this, it is possible to run a script that was compiled on another platform and contains calls to language elements not supported by the current platform.

Delrina Basic generates a runtime error when unsupported language elements are encountered during execution. For example, the following script will run without errors under Windows but generate a runtime error when run under UNIX:

```
Sub Main()
    MsgBox "Hello, world."
End Sub
```

If you trap a call to an unsupported function, the function will return one of the following values:

Data Type	Skipped Function Returns
Integer	0
Double	0.0
Single	0.0
Long	0
Date	December 31, 1899
Boolean	False
Variant	Empty
Object	Nothing

function

CSng

Syntax

```
CSng(expression)
```

Description

Converts expression to a Single.

Comments

- This function accepts any expression convertible to a Single, including strings. A runtime error is generated if expression is Null. Empty is treated as 0.0.
- A runtime error results if the passed expression is not within the valid range for Single.
- When passed a numeric expression, this function has the same effect as assigning the numeric expression to a Single.
- When used with variants, this function guarantees that the expression is converted to a Single variant (VarType 4).

Example

```
'This example displays the value of a String converted to a Single.

Sub Main()
    s$ = "100"
    MsgBox "The single value is: " & CSng(s$)
End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CDbl (function); CInt (function); CLng (function); CStr (function); CVar (function); CVErr (function); Single (data type).

function

CStr

Syntax

```
CStr(expression)
```

Description

Converts expression to a String.

Comments

- Unlike Str$ or Str, the string returned by CStr will not contain a leading space if the expression is positive. Further, the CStr function correctly recognizes thousands and decimal separators for your locale.
- Different data types are converted to String in accordance with the following rules:

Data Type	CStr Returns
Any numeric type	A string containing the number without the leading space for positive values.
Date	A string converted to a date using the short date format.
Boolean	A string containing either "True" or "False".
Null variant	A runtime error.
Empty variant	A zero-length string.

Example

'This example displays the value of a Double converted to a String.

```
Sub Main()
    s# = 123.456
    MsgBox "The string value is: " & CStr(s#)
End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CDbl (function); CInt (function); CLng (function); CSng (function); CVar (function); CVErr (function); String (data type); Str, Str$ (functions).

function

CurDir, CurDir$

Syntax

```
CurDir[$][(drive$)]
```

Description

Returns the current directory on the specified drive. If no drive$ is specified or drive$ is zero-length, then the current directory on the current drive is returned.

Comments

- CurDir$ returns a String, whereas CurDir returns a String variant.
- Delrina Basic generates a runtime error if drive$ is invalid.

Example

```
'This example saves the current directory, changes to the next higher
'directory, and displays the change; then restores the original directory
'and displays the change. Note: The dot designators will not work with
'all platforms.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    save$ = CurDir
    ChDir ("..")
    MsgBox "Old directory: " & save$ & crlf & "New directory: " & CurDir
    ChDir (save$)
    MsgBox "Directory restored to: " & CurDir
End Sub
```

See Also

ChDir (statement); ChDrive (statement); Dir, Dir$ (functions); MkDir (statement); RmDir (statement).

data type

Currency

Syntax

Currency

Description

A data type used to declare variables capable of holding fixed-point numbers with 15 digits to the left of the decimal point and 4 digits to the right.

Comments

- Currency variables are used to hold numbers within the following range:
  `-922,337,203,685,477.5808 <= currency <= 922,337,203,685,477.5807`
- Due to their accuracy, Currency variables are useful within calculations involving money.
- The type-declaration character for Currency is @.

Storage

Internally, currency values are 8-byte integers scaled by 10000. Thus, when appearing within a structure, currency values require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.

See Also

Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CCur (function).

function

CVar

Syntax

```
CVar(expression)
```

Description

Converts expression to a Variant.

Comments

- This function is used to convert an expression into a variant. Use of this function is not necessary (except for code documentation purposes) because assignment to variant variables automatically performs the necessary conversion:

```
Sub Main()
    Dim v As Variant
    v = 4 & "th"              'Assigns "4th" to v.
    MsgBox "You came in: " & v
    v = CVar(4 & "th")        'Assigns "4th" to v.
    MsgBox "You came in: " & v
End Sub
```

Example

'This example converts an expression into a Variant.

```
Sub Main()
    Dim s As String
    Dim a As Variant
    s = CStr("The quick brown fox ")
    msg = CVar(s & "jumped over the lazy dog.")
    MsgBox msg
End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CDbl (function); CInt (function); CLng (function); CSng (function); CStr (function); CVErr (function); Variant (data type).

function

CVErr

Syntax

```
CVErr(expression)
```

Description

Converts expression to an error.

Comments

- This function is used to convert an expression into a user-defined error number.
- A runtime error is generated under the following conditions:
- If expression is Null.
- If expression is a number outside the legal range for errors, which is as follows:

  0 <= expression <= 65535

- If expression is Boolean.
- If expression is a String that can't be converted to a number within the legal range.
- Empty is treated as 0.

Example

'This example simulates a user-defined error and displays the error number.

```
Sub Main()
    MsgBox "The error is: " & CStr(CVErr(2046))
End Sub
```

See Also

CCur (function); CBool (function); CDate, CVDate (functions); CDbl (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function), IsError (function).

data type

Date

Syntax

Date

Description

A data type capable of holding date and time values.

Comments

- Date variables are used to hold dates within the following range:

```
January 1, 100 00:00:00 <= date <= December 31, 9999 23:59:59

-6574340 <= date <= 2958465.99998843
```

- Internally, dates are stored as 8-byte IEEE double values. The integer part holds the number of days since December 31, 1899, and the fractional part holds the number of seconds as a fraction of the day. For example, the number 32874.5 represents January 1, 1990 at 12:00:00.
- When appearing within a structure, dates require 8 bytes of storage. Similarly, when used with binary or random files, 8 bytes of storage are required.
- There is no type-declaration character for Date.
- Date variables that haven't been assigned are given an initial value of 0 (i.e., December 31, 1899).

Date Literals

Literal dates are specified using number signs, as shown below:

```
Dim d As Date
d = #January 1, 1990#
```

The interpretation of the date string (i.e., January 1, 1990 in the above example) occurs at runtime, using the current country settings. This is a problem when interpreting dates such as 1/2/1990. If the date format is M/D/Y, then this date is January 2, 1990. If the date format is D/M/Y, then this date is February 1, 1990. To remove any ambiguity when interpreting dates, use the universal date format:

```
date_variable = #YY/MM/DD HH:MM:SS#
```

The following example specifies the date June 3, 1965 using the universal date format:

```
Dim d As Date
d = #1965/6/3 10:23:45#
```

See Also

Currency (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CDate, CVDate (functions).

function

Date, Date$

Syntax

Date[$][()]

Description

Returns the current system date.

Comments

- The Date$ function returns the date using the short date format. The Date function returns the date as a Date variant.
- Use the Date/Date$ statements to set the system date.

Note: In prior versions of Delrina Basic, the Date$ function returned the date using a fixed date format. The date is now returned using the current short date format (defined by the operating system), which may differ from the previous fixed format.

Example

```
'This example saves the current date to Cdate$, then changes the
'date and displays the result. It then changes the date back to the
'saved date and displays the result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    TheDate$ = Date
    Date = "01/01/95"
    MsgBox "Saved date is: " & TheDate$ & crlf & "Changed date is: " & Date
    Date = TheDate$
    MsgBox "Restored date to: " & TheDate$
End Sub
```

See Also

CDate, CVDate (functions); Time, Time$ (functions); Date, Date$ (statements); Now (function); Format, Format$ (functions); DateSerial (function); DateValue (function).

statement

Date, Date$

Syntax

```
Date[$] = newdate
```

Description

Sets the system date to the specified date.

Comments

- The Date$ statement requres a string variable using one of the following formats:

  ```
  MM-DD-YYYY
  MM-DD-YY
  MM/DD/YYYY
  MM/DD/YY,
  ```

  where MM is a two-digit month between 1 and 31, DD is a two-digit day between 1 and 31, and YYYY is a four-digit year between 1/1/100 and 12/31/9999.

- The Date statement converts any expression to a date, including string and numeric values. Unlike the Date$ statement, Date recognizes many different date formats, including abbreviated and full month names and a variety of ordering options. If newdate contains a time component, it is accepted, but the time is not changed. An error occurs if newdate cannot be interpreted as a valid date.

Example

```
'This example saves the current date to Cdate$, then changes the
'date and displays the result. It then changes the date back to the
'saved date and displays the result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    TheDate$ = Date
    Date = "01/01/95"
    MsgBox "Saved date is: " & TheDate$ & crlf & "Changed date is: " & Date
    Date = TheDate$
    MsgBox "Restored date to: " & TheDate$
End Sub
```

See Also

Date, Date$ (functions); Time, Time$ (statements).

function

DateAdd

Syntax

```
DateAdd(interval$, increment&, date)
```

Description

Returns a Date variant representing the sum of date and a specified number (increment) of time intervals (interval$).

Comments

- This function adds a specified number (increment) of time intervals (interval$) to the specified date (date). The following table describes the parameters to the DateAdd function:

Parameter	Description
interval$	String expression indicating the time interval used in the addition.
increment	Integer indicating the number of time intervals you wish to add. Positive values result in dates in the future; negative values result in dates in the past.
date	Any expression convertible to a Date. string expression. An example of a valid date/time string would be "January 1, 1993".

The interval$ parameter specifies what unit of time is to be added to the given date. It can be any of the following:

Time	Interval
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

- To add days to a date, you may use either day, day of the year, or weekday, as they are all equivalent ("d", "y", "w").
- The DateAdd function will never return an invalid date/time expression. The following example adds two months to December 31, 1992:

```
s# = DateAdd("m",2,"December 31,1992")
```

In this example, s is returned as the double-precision number equal to "February 28, 1993", not "February 31, 1993".

- Delrina Basic generates a runtime error if you try subtracting a time interval that is larger than the time value of the date.

Example

```
'This example gets today's date using the Date$ function; adds three
'years, two months, one week, and two days to it; and then displays the
'result in a dialog box.
```

```
Sub Main()
    Dim sdate$
    sdate$ = Date$
    NewDate# = DateAdd("yyyy",4,sdate$)
    NewDate# = DateAdd("m",3,NewDate#)
    NewDate# = DateAdd("ww",2,NewDate#)
    NewDate# = DateAdd("d",1,NewDate#)
    s$ = "Four years, three months, two weeks, and one day from now will be: "
    s$ = s$ & Format(NewDate#,"long date")
    MsgBox s$
End Sub
```

See Also

DateDiff (function).

function

DateDiff

Syntax

```
DateDiff(interval$,date1,date2)
```

Description

Returns a Date variant representing the number of given time intervals between date1 and date2.

Comments

- The following table describes the parameters:

Parameter	Description
interval$	String expression indicating the specific time interval you wish to find the difference between.
date1	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1994".
date2	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1994".

The following table lists the valid time interval strings and the meanings of each. The Format$ function uses the same expressions.

Time	Interval
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

To find the number of days between two dates, you may use either day or day of the year, as they are both equivalent ("d", "y").

The time interval weekday ("w") will return the number of weekdays occurring between date1 and date2, counting the first occurrence but not the last. However, if the time interval is week ("ww"), the function will return the number of calendar weeks between date1 and date2, counting the number of Sundays. If date1 falls on a Sunday, then that day is counted, but if date2 falls on a Sunday, it is not counted.

The DateDiff function will return a negative date/time value if date1 is a date later in time than date2.

Example

```
'This example gets today's date and adds ten days to it. It then
'calculates the difference between the two dates in days and weeks
'and displays the result.
```

```
Sub Main()
    today$ = Format(Date$,"Short Date")
    NextWeek = Format(DateAdd("d",14,today$),"Short Date")
    DifDays# = DateDiff("d",today$,NextWeek)
    DifWeek# = DateDiff("w",today$,NextWeek)
    s$ = "The difference between " & today$ & " and " & NextWeek
    s$ = s$ & " is: " & DifDays# & " days or " & DifWeek# & " weeks"
    MsgBox s$
End Sub
```

See Also

DateAdd (function).

function

DatePart

Syntax

```
DatePart(interval$,date)
```

Description

Returns an Integer representing a specific part of a date/time expression.

Comments

The DatePart function decomposes the specified date and returns a given date/time element. The following table describes the parameters:

Parameter	Description
interval$	String expression that indicates the specific time interval you wish to identify within the given date.
date	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1995".

The following table lists the valid time interval strings and the meanings of each. The Format$ function uses the same expressions.

Time	Interval
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

The weekday expression starts with Sunday as 1 and ends with Saturday as 7.

Example

```
'This example displays the parts of the current date.

Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
    today$ = Date$
    qt = DatePart("q",today$)
    yr = DatePart("yyyy",today$)
    mo = DatePart("m",today$)
    wk = DatePart("ww",today$)
    da = DatePart("d",today$)
    s$ = "The current date is:" & crlf & crlf
    s$ = s$ & "Quarter          : " & qt & crlf
    s$ = s$ & "Year         : " & yr & crlf
    s$ = s$ & "Month        : " & mo & crlf
    s$ = s$ & "Week         : " & wk & crlf
    s$ = s$ & "Day          : " & da & crlf
    MsgBox s$
End Sub
```

See Also

Day (function); Minute (function); Second (function); Month (function); Year (function); Hour (function); Weekday (function), Format (function).

function

DateSerial

Syntax

```
DateSerial(year,month,day)
```

Description

Returns a Date variant representing the specified date.

Comments

- The DateSerial function takes the following parameters:

Parameter	Description
year	Integer between 100 and 9999
month	Integer between 1 and 12
day	Integer between 1 and 31

Example

```
'This example converts a date to a real number representing the
'serial date in days since December 30, 1899 (which is day 0).

Sub Main()
    tdate# = DateSerial(1993,08,22)
    MsgBox "The DateSerial value for August 22, 1993, is: " & tdate#
End Sub
```

See Also

DateValue (function); TimeSerial (function); TimeValue (function); CDate, CVDate (functions).

function

DateValue

Syntax

```
DateValue(date_string$)
```

Description

Returns a Date variant representing the date contained in the specified string argument.

Example

```
'This example returns the day of the month for today's date.

Sub Main()
    tdate$ = Date$
    tday$ = DateValue(tdate$)
    MsgBox "The date value of " & tdate$ & " is: " & tday$
End Sub
```

See Also

TimeSerial (function); TimeValue (function); DateSerial (function).

Note

- Date specifications vary depending on the international settings contained in the "intl" section of the win.ini file. The date items must follow the ordering determined by the current date format settings in use by Windows.

function

Day

Syntax

```
Day(date)
```

Description

Returns the day of the month specified by date.

Comments

- The value returned is an Integer between 0 and 31 inclusive.
- The date parameter is any expression that converts to a Date.

Example

```
'This example gets the current date and then displays it.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    CurDate = Now()
    MsgBox "Today is day " & Day(CurDate) & " of the month." & crlf &   "Tomorrow is day " & Day(CurDate + 1) & "."
End Sub
```

See Also

Minute (function); Second (function); Month (function); Year (function); Hour (function); Weekday (function); DatePart (function).

function

DDB

Syntax

```
DDB(Cost, Salvage, Life, Period)
```

Description

Calculates the depreciation of an asset for a specified Period of time using the double-declining balance method.

Comments

- The double-declining balance method calculates the depreciation of an asset at an accelerated rate. The depreciation is at its highest in the first period and becomes progressively lower in each additional period. DDB uses the following formula to calculate the depreciation:

  DDB = ((`Cost - Total_depreciation_from_all_other_periods`) * 2) / `Life`

- The DDB function uses the following parameters:

Parameter	Description
Cost	Double representing the initial cost of the asset
Salvage	Double representing the estimated value of the asset at the end of its predicted useful life
Life	Double representing the predicted length of the asset's useful life
Period	Double representing the period for which you wish to calculate the depreciation

- Life and Period must be expressed using the same units. For example, if Life is expressed in months, then Period must also be expressed in months.

Example

```
'This example calculates the depreciation for capital equipment
'that cost $10,000, has a service life of ten years, and is worth
'$2,000 as scrap. The dialog box displays the depreciation for each
'of the first four years.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    s$ = "Depreciation Table" & crlf & crlf
    For yy = 1 To 4
        CurDep# = DDB(10000.0,2000.0,10,yy)
        s$ = s$ & "Year " & yy & " : " & CurDep# & crlf
    Next yy
    MsgBox s$
End Sub
```

See Also

Sln (function); SYD (function).

statement

DDERun

Syntax

```
DDERun channel, command$
```

Description

Runs a command in another application.

Comments

- The DDERun statement takes the following parameters:

Parameter	Description
channel	Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.
command$	String containing the command to be run. The format of command$ depends on the receiving application.

- If the receiving application does not run the instructions, Delrina Basic generates a runtime error.

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.
'The command strings being created contain Microsoft Excel macro
'commands and may be concantenated and sent as one string to speed
'things up.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)    ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)     'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"     'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"                'Send value to cell.
    'Retrieve value and display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")

    DDETerminate ch%
    Msgbox "Finished..."
End Sub
```

See Also

DDEInitiate (function); DDEPoke (statement); DDERequest, DDERequest$ (functions); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

function

DDEInitiate

Syntax

```
DDEInitiate(application$, topic$)
```

Description

Initializes a DDE link to another application and returns a unique number subsequently used to refer to the open DDE channel.

Comments

- The DDEInitiate statement takes the following parameters:

Parameter	Description
application$	String containing the name of the application (the server) with which a DDE conversation will be established.
topic$	String containing the name of the topic for the conversation. The possible values for this parameter are described in the documentation for the server application.

- This function returns 0 if Delrina Basic cannot establish the link. This will occur under any of the following circumstances:
- The specified application is not running.
- The topic was invalid for that application.
- Memory or system resources are insufficient to establish the DDE link.

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)   ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)    'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"    'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"              'Send value to cell.
    'Retrieve value and display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")

    DDETerminate ch%
    Msgbox "Finished..."
End Sub
```

See Also

DDERun (statement); DDEPoke (statement); DDERequest, DDERequest$ (functions); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

statement

DDEPoke

Syntax

```
DDEPoke channel, DataItem, value
```

Description

Sets the value of a data item in the receiving application associated with an open DDE link.

Comments

- The DDEPoke statement takes the following parameters:

Parameter	Description
channel	Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.
DataItem	Data item to be set. This parameter can be any expression convertible to a String. The format depends on the server.
value	The new value for the data item. This parameter can be any expression convertible to a String. The format depends on the server. A runtime error is generated if value is Null.

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)   ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)    'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"    'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"              'Send value to cell.
    'Retrieve value and display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")

    DDETerminate ch%
    Msgbox "Finished..."
End Sub
```

See Also

DDERun (statement); DDEInitiate (function); DDERequest, DDERequest$ (functions); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Note

The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

function

DDERequest, DDERequest$

Syntax

```
DDERequest[$](channel,DataItem$)
```

Description

Returns the value of the given data item in the receiving application associated with the open DDE channel.

Comments

- DDERequest$ returns a String, whereas DDERequest returns a String variant.
- The DDERequest/DDERequest$ functions take the following parameters:

Parameter	Description
channel	Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.
DataItem$	String containing the name of the data item to request. The format for this parameter depends on the server.

- The format for the returned value depends on the server.

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)    ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)     'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"     'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"                'Send value to cell.
    'Retrieve value and display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")

    DDETerminate ch%
    Msgbox "Finished..."
End Sub
```

See Also

DDERun (statement); DDEInitiate (function); DDEPoke (statement); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the DeIrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

statement

DDESend

Syntax

```
DDESend application$, topic$, DataItem, value
```

Description

Initiates a DDE conversation with the server as specified by application$ and topic$ and sends that server a new value for the specified item.

Comments

- The DDESend statement takes the following parameters:

Parameter	Description
application$	String containing the name of the application (the server) with which a DDE conversation will be established.
topic$	String containing the name of the topic for the conversation. The possible values for this parameter are described in the documentation for the server application.
DataItem	Data item to be set. This parameter can be any expression convertible to a String. The format depends on the server.
value	New value for the data item. This parameter can be any expression convertible to a String. The format depends on the server. A runtime error is generated if value is Null.

- The DDESend statement performs the equivalent of the following statements:

```
ch% = DDEInitiate(application$,topic$)
DDEPoke ch%,item,data
DDETerminate ch%
```

Example

```
'This example sets the content of the first cell in an Excel spreadsheet.

Sub Main()
    Dim cmd,ch%
    id = Shell("c:\excel5\excel.exe",3)    'Start Excel.

    On Error Goto ExcelError
    DDESend "Excel","Sheet1","R1C1","Payroll For August 1995"
    Msgbox "Finished..."
    Exit Sub

ExcelError:
    MsgBox "Error sending data to Excel."
    Exit Sub 'Reset error handler.
End Sub
```

See Also

DDERun (statement); DDEInitiate (function); DDEPoke (statement); DDERequest, DDERequest$ (functions); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

statement

DDETerminate

Syntax

```
DDETerminate channel
```

Description

Closes the specified DDE channel.

Comments

- The channel parameter is an Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.
- All open DDE channels are automatically terminated when the script ends.

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)    ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)     'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"     'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"                'Send value to cell.
    'Retrieve value and display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")

    DDETerminate ch%
    Msgbox "Finished..."
End Sub
```

See Also

DDERun (statement); DDEInitiate (function); DDEPoke (statement); DDERequest, DDERequest$ (functions); DDESend (function); DDETerminateAll (statement); DDETimeout (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

statement

DDETerminateAll

Syntax

DDETerminateAll

Description

Closes all open DDE channels.

Comments

- All open DDE channels are automatically terminated when the script ends.

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)    ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)     'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"     'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"                'Send value to cell.
    'Retrieve value and display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")

    DDETerminateAll
    Msgbox "Finished..."
End Sub
```

See Also

DDERun (statement); DDEInitiate (function); DDEPoke (statement); DDERequest, DDERequest$ (functions); DDESend (function); DDETerminate (statement); DDETimeout (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

statement

DDETimeout

Syntax

```
DDETimeout milliseconds
```

Description

Sets the number of milliseconds that must elapse before any DDE command times out.

Comments

- The milliseconds parameter is a Long and must be within the following range:
  `0 <= milliseconds <= 2,147,483,647`

The default is 10,000 (10 seconds).

Example

```
'This example sets and retrieves a cell in an Excel spreadsheet.
'The timeout has been set to wait 2 seconds for Excel to respond
'before timing out.

Sub Main()
    Dim cmd,q,ch%
    q = Chr(34)    ' Define quotation marks.

    id = Shell("c:\excel5\excel.exe",3)    'Start Excel.
    ch% = DDEInitiate("Excel","Sheet1")
    DDETimeout 2000            'Wait 2 seconds for Excel to respond

    On Error Resume Next
    cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]"    'Activate worksheet.
    DDERun ch%,cmd

    DDEPoke ch%,"R1C1","$1000.00"              'Send value to cell.
    'Retrieve value a nd display.
    MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1")
        DDETerminate ch%
        Msgbox "Finished..."
End Sub
```

See Also

DDERun (statement); DDEInitiate (function); DDEPoke (statement); DDERequest, DDERequest$ (functions); DDESend (function); DDETerminate (statement); DDETerminateAll (statement).

Note

- The DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the Delrina Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

statement

Declare

Syntax

```
Declare {Sub | Function} name[TypeChar] [CDecl | Pascal | System | StdCall]
_
     [Lib "LibName$" [Alias "AliasName$"]] [([ParameterList])] [As type]
```

Where ParameterList is a comma-separated list of the following (up to 30 parameters are allowed):

```
[Optional] [ByVal | ByRef] ParameterName[()] [As ParameterType]
```

Description

Creates a prototype for either an external routine or a Delrina Basic routine that occurs later in the source module or in another source module.

Comments

- Declare statements must appear outside of any Sub or Function declaration.
- Declare statements are only valid during the life of the script in which they appear.
- The Declare statement uses the following parameters:

Parameter	Description
name	Any valid Delrina Basic name. When you declare functions, you can include a type-declaration character to indicate the return type. This name is specified as a normal Delrina Basic keyword-i.e., it does not appear within quotes.
TypeChar	An optional type-declaration character used when defining the type of data returned from functions. It can be any of the following characters: #, !, $, @, %, or &. For external functions, the @ character is not allowed.

- Type-declaration characters can only appear with function declarations, and take the place of the As type clause.

Note: Currency data cannot be returned from external functions. Thus, the @ type-declaration character cannot be used when declaring external functions.

CDecl	Optional keyword indicating that the external subroutine or function uses the C calling convention. With C routines, arguments are pushed right to left on the stack and the caller performs stack cleanup.
Pascal	Optional keyword indicating that this external subroutine or function uses the Pascal calling convention. With Pascal routines, arguments are pushed left to right on the stack and the called function performs stack cleanup.
System	Optional keyword indicating that the external subroutine or function uses the System calling convention. With System routines, arguments are pushed right to left on the stack, the caller performs stack cleanup, and the number of arguments is specified in the AL register.
StdCall	Optional keyword indicating that the external subroutine or function uses the StdCall calling convention. With StdCall routines, arguments are pushed right to left on the stack and the called function performs stack cleanup.
LibName$	Must be specified if the routine is external. This parameter specifies the name of the library or code resource containing the external routine and must appear within quotes. The LibName$ parameter can include an optional path specifying the exact location of the library or code resource.
AliasName$	Alias name that must be given to provide the name of the routine if the name parameter is

not the routine's real name. For example, the following two statements declare the same routine:

Declare Function GetCurrentTime Lib "user" () As Integer

Declare Function GetTime Lib "user" Alias "GetCurrentTime" _As Integer

Use an alias when the name of an external routine conflicts with the name of a Delrina Basic internal routine or when the external routine name contains invalid characters.

The AliasName$ parameter must appear within quotes.

type — Indicates the return type for functions.

For external functions, the valid return types are: Integer, Long, String, Single, Double, Date, Boolean, and data objects.

Note: Currency, Variant, fixed-length strings, arrays, user-defined types, and OLE automation objects cannot be returned by external functions.

Optional — Keyword indicating that the parameter is optional. All optional parameters must be of type Variant. Furthermore, all parameters that follow the first optional parameter must also be optional.

If this keyword is omitted, then the parameter being defined is required when calling this subroutine or function.

ByVal — Optional keyword indicating that the caller will pass the parameter by value. Parameters passed by value cannot be changed by the called routine.

ByRef — Optional keyword indicating that the caller will pass the parameter by reference. Parameters passed by reference can be changed by the called routine. If neither ByVal or ByRef are specified, then ByRef is assumed.

ParameterName — Name of the parameter, which must follow Delrina Basic naming conventions:

1. Must start with a letter.

2. May contain letters, digits, and the underscore character (_). Punctuation and type-declaration characters are not allowed. The exclamation point (!) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character.

3. Must not exceed 80 characters in length.

Additionally, ParameterName can end with an optional type-declaration character specifying the type of that parameter (i.e., any of the following characters: %, &, !, #, @).

() — Indicates that the parameter is an array.

ParameterType — Specifies the type of the parameter (e.g., Integer, String, Variant, and so on). The As ParameterType clause should only be included if ParameterName does not contain a type-declaraction character.

In addition to the default Delrina Basic data types, ParameterType can specify any user-defined structure, data object, or OLE automation object. If the data type of the parameter is not known in advance, then the Any keyword can be used. This forces the Delrina Basic compiler to relax type checking, letting any data type to be passed in place of the given argument.

Declare Sub Convert Lib "mylib" (a As Any)

The Any data type can only be used when passing parameters to external routines.

Passing Parameters

By default, Delrina Basic passes arguments by reference. Many external routines require a value rather than a reference to a value. The ByVal keyword does this. For example, this C routine

```
void MessageBeep(int);
```

would be declared as follows:

```
Declare Sub MessageBeep Lib "user" (ByVal n As Integer)
```

As an example of passing parameters by reference, consider the following C routine which requires a pointer to an integer as the third parameter:

```
int SystemParametersInfo(int,int,int *,int);
```

This routine would be declared as follows (notice the ByRef keyword in the third parameter):

```
Declare Function SystemParametersInfo Lib "user" (ByVal action As Integer,_
    ByVal uParam As Integer,ByRef pInfo As Integer,_
    ByVal updateINI As Integer) As Integer
```

Strings can be passed by reference or by value. When they are passed by reference, a pointer to the internal handle to the Delrina Basic string is passed. When they are passed by value, Delrina Basic passes a 32-bit pointer to a null-terminated string (i.e., a C string). If an external routine modifies a passed string variable, then there must be sufficient space within the string to hold the returned characters. This can be accomplished using the Space function, as shown in the following example:

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal dirname$,ByVal length%)
    :
Dim s As String
s = Space(128)
GetWindowsDirectory s,128
```

Another alternative to ensure that a string has sufficient space is to declare the string with a fixed length:

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal dirname$,ByVal length%)
    :
Dim s As String * 128           'Declare a fixed-length string.
GetWindowsDirectory s,len(s)    'Pass it to an external subroutine.
```

Calling Conventions with External Routines

For external routines, the argument list must exactly match that of the referenced routine. When calling an external subroutine or function, Delrina Basic needs to be told how that routine expects to receive its parameters and who is responsible for cleanup of the stack.

Passing Null Pointers

To pass a null pointer to an external procedure, declare the parameter that is to receive the null pointer as type Any, then pass a long value 0 by value:

```
Declare Sub Foo Lib "sample" (ByVal lpName As Any)

Sub Main()
    Sub Foo "Hello"         'Pass a 32-bit pointer to a null-terminated string
    Sub Foo ByVal 0&            'Pass a null pointer
End Sub
```

Passing Data to External Routines

The following table shows how the different data types are passed to external routines:

Data Type	Is Passed As

ByRef Boolean	A 32-bit pointer to a 2-byte value containing -1 or 0.
ByVal Boolean	A 2-byte value containing -1 or 0.
ByVal Integer	A 32-bit pointer to a 2-byte short integer.
ByRef Integer	A 2-byte short integer.
ByVal Long	A 32-bit pointer to a 4-byte long integer.
ByRef Long	A 4-byte long integer.
ByRef Single	A 32-bit pointer to a 4-byte IEEE floating-point value (a float).
ByVal Single	A 4-byte IEEE floating-point value (a float).
ByRef Double	A 32-bit pointer to an 8-byte IEEE floating-point value (a double).
ByVal Double	An 8-byte IEEE floating-point value (a double).
ByVal String	A 32-bit pointer to a null-terminated string. With strings containing embedded nulls (Chr$(0)), it is not possible to determine which null represents the end of the string. Therefore, the first null is considered the string terminator. An external routine can freely change the content of a string. It cannot, however, write beyond the end of the null terminator.
ByRef String	A 32-bit pointer to a 2-byte internal value representing the string. This value can only be used by external routines written specifically for Delrina Basic.
ByRef Date	A 32-bit pointer to an 8-byte IEEE floating-point value (a double).
ByVal Date	An 8-byte IEEE floating-point value (a double).
ByRef Currency	A 32-bit pointer to an 8-byte integer scaled by 10000.
ByVal Currency	An 8-byte integer scaled by 10000.
ByRef Variant	A 32-bit pointer to a 16-byte internal variant structure. This structure contains a 2-byte type (the same as that returned by the VarType function), followed by 6 bytes of slop (for alignment), followed by 8 bytes containing the value.
ByVal Variant	A 16-byte variant structure. This structure contains a 2-byte type (the same as that returned by the VarType function), followed by 6 bytes of slop (for alignment), followed by 8 bytes containing the value.
ByVal Object	For data objects, a 32-bit pointer to a 4-byte unsigned long integer. This value can only be used by external routines written specifically for Delrina Basic. For OLE automation objects, a 32-bit pointer to an LPDISPATCH handle is passed.
ByRef Object	For data objects, a 32-bit pointer to a 4-byte unsigned long integer that references the object. This value can only be used by external routines written specifically for Delrina Basic. For OLE automation objects, a 32-bit pointer to a 4-byte internal ID is passed. This value can only be used by external routines written specifically for Delrina Basic.
User-defined type	A 32-bit pointer to the structure. User-defined types can only be passed by reference. It is important to remember that structures in Delrina Basic are packed on 2-byte boundaries, meaning that the individual structure members may not be aligned consistently with similar structures declared in C.
Arrays	A 32-bit pointer to a packed array of elements of the given type. Arrays can only be passed by reference.
Dialogs	Dialogs cannot be passed to external routines.

- Only variable-length strings can be passed to external routines; fixed-length strings are automatically converted to variable-length strings.
- Delrina Basic passes data to external functions consistent with that routine's prototype as defined by the Declare statement. There is one exception to this rule: you can override ByRef parameters using the ByVal keyword when passing individual parameters. The following example shows a number of

different ways to pass an Integer to an external routine called Foo:

```
Declare Sub Foo Lib "MyLib" (ByRef i As Integer)

Sub Main
    Dim i As Integer
    i = 6
    Foo 6              'Passes a temporary integer (value 6) by reference
    Foo i              'Passes variable "i" by reference
    Foo (i)        'Passes a temporary integer (value 6) by reference
    Foo i + 1      'Passes temporary integer (value 7) by reference
    Foo ByVal i    'Passes i by value
End Sub
```

The above example shows that the only way to override passing a value by reference is to use the ByVal keyword.

Note: Use caution when using the ByVal keyword in this way. The external routine Foo expects to receive a pointer to an Integer-a 32-bit value; using ByVal causes Delrina Basic to pass the Integer by value-a 16-bit value. Passing data of the wrong size to any external routine will have unpredictable results.

Example

```
Declare Function IsLoaded% Lib "Kernel" Alias "GetModuleHandle" (ByVal KName$)

Declare Function GetProfileString Lib "Kernel" (ByVal SName$,ByVal KName$,ByVal Def$,ByVal Ret$,ByVal Size%)
As Integer

Sub Main()
    SName$ = "Intl"                 'Win.ini section name.
    KName$ = "sCountry"             'Win.ini country setting.
    ret$ = String(255,0)          'Initialize return string.

    If GetProfileString(SName$,KName$,"",ret$,Len(ret$)) Then
        MsgBox "Your country setting is: " & ret$
    Else
        MsgBox "There is no country setting in your win.ini file."
    End If

    If IsLoaded("Progman") Then
        MsgBox "Progman is loaded."
    Else
        MsgBox "Progman is not loaded."
    End If
End Sub
```

See Also

Call (statement), Sub...End Sub (statement), Function...End Function (statement).

If the libname$ parameter does not contain an explicit path to the DLL, the following search will be performed for the DLL (in this order):

1. The current directory
2. The Windows directory
3. The Windows system directory
4. The directory containing Delrina Basic
5. All directories listed in the path environment variable

If the first character of aliasname$ is #, then the remainder of the characters specify the ordinal number of the routine to be called. For example, the following two statements are equivalent (under

Windows, GetCurrentTime is defined as ordinal 15 in the user.exe module):

```
Declare Function GetTime Lib "user" Alias "GetCurrentTime" () As Integer
Declare Function GetTime Lib "user" Alias "#15" () As Integer
```

The names of external routines declared using the CDecl keyword are usually preceeded with an underscore character. When Delrina Basic searches for your external routine by name, it first attempts to load the routine exactly as specified. If unsuccessful, Delrina Basic makes a second attempt by prepending an underscore character to the specified name. If both attempts fail, then Delrina Basic generates a runtime error.

Windows has a limitation that prevents Double, Single, and Date values from being returned from routines declared with the CDecl keyword. Routines that return data of these types should be declared Pascal.

statement

DefType

Syntax

```
DefInt letterrange
DefLng letterrange
DefStr letterrange
DefSng letterrange
DefDbl letterrange
DefCur letterrange
DefObj letterrange
DefVar letterrange
DefBool letterrange
DefDate letterrange
```

Description

Establishes the default type assigned to undeclared or untyped variables.

Comments

- The DefType statement controls automatic type declaration of variables. Normally, if a variable is encountered that hasn't yet been declared with the Dim, Public, or Private statement or does not appear with an explicit type-declaration character, then that variable is declared implicitly as a variant (DefVar A-Z). This can be changed using the DefType statement to specify starting letter ranges for type other than integer. The letterrange parameter is used to specify starting letters. Thus, any variable that begins with a specified character will be declared using the specified Type.
- The syntax for letterrange is:
  `letter [-letter] [,letter [-letter]]...`
- DefType variable types are superseded by an explicit type declarationusing either a type-declaration character or the Dim, Public, or Private statement.
- The DefType statement only affects how Delrina Basic compiles scripts and has no effect at runtime.
- The DefType statement can only appear outside all Sub and Function declarations.
- The following table describes the data types referenced by the different variations of the DefType statement:

Statement	Data Type
DefInt	Integer
DefLng	Long
DefStr	String
DefSng	Single
DefDbl	Double
DefCur	Currency
DefObj	Object
DefVar	Variant
DefBool	Boolean
DefDate	Date

Example

```
DefStr a-m
DefLng n-r
DefSng s-u
DefDbl v-w
DefInt x-z

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a = 100.52
    n = 100.52
    s = 100.52
    v = 100.52
    x = 100.52
    msg = "The values are:" & crlf & crlf
    msg = msg & "(String) a: " & a & crlf
    msg = msg & "(Long) n: " & n & crlf
    msg = msg & "(Single) s: " & s & crlf
    msg = msg & "(Double) v: " & v & crlf
    msg = msg & "(Integer) x: " & x & crlf
    MsgBox msg
End Sub
```

See Also

Currency (data type); Date (data type); Double (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); Integer (data type).

method

Desktop.ArrangeIcons

Syntax

Desktop.ArrangeIcons

Description

Reorganizes the minimized applications on the desktop.

Example

```
Sub Main()
    Desktop.ArrangeIcons
End Sub
```

See Also

Desktop.Cascade (method); Desktop.Tile (method).

method

Desktop.Cascade

Syntax

Desktop.Cascade

Description

Cascades all nonminimized windows.

Example

```
'This example cascades all the windows on the desktop. It first
'restores any minimized applications so that they are included in the
'cascade.

Sub Main()
    Dim apps$()
    AppList apps$
    For i = LBound(apps) To UBound(apps)
        AppRestore apps(i)
    Next i
    Desktop.Cascade
End Sub
```

See Also

Desktop.Tile (method); Desktop.ArrangeIcons (method).

method

Desktop.SetColors

Syntax

```
Desktop.SetColors ControlPanelItemName$
```

Description

Changes the system colors to one of a predefined color set.

Example

```
'This example lets the user to select any of the available Windows 'color schemes.

Sub Main()
    'Get color schemes from Windows
    Dim names$()
    ReadINISection "color schemes",names$,"CONTROL.INI"

SelectAgain:        'Let user to select color scheme
    color = SelectBox("Set Colors","Available Color Sets:",names$)
    If color <> -1 Then
        Desktop.SetColors names$(color)
        Goto SelectAgain
    End If
End Sub
```

See Also

Desktop.SetWallpaper (method).

Note

The names of the color sets are contained in the control.ini file.

method

Desktop.SetWallpaper

Syntax

```
Desktop.SetWallpaper filename$, isTile
```

Description

Changes the desktop wallpaper to the bitmap specified by filename$.

Comments

- The wallpaper will be tiled if isTile is True; otherwise, the bitmap will be centered on the desktop.
- To remove the wallpaper, set the filename$ parameter to "", as in the following example:

Desktop.SetWallpaper "",True

Example

'This example reads a list of .BMP files from the Windows directory 'and lets the user to select any of these as wallpaper.

```
Sub Main()
    Dim list$()

    ' Create the prefix for the bitmap filenames
    d$ = System.WindowsDirectory$
    If Right(d$,1) <> "\" Then d$ = d$ & "\"
    f$ = d$ & "*.BMP"

    FileList list$,f$      'Get list of bitmaps from Windows directory

    ' Were there any bitmaps?
    If ArrayDims(list$) = 0 Then
        MsgBox "There aren't any bitmaps in the Windows directory"
        Exit Sub
    End If

    'Add "(none)"
    ReDim Preserve list$ (UBound(list$) + 1)
    list$(UBound(list$)) = "(none)"

SelectAgain:        'Let user to select item
    paper = SelectBox("Set Wallpaper","Available Wallpaper:",list$)

    Select Case paper
        Case -1
            End
        Case UBound(list$)
            Desktop.SetWallPaper "",True
            Goto SelectAgain
        Case Else
            Desktop.SetWallPaper d$ & list$(paper),True
            Goto SelectAgain
    End Select
End Sub
```

See Also

Desktop.SetColors (method).

Note

The Desktop.SetWallpaper method makes permanent changes to the wallpaper by writing the new

wallpaper information to the win.ini file.

method

Desktop.Snapshot

Syntax

```
Desktop.Snapshot [spec]
```

Description

Takes a snapshot of a particular section of the screen and saves it to the Clipboard.

Comments

- The spec parameter is an Integer specifying the screen area to be saved. It can be any of the following:

0	Entire screen
1	Client area of the active application
2	Entire window of the active application
3	Client area of the active window
4	Entire window of the active window

- Before the snapshot is taken, each application is updated. This ensures that any application that is in the middle of drawing will have a chance to finish before the snapshot is taken.
- There is a slight delay if the specified window is large.

Example

```
'This example takes a snapshot of Program Manager and pastes the
'resulting bitmap into Windows Paintbrush.

Sub Main()
    AppActivate "Program Manager"          'Activate Program Manager.
    AppRestore
    Desktop.Snapshot 2                     'Place snapshot into Clipboard.
    id = Shell("pbrush.exe",3)             'Start new instance of PaintBrush.
    Menu "Edit.Paste"                      'Paste snapshot into Paintbrush.
End Sub
```

Note

Pictures are placed into the Clipboard in bitmap format.

method

Desktop.Tile

Syntax

Desktop.Tile

Description

Tiles all nonminimized windows.

Example

```
'This example tiles all the windows on the desktop. It first
'restores any minimized applications so that they are included in the
'tile.

Sub Main()
    Dim apps$()
    AppList apps$
    For i = LBound(apps) To UBound(apps)
        AppRestore apps(i)
    Next i
    Desktop.Tile
End Sub
```

See Also

Desktop.Cascade (method); Desktop.ArrangeIcons (method).

function

Dialog

Syntax

```
Dialog(DialogVariable [,[DefaultButton] [,Timeout]])
```

Description

Displays the dialog box associated with DialogVariable, returning an Integer indicating which button was clicked.

Comments

- The Dialog function returns any of the following values:

-1	The OK button was clicked.
0	The Cancel button was clicked.
>0	A push button was clicked. The returned number represents which button was clicked based on its order in the dialog box template (1 is the first push button, 2 is the second push button, and so on).

- The Dialog function accepts the following parameters:

Parameter	Description
DialogVariable	Name of a variable that has previously been dimensioned as a user dialog box. This is accomplished using the Dim statement: Dim MyDialog As MyTemplate All dialog variables are local to the Sub or Function in which they are defined. Private and public dialog variables are not allowed.
DefaultButton	An Integer specifying which button is to act as the default button in the dialog box. The value of DefaultButton can be any of the following:
-2	This value indicates that there is no default button.
-1	This value indicates that the OK button, if present, should be used as the default.
0	This value indicates that the Cancel button, if present, should be used as the default.
>0	This value indicates that the Nth button should be used as the default. This number is the index of a push button within the dialog box template. If DefaultButton is not specified, then -1 is used. If the number specified by DefaultButton does not correspond to an existing button, then there will be no default button. The default button appears with a thick border and is selected when the user presses Enter on a control other than a push button.
Timeout	An Integer specifying the number of milliseconds to display the dialog box before automatically dismissing it. If TimeOut is not specified or is equal to 0, then the dialog box will be displayed until dismissed by the user. If a dialog box has been dismissed due to a timeout, the Dialog function returns 0.

Example

```
'This example displays an abort/retry/ignore disk error dialog box.
```

```
Sub Main()
    Begin Dialog DiskErrorTemplate 16,32,152,48,"Disk Error"
        Text 8,8,100,8,"The disk drive door is open."
        PushButton 8,24,40,14,"Abort",.Abort
        PushButton 56,24,40,14,"Retry",.Retry
        PushButton 104,24,40,14,"Ignore",.Ignore
    End Dialog
    Dim DiskError As DiskErrorTemplate
    r% = Dialog(DiskError,3,0)
    MsgBox "You selected button: " & r%
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

statement

Dialog

Syntax

```
Dialog DialogVariable [,[DefaultButton] [,Timeout]]
```

Description

Same as the Dialog function, except that the Dialog statement does not return a value. (See Dialog [function].)

Example

'This example displays an Abort/Retry/Ignore disk error dialog box.

```
Sub Main()
    Begin Dialog DiskErrorTemplate 16,32,152,48,"Disk Error"
        Text 8,8,100,8,"The disk drive door is open."
        PushButton 8,24,40,14,"Abort",.Abort
        PushButton 56,24,40,14,"Retry",.Retry
        PushButton 104,24,40,14,"Ignore",.Ignore
    End Dialog
    Dim DiskError As DiskErrorTemplate
    Dialog DiskError,3,0
End Sub
```

See Also

Dialog (function).

statement

Dim

Syntax

```
Dim name [(<subscripts>)] [As [New] type] [,name [(<subscripts>)] [As [New] type]]...
```

Description

Declares a list of local variables and their corresponding types and sizes.

Comments

- If a type-declaration character is used when specifying name (such as %, @, &, $, or !), the optional [As type] expression is not allowed. For example, the following are allowed:

  Dim Temperature **As** Integer
  Dim Temperature%

- The subscripts parameter lets the declaration of dynamic and fixed arrays. The subscripts parameter uses the following syntax:

  ```
  [lower to] upper [,[lower to] upper]...
  ```

- The lower and upper parameters are integers specifying the lower and upper bounds of the array. If lower is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Delrina Basic supports a maximum of 60 array dimensions.
- The total size of an array (not counting space for strings) is limited to 64K.

Dynamic arrays are declared by not specifying any bounds:

Dim a**()**

- The type parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.
- A Dim statement within a subroutine or function declares variables local to that subroutine or function. If the Dim statement appears outside of any subroutine or function declaration, then that variable has the same scope as variables declared with the Private statement.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

```
Dim name As String * length
```

where length is a literal number specifying the string's length.

Implicit Variable Declaration

If Delrina Basic encounters a variable that has not been explicitly declared with Dim, then the variable will be implicitly declared using the specified type-declaration character (#, %, @, $, or &). If the variable appears without a type-declaration character, then the first letter is matched against any pending DefType statements, using the specified type if found. If no DefType statement has been encountered corresponding to the first letter of the variable name, then Variant is used.

Creating New Objects

The optional New keyword is used to declare a new instance of the specified data object. This keyword can only be used with data object types. Furthermore, this keyword cannot be used when declaring arrays.

At runtime, the application or extension that defines that object type is notified that a new object is being defined. The application responds by creating a new physical object (within the appropriate

context) and returning a reference to that object, which is immediately assigned to the variable being declared.

When that variable goes out of scope (i.e., the Sub or Function procedure in which the variable is declared ends), the application is notified. The application then performs some appropriate action, such as destroying the physical object.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
Integer	0
Long	0
Double	0.0
Single	0.0
Date	December 31, 1899 00:00:00
Currency	0.0
Boolean	False
Object	Nothing
Variant	Empty
String	"" (zero-length string)
User-defined type	Each element of the structure is given an initial value, as described above.
Arrays	Each element of the array is given an initial value, as described above.

Naming Conventions

Variable names must follow these naming rules:

1. Must start with a letter.
2. May contain letters, digits, and the underscore character (_); punctuation is not allowed. The exclamation point (!) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character.

   The last character of the name can be any of the following type-declaration characters: #, @, %, !, &, and $.
3. Must not exceed 80 characters in length.
4. Cannot be a reserved word.

Examples

```
'The following examples use the Dim statement to declare various
'variable types.
```

```
Sub Main()
    Dim i As Integer
    Dim l&                                  'long
    Dim s As Single
    Dim d#                                  'double
    Dim c$                                  'string
    Dim MyArray(10) As Integer              '10 element integer array
    Dim MyStrings$(2,10)                    '2-10 element string arrays
    Dim Filenames$(5 To 10)                 '6 element string array
    Dim Values(1 To 10,100 To 200)          '111 element variant array
End Sub
```

See Also

Redim (statement); Public (statement); Private (statement); Option Base (statement).

function

Dir, Dir$

Syntax

```
Dir$[(filespec$ [,attributes])]
```

Description

Returns a String containing the first or next file matching filespec$.

If filespec$ is specified, then the first file matching that filespec$ is returned. If filespec$ is not specified, then the next file matching the initial filespec$ is returned.

Comments

- Dir$ returns a String, whereas Dir returns a String variant.
- The Dir$/Dir functions take the following parameters:

Parameter	Description
filespec$	String containing a file specification. If this parameter is specified, then Dir$ returns the first file matching this file specification. If this parameter is omitted, then the next file matching the initial file specification is returned. If no path is specified in filespec$, then the current directory is used.
attributes	Integer specifying attributes of files you want included in the list, as described below. If omitted, then only the normal, read-only, and archive files are returned. An error is generated if Dir$ is called without first calling it with a valid filespec$. If there is no matching filespec$, then a zero-length string is returned.

Wildcards

The filespec$ argument can include wildcards, such as * and ?. The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complete searching patterns. The following table shows some examples:

This pattern	Matches these files	Doesn't match these files
S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
C*T	CAT	CAT.DOC CAP.TXT
C?T	CAT CUT	CAT.TXT CAPIT CT
*	(All files)	

Attributes

You can control which files are included in the search by specifying the optional attributes parameter. The Dir, Dir$ functions always return all normal, read-only, and archive files (ebNormal Or ebReadOnly

Or ebArchive). To include additional files, you can specify any combination of the following attributes (combined with the Or operator):

Constant	Value	Includes
ebNormal	0	Normal, Read-only, and archive files
ebHidden	2	Hidden files
ebSystem	4	System files
ebVolume	8	Volume label
ebDirectory	16	DOS subdirectories

Example

```
'This example uses Dir to fill a SelectBox with the first 10
'directory entries.

Const crlf = Chr$(13) + Chr$(10)
Option Base 1

Sub Main()
    Dim a$(10)
    i% = 1
    a(i%) = Dir("*.*")

    While (a(i%) <> "") and (i% < 10)
        i% = i% + 1
        a(i%) = Dir
    Wend

    r = SelectBox("Top 10 Directory Entries",,a)
End Sub
```

See Also

ChDir (statement); ChDrive (statement); CurDir, CurDir$ (functions); MkDir (statement); RmDir (statement); FileList (statement).

statement

DiskDrives

Syntax

```
DiskDrives array()
```

Description

Fills the specified String or Variant array with a list of valid drive letters.

Comments

- The array() parameter specifies either a zero- or a one-dimensioned array of strings or variants. The array can be either dynamic or fixed.
- If array() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound, UBound, and ArrayDims functions to determine the number and size of the new array's dimensions.
- If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.

Example

```
'This example builds and displays an array containing the first three
'available disk drives.

Sub Main()
    Dim drive$()
    DiskDrives drive$
    r% = SelectBox("Available Disk Drives",,drive$)
End Sub
```

See Also

ChDrive (statement); DiskFree (function).

function

DiskFree

Syntax

```
DiskFree&([drive$])
```

Description

Returns a Long containing the free space (in bytes) available on the specified drive.

Comments

If drive$ is zero-length or not specified, then the current drive is assumed.

Only the first character of the drive$ string is used.

Example

```
'This example uses DiskFree to set the value of i and then displays the
'result in a message box.

Sub Main()
    s$ = "c"
    i# = DiskFree(s$)
    MsgBox "Free disk space on drive '" & s$ & "' is: " & i#
End Sub
```

See Also

ChDrive (statement); DiskDrives (statement).

function

DlgControlId

Syntax

```
DlgControlId(ControlName$)
```

Description

Returns an Integer containing the index of the specified control as it appears in the dialog box template.

Comments

- The first control in the dialog box template is at index 0, the second is at index 1, and so on.
- The ControlName$ parameter contains the name of the .Identifier parameter associated with that control in the dialog box template.
- The Delrina Basic statements and functions that dynamically manipulate dialog box controls identify individual controls using either the .Identifier name of the control or the control's index. Using the index to refer to a control is slightly faster but results in code that is more difficult to maintain.

Example

```
'This example uses DlgControlId to verify which control was triggered
'and branches the dynamic dialog script accordingly.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 Then
        'Enable the next three controls.
        If DlgControlId(ControlName$) = 2 Then
            For i = 3 to 5
                DlgEnable i,DlgValue("CheckBox1")
            Next i
            DlgProc = 1         'Don't close the dialog box.
        End If
    ElseIf Action% = 1 Then
        'Set initial state upon startup
        For i = 3 to 5
            DlgEnable i,DlgValue("CheckBox1")
        Next i
    End If
End Function

Sub Main()
    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
        CheckBox 24,16,72,8,"Click Here",.CheckBox1
        CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2
        CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3
        CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4
        CheckBox 24,72,76,8,"Main Option 2",.CheckBox5
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText

(statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

function

DlgEnable

Syntax

DlgEnable(ControlName$ | ControlIndex)

Description

Returns True if the specified control is enabled; returns False otherwise.

Comments

- Disabled controls are dimmed and cannot receive keyboard or mouse input.
- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
- You cannot disable the control with the focus.

Example

'This example checks the status of a checkbox at the end of the dialog procedure and notifies the user accordingly.

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 Then
        'Enable the next three controls.
        If DlgControlId(ControlName$) = 2 Then
            For i = 3 to 5
                DlgEnable i,DlgValue("CheckBox1")
            Next i
            DlgProc = 1         'Don't close the dialog box.
        End If
    ElseIf Action% = 1 Then
        'Set initial state upon startup
        For i = 3 to 5
            DlgEnable i,DlgValue("CheckBox1")
        Next i
    End If

    If    DlgEnable(i) = True Then
        MsgBox "You do not have the required disk space.",ebExclamation,"Insufficient Disk Space"
    End If
End Function

Sub Main()
    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
        CheckBox 24,16,72,8,"Click Here",.CheckBox1
        CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2
        CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3
        CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4
        CheckBox 24,72,76,8,"Main Option 2",.CheckBox5
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgControl (statement); DlgEnable (statement); DlgFocus (function); DlgFocus (statement);

DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

statement

DlgEnable

Syntax

DlgEnable {ControlName$ | ControlIndex} [,isOn]

Description

Enables or disables the specified control.

Comments

- Disabled controls are dimmed and cannot receive keyboard or mouse input.
- The isOn parameter is an Integer specifying the new state of the control. It can be any of the following values:

0	The control is disabled.
1	The control is enabled.
Omitted	Toggles the control between enabled and disabled.

- Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).
- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```
'This example uses DlgEnable to turn on/off various dialog options.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 Then
        'Enable the next three controls.
        If DlgControlId(ControlName$) = 2 Then
            For i = 3 to 5
                DlgEnable i,DlgValue("CheckBox1")
            Next i
            DlgProc = 1         'Don't close the dialog box.
        End If
    ElseIf Action% = 1 Then
        'Set initial state upon startup
        For i = 3 to 5
            DlgEnable i,DlgValue("CheckBox1")
        Next i
    End If
End Function

Sub Main()
    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
        CheckBox 24,16,72,8,"Click Here",.CheckBox1
        CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2
        CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3
        CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4
        CheckBox 24,72,76,8,"Main Option 2",.CheckBox5
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

function

DlgFocus

Syntax

DlgFocus$[()]

Description

Returns a String containing the name of the control with the focus.

Comments

- The name of the control is the .Identifier parameter associated with the control in the dialog box template.

Example

```
'This code fragment makes sure that the control being disabled does not
'currently have the focus (otherwise, a runtime error would occur).

Sub Main()
    If DlgFocus = "Files" Then      'Does it have the focus?
        DlgFocus "OK"                   'Change the focus to another control.
    End If
    DlgEnable "Files",False             'Now we can disable the control.
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

statement

DlgFocus

Syntax

DlgFocus ControlName$ | ControlIndex

Description

Sets focus to the specified control.

Comments

- A runtime error results if the specified control is hidden, disabled, or nonexistent.
- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```
'This code fragment makes sure the user enters a correct value.
'If not, the control returns focus back to the TextBox for correction.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 and ControlName$ = "OK" Then
        If IsNumeric(DlgText$("TextBox1")) Then
            Msgbox "Duly Noted."
        Else
            Msgbox "Sorry, you must enter a number."
            DlgFocus "TextBox1"
            DlgProc = 1
        End If
    End If
End Function

Sub Main()
    Dim ListBox1$()
    Begin Dialog UserDialog ,,112,74,"Untitled",.DlgProc
        TextBox 12,20,88,12,.TextBox1
        OKButton 12,44,40,14
        CancelButton 60,44,40,14
        Text 12,11,88,8,"Enter Desired Salary:",.Text1
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

function

DlgListBoxArray

Syntax

DlgListBoxArray({ControlName$ | ControlIndex}, ArrayVariable)

Description

Fills a list box, combo box, or drop list box with the elements of an array, returning an Integer containing the number of elements that were actually set into the control.

Comments

- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
- The ArrayVariable parameter specifies a single-dimensioned array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.

Example

```
'This dialog function refills an array with files.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 1 Then
        Dim NewFiles$()                         'Create a new dynamic array.
        FileList NewFiles$,"c:\*.*"         'Fill the array with files.
        r% = DlgListBoxArray("Files",NewFiles$)     'Set items in the list box.
        DlgValue "Files",0              'Set the selection to the first item.
        DlgProc = 1                         'Don't close the dialog box.
    End If
End Function

Sub Main()
    Dim ListBox1$()
    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
        ListBox 8,12,112,72,ListBox1$,.Files
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

statement

DlgListBoxArray

Syntax

DlgListBoxArray {ControlName$ | ControlIndex}, ArrayVariable

Description

Fills a list box, combo box, or drop list box with the elements of an array.

Comments

- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
- The ArrayVariable parameter specifies a single-dimensioned array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.

Example

```
'This dialog function refills an array with files.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 1 Then
        Dim NewFiles$()                         'Create a new dynamic array.
        FileList NewFiles$,"c:\*.*"         'Fill the array with files.
        DlgListBoxArray "Files",NewFiles$'Set items in the list box.
        DlgValue "Files",0              'Set the selection to the first item.
        DlgProc = 1                         'Don't close the dialog box.
    End If
End Function

Sub Main()
    Dim ListBox1$()
    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
        ListBox 8,12,112,72,ListBox1$,.Files
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

function

DlgProc

Syntax

Function DlgProc(ControlName$, Action, SuppValue) [As Integer]

Description

Describes the syntax, parameters, and return value for dialog functions.

Comments

- Dialog functions are called by Delrina Basic during the processing of a custom dialog box. The name of a dialog function (DlgProc) appears in the Begin Dialog statement as the .DlgProc parameter.
- Dialog functions require the following parameters:

Parameter	Description
ControlName$	String containing the name of the control associated with Action.
Action	Integer containing the action that called the dialog function.
SuppValue	Integer of extra information associated with Action. For some actions, this parameter is not used.

- When Delrina Basic displays a custom dialog box, the user may click on buttons, type text into edit fields, select items from lists, and perform other actions. When these actions occur, Delrina Basic calls the dialog function, passing it the action, the name of the control on which the action occurred, and any other relevent information associated with the action.
- The following table describes the different actions sent to dialog functions:

Action	Description
1	This action is sent immediately before the dialog box is shown for the first time. This gives the dialog function a chance to prepare the dialog box for use. When this action is sent, ControlName$ contains a zero-length string, and SuppValue is 0. The return value from the dialog function is ignored in this case. **Before Showing the Dialog Box** After action 1 is sent, Delrina Basic performs additional processing before the dialog box is shown. Specifically, it cycles though the dialog box controls checking for visible picture or picture button controls. For each visible picture or picture button control, Delrina Basic attempts to load the associated picture. In addition to checking picture or picture button controls, Delrina Basic will automatically hide any control outside the confines of the visible portion of the dialog box. This prevents the user from tabbing to controls that cannot be seen. However, it does not prevent you from showing these controls with the DlgVisible statement in the dialog function.
2	This action is sent when: A button is clicked, such as OK, Cancel, or a push button. In this case, ControlName$ contains the name of the button. SuppValue contains 1 if an OK button was clicked and 2 if a Cancel button was clicked; SuppValue is undefined otherwise. If the dialog function returns 0 in response to this action, then the dialog box will be closed. Any other value causes Delrina Basic to continue dialog processing. A check box's state has been modified. In this case, ControlName$ contains the name of the check box, and SuppValue contains the new state of the check box (1 if on, 0 if off). An option button is selected. In this case, ControlName$ contains the name of the option button that was clicked, and SuppValue contains the index of the option button within the option button group (0-based). The current selection is changed in a list box, drop list box, or combo box. In this case, ControlName$ contains the name of the list box, combo box, or drop list box, and

	SuppValue contains the index of the new item (0 is the first item, 1 is the second, and so on).
3	This action is sent when the content of a text box or combo box has been changed. This action is only sent when the control loses focus. When this action is sent, ControlName$ contains the name of the text box or combo box, and SuppValue contains the length of the new content. The dialog function's return value is ignored with this action.
4	This action is sent when a control gains the focus. When this action is sent, ControlName$ contains the name of the control gaining the focus, and SuppValue contains the index of the control that lost the focus (0-based). The dialog function's return value is ignored with this action.
5	This action is sent continuously when the dialog box is idle. If the dialog function returns 1 in response to this action, then the idle action will continue to be sent. If the dialog function returns 0, then Delrina Basic will not send any additional idle actions. When the idle action is sent, ControlName$ contains a zero-length string, and SuppValue contains the number of times the idle action has been sent so far.
6	This action is sent when the dialog box is moved. The ControlName$ parameter contains a zero-length string, and SuppValue is 0. The dialog function's return value is ignored with this action.

- User-defined dialog boxes cannot be nested. In other words, the dialog function of one dialog box cannot create another user-defined dialog box. You can, however, invoke any built-in dialog box, such as MsgBox or InputBox$.
- Within dialog functions, you can use the following additional Delrina Basic statements and functions. These statements let you to manipulate the dialog box controls dynamically.

DlgVisible	DlgText$	DlgText
DlgSetPicture	DlgListBoxArray	DlgFocus
DlgEnable	DlgControlId	

Example

```
'This dialog function enables/disables a group of option buttons
'when a check box is clicked.

Function SampleDlgProc(ControlName$,Action%,SuppValue%)
    If Action% = 2 And ControlName$ = "Printing" Then
        DlgEnable "PrintOptions",SuppValue%
        SampleDlgProc = 1     'Don't close the dialog box.
    End If
End Function

Sub Main()
    Begin Dialog SampleDialogTemplate 34,39,106,45,"Sample",.SampleDlgProc
        OKButton 4,4,40,14
        CancelButton 4,24,40,14
        CheckBox 56,8,38,8,"Printing",.Printing
        OptionGroup .PrintOptions
            OptionButton 56,20,51,8,"Landscape",.Landscape
            OptionButton 56,32,40,8,"Portrait",.Portrait
    End Dialog
    Dim SampleDialog As SampleDialogTemplate
    SampleDialog.Printing = 1
    r% = Dialog(SampleDialog)
End Sub
```

See Also

Begin Dialog (statement).

statement

DlgSetPicture

Syntax

DlgSetPicture {ControlName$ | ControlIndex},PictureName$,PictureType

Description

Changes the content of the specified picture or picture button control.

Comments

- The DlgSetPicture statement accepts the following parameters:

Parameter	Description
ControlName$	String containing the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specified control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
PictureName$	String containing the name of the picture. If PictureType is 0, then this parameter specifies the name of the file containing the image. If PictureType is 10, then PictureName$ specifies the name of the image within the resource of the picture library. If PictureName$ is empty, then the current picture associated with the specified control will be deleted. Thus, a technique for conserving memory and resources would involve setting the picture to empty before hiding a picture control.
PictureType	Integer specifying the source for the image. The following sources are supported: 0 The image is contained in a file on disk. 10 The image is contained in the picture library specified by the Begin Dialog statement. When this type is used, the PictureName$ parameter must be specified with the Begin Dialog statement.

Examples

```
Sub Main()
    DlgSetPicture "Picture1","\windows\checks.bmp",0    'Set picture from a file.

    DlgSetPicture 27,"FaxReport",10          'Set control 10's image
                                             'from a library.
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function), Picture (statement), PictureButton (statement).

statement

DlgText

Syntax

DlgText {ControlName$ | ControlIndex}, NewText$

Description

Changes the text content of the specified control.

Comments

- The effect of this statement depends on the type of the specified control:

Control Type	Effect of DlgText
Picture	Runtime error.
Option group	Runtime error.
Drop list box	Sets the current selection to the item matching NewText$. If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with NewText$. If no match is found, then the selection is removed.
OK button	Sets the label of the control to NewText$.
Cancel button	Sets the label of the control to NewText$.
Push button	Sets the label of the control to NewText$.
List box	Sets the current selection to the item matching NewText$. If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with NewText$. If no match is found, then the selection is removed.
Combo box	Sets the content of the edit field of the combo box to NewText$.
Text	Sets the label of the control to NewText$.
Text box	Sets the content of the text box to NewText$.
Group box	Sets the label of the control to NewText$.
Option button	Sets the label of the control to NewText$.

- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```
Sub Main()
    DlgText "GroupBox1","Save Options"        'Change text of group box 1.

    If DlgText$(9) = "Save Options" Then
        DlgText 9,"Editing Options"           'Change text to "Editing Options".
    End If
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

function

DlgText$

Syntax

DlgText$(ControlName$ | ControlIndex)

Description

Returns the text content of the specified control.

Comments

- The text returned depends on the type of the specified control:

Control Type	Value Returned by DlgText$
Picture	No value is returned. A runtime error occurs.
Option group	No value is returned. A runtime error occurs.
Drop list box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
OK button	Returns the label of the control.
Cancel button	Returns the label of the control.
Push button	Returns the label of the control.
List box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
Combo box	Returns the content of the edit field portion of the combo box.
Text	Returns the label of the control.
Text box	Returns the content of the control.
Group box	Returns the label of the control.
Option button	Returns the label of the control.

- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```
'This code fragment makes sure the user enters a correct value.
'If not, the control returns focus back to the TextBox for correction.

Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 and ControlName$ = "OK" Then
        If IsNumeric(DlgText$("TextBox1")) Then
            Msgbox "Duly Noted."
        Else
            Msgbox "Sorry, you must enter a number."
            DlgFocus "TextBox1"
            DlgProc = 1
        End If
    End If
End Function
```

```
Sub Main()
    Dim ListBox1$()
    Begin Dialog UserDialog ,,112,74,"Untitled",.DlgProc
        TextBox 12,20,88,12,.TextBox1
        OKButton 12,44,40,14
        CancelButton 60,44,40,14
        Text 12,11,88,8,"Enter Desired Salary:",.Text1
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

function

DlgValue

Syntax

DlgValue(ControlName$ | ControlIndex)

Description

Returns an Integer indicating the value of the specified control.

Comments

- The value of any given control depends on its type, according to the following table:

Control Type	DlgValue Returns
Option group	The index of the selected option button within the group (0 is the first option button, 1 is the second, and so on).
List box	The index of the selected item.
Drop list box	The index of the selected item.
Check box	1 if the check box is checked; 0 otherwise.

- A runtime error is generated if DlgValue is used with controls other than those listed in the above table.
- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```
'This code fragment toggles the value of a check box.

Sub Main()
    If DlgValue("MyCheckBox") = 1 Then
        DlgValue "MyCheckBox",0
    Else
        DlgValue "MyCheckBox",1
    End If
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

statement

DlgValue

Syntax

DlgValue {ControlName$ | ControlIndex},Value

Description

Changes the value of the given control.

Comments

The value of any given control is an Integer and depends on its type, according to the following table:

Control Type	Description of Value
Option group	The index of the new selected option button within the group (0 is the first option button, 1 is the second, and so on).
List box	The index of the new selected item.
Drop list box	The index of the new selected item.
Check box	1 if the check box is to be checked; 0 if the check is to be removed.

- A runtime error is generated if DlgValue is used with controls other than those listed in the above table.
- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```
'This code fragment toggles the value of a check box.

Sub Main()
    If DlgValue("MyCheckBox") = 1 Then
        DlgValue "MyCheckBox",0
    Else
        DlgValue "MyCheckBox",1
    End If
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgVisible (statement); DlgVisible (function).

function

DlgVisible

Syntax

DlgVisible(ControlName$ | ControlIndex)

Description

Returns True if the specified control is visible; returns False otherwise.

The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the template (0 is the first control in the template, 1 is the second, and so on).

A runtime error is generated if DlgVisible is called with no user dialog is active.

Example

```
Sub Main()
    If DlgVisible("Portrait") Then Beep

    If DlgVisible(10) And DlgVisible(12) Then
        MsgBox "The 10th and 12th controls are visible."
    End If
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (function).

statement

DlgVisible

Syntax

DlgVisible {ControlName$ | ControlIndex} [,isOn]

Description

Hides or shows the specified control.

Comments

- Hidden controls cannot be seen in the dialog box and cannot receive the focus using Tab.
- The isOn parameter is an Integer specifying the new state of the control. It can be any of the following values:

1	The control is shown.
0	The control is hidden.
Omitted	Toggles the visibility of the control.

- Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).
- The ControlName$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Picture Caching

When the dialog box is first created and before it is shown, Delrina Basic calls the dialog function with action set to 1. At this time, no pictures have been loaded into the picture controls contained in the dialog box template. After control returns from the dialog function and before the dialog box is shown, Delrina Basic will load the pictures of all visible picture controls. Thus, it is possible for the dialog function to hide certain picture controls, which prevents the associated pictures from being loaded and causes the dialog box to load faster. When a picture control is made visible for the first time, the associated picture will then be loaded.

Example

```
'This example creates a dialog box with two panels. The DlgVisible
'statement is used to show or hide the controls of the different
'panels.

Sub EnableGroup(start%,finish%)
    For i = 6 To 13                         'Disable all options.
        DlgVisible i,False
    Next i
    For i = start% To finish%               'Enable only the right ones.
        DlgVisible i,True
    Next i
End Sub
```

```
Function DlgProc(ControlName$,Action%,SuppValue%)
    If Action% = 1 Then
        DlgValue "WhichOptions",0        'Set to save options.
        EnableGroup 6,8                  'Enable the save options.
    End If
    If Action% = 2 And ControlName$ = "SaveOptions" Then
        EnableGroup 6,8                  'Enable the save options.
        DlgProc = 1                      'Don't close the dialog box.
    End If
    If Action% = 2 And ControlName$ = "EditingOptions" Then
        EnableGroup 9,13                     'Enable the editing options.
        DlgProc = 1                      'Don't close the dialog box.
    End If
End Function

Sub Main()
    Begin Dialog OptionsTemplate 33,33,171,134,"Options",.DlgProc
        'Background (controls 0-5)
        GroupBox 8,40,152,84,""
        OptionGroup .WhichOptions
            OptionButton 8,8,59,8,"Save Options",.SaveOptions
            OptionButton 8,20,65,8,"Editing Options",.EditingOptions
        OKButton 116,7,44,14
        CancelButton 116,24,44,14

        'Save options (controls 6-8)
        CheckBox 20,56,88,8,"Always create backup",.CheckBox1
        CheckBox 20,68,65,8,"Automatic save",.CheckBox2
        CheckBox 20,80,70,8,"Let overwriting",.CheckBox3

        'Editing options (controls 9-13)
        CheckBox 20,56,65,8,"Overtype mode",.OvertypeMode
        CheckBox 20,68,69,8,"Uppercase only",.UppercaseOnly
        CheckBox 20,80,105,8,"Automatically check syntax",.AutoCheckSyntax
        CheckBox 20,92,73,8,"Full line selection",.FullLineSelection
        CheckBox 20,104,102,8,"Typing replaces selection",.TypingReplacesText
    End Dialog

    Dim OptionsDialog As OptionsTemplate
    Dialog OptionsDialog
End Sub
```

See Also

DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement).

statement

Do...Loop

Syntax 1

Do {While | Until} condition statements Loop

Syntax 2

```
Do
    statements
Loop {While | Until} condition
```

Syntax 3

```
Do
    statements
Loop
```

Description

Repeats a block of Delrina Basic statements while a condition is True or until a condition is True.

Comments

- If the {While | Until} conditional clause is not specified, then the loop repeats the statements forever (or until Delrina Basic encounters an Exit Do statement).
- The condition parameter specifies any Boolean expression.

Examples

```
Sub Main()
    'This first example uses the Do...While statement, which performs
    'the iteration, then checks the condition, and repeats if the
    'condition is True.

    Dim a$(100)
    i% = -1
    Do
        i% = i% + 1
        If i% = 0 Then
            a(i%) = Dir("*")
        Else
            a(i%) = Dir
        End If
    Loop While(a(i%) <> "" And i% <= 99)
    r% = SelectBox(i% & " files found",,a)
End Sub



Sub Main()
    'This second example uses the Do While...Loop, which checks the
    'condition and then repeats if the condition is True.
```

```
    Dim a$(100)
    i% = 0
    a(i%) = Dir("*")
    Do While (a(i%) <> "") And (i% <= 99)
        i% = i% + 1
        a(i%) = Dir
    Loop
    r% = SelectBox(i% & " files found",,a)
End Sub


Sub Main()
    'This third example uses the Do Until...Loop, which does the
    'iteration and then checks the condition and repeats if the
    'condition is True.

    Dim a$(100)
    i% = 0
    a(i%) = Dir("*")
    Do Until (a(i%) = "") Or (i% = 100)
        i% = i% + 1
        a(i%) = Dir
    Loop
    r% = SelectBox(i% & " files found",,a)
End Sub


Sub Main()
    'This last example uses the Do...Until Loop, which performs the
    'iteration first, checks the condition, and repeats if the
    'condition is True.

    Dim a$(100)
    i% = -1
    Do
        i% = i% + 1
        If i% = 0 Then
            a(i%) = Dir("*")
        Else
            a(i%) = Dir
        End If
    Loop Until (a(i%) = "") Or (i% = 100)
    r% = SelectBox(i% & " files found",,a)
End Sub
```

See Also

For...Next (statement); While ...WEnd (statement).

Note

Due to errors in program logic, you can inadvertently create infinite loops in your code. Under Windows you can break out of infinite loops using Ctrl+Break.

function

DoEvents

Syntax

DoEvents[()]

Description

Yields control to other applications, returning an Integer 0.

Comments

- This statement yields control to the operating system, letting other applications to process mouse, keyboard, and other messages.
- If a SendKeys statement is active, this statement waits until all the keys in the queue have been processed.

Example

```
'The following routine explicitly yields to let other applications
'to run and refresh on a regular basis.

Sub Main()
    Open "test.txt" For Output As #1
    For i = 1 To 10000
        Print #1,"This is a test of the system and such."
        r = DoEvents
    Next i
    MsgBox "The DoEvents return value is: " & r
    Close #1
End Sub
```

See Also

DoEvents (statement).

statement

DoEvents

Syntax

DoEvents

Description

Yields control to other applications.

Comments

- This statement yields control to the operating system, letting other applications to process mouse, keyboard, and other messages.
- If a SendKeys statement is active, this statement waits until all the keys in the queue have been processed.

Examples

```
'This first example shows a script that takes a long time and hogs
'the system. The following routine explicitly yields to let other
'applications to run and refresh on a regular basis.

Sub Main()
    Open "test.txt" For Output As #1
    For i = 1 To 10000
        Print #1,"This is a test of the system and stuff."
        DoEvents
    Next i
    Close #1
End Sub



'In this second example, the DoEvents statement is used to wait until
'the queue has been completely flushed.

Sub Main()
    id = Shell("notepad.exe",3)       'Start new instance of Notepad.
    SendKeys "This is a test.",False      'Send some keys.
    DoEvents                                    'Wait for the keys to play back.
End Sub
```

See Also

DoEvents (function).

statement

DoKeys

Syntax

```
DoKeys KeyString$ [,time]
```

Description

Simulates the pressing of the specified keys.

Comments

- The DoKeys statement accepts the following parameters:

Parameter	Description
KeyString$	String containing the keys to be sent. The format for KeyString$ is described under the SendKeys statement.
time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString$ parameter. It must be within the following range: 0 <= time <= 32767 For example, if time is 5000 (5 seconds) and the KeyString$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

Example

```
'This code fragment plays back the time and date into Notepad.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    id = Shell("notepad.exe",3)       'Start new instance of Notepad.
    t$ = time$
    d$ = date$
    DoKeys "The time is: " & t$ & "." & crlf
    DoKeys "The date is: " & d$ & "."
End Sub
```

See Also

SendKeys (statement); QueKeys (statement); QueKeyDn (statement); QueKeyUp (statement).

Note

- This statement uses the Windows journalizing mechanism to play keystrokes into the Windows environment.

data type

Double

Syntax

Double

Description

A data type used to declare variables capable of holding real numbers with 15-16 digits of precision.

Comments

- Double variables are used to hold numbers within the following ranges:

Sign	Range
Negative	-1.797693134862315E308 <= double <= -4.94066E-324
Positive	4.94066E-324 <= double <= 1.797693134862315E308

- The type-declaration character for Double is #.

Storage

Internally, doubles are 8-byte (64-bit) IEEE values. Thus, when appearing within a structure, doubles require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.

Each Double consists of the following

A 1-bit sign

An 11-bit exponent

A 53-bit significand (mantissa)

See Also

Currency (data type); Date (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CDbl (function).

statement

DropListBox

Syntax

DropListBox X, Y, width, height, ArrayVariable, .Identifier

Description

Creates a drop list box within a dialog box template.

Comments

- When the dialog box is invoked, the drop list box will be filled with the elements contained in ArrayVariable. Drop list boxes are similar to combo boxes, with the following exceptions:
- The list box portion of a drop list box is not opened by default. The user must open it by clicking the down arrow.
- The user cannot type into a drop list box. Only items from the list box may be selected. With combo boxes, the user can type the name of an item from the list directly or type the name of an item that is not contained within the combo box.
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The DropListBox statement requires the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
ArrayVariable	Single-dimensioned array used to initialize the elements of the drop list box. If this array has no dimensions, then the drop list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the drop list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax: DialogVariable.Identifier

Example

```
'This example lets the user to choose a field name from a drop
'list box.
```

```
Sub Main()
    Dim FieldNames$(4)
    FieldNames$(0) = "Last Name"
    FieldNames$(1) = "First Name"
    FieldNames$(2) = "Zip Code"
    FieldNames$(3) = "State"
    FieldNames$(4) = "City"
    Begin Dialog FindTemplate 16,32,168,48,"Find"
        Text 8,8,37,8,"&Find what:"
        DropListBox 48,6,64,80,FieldNames,.WhichField
        OKButton 120,7,40,14
        CancelButton 120,27,40,14
    End Dialog
    Dim FindDialog As FindTemplate
    FindDialog.WhichField = 1
    Dialog FindDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

constant

ebAbort

Description

Returned by the MsgBox function when the Abort button is chosen.

Comments

- This constant is equal to 3.

Example

```
'This example displays a dialog box with Abort, Retry, and Ignore
'buttons.

Sub Main()
Again:
    rc% = MsgBox("Do you want to continue?",ebAbortRetryIgnore)
    If rc% = ebAbort or rc% = ebIgnore Then
        End
    ElseIf rc% = ebRetry Then
        Goto Again
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebAbortRetryIgnore

Description

Used by the MsgBox statement and function.

Comments

- This constant is equal to 2.

Example

```
'This example displays a dialog box with Abort, Retry, and Ignore
'buttons.

Sub Main()
Again:
    rc% = MsgBox("Do you want to continue?",ebAbortRetryIgnore)
    If rc% = ebAbort or rc% = ebIgnore Then
        End
    ElseIf rc% = ebRetry Then
        Goto Again
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebApplicationModal

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 0.

Example

```
'This example displays an application-modal dialog box (which is the
'default).

Sub Main()
    MsgBox "This is application-modal.",ebOKOnly Or ebApplicationModal
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebArchive

Description

Bit position of a file attribute indicating that a file hasn't been backed up.

Comments

- This constant is equal to 32.

Example

```
'This example dimensions an array and fills it with filenames with the
'Archive bit set.

Sub Main()
    Dim s$()
    FileList s$,"*",ebArchive
    a% = SelectBox("Archived Files", "Choose one", s$)
    If a% >= 0 Then                 'If a% is -1, then the user pressed Cancel.
        MsgBox "You selected Archive file: " & s$(a)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebBold

Description

Used with the Text and TextBox statement to specify a bold font.

Comments

- This constant is equal to 2.

Example

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Bold Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBold
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebBold
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

See Also

Text (statement), TextBox (statement).

constant

ebBoldItalic

Description

Used with the Text and TextBox statement to specify a bold-italic font.

Comments

- This constant is equal to 6.

Example

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Bold-Italic Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBoldItalic
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebBoldItalic
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

See Also

Text (statement), TextBox (statement).

constant

ebBoolean

Description

Number representing the type of a Boolean variant.

Comments

- This constant is equal to 11.

Example

```
Sub Main()
    Dim MyVariant as variant
    MyVariant = True
    If VarType(MyVariant) = ebBoolean Then
        MyVariant = 5.5
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebCancel

Description

Returned by the MsgBox function when the Cancel button is chosen.

Comments

- This constant is equal to 2.

Example

```
Sub Main()
    'Invoke MsgBox and check whether the Cancel button was pressed.
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel)
    If rc% = ebCancel Then
        MsgBox "The user clicked Cancel."
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebCritical

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 16.

Example

```
Sub Main()
'Invoke MsgBox with Abort, Retry, and Ignore buttons and a Stop icon.
      rc% = MsgBox("Disk drive door is open.",ebAbortRetryIgnore Or      ebCritical)
       If rc% = 3 Then
            'The user selected Abort from the dialog box.
            MsgBox "The user clicked Abort."
      End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebCurrency

Description

Number representing the type of a Currency variant.

Comments

- This constant is equal to 6.

Example

```
'This example checks to see whether a variant is of type Currency.

Sub Main()
    Dim MyVariant
    If VarType(MyVariant) = ebCurrency Then
        MsgBox "Variant is Currency."
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebDataObject

Description

Number representing the type of a data object variant.

Comments

- This constant is equal to 13.

Example

```
'This example checks to see whether a variable is a data object.

Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebDataObject Then
        MsgBox "Variant contains a data object."
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebError

Description

Number representing the type of an error variant.

Comments

- This constant is equal to 10.

Example

```
'This example checks to see whether a variable is an error.

Function Div(ByVal a As Variant,ByVal b As Variant) As Variant
    On Error Resume Next
    Div = a / b
    If Err <> 0 Then Div = CVErr(Err)
End Function

Sub Main()
    a = InputBox("Please enter 1st number","Division Sample")
    b = InputBox("Please enter 2nd number","Division Sample")

    res = Div(a,b)

    If VarType(res) = ebError Then
        res = CStr(res)
        res = Error(Mid(res,7,Len(res)))
        MsgBox "" & res & "" occurred"
    Else
        MsgBox "The result of the division is: " & res
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebDate

Description

Number representing the type of a Date variant.

Comments

- This constant is equal to 7.

Example

```
Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebDate Then
        MsgBox "This variable is a Date type!"
    Else
        MsgBox "This variable is not a Date type!"
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebDefaultButton1

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 0.

Example

```
'This example invokes MsgBox with the focus on the OK button by default.

Sub Main()
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or     ebDefaultButton1)
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebDefaultButton2

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 256.

Example

```
'This example invokes MsgBox with the focus on the Cancel button by
'default.

Sub Main()
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or     ebDefaultButton2)
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebDefaultButton3

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 512.

Example

```
'This example invokes MsgBox with the focus on the Ignore button by
'default.

Sub Main()
    rc% = MsgBox("Disk drive door open.",ebAbortRetryIgnore Or  ebDefaultButton3)
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebDirectory

Description

Bit position of a file attribute indicating that a file is a directory entry.

Comments

- This constant is equal to 16.

Example

```
'This example dimensions an array and fills it with directory names
'using the ebDirectory constant.

Sub Main()
    Dim s$()
    FileList s$,"c:\*",ebDirectory
    a% = SelectBox("Directories", "Choose one:", s$)
    If a% >= 0 Then
        MsgBox "You selected directory: " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebDos

Description

Used with the AppType or FileType functions to indicate a DOS application.

Comments

- This constant is equal to 1.

Example

```
'This example detects whether a DOS program was selected.

Sub Main()
    s$ = OpenFilename$("Run","Programs:*.exe")
    If s$ <> "" Then
        If FileType(s$) = ebDos Then
            MsgBox "You selected a DOS exe file."
        End If
    End If
End Sub
```

See Also

AppType (function); FileType (function).

constant

ebWin16

Description

Used with the Basic.OS property to indicate the 16-bit Windows version of Delrina Basic.

Comments

- This constant is equal to 0.
- The Basic.OS property returns this value when Delrina Basic is running under the Windows 3.1 operating system

Example

```
Sub Main()
    If Basic.OS = ebWin16 Then MsgBox "Running under Windows 3.1."
End Sub
```

constant

ebDouble

Description

Number representing the type of a Double variant.

Comments

- This constant is equal to 5.

Example

See ebSingle (constant).

See Also

MsgBox (function); MsgBox (statement); VarType (function); Variant (data type).

constant

ebEmpty

Description

Number representing the type of an Empty variant.

Comments

- This constant is equal to 0.

Example

```
Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebEmpty Then
        MsgBox "This variant has not been assigned a value yet!"
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebExclamation

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 48.

Example

```
'This example displays a dialog box with an OK button and an
'exclamation icon.

Sub Main()
    MsgBox "Out of memory saving to disk.",ebOKOnly Or ebExclamation
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebHidden

Description

Bit position of a file attribute indicating that a file is hidden.

Comments

- This constant is equal to 2.

Example

```
'This example dimensions an array and fills it with filenames using
'the ebHidden attribute.

Sub Main()
    Dim s$()
    FileList s$,"*",ebHidden
    If ArrayDims(s$) = 0 Then
        MsgBox "No hidden files found!"
        End
    End If
    a% = SelectBox("Hidden Files","Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected hidden file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebIgnore

Description

Returned by the MsgBox function when the Ignore button is chosen.

Comments

- This constant is equal to 5.

Example

```
'This example displays a critical error dialog box and sees what the
'user wants to do.

Sub Main()
    rc% = MsgBox("Printer out of paper.",ebAbortRetryIgnore)
    If rc% = ebIgnore Then
        'Continue printing here.
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebInformation

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 64.

Example

```
'This example displays a dialog box with the Information icon.

Sub Main()
    MsgBox "You just deleted your file!",ebOKOnly Or ebInformation
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebInteger

Description

Number representing the type of an Integer variant.

Comments

- This constant is equal to 2.

Example

```
'This example defines a function that returns True if a variant
'contains an Integer value (either a 16-bit or 32-bit Integer).

Function IsInteger(v As Variant) As Boolean
    If VarType(v) = ebInteger Or VarType(v) = ebLong Then
        IsInteger = True
    Else
        IsInteger = False
    End If
End Function

Sub Main()
    Dim i as Integer
    i = 123
    If IsInteger(i) then
        Msgbox "i is an Integer."
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebItalic

Description

Used with the Text and TextBox statement to specify an italic font.

Comments

- This constant is equal to 4.

Example

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Italic Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebItalic
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebItalic
        OKButton 96,110,40,14
    End Dialog

    Dim a As UserDialog
    Dialog a
End Sub
```

See Also

Text (statement), TextBox (statement).

constant

ebLandscape

Description

Used with the PrinterSetOrientation statement to align the paper horizontally.

Comments

- This constant is equal to 2.

Example

```
'This example sets the printer orientation to landscape.

Sub Main()
    PrinterSetOrientation ebLandscape
    Msgbox "Printer set to landscape."
End Sub
```

See Also

PrinterSetOrientation (statement); PrinterGetOrientation (function).

constant

ebLeftButton

Description

Used with the QueMouseXX commands to represent the left button.

Comments

- This constant is equal to 1.

Example

```
'This example double-clicks the left mouse button.

Sub Main()
    QueMouseClick ebLeftButton,1000,1875
End Sub
```

See Also

QueButtonDn (statement); QueButtonUp (statement); QueMouseClick (statement); QueMouseDblClk (statement); QueMouseDblDn (statement).

constant

ebLong

Description

Number representing the type of a Long variant.

Comments

- This constant is equal to 3.

Example

See ebInteger (constant).

See Also

VarType (function); Variant (data type).

constant

ebMaximized

Description

Used with the AppSetState and AppGetState statements to indicate a maximized window state.

Comments

- This constant is equal to 1.

Example

```
'This example minimizes the current application if it is maximized.

Sub Main()
    If AppGetState = ebMaximized Then AppMinimize
End Sub
```

See Also

AppSetState (statement); AppGetState (function).

constant

ebMinimized

Description

Used with the AppSetState and AppGetState statements to indicate a minimized window state.

Comments

- This constant is equal to 2.

Example

```
'This example restores the current application if it is minimized.

Sub Main()
    If AppGetState = ebMinimized Then
        AppMaximize
    Else
        AppMinimize
    End If
End Sub
```

See Also

AppSetState (statement); AppGetState (function).

constant

ebNo

Description

Returned by the MsgBox function when the No button is chosen.

Comments

- This constant is equal to 7.

Example

```
'This example asks a question and queries the user's response.

Sub Main()
    rc% = MsgBox("Do you want to update the glossary?",ebYesNo)
    If rc% = ebNo Then
        MsgBox "The user clicked 'No'."          'Don't update glossary.
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebNone

Description

Bit value used to select files with no other attributes.

Comments

- This value can be used with the Dir$ and FileList commands. These functions will return only files with no attributes set when used with this constant. This constant is equal to 64.

Example

```
'This example dimensions an array and fills it with filenames with no
'attributes set.

Sub Main()
    Dim s$()
    FileList s$,"*",ebNone
    If ArrayDims(s$) = 0 Then
        MsgBox "No files found without attributes!"
        End
    End If
    a% = SelectBox("No Attributes", "Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebNormal

Description

Used to search for "normal" files.

Comments

- This value can be used with the Dir$ and FileList commands and will return files with the Archive, Volume, ReadOnly, or no attributes set. It will not match files with Hidden, System, or Directory attributes. This constant is equal to 0.

Example

```
'This example dimensions an array and fills it with filenames with
'Normal attributes.

Sub Main()
    Dim s$()
    FileList s$,"*", ebNormal
    If ArrayDims(s$) = 0 Then
        MsgBox "No filesfound!"
        End
    End If
    a% = SelectBox("Normal Files", "Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebNull

Description

Number representing the type of a Null variant.

Comments

- This constant is equal to 1.

Example

```
Sub Main()
    Dim MyVariant
    MyVariant = Null
    If VarType(MyVariant) = ebNull Then
        MsgBox "This variant is Null"
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebObject

Description

Number representing the type of an Object variant (an OLE automation object).

Comments

- This constant is equal to 9.

Example

```
Sub Main()
    Dim MyVariant
    If VarType(MyVariant) = ebObject Then
        MsgBox MyVariant.Value
    Else
        MsgBox "'MyVariant' is not an object."
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebOK

Description

Returned by the MsgBox function when the OK button is chosen.

Comments

- This constant is equal to 1.

Example

```
'This example displays a dialog box that lets the user to cancel.

Sub Main()
    rc% = MsgBox("Are you sure you want to exit Windows?",ebOKCancel)
    If rc% = ebOK Then System.Exit
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebOKCancel

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 1.

Example

```
'This example displays a dialog box that lets the user to cancel.

Sub Main()
    rc% = MsgBox("Are you sure you want to exit Windows?",ebOKCancel)
    If rc% = ebOK Then System.Exit
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebOKOnly

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 0.

Example

'This example informs the user of what is going on (no options).

```
Sub Main()
    MsgBox "The system has been reset.",ebOKOnly
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebPortrait

Description

Used with the PrinterSetOrientation statement to align the paper vertically.

Comments

- This constant is equal to 1.

Example

```
'This example changes the printer's orientation to portrait.

Sub Main()
    PrinterSetOrientation ebPortrait
End Sub
```

See Also

PrinterSetOrientation (statement); PrinterGetOrientation (function).

constant

ebQuestion

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 32.

Example

```
'This example displays a dialog box with OK and Cancel buttons and a
'question icon.

Sub Main()
    rc% = MsgBox("OK to delete file?",ebOKCancel Or ebQuestion)
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebReadOnly

Description

Bit position of a file attribute indicating that a file is read-only.

Comments

- This constant is equal to 1.

Example

```
'This example dimensions an array and fills it with filenames with
'ReadOnly attributes.

Sub Main()
    Dim s$()
    FileList s$, "*", ebReadOnly
    If ArrayDims(s$) = 0 Then
        MsgBox "No read only files found!"
        End
    End If
    a% = SelectBox("ReadOnly", "Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebRegular

Description

Used with the Text and TextBox statement to specify an normal-styled font (i.e., neither bold or italic).

Comments

- This constant is equal to 1.

Example

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Regular Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebRegular
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebRegular
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

See Also

Text (statement), TextBox (statement).

constant

ebRestored

Description

Used with the AppSetState and AppGetState statements to indicate a normal window state.

Comments

- This constant is equal to 3.

Example

```
'This example minimizes the current application only if it is
'restored.

Sub Main()
    state% = AppGetState
    If state% = ebRestored Then
        AppMinimize
    End If
End Sub
```

See Also

AppSetState (statement); AppGetState (function).

constant

ebRetry

Description

Returned by the MsgBox function when the Retry button is chosen.

Comments

- This constant is equal to 4.

Example

```
'This example displays a Retry message box.

Sub Main()
    rc% = MsgBox("Unable to open file.",ebRetryCancel)
    If rc% = ebRetry Then
        MsgBox "User selected Retry."
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebRetryCancel

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 5.

Example

```
'This example invokes a dialog box with Retry and Cancel buttons.

Sub Main()
    rc% = MsgBox("Unable to open file.",ebRetryCancel)
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebRightButton

Description

Used with the QueMouseXX commands to represent the right button.

Comments

- This constant is equal to 2.

Example

```
'This example clicks the right mouse button at 1000,1200.

Sub Main()
    QueMouseClick ebRightButton,1000,1200
End Sub
```

See Also

QueButtonDn (statement); QueButtonUp (statement); QueMouseClick (statement); QueMouseDblClk (statement); QueMouseDblDn (statement).

constant

ebSingle

Description

Number representing the type of a Single variant.

Comments

- This constant is equal to 4.

Example

```
'This example defines a function that returns True if the passed
'variant is a Real number.

Function IsReal(v As Variant) As Boolean
    If VarType(v) = ebSingle Or VarType(v) = ebDouble Then
        IsReal = True
    Else
        IsReal = False
    End If
End Function

Sub Main()
    Dim i as Integer
    i = 123
    If IsReal(i) then
        Msgbox "i is Real."
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebString

Description

Number representing the type of a String variant.

Comments

- This constant is equal to 8.

Example

```
Sub Main()
    Dim MyVariant as variant
    MyVariant = "This is a test."
    If VarType(MyVariant) = ebString Then
        MsgBox "Variant is a string."
    End If
End Sub
```

See Also

VarType (function); Variant (data type).

constant

ebSystem

Description

Bit position of a file attribute indicating that a file is a system file.

Comments

- This constant is equal to 4.

Example

```
'This example dimensions an array and fills it with filenames with
'System attributes.

Sub Main()
    Dim s$()
    FileList s$,"*",ebSystem
    a% = SelectBox("System Files", "Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebSystemModal

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 4096.

Example

```
Sub Main()
    MsgBox "All applications are halted!",ebSystemModal
End Sub
```

See Also

ebApplicationModal (constant); Constants (topic); MsgBox (function); MsgBox (statement).

constant

ebVariant

Description

Number representing the type of a Variant.

Comments

- Currently, it is not possible for variants to use this subtype. This constant is equal to 12.

See Also

VarType (function); Variant (data type).

constant

ebVolume

Description

Bit position of a file attribute indicating that a file is the volume label.

Comments

- This constant is equal to 8.

Example

```
'This example dimensions an array and fills it with filenames with
'Volume attributes.

Sub Main()
    Dim s$()
    FileList s$, "*", ebVolume
    If ArrayDims(s$) > 0 Then
        MsgBox "The volume name is: " & s(1)
    Else
        MsgBox "No volumes found."
    End If
End Sub
```

See Also

Dir, Dir$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

constant

ebWindows

Description

Used with the AppType function to indicate a Windows application.

Comments

- This constant is equal to 2.

Example

```
'This example determines whether a Windows application was selected.

Sub Main()
    s$ = OpenFilename$("Run","Programs:*.exe")
    If s$ <> "" Then
        If FileType(s$) = ebWindows Then
            MsgBox "You selected a Windows .exe file."
        End If
    End If
End Sub
```

See Also

AppGetType (function); AppFileType (function).

constant

ebYes

Description

Returned by the MsgBox function when the Yes button is chosen.

Comments

- This constant is equal to 6.

Example

```
'This example queries the user for a response.

Sub Main()
    rc% = MsgBox("Overwrite file?",ebYesNoCancel)
    If rc% = ebYes Then
        MsgBox "You elected to overwrite the file."
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebYesNo

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 4.

Example

```
'This example displays a dialog box with Yes and No buttons.

Sub Main()
    rc% = MsgBox("Are you sure you want to remove all formatting?",ebYesNo)
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

constant

ebYesNoCancel

Description

Used with the MsgBox statement and function.

Comments

- This constant is equal to 3.

Example

```
'This example displays a dialog box with Yes, No, and Cancel buttons.

Sub Main()
    rc% = MsgBox("Format drive C:?",ebYesNoCancel)
    If rc% = ebYes Then
        MsgBox "The user chose Yes."
    End If
End Sub
```

See Also

MsgBox (function); MsgBox (statement).

function

EditEnabled

Syntax

```
EditEnabled(name$ | id)
```

Description

Returns True if the given text box is enabled within the active window or dialog box; returns False otherwise.

Comments

- The EditEnabled function takes the following parameters:

Parameter	Description
name$	String containing the name of the text box. The name of a text box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a text box.
id	Integer specifying the ID of the text box.

- A runtime error is generated if a text box control with the given name or ID cannot be found within the active window.
- If enabled, the text box can be given the focus using the ActivateControl statement.

Note: The EditEnabled function is used to determine whether a text box is enabled in another application's dialog box. Use the DlgEnable function in dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

EditExists (function); GetEditText$ (function); SetEditText (statement).

function

EditExists

Syntax

```
EditExists(name$ | id)
```

Description

Returns True if the given text box exists within the active window or dialog box; returns False otherwise.

Comments

- The EditExists function takes the following parameters:

Parameter	Description
name$	String containing the name of the text box. The name of a text box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a text box.
id	Integer specifying the ID of the text box.

- A runtime error is generated if a text box control with the given name or ID cannot be found within the active window.
- If there is no active window, False will be returned.

Note: The EditExists function is used to determine whether a text box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                    'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

EditEnabled (function); GetEditText$ (function); SetEditText (statement).

constant

Empty

Description

Constant representing a variant of type 0.

Comments

- The Empty value has special meaning indicating that a Variant is uninitialized.
- When Empty is assigned to numbers, the value 0 is assigned. When Empty is assigned to a String, the string is assigned a zero-length string.

Example

```
Sub Main()
    Dim a As Variant
    a = Empty
    MsgBox "This string is" & a & "concatenated with Empty"
    MsgBox "5 + Empty = " & (5 + a)
End Sub
```

See Also

Null (constant); Variant (data type); VarType (function).

statement

End

Syntax

End

Description

Terminates execution of the current script, closing all open files.

Example

```
'This example uses the End statement to stop execution.

Sub Main()
    MsgBox "The next line will terminate the script."
    End
End Sub
```

See Also

Close (statement); Stop (statement); Exit For (statement); Exit Do (statement); Exit Function (statement); Exit Sub (function).

function

Environ, Environ$

Syntax

```
Environ[$](variable$ | VariableNumber)
```

Description

Returns the value of the specified environment variable.

Comments

- Environ$ returns a String, whereas Environ returns a String variant.
- If variable$ is specified, then this function looks for that variable$ in the environment. If the variable$ name cannot be found, then a zero-length string is returned.
- If VariableNumber is specified, then this function looks for the Nth variable within the environment (the first variable being number 1). If there is no such environment variable, then a zero-length string is returned. Otherwise, the entire entry from the environment is returned in the following format:
- variable = value

Example

```
'This example looks for the DOS Comspec variable and displays the
'value in a dialog box.

Sub Main()
    Dim a$(1)
    a$(1) = Environ("COMSPEC")
    MsgBox "The DOS Comspec variable is set to: " & a$(1)
End Sub
```

See Also

Command, Command$ (functions).

function

EOF

Syntax

```
EOF(filenumber)
```

Description

Returns True if the end-of-file has been reached for the given file; returns False otherwise.

Comments

- The filenumber parameter is an Integer used by Delrina Basic to refer to the open file-the number passed to the Open statement.
- With sequential files, EOF returns True when the end of the file has been reached (i.e., the next file read command will result in a runtime error).
- With Random or Binary files, EOF returns True after an attempt has been made to read beyond the end of the file. Thus, EOF will only return True when Get was unable to read the entire record.

Example

```
'This example opens the autoexec.bat file and reads lines from the
'file until the end-of-file is reached.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    file$ = "c:\autoexec.bat"
    Open file$ For Input As #1
    Do While Not EOF(1)
        Line Input #1,newline
    Loop
    Close
    MsgBox "The last line of "" & file$ "" is:" & crlf & crlf & newline
End Sub
```

See Also

Open (statement); LOF (function).

operator

Eqv

Syntax

expression1 Eqv expression2

Description

Performs a logical or binary equivalence on two expressions.

Comments

- If both expressions are either Boolean, Boolean variants, or Null variants, then a logical equivalence is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
False	True	False
False	False	True

If either expression is Null, then Null is returned.

Binary Equivalence

If the two expressions are Integer, then a binary equivalence is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary equivalence is then performed, returning a Long result.

Binary equivalence forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:

```
1     Eqv 1   =   1       Example:
0     Eqv 1   =   0       5    01101001
1     Eqv 0   =   0       6    10101010
0     Eqv 0   =   1       Eqv 00101000
```

Example

```
'This example assigns False to A, performs some equivalent operations,
'and displays a dialog box with the result. Since A is equivalent to
'False, and False is equivalent to 0, and by definition, A = 0, then
'the dialog box will display "A is False."

Sub Main()
    a = False
    If ((a Eqv False) And (False Eqv 0) And (a = 0)) Then
        MsgBox "a is False."
    Else
        MsgBox "a is True."
    End If
End Sub
```

See Also

Operator Precedence (topic); Or (operator); Xor (operator); Imp (operator); And (operator).

statement

Erase

Syntax

Erase array1 [,array2]...

Description

Erases the elements of the specified arrays.

Comments

- For dynamic arrays, the elements are erased, and the array is redimensioned to have no dimensions (and therefore no elements). For fixed arrays, only the elements are erased; the array dimensions are not changed.
- After a dynamic array is erased, the array will contain no elements and no dimensions. Thus, before the array can be used by your program, the dimensions must be reestablished using the Redim statement.
- Up to 32 parameters can be specified with the Erase statement.
- The meaning of erasing an array element depends on the type of the element being erased:

Element Type	What Erase Does
Integer	Sets the element to 0.
Boolean	Sets the element to False.
Long	Sets the element to 0.
Double	Sets the element to 0.0.
Date	Sets the element to December 30, 1899.
Single	Sets the element to 0.0.
String (variable-length)	Frees the string, then sets the element to a zero-length string.
String (fixed-length)	Sets every character of each element to zero (Chr$(0)).
Object	Decrements the reference count and sets the element to Nothing.
Variant	Sets the element to Empty.
User-defined type	Sets each structure element as a separate variable.

Example

'This example fills an array with a list of available disk drives, displays the list, erases the array and then redisplays the list.

```
Sub Main()
    Dim a$(10)          'Declare an array.
    DiskDrives a        'Fill element 1 with a list of available disk drives.
    r = SelectBox("Array Before Erase",,a)
    Erase a$                    'Erase all elements in the array.
    r = SelectBox("Array After Erase",,a)
End Sub
```

See Also

Redim (statement); Arrays (topic).

function

Erl

Syntax

Erl[()]

Description

Returns the line number of the most recent error.

Comments

- The first line of the script is 1, the second line is 2, and so on.
- The internal value of Erl is reset to 0 with any of the following statements: Resume, Exit Sub, Exit Function. Thus, if you want to use this value outside an error handler, you must assign it to a variable.

Example

```
'This example generates an error and then determines the line
'on which the error occurred.

Sub Main()
    Dim i As Integer
    On Error Goto Trap1
    i = 32767                   'Generate an error--overflow.
    i = i + 1
    Exit Sub
Trap1:
    MsgBox "Error on line: " & Erl
    Exit Sub                    'Reset the error handler.
End Sub
```

See Also

Err (function); Error, Error$ (functions); Error Handling (topic).

function

Err

Syntax

Err[()]

Description

Returns an Integer representing the error that caused the current error trap.

Comments

- The Err function can only be used while within an error trap.
- The internal value of Err is reset to 0 with any of the following statements: Resume, Exit Sub, Exit Function. Thus, if you want to use this value outside an error handler, you must assign it to a variable.

Example

```
'This example forces error 10, with a subsequent transfer to
'the TestError label. TestError tests the error and, if not
'error 55, resets Err to 999 (user-defined error) and returns to
'the Main subroutine.

Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: '" & Err & " - " & Error$ & "'"
    Exit Sub

TestError:
    If Err = 55 Then                'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error '" & Err & "' has occurred!"
        Err = 999
    End If
    Resume Next
End Sub
```

See Also

Erl (function); Error, Error$ (functions); Error Handling (topic).

statement

Err

Syntax

```
Err = value
```

Description

Sets the value returned by the Err function to a specific Integer value.

Comments

- Only positive values less than or equal to 32767 can be used.
- Setting value to -1 has the side effect of resetting the error state. This lets you to perform error trapping within an error handler. The ability to reset the error handler while within an error trap is not standard Basic. Normally, the error handler is reset only with the Resume, Exit Sub, or Exit Function statement.

Example

```
'This example forces error 10, with a subsequent transfer to
'the TestError label. TestError tests the error and, if not
'error 55, resets Err to 999 (user-defined error) and returns to
'the Main subroutine.

Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: "' & Err() & " - " & Error$ & """
    Exit Sub

TestError:
    If Err = 55 Then                'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error "' & Err & "' has occurred."
        Err = 999
    End If
    Resume Next
End Sub
```

See Also

Error (statement); Error Handling (topic).

statement

Error

Syntax

```
Error errornumber
```

Description

Simulates the occurrence of the given runtime error.

Comments

- The errornumber parameter is any Integer containing either a built-in error number or a user-defined error number. The Err function can be used within the error trap handler to determine the value of the error.

Example

```
'This example forces error 10, with a subsequent transfer to
'the TestError label. TestError tests the error and, if not
'error 55, resets Err to 999 (user-defined error) and returns to
'the Main subroutine.

Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: '" & Err() & " - " & Error$ & "'"
    Exit Sub

TestError:
    If Err = 55 Then                'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error '" & Err & "' has occurred."
        Err = 999
    End If
    Resume Next
End Sub
```

See Also

Err (statement); Error Handling (topic).

topic

Error Handling

Error Handlers

Delrina Basic supports nested error handlers. When an error occurs within a subroutine, Delrina Basic checks for an On Error handler within the currently executing subroutine or function. An error handler is defined as follows:

```
Sub foo()
    On Error Goto catch
    'Do something here.
    Exit Sub

catch:
    'Handle error here.
End Sub
```

Error handlers have a life local to the procedure in which they are defined. The error is reset when (1) another On Error statement is encountered, (2) an error occurs, or (3) the procedure returns.

Cascading Errors

If a runtime error occurs and no On Error handler is defined within the currently executing procedure, then Delrina Basic returns to the calling procedure and runs the error handler there. This process repeats until a procedure is found that contains an error handler or until there are no more procedures. If an error is not trapped or if an error occurs within the error handler, then Delrina Basic displays an error message, halting execution of the script.

Once an error handler has control, it must address the condition that caused the error and resume execution with the Resume statement. This statement resets the error handler, transferring execution to an appropriate place within the current procedure. An error is displayed if a procedure exits without first executing Resume or Exit.

Visual Basic Compatibility

Where possible, Delrina Basic has the same error numbers and error messages as Visual Basic. This is useful for porting scripts between environments.

Handling errors in Delrina Basic involves querying the error number or error text using the Error$ or Err function. Since this is the only way to handle errors in Delrina Basic, compatibility with Visual Basic's error numbers and messages is essential.

Delrina Basic errors fall into three categories:

1. **Visual Basic-compatible errors:** These errors, numbered between 0 and 799, are numbered and named according to the errors supported by Visual Basic.
2. **Delrina Basic errors:** These errors, numbered from 800 to 999, are unique to Delrina Basic.
3. **User-defined errors:** These errors, equal to or greater than 1,000, are available for use by extensions or by the script itself.

You can intercept trappable errors using Delrina Basic's On Error construct. Almost all errors in Delrina Basic are trappable except for various system errors.

function

Error, Error$

Syntax

```
Error[$][(errornumber)]
```

Description

Returns a String containing the text corresponding to the given error number or the most recent error.

Comments

- Error$ returns a String, whereas Error returns a String variant.
- The errornumber parameter is an Integer containing the number of the error message to retrieve. If this parameter is omitted, then the function returns the text corresponding to the most recent runtime error. If no runtime error has occurred, then a zero-length string is returned.
- If the Error statement was used to generate a user-defined runtime error, then this function will return a zero-length string ("").

Example

```
'This example forces error 10, with a subsequent transfer to
'the TestError label. TestError tests the error and, if not
'error 55, resets Err to 999 (user-defined error) and returns to
'the Main subroutine.

Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: "" & Err & " - " & Error & """
    Exit Sub

TestError:
    If Err = 55 Then                'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error "" & Err & "" has occurred."
        Err = 999
    End If
    Resume Next
End Sub
```

See Also

Erl (function); Err (function); Error Handling (topic).

statement

Exit Do

Syntax

Exit Do

Description

Causes execution to continue on the statement following the Loop clause.

Comments

- This statement can only appear within a Do...Loop statement.

Example

```
'This example will load an array with directory entries unless there
'are more than ten entries--in which case, the Exit Do terminates
'the loop.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$(5)
    Do
        i% = i% + 1
        If i% = 1 Then
            a(i%) = Dir("*")
        Else
            a(i%) = Dir
        End If
        If i% >= 5 Then Exit Do
    Loop While (a(i%) <> "")

    If i% = 5 Then
        MsgBox i% & " directory entries processed!"
    Else
        MsgBox "Less than " & i% & " entries processed!"
    End If
End Sub
```

See Also

Stop (statement); Exit For (statement); Exit Function (statement); Exit Sub (statement); End (function); Do...Loop (statement).

statement

Exit For

Syntax

Exit For

Description

Causes execution to exit the innermost For loop, continuing execution on the line following the Next statement.

Comments

- This statement can only appear within a For...Next block.

Example

```
'This example enters a large user-defined cycle, performs a
'calulation and exits the For...Next loop when the result exceeds a
'certain value.

Const critical_level = 500

Sub Main()
    num = InputBox("Please enter the number of cycles","Cycles")
    For i = 1 To Val(num)
        newpressure = i * 2
        If newpressure >= critical_level Then Exit For
    Next i

    MsgBox "The valve pressure is: " & newpressure
End Sub
```

See Also

Stop (statement); Exit Do (statement); Exit Function (statement); Exit Sub (statement); End (statement); For...Next (statement).

statement

Exit Function

Syntax

Exit Function

Description

Causes execution to exit the current function, continuing execution on the statement following the call to this function.

Comments

- This statement can only appear within a function.

Example

```
'This function displays a message and then terminates with Exit
'Function.

Function Test_Exit() As Integer
    MsgBox "Testing function exit, returning to Main()."
    Test_Exit = 0
    Exit Function
    MsgBox "This line should never run."
End Function

Sub Main()
    a% = Test_Exit()
    MsgBox "This is the last line of Main()."
End Sub
```

See Also

Stop (statement); Exit For (statement); Exit Do (statement); Exit Sub (statement); End (statement); Function...End Function (statement).

statement

Exit Sub

Syntax

Exit Sub

Description

Causes execution to exit the current subroutine, continuing execution on the statement following the call to this subroutine.

Comments

- This statement can appear anywhere within a subroutine. It cannot appear within a function.

Example

```
'This example displays a dialog box and then exits. The last line
'should never run because of the Exit Sub statement.

Sub Main()
    MsgBox "Terminating Main()."
    Exit Sub
    MsgBox "Still here in Main()."
End Sub
```

See Also

Stop (statement); Exit For (statement); Exit Do (statement); Exit Function (statement); End (function); Sub...End Sub (statement).

function

Exp

Syntax

```
Exp(value)
```

Description

Returns the value of e raised to the power of value.

Comments

- The value parameter is a Double within the following range:
  `0 <= value <= 709.782712893.`
- A runtime error is generated if value is out of the range specified above.
- The value of e is 2.71828.

Example

```
'This example assigns a to e raised to the 12.4 power and displays it
'in a dialog box.

Sub Main()
    a# = Exp(12.4)
    MsgBox "e to the 12.4 power is: " & a#
End Sub
```

See Also

Log (function).

topic

Expression Evaluation

Delrina Basic lets expressions to involve data of different types. When this occurs, the two arguments are converted to be of the same type by promoting the less precise operand to the same type as the more precise operand. For example, Delrina Basic will promote the value of i% to a Double in the following expression:

```
result# = i% * d#
```

In some cases, the data type to which each operand is promoted is different than that of the most precise operand. This is dependent on the operator and the data types of the two operands and is noted in the description of each operator.

If an operation is performed between a numeric expression and a String expression, then the String expression is usually converted to be of the same type as the numeric expression. For example, the following expression converts the String expression to an Integer before performing the multiplication:

```
result = 10 * "2"          'Result is equal to 20.
```

There are exceptions to this rule as noted in the description of the indicidual operators.

Type Coercion

Delrina Basic performs numeric type conversion automatically. Automatic conversions sometimes result in overflow errors, as shown in the following example:

```
d# = 45354
i% = d#
```

In this example, an overflow error is generated because the value contained in d# is larger than the maximum size of an Integer.

Rounding

When floating-point values (Single or Double) are converted to integer values (Integer or Long), the fractional part of the floating-point number is lost, rounding to the nearest integer value. Delrina Basic uses Baker's rounding:

- If the fractional part is larger than .5, the number is rounded up.
- If the fractional part is smaller than .5, the number is rounded down.
- If the fractional part is equal to .5, then the number is rounded up if it is odd and down if it is even.

The following table shows sample values before and after rounding:

Before Rounding	After Rounding to Whole Number
2.1	2
4.6	5
2.5	2
3.5	4

Default Properties

When an OLE object variable or an Object variant is used with numerical operators such as addition or subtraction, then the default property of that obect is automatically retrieved. For example, consider the following:

```
Dim Excel As Object
Set Excel = GetObject(,"Excel.Application")
MsgBox "This application is " & Excel
```

The above example displays This application is Microsoft Excel in a dialog box. When the variable

Excel is used within the expression, the default property is automatically retrieved, which, in this case, is the string Microsoft Excel. Considering that the default property of the Excel object is .Value, then the following two statements are equivalent:

```
MsgBox "This application is " & Excel
MsgBox "This application is " & Excel.Value
```

constant

False

Description

Boolean constant whose value is False.

Comments

- Used in conditionals and Boolean expressions.

Example

```
'This example assigns False to a, performs some equivalent operations,
'and displays a dialog box with the result. Since a is equivalent to
'False, and False is equivalent to 0, and by definition, a = 0, then
'the dialog box will display "a is False."

Sub Main()
    a = False
    If ((a = False) And (False Eqv 0) And (a = 0)) Then
        MsgBox "a is False."
    Else
        MsgBox "a is True."
    End If
End Sub
```

See Also

True (constant); Constants (topic); Boolean (data type).

function

FileAttr

Syntax

```
FileAttr(filenumber, attribute)
```

Description

Returns an Integer specifying the file mode (if attribute is 1) or the operating system file handle (if attribute is 2).

Comments

- The FileAttr function takes the following parameters:

Parameter	Description
filenumber	Integer value used by Delrina Basic to refer to the open file-the number passed to the Open statement.
attribute	Integer specifying the type of value to be returned. If attribute is 1, then one of the following values is returned: 1 Input 2 Output 4 Random 8 Append 32 Binary If attribute is 2, then the operating system file handle is returned. On most systems, this is a special Integer value identifying the file.

Example

```
'This example opens a file for input, reads the file attributes, and
'determines the file mode for which it was opened. The result is
'displayed in a dialog box.

Sub Main()
    Open "c:\autoexec.bat" For Input As #1
    a% = FileAttr(1,1)
    Select Case a%
        Case 1
            MsgBox "Opened for input."
        Case 2
            MsgBox "Opened for output."
        Case 4
            MsgBox "Opened for random."
        Case 8
            MsgBox "Opened for append."
        Case 32
            MsgBox "Opened for binary."
        Case Else
            MsgBox "Unknown file mode."
        End Select
    a% = FileAttr(1,2)
    MsgBox "File handle is: " & a%
    Close
End Sub
```

See Also

FileLen (function); GetAttr (function); FileType (function); FileExists (function); Open (statement); SetAttr (statement).

statement

FileCopy

Syntax

```
FileCopy source$, destination$
```

Description

Copies a source$ file to a destination$ file.

Comments

- The FileCopy function takes the following parameters:

Parameter	Description
source$	String containing the name of a single file to copy. The source$ parameter cannot contain wildcards (? or *) but may contain path information.
destination$	String containing a single, unique destination file, which may contain a drive and path specification.

- The file will be copied and renamed if the source$ and destination$ filenames are not the same.
- Some platforms do not support drive letters and may not support dots to indicate current and parent directories.

Example

```
'This example copies the autoexec.bat file to "autoexec.sav", then
'opens the copied file and tries to copy it again--which generates an
'error.

Sub Main()
    On Error Goto ErrHandler
    FileCopy "c:\autoexec.bat","c:\autoexec.sav"
    Open "c:\autoexec.sav" For Input As # 1
    FileCopy "c:\autoexec.sav","c:\autoexec.sv2"
    Close
    Exit Sub

ErrHandler:
    If Err = 55 Then                'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "An unspecified file copy error has occurred."
    End If
    Resume Next
End Sub
```

See Also

Kill (statement); Name (statement).

function

FileDateTime

Syntax

```
FileDateTime(filename$)
```

Description

Returns a Date variant representing the date and time of the last modification of a file.

Comments

- This function retrieves the date and time of the last modification of the file specified by filename$ (wildcards are not leted). A runtime error results if the file does not exist. The value returned can be used with the date/time functions (i.e., Year, Month, Day, Weekday, Minute, Second, Hour) to extract the individual elements.

Example

```
'This example gets the file date/time of the autoexec.bat file and
'displays it in a dialog box.

Sub Main()
    If FileExists("c:\autoexec.bat") Then
        a# = FileDateTime("c:\autoexec.bat")
        MsgBox "The date/time information for the file is: " & Year(a#) & "-" & Month(a#) & "-" & Day(a#)
    Else
        MsgBox "The file does not exist."
    End If
End Sub
```

See Also

FileLen (function); GetAttr (function); FileType (function); FileAttr (function); FileExists (function).

statement

FileDirs

Syntax

```
FileDirs array() [,dirspec$]
```

Description

Fills a String or Variant array with directory names from disk.

Comments

- The FileDirs statement takes the following parameters:

Parameter	Description
array()	Either a zero- or a one-dimensioned array of strings or variants. The array can be either dynamic or fixed. If array() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound, UBound, and ArrayDims functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.
dirspec$	String containing the file search mask, such as: t*. c:*.* If this parameter is omitted, then * is used, which fills the array with all the subdirectory names within the current directory.

Example

```
'This example fills an array with directory entries and displays the
'first one.

Sub Main()
    Dim a$()
    FileDirs a$,"c:\*.*"
    MsgBox "The first directory is: " & a$(0)
End Sub
```

See Also

FileList (statement); Dir, Dir$ (functions); CurDir, CurDir$ (functions); ChDir (statement).

function

FileExists

Syntax

```
FileExists(filename$)
```

Description

Returns True if filename$ exists; returns False otherwise.

Comments

- This function determines whether a given filename$ is valid.
- This function will return False if filename$ specifies a subdirectory.

Example

```
'This example checks to see whether there is an autoexec.bat file in
'the root directory of the C drive, then displays either its date and
'time of creation or the fact that it does not exist.

Sub Main()
    If FileExists("c:\autoexec.bat") Then
        Msgbox "This file exists!"
    Else
        MsgBox "File does not exist."
    End If
End Sub
```

See Also

FileLen (function); GetAttr (function); FileType (function); FileAttr (function); FileParse$ (function).

function

FileLen

Syntax

```
FileLen(filename$)
```

Description

Returns a Long representing the length of filename$ in bytes.

Comments

- This function is used in place of the LOF function to retrieve the length of a file without first opening the file. A runtime error results if the file does not exist.

Example

```
'This example checks to see whether there is a c:\autoexec.bat file
'and, if there is, displays the length of the file.

Sub Main()
    file$ = "c:\autoexec.bat"
    If FileExists(file$) And FileLen(file$) <> 0) Then
        b% = FileLen(file$)
        MsgBox """ & file$ & "' is " & b% & " bytes."
    Else
        MsgBox """ & file$ & "' does not exist."
    End If
End Sub
```

See Also

GetAttr (function); FileType (function); FileAttr (function); FileParse$ (function); FileExists (function); Loc (function).

statement

FileList

Syntax

```
FileList array() [,[filespec$] [,[include_attr] [,exclude_attr]]]
```

Description

Fills a String or Variant array with filenames from disk.

Comments

- The FileList function takes the following parameters:

Parameter	Description
array()	Either a zero- or a one-dimensioned array of strings or variants. The array can be either dynamic or fixed. If array() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound, UBound, and ArrayDims functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.
filespec$	String specifying which filenames are to be included in the list. The filespec$ parameter can include wildcards, such as * and ?. If this parameter is omitted, then * is used.
include_attr	Integer specifying attributes of files you want included in the list. It can be any combination of the attributes listed below. If this parameter is omitted, then the value 97 is used (ebReadOnly Or ebArchive Or ebNone).
exclude_attr	Integer specifying attributes of files you want excluded from the list. It can be any combination of the attributes listed below. If this parameter is omitted, then the value 18 is used (ebHidden Or ebDirectory). In other words, hidden files and subdirectories are excluded from the list.
Wildcards	The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complete searching patterns. The following table shows some examples:

This Pattern	Matches These Files	Doesn't Match These Files
S.TXT	SAMPLE.TXT GOOSE.TXT	SAMPLE SAMPLE.DAT SAMS.TXT
C*T.TXT	CAT.TXT ACATS.TXT	CAP.TXT
C*T	CAT CAP.TXT	CAT.DOC
C?T CUT	CAT CAPIT CT	CAT.TXT
*	(All files)	

File Attributes

These numbers can be any combination of the following:

Constant	Value	Includes
ebNormal	0	Read-only, archive, subdir, none
ebReadOnly	1	Read-only files
ebHidden	2	Hidden files
ebSystem	4	System files
ebVolume	8	Volume label
ebDirectory	16	DOS subdirectories
ebArchive	32	Files that have changed since the last backup
ebNone	64	Files with no attributes

Example

```
'This example fills an array a with the directory of the current drive
'for all files that have normal or no attributes and excludes those
'with system attributes. The dialog box displays four filenames from
'the array.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$()
    FileList a$,"*.*",(ebNormal + ebNone),ebSystem
    If ArrayDims(a$) > 0 Then
        r = SelectBox("FileList","The files you filtered are:",a$)
    Else
        MsgBox "No files found."
    End If
End Sub
```

See Also

FileDirs (statement); Dir, Dir$ (functions).

function

FileParse$

Syntax

```
FileParse$(filename$[, operation])
```

Description

Returns a String containing a portion of filename$ such as the path, drive, or file extension.

Comments

- The filename$ parameter can specify any valid filename (it does not have to exist). For example:

  ..\test.dat
  c:\sheets\test.dat
  test.dat

- A runtime error is generated if filename$ is a zero-length string.
- The optional operation parameter is an Integer specifying which portion of the filename$ to extract. It can be any of the following values.

Value	Meaning	Example
0	Full name	c:\sheets\test.dat
1	Drive	c
2	Path	c:\sheets
3	Name	test.dat
4	Root	test
5	Extension	dat

- If operation is not specified, then the full name is returned. A runtime error will result if operation is not one of the above values.
- A runtime error results if filename$ is empty.

Example

```
'This example parses the file string "c:\temp\autoexec.bat" into its
'component parts and displays them in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$(5)
    file$ = "c:\temp\autoexec.bat"
    For i = 1 To 5
        a$(i) = FileParse$(file$,i)
    Next i

    msg = "The breakdown of '" & file$ & "' is:" & crlf & crlf
    msg = msg & a$(1) & crlf & a$(2) & crlf & a$(3) & crlf & a$(4) & crlf & a$(5)
    MsgBox msg
End Sub
```

See Also

FileLen (function); GetAttr (function); FileType (function); FileAttr (function); FileExists (function).

function

FileType

Syntax

```
FileType(filename$)
```

Description

Returns the type of the specified file.

Example

```
'This example looks at c:\windows\winfile.exe and determines whether
'it is a DOS or a Windows file. The result is displayed in a dialog
'box.

Sub Main()
    file$ = "notepad.exe"
    a = FileType(file$)
    If a = ebDos Then
        MsgBox "" & file$ & "' is a DOS file."
    Else
        MsgBox "" & file$ & "' is a Windows file of type '" & a & ""
    End If
End Sub
```

See Also

FileLen (function); GetAttr (function); FileAttr (function); FileExists (function).

Note

- Currently, only files with a ".exe" extension can be used with this function. Files with a ".com" or ".bat" extension will return 3 (unknown).

function

Fix

Syntax

```
Fix(number)
```

Description

Returns the integer part of number.

Comments

- This function returns the integer part of the given value by removing the fractional part. The sign is preserved.
- The Fix function returns the same type as number, with the following exceptions:
- If number is Empty, then an Integer variant of value 0 is returned.
- If number is a String, then a Double variant is returned.
- If number contains no valid data, then a Null variant is returned.

Example

```
'This example returns the fixed part of a number and assigns it to b,
'then displays the result in a dialog box.

Sub Main()
    a# = -19923.45
    b% = Fix(a#)
    MsgBox "The fixed portion of -19923.45 is: " & b%
End Sub
```

See Also

Int (function); CInt (function).

statement

For...Next

Syntax

```
For counter = start To end [Step increment]
     [statements]
     [Exit For]
     [statements]
Next [counter [,nextcounter]... ]
```

Description

Repeats a block of statements a specified number of times, incrementing a loop counter by a given increment each time through the loop.

Comments

- The For statement takes the following parameters:

Parameter	Description
counter	Name of a numeric variable. Variables of the following types can be used: Integer, Long, Single, Double, Variant.
start	Initial value for counter. The first time through the loop, counter is assigned this value.
end	Final value for counter. The statements will continue executing until counter is equal to end.
increment	Amount added to counter each time through the loop. If end is greater than start, then increment must be positive. If end is less than start, then increment must be negative. If increment is not specified, then 1 is assumed. The expression given as increment is evaluated only once. Changing the step during execution of the loop will have no effect.
statements	Any number of Delrina Basic statements.

- The For...Next statement continues executing until an Exit For statement is encountered when counter is greater than end.
- For...Next statements can be nested. In such a case, the Next [counter] statement applies to the innermost For...Next**.**
- The Next clause can be optimized for nested next loops by separating each counter with a comma. The ordering of the counters must be consistent with the nesting order (innermost counter appearing before outermost counter). The following example shows two equivalent For statements:

```
For i = 1 To 10                 For i = 1 To 10
For j = 1 To 10                 For j = 1 To 10
Next j                          Next j,i
Next i
```

- A Next clause appearing by itself (with no counter variable) matches the innermost For loop.
- The counter variable can be changed within the loop but will have no effect on the number of times the loop will run.

Example

```
Sub Main()
    'This example constructs a truth table for the OR statement      'using nested For...Next loops.
    msg = "Logic table for Or:" & crlf & crlf
    For x = -1 To 0
        For y = -1 To 0
            z = x Or y
            msg = msg & CBool(x) & " Or "
            msg = msg & CBool(y) & " = "
            msg = msg & CBool(z) & Basic.Eoln$
        Next y
    Next x
    MsgBox msg
End Sub
```

See Also

Do...Loop (statement); While...WEnd (statement).

Note

- Due to errors in program logic, you can inadvertently create infinite loops in your code. Under Windows you can break out of infinite loops using Ctrl+Break.

function

Format, Format$

Syntax

```
Format[$](expression [,Userformat$])
```

Description

Returns a String formatted to user specification.

Comments

- Format$ returns a String, whereas Format returns a String variant.
- The Format$/Format functions take the following parameters:

Parameter	Description
expression	String or numeric expression to be formatted.
Userformat$	Format expression that can be either one of the built-in Delrina Basic formats or a user-defined format consisting of characters that specify how the expression should be displayed. String, numeric, and date/time formats cannot be mixed in a single Userformat$ expression. If Userformat$ is omitted and the expression is numeric, then these functions perform the same function as the Str$ or Str statements, except that they do not preserve a leading space for positive values.

- If expression is Null, then a zero-length string is returned.

Built-In Formats

To format numeric expressions, you can specify one of the built-in formats. There are two categories of built-in formats: one deals with numeric expressions and the other with date/time values.The following tables list the built-in numeric and date/time format strings, followed by an explanation of what each does.

Format	Description
General number	Display the numeric expression as is, with no additional formatting.
Currency	Displays the numeric expression as currency, with thousands separator if necessary.
Fixed	Displays at least one digit to the left of the decimal separator and two digits to the right.
Standard	Displays the numeric expression with thousands separator if necessary. Displays at least one digit to the left of the decimal separator and two digits to the right.
Percent	Displays the numeric expression multiplied by 100. A percent sign (%) will appear at the right of the formatted output. Two digits are displayed to the right of the decimal separator.
Scientific	Displays the number using scientific notation. One digit appears before the decimal separator and two after.
Yes/No	Displays No if the numeric expression is 0. Displays Yes for all other values.
True/False	Displays False if the numeric expression is 0. Displays True for all other values.
On/Off	Displays Off if the numeric expression is 0. Displays On for all other values.

Date/Time Formats

Format	Description
General date	Displays the date and time. If there is no fractional part in the numeric expression, then only the date is displayed. If there is no integral part in the numeric expression, then only the time is displayed. Output is in the following form: 1/1/95 01:00:00 AM.

Long date	Displays a long date.
Medium date	Displays a medium date-prints out only the abbreviated name of the month.
Short date	Displays a short date.
Long time	Displays the long time. The default is: h:mm:ss.
Medium time	Displays the time using a 12-hour clock. Hours and minutes are displayed, and the AM/PM designator is at the end.
Short time	Displays the time using a 24-hour clock. Hours and minutes are displayed.

User-Defined Formats

In addition to the built-in formats, you can specify a user-defined format by using characters that have special meaning when used in a format expression. The following tables list the characters you can use for numeric, string, and date/time formats and explain their functions.

Numeric Formats

Character	Meaning
Empty string	Displays the numeric expression as is, with no additional formatting.
0	This is a digit placeholder. Displays a number or a 0. If a number exists in the numeric expression in the position where the 0 appears, the number will be displayed. Otherwise, a 0 will be displayed. If there are more 0s in the format string than there are digits, the leading and trailing 0s are displayed without modification.
#	This is a digit placeholder. Displays a number or nothing. If a number exists in the numeric expression in the position where the number sign appears, the number will be displayed. Otherwise, nothing will be displayed. Leading and trailing 0s are not displayed.
.	This is the decimal placeholder. Designates the number of digits to the left of the decimal and the number of digits to the right. The character used in the formatted string depends on the decimal placeholder, as specified by your locale.
%	This is the percentage operator. The numeric expression is multiplied by 100, and the percent character is inserted in the same position as it appears in the user-defined format string.
,	This is the thousand separator. The common use for the thousands separator is to separate thousands from hundreds. To specify this use, the thousands separator must be surrounded by digit placeholders. Commas appearing before any digit placeholders are specified are just displayed. Adjacent commas with no digit placeholders specified between them and the decimal mean that the number should be divided by 1,000 for each adjacent comma in the format string. A comma immediately to the left of the decimal has the same function. The actual thousands separator character used depends on the character specified by your locale.
E-E+e-e+	These are the scientific notation operators, which display the number in scientific notation. At least one digit placeholder must exist to the left of E-, E+, e-, or e+. Any digit placeholders displayed to the left of E-, E+, e-, or e+ determine the number of digits displayed in the exponent. Using E+ or e+ places a + in front of positive exponents and a - in front of negative exponents. Using E- or e- places a - in front of negative exponents and nothing in front of positive exponents.
:	This is the time separator. Separates hours, minutes, and seconds when time values are being formatted. The actual character used depends on the character specified by your locale.
/	This is the date separator. Separates months, days, and years when date values are being formatted. The actual

	character used depends on the character specified by your locale.
-+$()space	These are the literal characters you can display. To display any other character, you should precede it with a backslash or enclose it in quotes.
\	This designates the next character as a displayed character. To display characters, precede them with a backslash. To display a backslash, use two backslashes. Double quotation marks can also be used to display characters. Numeric formatting characters, date/time formatting characters, and string formatting characters cannot be displayed without a preceding backslash.
"ABC"	Displays the text between the quotation marks, but not the quotation marks. To designate a double quotation mark within a format string, use two adjacent double quotation marks.
*	This will display the next character as the fill character. Any empty space in a field will be filled with the specified fill character.

- Numeric formats can contain one to three parts. Each part is separated by a semicolon. If you specify one format, it applies to all values. If you specify two formats, the first applies to positive values and the second to negative values. If you specify three formats, the first applies to positive values, the second to negative values, and the third to 0s. If you include semicolons with no format between them, the format for positive values is used.

String Formats

Character	Meaning
@	This is a character placeholder. Displays a character if one exists in the expression in the same position; otherwise, displays a space. Placeholders are filled from right to left unless the format string specifies left to right.
&	This is a character placeholder. Displays a character if one exists in the expression in the same position; otherwise, displays nothing. Placeholders are filled from right to left unless the format string specifies left to right.
<	This character forces lowercase. Displays all characters in the expression in lowercase.
>	This character forces uppercase. Displays all characters in the expression in uppercase.
!	This character forces placeholders to be filled from left to right. The default is right to left.

Date/Time Formats

Character	Meaning
c	Displays the date as ddddd and the time as ttttt. Only the date is displayed if no fractional part exists in the numeric expression. Only the time is displayed if no integral portion exists in the numeric expression.
d	Displays the day without a leading 0 (1-31).
dd	Displays the day with a leading 0 (01-31).
ddd	Displays the day of the week abbreviated (Sun-Sat).
dddd	Displays the day of the week (Sunday-Saturday).
ddddd	Displays the date as a short date.
dddddd	Displays the date as a long date.
w	Displays the number of the day of the week (1-7). Sunday is 1; Saturday is 7.

ww	Displays the week of the year (1-53).
m	Displays the month without a leading 0 (1-12). If m immediately follows h or hh, m is treated as minutes (0-59).
mm	Displays the month with a leading 0 (01-12). If mm immediately follows h or hh, mm is treated as minutes with a leading 0 (00-59).
mmm	Displays the month abbreviated (Jan-Dec).
mmmm	Displays the month (January-December).
q	Displays the quarter of the year (1-4).
y	Displays the day of the year (1-366).
yy	Displays the year, not the century (00-99).
yyyy	Displays the year (1000-9999).
h	Displays the hour without a leading 0 (0-24).
hh	Displays the hour with a leading 0 (00-24).
n	Displays the minute without a leading 0 (0-59).
nn	Displays the minute with a leading 0 (00-59).
s	Displays the second without a leading 0 (0-59).
ss	Displays the second with a leading 0 (00-59).
ttttt	Displays the time. A leading 0 is displayed if specified by your locale.
AM/PM	Displays the time using a 12-hour clock. Displays an uppercase AM for time values before 12 noon. Displays an uppercase PM for time values after 12 noon and before 12 midnight.
am/pm	Displays the time using a 12-hour clock. Displays a lowercase am or pm at the end.
A/P	Displays the time using a 12-hour clock. Displays an uppercase A or P at the end.
a/p	Displays the time using a 12-hour clock. Displays a lowercase a or p at the end.
AMPM	Displays the time using a 12-hour clock. Displays the string s1159 for values before 12 noon and s2359 for values after 12 noon and before 12 midnight.

Example

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a# = 1199.234
    msg = "Some general formats for '" & a# & "' are:" & crlf & crlf
    msg = msg & Format(a#,"General Number") & crlf
    msg = msg & Format(a#,"Currency") & crlf
    msg = msg & Format(a#,"Standard") & crlf
    msg = msg & Format(a#,"Fixed") & crlf
    msg = msg & Format(a#,"Percent") & crlf
    msg = msg & Format(a#,"Scientific") & crlf
    msg = msg & Format(True,"Yes/No") & crlf
    msg = msg & Format(True,"True/False") & crlf
    msg = msg & Format(True,"On/Off") & crlf
    msg = msg & Format(a#,"0,0.00") & crlf
    msg = msg & Format(a#,"##,###,###.###") & crlf
    MsgBox msg
```

```
    da$ = Date$
    msg = "Some date formats for '" & da$ & "' are:" & crlf & crlf
    msg = msg & Format(da$,"General Date") & crlf
    msg = msg & Format(da$,"Long Date") & crlf
    msg = msg & Format(da$,"Medium Date") & crlf
    msg = msg & Format(da$,"Short Date") & crlf
    MsgBox msg

    ti$ = Time$
    msg = "Some time formats for '" & ti$ & "' are:" & crlf & crlf
    msg = msg & Format(ti$,"Long Time") & crlf
    msg = msg & Format(ti$,"Medium Time") & crlf
    msg = msg & Format(ti$,"Short Time") & crlf
    MsgBox msg
End Sub
```

See Also

Str, Str$ (functions); CStr (function).

Note

- Under Windows default date/time formats are read from the [Intl] section of the win.ini file.

function

FreeFile

Syntax

FreeFile[()]

Description

Returns an Integer containing the next available file number.

Comments

- The number returned is suitable for use in the Open statement and will always be between 1 and 255 inclusive.

Example

```
'This example assigns A to the next free file number and displays it
'in a dialog box.

Sub Main()
    a = FreeFile
    MsgBox "The next free file number is: " & a
End Sub
```

See Also

FileAttr (function); Open (statement).

statement

Function...End Function

Syntax

```
[Private | Public] [Static] Function name[(arglist)] [As ReturnType]
     [statements]
End Sub
```

where arglist is a comma-separated list of the following (up to 30 arguments are allowed):

```
[Optional] [ByVal | ByRef] parameter [()] [As type]
```

Description

Creates a user-defined function.

Comments

- The Function statement has the following parts:

Part	Description
Private	Indicates that the function being defined cannot be called from other scripts.
Public	Indicates that the function being defined can be called from other scripts. If both the Private and Public keywords are missing, then Public is assumed.
Static	Recognized by the compiler but currently has no effect.
name	Name of the function, which must follow Delrina Basic naming conventions: 1. Must start with a letter. 2. May contain letters, digits, and the underscore character (_). Punctuation and type-declaration characters are not allowed. The exclamation point (!) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character. 3. Must not exceed 80 characters in length. Additionally, the name parameter can end with an optional type-declaration character specifying the type of data returned by the function (i.e., any of the following characters: %, &, !, #, @).
Optional	Keyword indicating that the parameter is optional. All optional parameters must be of type Variant. Furthermore, all parameters that follow the first optional parameter must also be optional. If this keyword is omitted, then the parameter is required.

Note: You can use the IsMissing function to determine if an optional parameter was actually passed by the caller.

ByVal	Keyword indicating that parameter is passed by value.
ByRef	Keyword indicating that parameter is passed by reference. If neither the ByVal nor the ByRef keyword is given, then ByRef is assumed.
parameter	Name of the parameter, which must follow the same naming conventions as those used by variables. This name can include a type-declaration character, appearing in place of As type.
type	Type of the parameter (i.e., Integer, String, and so on). Arrays are indicated with parentheses. For example, an array of integers would be declared as follows: Function Test(a() As Integer) End Function
ReturnType	Type of data returned by the function. If the return type is not given, then Variant is

assumed. The ReturnType can only be specified if the function name (i.e., the name parameter) does not contain an explicit type-declaration character.

- A function returns to the caller when either of the following statements is encountered:

  ```
  End Function
  Exit Function
  ```

- Functions can be recursive.

Returning Values from Functions

To assign a return value, an expression must be assigned to the name of the function, as shown below:

```
Function TimesTwo(a As Integer) As Integer
    TimesTwo = a * 2
End Function
```

If no assignment is encountered before the function exits, then one of the following values is returned:

Value	Data Type Returned by the Function
0	Integer, Long, Single, Double, Currency
Zero-length string	String
Nothing	Object (or any data object)
Empty	Variant
December 30, 1899	Date
False	Boolean

The type of the return value is determined by the As ReturnType clause on the Function statement itself. As an alternative, a type-declaration character can be added to the Function name. For example, the following two definitions of Test both return String values:

```
Function Test() As String
    Test = "Hello, world"
End Function

Function Test$()
    Test = "Hello, world"
End Function
```

Passing Parameters to Functions

Parameters are passed to a function either by value or by reference, depending on the declaration of that parameter in arglist. If the parameter is declared using the ByRef keyword, then any modifications to that passed parameter within the function change the value of that variable in the caller. If the parameter is declared using the ByVal keyword, then the value of that variable cannot be changed in the called function. If neither the ByRef or ByVal keywords are specified, then the parameter is passed by reference.

You can override passing a parameter by reference by enclosing that parameter within parentheses. For instance, the following example passes the variable j by reference, regardless of how the third parameter is declared in the arglist of UserFunction:

```
i = UserFunction(10,12,(j))
```

Optional Parameters

Delrina Basic lets you to skip parameters when calling functions, as shown in the following example:

```
Function Test(a%,b%,c%) As Variant
End Function

Sub Main
a = Test(1,,4)       'Parameter 2 was skipped.
End Sub
```

You can skip any parameter with the following restrictions:

1. The call cannot end with a comma. For instance, using the above example, the following is not valid:

   ```
   a = Test(1,,)
   ```

2. The call must contain the minimum number of parameters as requred by the called function. For instance, using the above example, the following are invalid:

   ```
   a = Test(,1)   'Only passes two out of three required parameters.
   a = Test(1,2)  'Only passes two out of three required parameters.
   ```

When you skip a parameter in this manner, Delrina Basic creates a temporary variable and passes this variable instead. The value of this temporary variable depends on the data type of the corresponding parameter in the argument list of the called function, as described in the following table:

Value	Data Type
0	Integer, Long, Single, Double, Currency
Zero-length string	String
Nothing	Object (or any data object)
Error	Variant
December 30, 1899	Date
False	Boolean

Within the called function, you will be unable to determine if a parameter was skipped unless the parameter was declared as a variant in the argument list of the function. In this case, you can use the IsMissing function to determine if the parameter was skipped:

```
Function Test(a,b,c)
    If IsMissing(a) Or IsMissing(b) Then Exit Sub
End Function
```

Example

```
Function Factorial(n%) As Integer
    'This function calculates N! (N-factoral).
    f% = 1
    For i = n To 2 Step -1
        f = f * i
    Next i
    Factorial = f
End Function
```

```
Sub Main()
    'This example calls user-defined function Factoral and displays the
    'result in a dialog box.
    a% = 0
    Do While a% < 2
        a% = Val(InputBox("Enter an integer number greater than 2.","Compute Factorial"))
    Loop
    b# = Factorial(a%)
    MsgBox "The factoral of " & a% & " is: " & b#
End Sub
```

See Also

Sub...End Sub (statement)

function

Fv

Syntax

```
Fv(Rate, Nper, Pmt,Pv,Due)
```

Description

Calculates the future value of an annuity based on periodic fixed payments and a constant rate of interest.

Comments

- An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.
- The Fv function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. Make sure that annual rates are normalized for monthly periods (divided by 12).
NPer	Double representing the total number of payments (periods) in the annuity.
Pmt	Double representing the amount of each payment per period. Payments are entered as negative values, whereas receipts are entered as positive values.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, whereas in the case of a retirement annuity, the present value would be the amount of the fund.
Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.

- Rate and NPer values must be expressed in the same units. If Rate is expressed as a percentage per month, then NPer must also be expressed in months. If Rate is an annual rate, then the NPer must also be given in years.
- Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example

```
'This example calculates the future value of 100 dollars paid
'periodically for a period of 10 years (120 months) at a rate of 10%
'per year (or .10/12 per month) with payments made on the first of the
'month. The value is displayed in a dialog box. Note that payments are
'negative values.

Sub Main()
    a# = Fv((.10/12),120,-100.00,0,1)
    MsgBox "Future value is: " & Format(a#,"Currency")
End Sub
```

See Also

IRR (function); MIRR (function); Npv (function); Pv (function).

statement

Get

Syntax

```
Get [#] filenumber, [recordnumber], variable
```

Description

Retrieves data from a random or binary file and stores that data into the specified variable.

Comments

- The Get statement accepts the following parameters:

Parameter	Description
filenumber	Integer used by Delrina Basic to identify the file. This is the same number passed to the Open statement.
recordnumber	Long specifying which record is to be read from the file. For binary files, this number represents the first byte to be read starting with the beginning of the file (the first byte is 1). For random files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647. If the recordnumber parameter is omitted, the next record is read from the file (if no records have been read yet, then the first record in the file is read). When this parameter is omitted, the commas must still appear, as in the following example: Get #1,,recvar If recordnumber is specified, it overrides any previous change in file position specified with the Seek statement.
variable	Variable into which data will be read. The type of the variable determines how the data is read from the file, as described below.

- With random files, a runtime error will occur if the length of the data being read exceeds the reclen parameter specified with the Open statement. If the length of the data being read is less than the record length, the file pointer is advanced to the start of the next record. With binary files, the data elements being read are contiguousthe file pointer is never advanced.

Variable Types

- The type of the variable parameter determines how data will be read from the file. It can be any of the following types:

Variable Type	File Storage Description
Integer	2 bytes are read from the file.
Long	4 bytes are read from the file.
String (variable-length)	In binary files, variable-length strings are read by first determining the specified string variable's length and then reading that many bytes from the file. For example, to read a string of eight characters: s$ = String(8," ") **Get #**1,,s$ In random files, variable-length strings are read by first reading a 2-byte length and then reading that many characters from the file.
String (fixed-length)	Fixed-length strings are read by reading a fixed number of characters from the file equal to the string's declared length.
Double	8 bytes are read from the file (IEEE format).

Single	4 bytes are read from the file (IEEE format).
Date	8 bytes are read from the file (IEEE double format).
Boolean	2 bytes are read from the file. Nonzero values are True, and zero values are False.
Variant	A 2-byte VarType is read from the file, which determines the format of the data that follows. Once the VarType is known, the data is read individually, as described above. With user-defined errors, after the 2-byte VarType, a 2-byte unsigned integer is read and assigned as the value of the user-defined error, followed by 2 additional bytes of information about the error. The exception is with strings, which are always preceded by a 2-byte string length.
User-defined types	Each member of a user-defined data type is read individually. In binary files, variable-length strings within user-defined types are read by first reading a 2-byte length followed by the string's content. This storage is different from variable-length strings outside of user-defined types. When reading user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.
Arrays	Arrays cannot be read from a file using the Get statement.
Objects	Object variables cannot be read from a file using the Get statement.

Example

```
'This example opens a file for random write, then writes ten
'records into the file with the values 10...50. Then the file
'is closed and reopened in random mode for read, and the
'records are read with the Get statement. The result is displayed
'in a message box.

Sub Main()
    Open "test.dat" For Random Access Write As #1
    For x = 1 to 10
        y = x * 10
        Put #1,x,y
    Next x
    Close

    Open "test.dat" For Random Access Read As #1
    For y = 1 to 5
        Get #1,y,x
        msg = msg & "Record " & y & ": " & x & Basic.Eoln$
    Next y
    Close

    MsgBox msg
End Sub
```

See Also

Open (statement); Put (statement); Input# (statement); Line Input# (statement); Input, Input$ (functions).

function

GetAttr

Syntax

```
GetAttr(filename$)
```

Description

Returns an Integer containing the attributes of the specified file.

Comments

- The attribute value returned is the sum of the attributes set for the file. The value of each attribute is as follows:

Constant	Value	Includes
ebNormal	0	Read-only files, archive files, subdirectories, and files with no attributes.
ebReadOnly	1	Read-only files
ebHidden	2	Hidden files
ebSystem	4	System files
ebVolume	8	Volume label
ebDirectory	16	DOS subdirectories
ebArchive	32	Files that have changed since the last backup
ebNone	64	Files with no attributes

- To deterimine whether a particular attribute is set, you can And the values shown above with the value returned by GetAttr. If the result is True, the attribute is set, as shown below:

```
Sub Main()
    Dim w As Integer
    w = GetAttr("sample.txt")
    If w And ebReadOnly Then MsgBox "This file is read-only."
End Sub
```

Example

```
'This example tests to see whether the file test.dat exists.
'If it does not, then it creates the file. The file attributes are 'then retrieved with the GetAttr function, and the result is
'displayed.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a()
    FileList a,"*.*"

Again:
    msg = ""
    r = SelectBox("Attribute Checker","Select File:",a)
    If r = -1 Then
        End
    Else
        y% = GetAttr(a(r))
    End If
```

```
    If y% = 0 Then msg = msg & "This file has no special attributes." & crlf
    If y% And ebReadOnly Then msg = msg & "The read-only bit is set." & crlf
    If y% And ebHidden Then msg = msg & "The hidden bit is set." & crlf
    If y% And ebSystem Then msg = msg & "The system bit is set." & crlf
    If y% And ebVolume Then msg = msg & "The volume bit is set." & crlf
    If y% And ebDirectory Then msg = msg & "The directory bit is set." & crlf
    If y% And ebArchive Then msg = msg & "The archive bit is set."

    MsgBox msg
    Goto Again
End Sub
```

See Also

SetAttr (statement); FileAttr (function).

Notes

These attributes are the same as those used by DOS.

function

GetCheckBox

Syntax

```
GetCheckBox(name$ | id)
```

Description

Returns an Integer representing the state of the specified check box.

Comments

- This function is used to determine the state of a check box, given its name or ID. The returned value will be one of the following:

Returned Value	Description
0	Check box contains no check.
1	Check box contains a check.
2	Check box is grayed.

- The GetCheckBox function takes the following parameters:

Parameter	Description
name$	String containing the name of the check box.
id	Integer specifying the ID of the check box.

Note: The GetCheckBox function is used to retrieve the state of a check box in another application's dialog box. Use the DlgValue function to retrieve the state of a check box in a dynamic dialog box.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each
Job",False
    End If
End Sub
```

See Also

CheckBoxExists (function); CheckBoxEnabled (function); SetCheckBox (statement); DlgValue (function).

function

GetComboBoxItem$

Syntax

```
GetComboBoxItem$(name$ | id [,ItemNumber])
```

Description

Returns a String containing the text of an item within a combo box.

Comments

- The GetComboBoxItem$ function takes the following parameters:

Parameter	Description
name$	String specifying the name of the combo box containing the item to be returned. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
id	Integer specifying the ID of the combo box containing the item to be returned.
ItemNumber	Integer containing the line number of the desired combo box item to be returned. If omitted, then the currently selected item in the combo box is returned.

- The combo box must exist within the current window or dialog box; otherwise, a runtime error is generated.
- A zero-length string will be returned if the combo box does not contain textual items.

Note: The GetComboBoxItem$ function is used to retrieve the current item of a combo box in another application's dialog box. Use the DlgText function to retrieve the current item of a combo box in a dynamic dialog box.

Example

```
'This code fragment retrieves the last item from a combo box.

    last% = GetComboBoxItemCount("Directories:")
    s$ = GetComboBoxItem$("Directories:",last% - 1)      'Number is 0-based
    MsgBox "The last item in the combo box is " & s$
```

See Also

ComboBoxEnabled (function); ComboBoxExists (function); GetComboBoxItemCount (function); SelectComboBoxItem (statement).

function

GetComboBoxItemCount

Syntax

```
GetComboBoxItemCount(name$ | id)
```

Description

Returns an Integer containing the number of items in the specified combo box.

Comments

- The GetComboBoxItemCount function takes the following parameters:

Parameter	Description
name$	String containing the name of the combo box. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
id	Integer specifying the ID of the combo box.

- A runtime error is generated if the specified combo box does not exist within the current window or dialog box.

Note: The GetComboBoxItemCount function is used to determine the number of items in a combo box in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This code fragment copies all the items out of a combo box and into an
'array.

    Dim MyList$()
    last% = GetComboBoxItemCount("Directories:")
    ReDim MyList$(0 To last - 1)
    For i = 0 To last - 1
        MyList$(i) = GetComboBoxItem$("Directories:",i)
    Next i
```

See Also

ComboBoxEnabled (function); ComboBoxExists (function); GetComboBoxItem$ (function); SelectComboBoxItem (statement).

function

GetEditText$

Syntax

```
GetEditText$(name$ | id)
```

Description

Returns a String containing the content of the specified text box control.

Comments

- The GetEditText$ function takes the following parameters:

Parameter	Description
name$	String containing the name of the text box whose content will be returned. The name of a text box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a text box. A runtime error is generated if a text box with that name cannot be found within the active window.
id	Integer specifying the ID of the text box whose content will be returned.

- A runtime error is generated if a text box control with the given name or ID cannot be found within the active window.

Note: The GetEditText$ function is used to retrieve the content of a text box in another application's dialog box. Use the DlgText$ function to retrieve the content of a text box in a dynamic dialog box.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                    'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

EditEnabled (function); EditExists (function); SetEditText (statement).

function

GetListBoxItem$

Syntax

```
GetListBoxItem$(name$ | id,[item])
```

Description

Returns a String containing the specified item in a list box.

Comments

- The GetListBoxItem$ function takes the following parameters:

Parameter	Description
name$	String specifying the name of the list box containing the item to be returned. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
id	Integer specifying the ID of the list box containing the item to be returned.
item	Integer containing the line number of the desired list box item to be returned. This number must be between 1 and the number of items in the list box. If omitted, then the currently selected item in the list box is returned.

- A runtime error is generated if the specified list box cannot be found within the active window.

Note: The GetListBoxItem$ function is used to retreive an item from a list box in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This code fragment sees whether my name appears as an item in the "Users"
'list box.

    last% = GetListBoxItemCount("Users")
    IsThere = False
    For i = 0 To last% - 1          'Number is zero-based.
        If GetListBoxItem$("Users",i) = Net.User$ Then isThere = True
    Next i
    If IsThere Then MsgBox "I am a member!",ebOKOnly
```

See Also

GetListBoxItemCount (function); ListBoxEnabled (function); ListBoxExists (function); SelectListBoxItem (statement).

function

GetListBoxItemCount

Syntax

```
GetListBoxItemCount(name$ | id)
```

Description

Returns an Integer containing the number of items in a specified list box.

Comments

- The GetListBoxItemCount function takes the following parameters:

Parameter	Description
name$	String containing the name of the list box. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
id	Integer specifying the ID of the list box.

- A runtime error is generated if the specified list box cannot be found within the active window.

Note: The GetListBoxItemCount function is used to retrieve the number of items in a list box in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This code fragment sees whether my name appears as an item in the "Users"
'list box.

    last% = GetListBoxItemCount("Users")
    IsThere = False
    For i = 0 To last% - 1          'Number is zero-based.
        If GetListBoxItem$("Users",i) = Net.User$ Then isThere = True
    Next i
    If IsThere Then MsgBox "I am a member!",ebOKOnly
```

See Also

GetListBoxItem$ (function); ListBoxEnabled (function); ListBoxExists (function); SelectListBoxItem (statement).

function

GetObject

Syntax

```
GetObject(filename$ [,class$])
```

Description

Returns the object specified by filename$ or returns a previously instantiated object of the given class$.

Comments

- This function is used to retrieve an existing OLE automation object, either one that comes from a file or one that has previously been instantiated.
- The filename$ argument specifies the full pathname of the file containing the object to be activated. The application associated with the file is determined by OLE at runtime. For example, suppose that a file called c:\docs\resume.doc was created by a word processor called wordproc.exe. The following statement would invoke wordproc.exe, load the file called c:\docs\resume.doc, and assign that object to a variable:

```
Dim doc As Object
Set doc = GetObject("c:\docs\resume.doc")
```

- To activate a part of an object, add an exclamation point to the filename followed by a string representing the part of the object that you want to activate. For example, to activate the first three pages of the document in the previous example:

```
Dim doc As Object
Set doc = GetObject("c:\docs\resume.doc!P1-P3")
```

- The GetObject function behaves differently depending on whether the first parameter is omitted. The following table summarizes the different beheviors of GetObject:

Filename$	Class$	GetObject Returns
Omitted	Specified	Reference to an existing instance of the specified object. A runtime error results if the object is not already loaded.
""	Specified	Reference to a new object (as specified by class$). A runtime error occurs if an object of the specified class cannot be found. This is the same as CreateObject.
Specified	Omitted	Default object from filename$. The application to activate is determined by OLE based on the given filename.
Specified	Specified	Object given by class$ from the file given by filename$. A runtime error occurs if an object of the given class cannot be found in the given file.

Examples

```
'This first example instantiates the existing copy of Excel.

Sub Main()
    Dim Excel As Object
    Set Excel = GetObject(,"Excel.Application")

    'This second example loads the OLE server associated with a document.

    Dim MyObject As Object
    Set MyObject = GetObject("c:\documents\resume.doc")
End Sub
```

See Also

CreateObject (function); Object (data type).

function

GetOption

Syntax

```
GetOption(name$ | id)
```

Description

Returns True if the option is set; returns False otherwise.

Comments

- The GetOption function takes the following parameters:

Parameter	Description
name$	String containing the name of the option button.
id	Integer containing the ID of the option button. The id must be used when the name of the option button is not known in advance.

- The option button must exist within the current window or dialog box.
- A runtime error will be generated if the specified option button does not exist.

Note: The GetOption function is used to retrieve the state of an option button in another application's dialog box. Use the DlgValue function to retrieve the state of an option button in a dynamic dialog box.

Example

```
'This example figures out which option is set in the Desktop dialog
'box of the Control Panel.

Sub Main()
    id = Shell("control.exe",1)                 'Run the Control Panel.
    Menu "Settings.Desktop"                         'Select Desktop dialog box.
    WinActivate "Control Panel|Desktop"         'Activate it.
    If GetOption("Tile") Then                   'Retrieve which option is set.
        MsgBox "Your wallpaper is tiled."       'The Tile option is currently set.
    ElseIf GetOption("Center")
        MsgBox "Your wallpaper is centered."    'The Centered option is currently set.
    End If
End Sub
```

See Also

OptionEnabled (function); OptionExists (function); SetOption (statement).

statement

Global

Description

See Public (statement).

statement

GoSub

Syntax

```
GoSub label
```

Description

- Causes execution to continue at the specified label.

Comments

- Execution can later be returned to the statement following the GoSub by using the Return statement.
- The label parameter must be a label within the current function or subroutine. GoSub outside the context of the current function or subroutine is not allowed.

Example

```
'This example gets a name from the user and then branches to a
'subroutine to check the input. If the user clicks Cancel or enters a
'blank name, the program terminates; otherwise, the name is set to
'MICHAEL, and a message is displayed.

Sub Main()
    uname$ = Ucase$(InputBox$("Enter your name:","Enter Name"))
    GoSub CheckName
    MsgBox "I'm looking for MICHAEL, not " & uname$
    Exit Sub

CheckName:
    If (uname$ = "") Then
        GoSub BlankName
    ElseIf uname$ = "MICHAEL" Then
        GoSub RightName
    Else
        GoSub OtherName
    End If
    Return

BlankName:
    MsgBox "No name? Clicked Cancel? I'm shutting down."
    Exit Sub
RightName:
    Msgbox "Hey, MIKE where have you been?"
    End
OtherName:
    Return
End Sub
```

See Also

Goto (statement); Return (statement).

statement

Goto

Syntax

```
Goto label
```

Description

Transfers execution to the line containing the specified label.

Comments

- The compiler will produce an error if label does not exist.
- The label must appear within the same subroutine or function as the Goto.
- Labels are identifiers that follow these rules:
  1. Must begin with a letter.
  2. May contain letters, digits, and the underscore character.
  3. Must not exceed 80 characters in length.
  4. Must be followed by a colon (:).
- Labels are not case-sensitive.

Example

```
'This example gets a name from the user and then branches to a
'statement, depending on the input name. If the name is not MICHAEL,
'it is reset to MICHAEL unless it is null or the user clicks Cancel--
'in which case, the program displays a message and terminates.

Sub Main()
    uname$ = UCase(InputBox("Enter your name:","Enter Name"))
    If uname$ = "MICHAEL" Then
        Goto RightName
    Else
        Goto WrongName
    End If

WrongName:
    If (uname$ = "") Then
        MsgBox "No name? Clicked Cancel? I'm shutting down."
    Else
        MsgBox "I am renaming you MICHAEL!"
        uname$ = "MICHAEL"
        Goto RightName
    End If
    Exit Sub

RightName:
    MsgBox "Hello, " & uname$
End Sub
```

See Also

GoSub (statement); Call (statement).

Notes

To break out of an infinite loop, press Ctrl+Break.

statement

GroupBox

Syntax

```
GroupBox X,Y,width,height,title$ [,.Identifier]
```

Description

Defines a group box within a dialog box template.

Comments

- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The group box control is used for static display onlythe user cannot interact with a group box control.
- Separator lines can be created using group box controls. This is accomplished by creating a group box that is wider than the width of the dialog box and extends below the bottom of the dialog boxi.e., three sides of the group box are not visible.
- If title$ is a zero-length string, then the group box is drawn as a solid rectangle with no title.
- The GroupBox statement requires the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
title$	String containing the label of the group box. If title$ is a zero-length string, then no title will appear.
.Identifier	Optional parameter that specifies the name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words of title$ are used.

Example

```
'This example shows the GroupBox statement being used both for grouping
'and as a separator line.

Sub Main()
    Begin Dialog OptionsTemplate 16,32,128,84,"Options"
        GroupBox 4,4,116,40,"Window Options"
        CheckBox 12,16,60,8,"Show &Toolbar",.ShowToolbar
        CheckBox 12,28,68,8,"Show &Status Bar",.ShowStatusBar
        GroupBox -12,52,152,48," ",.SeparatorLine
        OKButton 16,64,40,14,.OK
        CancelButton 68,64,40,14,.Cancel
    End Dialog
    Dim OptionsDialog As OptionsTemplate
    Dialog OptionsDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

function

Hex, Hex$

Syntax

```
Hex[$](number)
```

Description

Returns a String containing the hexadecimal equivalent of number.

Comments

- Hex$ returns a String, whereas Hex returns a String variant.
- The returned string contains only the number of hexadecimal digits necessary to represent the number, up to a maximum of eight.
- The number parameter can be any type but is rounded to the nearest whole number before converting to hex. If the passed number is an integer, then a maximum of four digits are returned; otherwise, up to eight digits can be returned.
- The number parameter can be any expression convertible to a number. If number is Null, then Null is returned. Empty is treated as 0.

Example

'This example accepts a number and displays the decimal and 'hexadecimal equivalent until the input number is 0 or invalid.

```
Sub Main()
    Do
        xs$ = InputBox("Enter a number to convert:","Hex Convert")
        x = Val(xs$)
        If x <> 0 Then
            MsgBox "Decimal: " & x & "     Hex: " & Hex(x)
        Else
            MsgBox "Goodbye."
        End If
    Loop While x <> 0
End Sub
```

See Also

Oct, Oct$ (functions).

statement

HLine

Syntax

```
HLine [lines]
```

Description

Scrolls the window with the focus left or right by the specified number of lines.

Comments

- The lines parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled right by one line.

Example

```
'This example scrolls the Notepad window to the left by three
'"amounts." Each "amount" is equivalent to clicking the right arrow
'of the horizontal scroll bar once.

Sub Main()
    If AppFind$("Notepad") = "" Then
        id = Shell("notepad.exe",3)
    Else
        AppActivate "Notepad"
        AppMaximize
    End If
    HLine 3        'Move 3 lines into Notepad.
End Sub
```

See Also

HPage (statement); HScroll (statement).

function

Hour

Syntax

```
Hour(time)
```

Description

Returns the hour of the day encoded in the specified time parameter.

Comments

- The value returned is as an Integer between 0 and 23 inclusive.
- The time parameter is any expression that converts to a Date.

Example

```
'This example takes the current time; extracts the hour,
'minute, and second; and displays them as the current time.

Sub Main()
    Msgbox "It is now hour " & Hour(Time) & " of today."
End Sub
```

See Also

Day (function); Minute (function); Second (function); Month (function); Year (function); Weekday (function); DatePart (function).

statement

HPage

Syntax

```
HPage [pages]
```

Description

Scrolls the window with the focus left or right by the specified number of pages.

Comments

- The pages parameter is an Integer specifying the number of pages to scroll. If this parameter is omitted, then the window is scrolled right by one page.

Example

```
'This example scrolls the Notepad window to the left by three
'"amounts." Each "amount" is equivalent to clicking within the
'horizontal scroll bar on the right side of the thumb mark.

Sub Main()
    If AppFind$("Notepad") = "" Then
        id = Shell("notepad.exe",3)
    Else
        AppActivate "Notepad"
        AppMaximize
    End If
    HPage 3      'Move 3 pages down in Notepad.
End Sub
```

See Also

HLine (statement); HScroll (statement).

statement

HScroll

Syntax

```
HScroll percentage
```

Description

Sets the thumb mark on the horizontal scroll bar attached to the current window.

Comments

- The position is given as a percentage of the total range associated with that scroll bar. For example, if the percentage parameter is 50, then the thumb mark is positioned in the middle of the scroll bar.

Example

```
'This example centers the thumb mark on the horizontal scroll bar of
'the Notepad window.

Sub Main()
    If AppFind$("Notepad") = "" Then
        id = Shell("notepad.exe",3)
    Else
        AppActivate "Notepad"
        AppMaximize
    End If
    HScroll 50          'Jump to the middle of the Notepad document.
End Sub
```

See Also

HLine (statement); HPage (statement).

object

HWND

Syntax

```
Dim name As HWND
```

Description

A data type used to hold window objects.

Comments

- This data type is used to hold references to physical windows in the operating environment. The following commands operate on HWND objects:

  WinActivate WinClose WinFind WinList
  WinMaximize WinMinimize WinMove WinRestore
  WinSize

- The above language elements support both string and HWND window specifications.

Example

'This example activates the "Main" MDI window within Program Manager.

```
Sub Main()
    Dim ProgramManager As HWND
    Dim ProgramManagerMain As HWND
    Set ProgramManager = WinFind("Program Manager")
    If ProgramManager Is Not Nothing Then
        WinActivate ProgramManager
        WinMaximize ProgramManager
        Set ProgramManagerMain = WinFind("Program Manager|Main")
        If ProgramManagerMain Is Not Nothing Then
            WinActivate ProgramManagerMain
            WinRestore ProgramManagerMain
        Else
            MsgBox "Your Program Manager doesn't have a Main group."
        End If
    Else
        MsgBox "Program Manager is not running."
    End If
End Sub
```

See Also

HWND.Value (property); WinFind (function); WinActivate (function).

property

HWND.Value

Syntax

```
window.Value
```

Description

The default property of an HWND object that returns a Variant containing a HANDLE to the physical window of an HWND object variable.

Comments

- The Value property is used to retrieve the operating environment-specific value of a given HWND object. The size of this value depends on the operating environment in which the script is executing and thus should always be placed into a Variant variable.
- This property is read-only.

Example

```
'This example displays a dialog box containing the class name of
'Program Manager's Main window. It does so using the .Value property,
'passing it directly to a Windows' external routine.

Declare Sub GetClassName Lib "user" (ByVal Win%,ByVal ClsName$,ByVal ClsNameLen%)

Sub Main()
    Dim ProgramManager As HWND
    Set ProgramManager = WinFind("Program Manager")
    ClassName$ = Space(40)
    GetClassName ProgramManager.Value,ClassName$,Len(ClassName$)
    MsgBox "The program classname is: " & ClassName$
End Sub
```

See Also

HWND (data type).

Note

- This value is an Integer.

statement

If...Then...Else

Syntax 1

```
If condition Then statements [Else else_statements]
```

Syntax 2

```
If condition Then
  [statements]
[ElseIf else_condition Then
  [elseif_statements]]
[Else
  [else_statements]]
End If
```

Description

Conditionally runs a statement or group of statements.

Comments

- The single-line conditional statement (syntax 1) has the following parameters:

Parameter	Description
condition	Any expression evaluating to a Boolean value.
statements	One or more statements separated with colons. This group of statements is run when condition is True.
else_statements	One or more statements separated with colons. This group of statements is run when condition is False.

- The multiline conditional statement (syntax 2) has the following parameters:

Parameter	Description
condition	Any expression evaluating to a Boolean value.
statements	One or more statements to be run when condition is True.
else_condition	Any expression evaluating to a Boolean value. The else_condition is evaluated if condition is False.
elseif_statements	One or more statements to be exected when condition is False and else_condition is True.
else_statements	One or more statements to be run when both condition and else_condition are False.

- There can be as many ElseIf conditions as required.

Example

```
'This example inputs a name from the user and checks to see whether it
'is MICHAEL or MIKE using three forms of the If...Then...Else
'statement. It then branches to a statement that displays a welcome
'message depending on the user's name.

Sub Main()
    uname$ = UCase(InputBox("Enter your name:","Enter Name"))
    If uname$ = "MICHAEL" Then GoSub MikeName

    If uname$ = "MIKE" Then
        GoSub MikeName
        Exit Sub
    End If
```

```
    If uname$ = "" Then
        MsgBox "Since you don't have a name, I'll call you MIKE!"
        uname$ = "MIKE"
        GoSub MikeName
    ElseIf uname$ = "MICHAEL" Then
        GoSub MikeName
    Else
        GoSub OtherName
    End If
    Exit Sub

MikeName:
    MsgBox "Hello, MICHAEL!"
    Return

OtherName:
    MsgBox "Hello, " & uname$ & "!"
    Return
End Sub
```

See Also

Choose (function); Switch (function); IIf (function); Select...Case (statement).

function

IIf

Syntax

```
IIf(condition,TrueExpression,FalseExpression)
```

Description

Returns TrueExpression if condition is True; otherwise, returns FalseExpression.

Comments

- Both expressions are calculated before IIf returns.
- The IIf function is shorthand for the following construct:

```
If condition Then
variable = TrueExpression
Else
variable = FalseExpression
End If
```

Example

```
Sub Main()
    s$ = "Car"
    MsgBox "You have a " & IIf(s$ = "Car","nice car.","nice non-car.")
End Sub
```

See Also

Choose (function); Switch (function); If...Then...Else (statement); Select...Case (statement).

operator

Imp

Syntax

```
expression1 Imp expression2
```

Description

Performs a logical or binary implication on two expressions.

Comments

- If both expressions are either Boolean, Boolean variants, or Null variants, then a logical implication is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Binary Implication

If the two expressions are Integer, then a binary implication is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary implication is then performed, returning a Long result.

Binary implication forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:

```
1       Imp 1   =   1       Example:
0       Imp 1   =   1       5    01101001
1       Imp 0   =   0       6    10101010
0       Imp 0   =   1       Imp 10111110
```

Example

```
'This example compares the result of two expressions to determine
'whether one implies the other.

Sub Main()
    a = 10 : b = 20 : c = 30 : d = 40

    If (a < b) Imp (c < d) Then
        MsgBox "a is less than b implies that c is less than d."
    Else
        MsgBox "a is less than b does not imply that c is less than d."
    End If

    If (a < b) Imp (c > d) Then
        MsgBox "a is less than b implies that c is greater than d."
    Else
        MsgBox "a is less than b does not imply that c is greater than d."
    End If
End Sub
```

See Also

Operator Precedence (topic); Or (operator); Xor (operator); Eqv (operator); And (operator).

statement

Inline

Syntax

```
Inline name [parameters]
     anytext
End Inline
```

Description

Enables execution or interpretation of a block of text.

Comments

- The Inline statement takes the following parameters:

Parameter	Description
name	Identifier specifying the type of inline statement.
parameters	Comma-separated list of parameters.
anytext	Text to be run by the Inline statement. This text must be in a format appropriate for execution by the Inline statement. The end of the text is assumed to be the first occurrence of the words End Inline appearing on a line.

Example

```
Sub Main()
    Inline WinScript
        -- This is a Windows comment.
        Beep
        Display Dialog "Windows" buttons "OK" default button "OK"
        Display Dialog Current Date
    End Inline
End Sub
```

statement

Input#

Syntax

```
Input [#]filenumber%,variable[,variable]...
```

Description

Reads data from the file referenced by filenumber into the given variables.

Comments

- Each variable must be type-matched to the data in the file. For example, a String variable must be matched to a string in the file.
- The following parsing rules are observed while reading each variable in the variable list:
  1. Leading white space is ignored (spaces and tabs).
  2. When reading String variables, if the first character on the line is a quotation mark, then characters are read up to the next quoation mark or the end of the line, whichever comes first. Blank lines are read as empty strings. If the first character read is not a quoation mark, then characters are read up to the first comma or the end of the line, whichever comes first. String delimiters (quotes, comma, end-of-line) are not included in the returned string.
  3. When reading numeric variables, scanning of the number stops when the first nonnumber character (such as a comma, a letter, or any other unexpected character) is encountered. Numeric errors are ignored while reading numbers from a file. The resultant number is automatically converted to the same type as the variable into which the value will be placed. If there is an error in conversion, then 0 is stored into the variable.

     After reading the number, input is skipped up to the next delimiter-a comma, an end-of-line, or an end-of-file.

     Numbers must adhere to any of the following syntaxes:

     ```
     [-|+]digits[.digits][E[-|+]digits][!|#|%|&|@]

     &Hhexdigits[!|#|%|&]

     &[O]octaldigits[!|#|%|&|@]
     ```

  4. When reading Boolean variables, the first character must be #; otherwise, a runtime error occurs. If the first character is #, then input is scanned up to the next delimiter (a comma, an end-of-line, or an end-of-file). If the input matches #FALSE#, then False is stored in the Boolean; otherwise True is stored.
  5. When reading Date variables, the first character must be #; otherwise, a runtime error occurs. If the first character is #, then the input is scanned up to the next delimiter (a comma, an end-of-line, or an end-of-file). If the input ends in a # and the text between the #'s can be correctly interpreted as a date, then the date is stored; otherwise, December 31, 1899, is stored.

     Normally, dates that follow the universal date format are input from sequential files. These dates use this syntax:

     ```
     #YYYY-MM-DD HH:MM:SS#
     ```

     where YYYY is a year between 100 and 9999, MM is a month between 1 and 12, DD is a day between 1 and 31, HH is an hour between 0 and 23, MM is a minute between 0 and 59, and SS is a second between 0 and 59.
  6. When reading Variant variables, if the data begins with a quotation mark, then a string is read consisting of the characters between the opening quotation mark and the closing quoation mark, end-of-line, or end-of-file.

     If the input does not begin with a quotation mark, then input is scanned up to the next comma, end-

of-line, or end-of-file and a determination is made as to what data is being represented. If the data cannot be represented as a number, Date, Error, Boolean, or Null, then it is read as a string.

If an error occurs in interpretation of the data as a particular type, then that data is read as a String variant.

When reading numbers into variants, the optional type-declaration character determines the VarType of the resulting variant. If no type-declaration character is specified, then Delrina Basic will read the number according to the following rules:

Rule 1: If the number contains a decimal point or an exponent, then the number is read as Currency. If there is an error converting to Currency, then the number is treated as a Double.

Rule 2: If the number does not contain a decimal point or an exponent, then the number is stored in the smallest of the following data types that most accurately represents that value: Integer, Long, Currency, Double.

7. End-of-line is interpreted as either a single line feed, a single carriage return, or a carriage-return/line-feed pair. Thus, text files from any platform can be interpreted using this command.

- The filenumber parameter is a number that is used by Delrina Basic to refer to the open filethe number passed to the Open statement.
- The filenumber must reference a file opened in Input mode. It is good practice to use the Write statement to write date elements to files read with the Input statement to ensure that the variable list is consistent between the input and output routines.

Example

```
'This example creates a file called test.dat and writes a series of
'variables into it. Then the variables are read using the Input#
'function.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Open "test.dat" For Output As #1
    Write #1,2112,"David","McCue","123-45-6789"
    Close

    Open "test.dat" For Input As #1
    Input #1,x%,s1$,s2$,s3$
    msg = "Employee #" & x% & " Personal Information" & crlf & crlf
    msg =   msg & "First Name: " & s1$ & crlf
    msg = msg & "Last Name: "& s2$ & crlf
    msg = msg & "Social Security Number: " & s3$
    MsgBox msg
    Close

    Kill "test.dat"
End Sub
```

See Also

Open (statement); Get (statement); Line Input# (statement); Input, Input$ (functions).

function

Input, Input$

Syntax

```
Input[$](numbytes,[#]filenumber)
```

Description

Returns numbytes characters read from a given sequential file.

Comments

- Input$ returns a String, whereas Input returns a String variant.
- The Input/Input$ functions require the following parameters:

Parameter	Description
numbytes	Integer containing the number of bytes to be read from the file.
filenumber	Integer referencing a file opened in either Input or Binary mode. This is the same number passed to the Open statement.

- This function reads all characters, including spaces and end-of-lines.

Example

```
'This example opens the autoexec.bat file and displays it in a
'dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    file$ = "c:\autoexec.bat"
    x& = FileLen(file$)

    If x& > 0 Then
        Open file$ For Input As #1
    Else
        MsgBox "" & file$ & "' not found or empty."
        Exit Sub
    End If

    'use the file length to read the file in
    If x& > 80 Then
        ins = Input(80,1)
    Else
        ins = Input(x&,1)
    End If
    Close

    MsgBox UCase(file$) & crlf & crlf & "File length: " & x& & crlf & "Contents:" & crlf & ins
End Sub
```

See Also

Open (statement); Get (statement); Input# (statement); Line Input# (statement).

function

InputBox, InputBox$

Syntax

```
InputBox[$](prompt [,[title] [,[default] [,X,Y]]])
```

Description

Displays a dialog box with a text box into which the user can type.

Comments

- The content of the text box is returned as a String (in the case of InputBox$) or as a String variant (in the case of InputBox). A zero-length string is returned if the user selects Cancel.
- The InputBox/InputBox$ functions take the following parameters:

Parameter	Description
prompt	Text to be displayed above the text box. The prompt parameter can contain multiple lines, each separated with an end-of-line (a carriage return, line feed, or carriage-return/line-feed pair). A runtime error is generated if prompt is Null.
title	Caption of the dialog box. If this parameter is omitted, then no title appears as the dialog box's caption. A runtime error is generated if title is Null.
default	Default response. This string is initially displayed in the text box. A runtime error is generated if default is Null.
X, Y	Integer coordinates, given in twips (twentieths of a point), specifying the upper left corner of the dialog box relative to the upper left corner of the screen. If the position is omitted, then the dialog box is positioned on or near the application executing the script.

Example

```
Sub Main()
    s$ = InputBox("File to copy:","Copy","sample.txt")
End Sub
```

See Also

MsgBox (statement); AskBox$ (function); AskPassword$ (function); OpenFilename$ (function); SaveFilename$ (function); SelectBox (function); AnswerBox (function).

function

InStr

Syntax

```
InStr([start,] search, find [,compare])
```

Description

Returns the first character position of string find within string search.

Comments

- The InStr function takes the following parameters:

Parameter	Description
start	Integer specifying the character position where searching begins. The start parameter must be between 1 and 32767.
	If this parameter is omitted, then the search starts at the beginning (start = 1).
search	Text to search. This can be any expression convertible to a String.
find	Text for which to search. This can be any expression convertible to a String.
compare	Integer controlling how string comparisons are performed:
0	String comparisons are case-sensitive.
1	String comparisons are case-insensitive.
Any other value	A runtime error is produced.
	If this parameter is omitted, then string comparisons use the current Option Compare setting. If no Option Compare statement has been encountered, then Binary is used (i.e., string comparisons are case-sensitive).
	If the string is found, then its character position within search is returned, with 1 being the character position of the first character. If find is not found, or start is greater than the length of search, or search is zero-length, then 0 is returned.

Example

```
'This example checks to see whether one string is in another and,
'if it is, then it copies the string to a variable and displays the
'result.

Sub Main()
    a$ = "This string contains the name Stuart and other characters."
    x% = InStr(a$,"Stuart",1)
    If x% <> 0 Then
        b$ = Mid(a$,x%,6)
        MsgBox b$ & " was found."
        Exit Sub
    Else
        MsgBox "Stuart not found."
    End If
End Sub
```

See Also

Mid, Mid$ (functions); Option Compare (statement); Item$ (function); Word$ (function); Line$ (function).

function

Int

Syntax

```
Int(number)
```

Description

Returns the integer part of number.

Comments

- This function returns the integer part of a given value by returning the first integer less than the number. The sign is preserved.
- The Int function returns the same type as number, with the following exceptions:
- If number is Empty, then an Integer variant of value 0 is returned.
- If number is a String, then a Double variant is returned.
- If number is Null, then a Null variant is returned.

Example

```
'This example extracts the integer part of a number.

Sub Main()
    a# = -1234.5224
    b% = Int(a#)
    MsgBox "The integer part of -1234.5224 is: " & b%
End Sub
```

See Also

Fix (function); CInt (function).

data type

Integer

Syntax

Integer

Description

A data type used to declare whole numbers with up to four digits of precision.

Comments

- Integer variables are used to hold numbers within the following range:
  ```
  -32768 <= integer <= 32767
  ```
- Internally, integers are 2-byte short values. Thus, when appearing within a structure, integers require 2 bytes of storage. When used with binary or random files, 2 bytes of storage are required.
- When passed to external routines, Integer values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack.
- The type-declaration character for Integer is %.

See Also

Currency (data type); Date (data type); Double (data type); Long (data type), Object (data type), Single (data type), String (data type), Variant (data type), Boolean (data type), DefType (statement), CInt (function).

function

IPmt

Syntax

```
IPmt(Rate, Per, Nper, Pv, Fv, Due)
```

Description

Returns the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Comments

- An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages, monthly savings plans, and retirement plans.
- The following table describes the different parameters:

Parameter	Description
Rate	Double representing the interest rate per period. If the payment periods are monthly, be sure to divide the annual interest rate by 12 to get the monthly rate.
Per	Double representing the payment period for which you are calculating the interest payment. If you want to know the interest paid or received during period 20 of an annuity, this value would be 20.
Nper	Double representing the total number of payments in the annuity. This is usually expressed in months, and you should be sure that the interest rate given above is for the same period that you enter here.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan because that is the amount of cash you have in the present. In the case of a retirement plan, this value would be the current value of the fund because you have a set amount of principal in the plan.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be zero because you will have paid it off. In the case of a savings plan, the future value would be the balance of the account after all payments are made.
Due	Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period (usually, the end of the month). If this value is 1, then payments are due at the start of each period (the beginning of the month).

- Rate and Nper must be in expressed in the same units. If Rate is expressed in percentage paid per month, then Nper must also be expressed in months. If Rate is an annual rate, then the period given in Nper should also be in years or the annual Rate should be divided by 12 to obtain a monthly rate.
- If the function returns a negative value, it represents interest you are paying out, whereas a positive value represents interest paid to you.

Example

```
'This example calculates the amount of interest paid on a $1,000.00
'loan financed over 36 months with an annual interest rate of 10%.
'Payments are due at the beginning of the month. The interest paid
'during the first 10 months is displayed in a table.

Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
    For x = 1 to 10
        ipm# = IPmt((.10/12),x,36,1000,0,1)
        msg = msg & Format(x,"00") & " : " & Format(ipm#," 0,0.00") & crlf
    Next x
    MsgBox msg
End Sub
```

See Also

NPer (function); Pmt (function); PPmt (function); Rate (function).

function

IRR

Syntax

```
IRR(ValueArray(),Guess)
```

Description

Returns the internal rate of return for a series of periodic payments and receipts.

Comments

- The internal rate of return is the equivalent rate of interest for an investment consisting of a series of positive and/or negative cash flows over a period of regular intervals. It is usually used to project the rate of return on a business investment that requires a capital investment up front and a series of investments and returns on investment over time.
- The IRR function requires the following parameters:

Parameter	Description
ValueArray()	Array of Double numbers that represent payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one negative value to indicate the initial investment (negative value) and the amount earned by the investment (positive value).
Guess	Double containing your guess as to the value that the IRR function will return. The most common guess is .1 (10 percent).

- The value of IRR is found by iteration. It starts with the value of Guess and cycles through the calculation adjusting Guess until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, IRR fails, and the user must pick a better guess.

Example

```
'This example illustrates the purchase of a lemonade stand for $800
'and a series of incomes from the sale of lemonade over 12 months.
'The projected incomes for this example are generated in two
'For...Next Loops, and then the internal rate of return is calculated
'and displayed. (Not a bad investment!)

Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
    Dim valu#(12)
    valu(1) = -800                          'Initial investment
    msg = valu#(1) & ", "

    'Calculate the second through fifth months' sales.
    For x = 2 To 5
        valu(x) = 100 + (x * 2)
        msg = msg & valu(x) & ", "
    Next x

    'Calcluate the sixth through twelfth months' sales.
    For x = 6 To 12
        valu(x) = 100 + (x * 10)
        msg = msg & valu(x) & ", "
    Next x

    'Calcluate the equivalent investment return rate.
    retrn# = IRR(valu,.1)
    msg = "The values: " & crlf & msg & crlf & crlf
    MsgBox msg & "Return rate: " & Format(retrn#,"Percent")
End Sub
```

See Also

Fv (function); MIRR (function); Npv (function); Pv (function).

operator

Is

Syntax

```
object Is [object | Nothing]
```

Description

Returns True if the two operands refer to the same object; returns False otherwise.

Comments

- This operator is used to determine whether two object variables refer to the same object. Both operands must be object variables of the same type (i.e., the same data object type or both of type Object).
- The Nothing constant can be used to determine whether an object variable is uninitialized:

```
If MyObject Is Nothing Then MsgBox "MyObject is uninitialized."
```

- Uninitialized object variables reference no object.

Example

```
'This function inserts the date into a Microsoft Word document.

Sub InsertDate(ByVal WinWord As Object)
    If WinWord Is Nothing Then
        MsgBox "Object variant is not set."
    Else
        WinWord.Insert Date$
    End If
End Sub

Sub Main()
    Dim WinWord As Object
    On Error Resume Next
    WinWord = CreateObject("word.basic")
    InsertDate WinWord
End Sub
```

See Also

Operator Precedence (topic); Like (operator).

Note

- When comparing OLE automation objects, the Is operator will only return True if the operands reference the same OLE automation object. This is different from data objects. For example, the following use of Is (using the object class called excel.application) returns True:

```
Dim a As Object
Dim b As Object
a = CreateObject("excel.application")
b = a
If a Is b Then Beep
```

The following use of Is will return False, even though the actual objects may be the same:

```
Dim a As Object
Dim b As Object
a = CreateObject("excel.application")
b = GetObject(,"excel.application")
If a Is b Then Beep
```

The Is operator may return False in the above case because, even though a and b reference the same object, they may be treated as different objects by OLE 2.0 (this is dependent on the OLE 2.0 server application).

function

IsDate

Syntax

```
IsDate(expression)
```

Description

Returns True if expression can be legally converted to a date; returns False otherwise.

Example

```
Sub Main()
    Dim a As Variant
Retry:
    a = InputBox("Enter a date.","Enter Date")
    If IsDate(a) Then
        MsgBox Format(a,"long date")
    Else
        Msgbox "Not quite, please try again!"
        Goto Retry
    End If
End Sub
```

See Also

Variant (data type); IsEmpty (function); IsError (function); IsObject (function); VarType (function); IsNull (function).

function

IsEmpty

Syntax

```
IsEmpty(expression)
```

Description

Returns True if expression is a Variant variable that has never been initialized; returns False otherwise.

Comments

- The IsEmpty function is the same as the following:
  ```
  (VarType(expression) = ebEmpty)
  ```

Example

```
Sub Main()
    Dim a As Variant
    If IsEmpty(a) Then
        a = 1.0#            'Give uninitialized data a Double value 0.0.
        MsgBox "The variable has been initialized to: " & a
    Else
        MsgBox "The variable was already initialized!"
    End If
End Sub
```

See Also

Variant (data type); IsDate (function); IsError (function); IsObject (function); VarType (function); IsNull (function).

function

IsError

Syntax

```
IsError(expression)
```

Description

Returns True if expression is a user-defined error value; returns False otherwise.

Example

```
'This example creates a function that divides two numbers. If there
'is an error dividing the numbers, then a variant of type "error" is
'returned. Otherwise, the function returns the result of the division.
'The IsError function is used to determine whether the function
'encountered an error.

Function Div(ByVal a,ByVal b) As Variant
    If b = 0 Then
        Div = CVErr(2112)        'Return a special error value.
    Else
        Div = a / b              'Return the division.
    End If
End Function

Sub Main()
    Dim a As Variant
    a = Div(10,12)
    If IsError(a) Then
        MsgBox "The following error occurred: " & CStr(a)
    Else
        MsgBox "The result of the division is: " & a
    End If
End Sub
```

See Also

Variant (data type); IsEmpty (function); IsDate (function); IsObject (function); VarType (function); IsNull (function).

function

IsMissing

Syntax

```
IsMissing(variable)
```

Description

Returns True if variable was passed to the current subroutine or function; returns False if omitted.

Comments

- The IsMissing is used with variant variables passed as optional parameters (using the Optional keyword) to the current subroutine or function. For non-variant variables or variables that were not declared with the Optional keyword, IsMissing will always return True.

Example

```
'The following function runs an application and optionally minimizes it. If
'the optional isMinimize parameter is not specified by the caller, then the
'application is not minimized.

Sub Test(AppName As String,Optional isMinimize As Variant)
    app = Shell(AppName)
    If Not IsMissing(isMinimize) Then
        AppMinimize app
    Else
        AppMaximize app
    End If
End Sub

Sub Main
    Test "notepad.exe"                  'Maximize this application
    Test "notepad.exe",True         'Mimimize this application
End Sub
```

See Also

Declare (statement), Sub...End Sub (statement), Function...End Function (statement)

function

IsNull

Syntax

```
IsNull(expression)
```

Description

Returns True if expression is a Variant variable that contains no valid data; returns False otherwise.

Comments

- The IsNull function is the same as the following:
  ```
  (VarType(expression) = ebNull)
  ```

Example

```
Sub Main()
    Dim a As Variant        'Initialized as Empty
    If IsNull(a) Then MsgBox "The variable contains no valid data."
    a = Empty * Null
    If IsNull(a) Then MsgBox "Null propagated through the expression."
End Sub
```

See Also

Empty (constant); Variant (data type); IsEmpty (function); IsDate (function); IsError (function); IsObject (function); VarType (function).

function

IsNumeric

Syntax

```
IsNumeric(expression)
```

Description

Returns True if expression can be converted to a number; returns False otherwise.

Comments

- If passed a number or a variant containing a number, then IsNumeric always returns True.
- If a String or String variant is passed, then IsNumeric will return True only if the string can be converted to a number. The following syntaxes are recognized as valid numbers:

```
&Hhexdigits[&|%|!|#|@]

&[O]octaldigits[&|%|!|#|@]

[-|+]digits[.[digits]][E[-|+]digits][!|%|&|#|@]
```

- If an Object variant is passed, then the default property of that object is retrieved and one of the above rules is applied.
- IsNumeric returns False if expression is a Date.

Example

```
Sub Main()
    Dim s$ As String
    s$ = InputBox("Enter a number.","Enter Number")

    If IsNumeric(s$) Then
        MsgBox "You did good!"
    Else
        MsgBox "You didn't do so good!"
    End If
End Sub
```

See Also

Variant (data type); IsEmpty (function); IsDate (function); IsError (function); IsObject (function); VarType (function); IsNull (function).

function

IsObject

Syntax

```
IsObject(expression)
```

Description

Returns True if expression is a Variant variable containing an Object; returns False otherwise.

Example

'This example will attempt to find a running copy of Excel and create 'a Excel object that can be referenced as any other object in 'Delrina Basic.

```
Sub Main()
    Dim v As Variant
    On Error Resume Next
    Set v = GetObject(,"Excel.Application")

    If IsObject(v) Then
        MsgBox "The default object value is: " & v = v.Value        'Access value property of the object.
    Else
        MsgBox "Excel not loaded."
    End If
End Sub
```

See Also

Variant (data type); IsEmpty (function); IsDate (function); IsError (function); VarType (function); IsNull (function).

function

Item$

Syntax

```
Item$(text$,first,last [,delimiters$])
```

Description

Returns all the items between first and last within the specified formatted text list.

Comments

- The Item$ function takes the following parameters:

Parameter	Description
text$	String containing the text from which a range of items is returned.
first	Integer containing the index of the first item to be returned. If first is greater than the number of items in text$, then a zero-length string is returned.
last	Integer containing the index of the last item to be returned. All of the items between first and last are returned. If last is greater than the number of items in text$, then all items from first to the end of text are returned.
delimiters$	String containing different item delimiters. By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the delimiters$ parameter.

Example

```
'This example creates two delimited lists and extracts a range from
'each, then displays the result in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    ilist$ = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15"
    slist$ = "1/2/3/4/5/6/7/8/9/10/11/12/13/14/15"
    list1$ = Item$(ilist$,5,12)
    list2$ = Item$(slist$,2,9,"/")
    MsgBox "The returned lists are: " & crlf & list1$ & crlf & list2$
End Sub
```

See Also

ItemCount (function); Line$ (function); LineCount (function); Word$ (function); WordCount (function).

function

ItemCount

Syntax

```
ItemCount(text$ [,delimiters$])
```

Description

Returns an Integer containing the number of items in the specified delimited text.

Comments

- Items are substrings of a delimited text string. Items, by default, are separated by commas and/or end-of-lines. This can be changed by specifying different delimiters in the delimiters$ parameter. For example, to parse items using a backslash:

```
n = ItemCount(text$,"\")
```

Example

```
'This example creates two delimited lists and then counts the number
'of items in each. The counts are displayed in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    ilist$ = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15"
    slist$ = "1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19"

    l1% = ItemCount(ilist$)
    l2% = ItemCount(slist$,"/")
    msg = "The first lists contains: " & l1% & " items." & crlf
    msg = msg & "The second list contains: " & l2% & " items."
    MsgBox msg
End Sub
```

See Also

Item$ (function); Line$ (function); LineCount (function); Word$ (function); WordCount (function).

topic

Keywords

A keyword is any word or symbol recognized by Delrina Basic as part of the language. All of the following are keywords:

- Built-in subroutine names, such as MsgBox and Print.
- Built-in function names, such as Str$, CDbl, and Mid$.
- Special keywords, such as To, Next, Case, and Binary.
- Names of any extended language elements.

Restrictions

All keywords are reserved by Delrina Basic, in that you cannot create a variable, function, constant, or subroutine with the same name as a keyword. However, you are free to use all keywords as the names of structure members.

statement

Kill

Syntax

```
Kill filespec$
```

Description

Deletes all files matching filespec$.

Comments

- The filespec$ argument can include wildcards, such as * and ?. The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complex searching patterns. The following table shows some examples.

This Pattern	Matches These Files	Doesn't Match These Files
S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
C*T.TXT	CAT.TXT CAP.TXT ACATS.TXT	
C*T	CAT CAT.DOC CAP.TXT	
C?T	CAT CAT.TXT CUT CAPIT CT	
*	(All files)	

Example

```
'This example looks to see whether file test1.dat exists. If it does not,
'then it creates both test1.dat and test2.dat. The existence of the files
'is tested again; if they exist, a message is generated, and then
'they are deleted. The final test looks to see whether they are still
'there and displays the result.

Sub Main()
    If Not FileExists("test1.dat") Then
        Open "test1.dat" For Output As #1
        Open "test2.dat" For Output As #2
        Close
    End If

    If FileExists ("test1.dat") Then
        MsgBox "File test1.dat exists."
        Kill "test?.dat"
    End If

    If FileExists ("test1.dat") Then
        MsgBox "File test1.dat still exists."
    Else
        MsgBox "test?.dat sucessfully deleted."
    End If
End Sub
```

See Also

Name (statement).

Note

- Notice that Delrina Basic's filename matching is different than DOS's. The pattern "*.*" under DOS matches all files. With Delrina Basic, this pattern matches only files that have file extensions.
- This function behaves the same as the "del" command in DOS.

function

LBound

Syntax

```
LBound(ArrayVariable() [,dimension])
```

Description

Returns an Integer containing the lower bound of the specified dimension of the specified array variable.

Comments

- The dimension parameter is an integer specifying the desired dimension. If this parameter is not specified, then the lower bound of the first dimension is returned.
- The LBound function can be used to find the lower bound of a dimension of an array returned by an OLE automation method or property:

```
LBound(object.property [,dimension])

LBound(object.method [,dimension])
```

Examples

```
Sub Main()
    'This example dimensions two arrays and displays their lower bounds.

    Dim a(5 To 12)
    Dim b(2 To 100,9 To 20)

    lba = LBound(a)
    lbb = LBound(b,2)
    MsgBox "The lower bound of a is: " & lba & " The lower bound of b is: " & lbb

    'This example uses LBound and UBound to dimension a dynamic array to
    'hold a copy of an array redimmed by the FileList statement.

    Dim fl$()
    FileList fl$,"*.*"
    count = UBound(fl$)
    If ArrayDims(a) Then
        Redim nl$(LBound(fl$) To UBound(fl$))
        For x = 1 To count
            nl$(x) = fl$(x)
        Next x
        MsgBox "The last element of the new array is: " & nl$(count)
    End If
End Sub
```

See Also

UBound (function); ArrayDims (function); Arrays (topic).

function

LCase, LCase$

Syntax

```
LCase[$](text)
```

Description

Returns the lowercase equivalent of the specified string.

Comments

- LCase$ returns a String, whereas LCase returns a String variant.
- Null is returned if text is Null.

Example

```
'This example shows the LCase function used to change uppercase names
'to lowercase with an uppercase first letter.

Sub Main()
    lname$ = "WILLIAMS"
    fl$ = Left(lname$,1)
    rest$ = Mid(lname$,2,Len(lname$))
    lname$ = fl$ & LCase(rest$)
    MsgBox "The converted name is: " & lname$
End Sub
```

See Also

UCase, UCase$ (functions).

function

Left, Left$

Syntax

```
Left[$](text,NumChars)
```

Description

Returns the leftmost NumChars characters from a given string.

Comments

- Left$ returns a String, whereas Left returns a String variant.
- NumChars is an Integer value specifying the number of character to return. If NumChars is 0, then a zero-length string is returned. If NumChars is greater than or equal to the number of characters in the specified string, then the entire string is returned.
- Null is returned if text is Null.

Example

```
'This example shows the Left$ function used to change uppercase names
'to lowercase with an uppercase first letter.

Sub Main()
    lname$ = "WILLIAMS"
    fl$ = Left(lname$,1)
    rest$ = Mid(lname$,2,Len(lname$))
    lname$ = fl$ & LCase(rest$)
    MsgBox "The converted name is: " & lname$
End Sub
```

See Also

Right, Right$ (functions).

function

Len

Syntax

```
Len(expression)
```

Description

Returns the number of characters in expression or the number of bytes required to store the specified variable.

Comments

- If expression evaluates to a string, then Len returns the number of characters in a given string or 0 if the string is empty. When used with a Variant variable, the length of the variant when converted to a String is returned. If expression is a Null, then Len returns a Null variant.
- If used with a non-String or non-Variant variable, the function returns the number of bytes occupied by that data element.
- When used with user-defined data types, the function returns the combined size of each member within the structure. Since variable-length strings are stored elsewhere, the size of each variable-length string within a structure is 2 bytes.
- The following table describes the sizes of the individual data elements:

Data Element	Size
Integer	2 bytes.
Long	4 bytes.
Float	4 bytes.
Double	8 bytes.
Currency	8 bytes.
String (variable-length)	Number of characters in the string.
String (fixed-length)	The length of the string as it appears in the string's declaration.
Objects	0 bytes. Both data object variables and variables of type Object are always returned as 0 size.
User-defined type	Combined size of each structure member. Variable-length strings within structures require 2 bytes of storage. Arrays within structures are fixed in their dimensions. The elements for fixed arrays are stored within the structure and therefore require the number of bytes for each array element multiplied by the size of each array dimension: element_size * dimension1 * dimension2...

- The Len function always returns 0 with object variables or any data object variable.

Examples

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    'This example shows the Len function used in a routine to change
    'uppercase names to lowercase with an uppercase first letter.
    lname$ = "WILLIAMS"
    fl$ = Left(lname$,1)
    ln% = Len(lname$)
    rest$ = Mid(lname$,2,ln%)
    nname$ = fl$ & LCase(rest$)
    MsgBox "The proper case for " & lname$ & " is " & nname$ & "."
```

```
    'This example returns a table of lengths for standard numeric types.
    Dim lns(4)
    a% = 100 : b& = 200 : c! = 200.22 : d# = 300.22
    lns(1) = Len(a%)
    lns(2) = Len(b&)
    lns(3) = Len(c!)
    lns(4) = Len(d#)
    msg = "Lengths (in bytes) of standard types:" & crlf & crlf
    msg = msg & "Integer: " & lns(1) & crlf
    msg = msg & "Long: " & lns(2) & crlf
    msg = msg & "Single: " & lns(3) & crlf
    msg = msg & "Double: " & lns(4) & crlf
    MsgBox msg
End Sub
```

See Also

InStr (function).

statement

Let

Syntax

```
[Let] variable = expression
```

Description

Assigns the result of an expression to a variable.

Comments

- The use of the word Let is supported for compatibility with other implementations of Delrina Basic. Normally, this word is dropped.
- When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This happens when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:

```
Dim amount As Long
Dim quantity As Integer

amount = 400123      'Assign a value out of range for int.
quantity = amount    'Attempt to assign to Integer.
```

When performing an automatic data conversion, underflow is not an error.

Example

```
Sub Main()
    Let a$ = "This is a string."
    Let b% = 100
    Let c# = 1213.3443
End Sub
```

See Also

= (keyword); Expression Evaluation (topic).

operator

Like

Syntax

```
expression Like pattern
```

Description

Compares two strings and returns True if the expression matches the given pattern; returns False otherwise.

Comments

- Case sensitivity is controlled by the Option Compare setting.
- The pattern expression can contain special characters that let more flexible matching:

Character	Evaluates To
?	Matches a single character.
*	Matches one or more characters.
#	Matches any digit.
[range]	Matches if the character in question is within the specified range.
[!range]	Matches if the character in question is not within the specified range.

- A range specifies a grouping of characters. To specify a match of any of a group of characters, use the syntax [ABCDE]. To specify a range of characters, use the syntax [A-Z]. Special characters must appear within brackets, such as []*?#.
- If expression or pattern is not a string, then both expression and pattern are converted to String variants and compared, returning a Boolean variant. If either variant is Null, then Null is returned.
- The following table shows some examples:

expression	**True** `If pattern Is`	**False** `If pattern Is`
"EBW"	"E*W", "E*"	"E*B"
"Delrina Basic"	"B*[r-t]icScript"	"B[r-t]ic"
"Version"	"V[e]?s*n"	"V[r]?s*N"
"2.0"	"#.#", "#?#"	"###", "#?[!0-9]"
"[ABC]"	"[[]*]"	"[ABC]", "[*]"

Example

```
'This example demonstrates various uses of the Like function.

Sub Main()
    a$ = "This is a string variable of 123456 characters"
    b$ = "123.45"
    If a$ Like "[A-Z][g-i]*" Then MsgBox "The first comparison is True."
    If b$ Like "##3.##" Then MsgBox "The second comparison is True."
    If a$ Like "*variable*" Then MsgBox "The third comparison is True."
End Sub
```

See Also

Operator Precedence (topic); Is (operator); Option Compare (statement).

statement

Line Input#

Syntax

```
Line Input [#]filenumber,variable
```

Description

Reads an entire line into the given variable.

Comments

- The filenumber parameter is a number that is used by Delrina Basic to refer to the open filethe number passed to the Open statement. The filenumber must reference a file opened in Input mode.
- The file is read up to the next end-of-line, but the end-of-line character(s) is (are) not returned in the string. The file pointer is positioned after the terminating end-of-line.
- The variable parameter is any string or variant variable reference. This statement will automatically declare the variable if the specified variable has not yet been used or dimensioned.
- This statement recognizes either a single line feed or a carriage-return/line-feed pair as the end-of-line delimiter.

Example

```
'This example reads five lines of the autoexec.bat file and displays
'them in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    file$ = "c:\autoexec.bat"
    Open file$ For Input As #1
    For x = 1 To 5
        Line Input #1,lin$
        msg = msg & lin$ & crlf
    Next x
    MsgBox "The first 5 lines of '" & file$ & "' are:" & crlf & crlf & msg
End Sub
```

See Also

Open (statement); Get (statement); Input# (statement); Input, Input$ (functions).

topic

Line Numbers

Line numbers are not supported by Delrina Basic.

As an alternative to line numbers, you can use meaningful labels as targets for absolute jumps, as shown below:

```
Sub Main()
    Dim i As Integer
    On Error Goto MyErrorTrap
    i = 0
LoopTop:
    i = i + 1
    If i < 10 Then Goto LoopTop
MyErrorTrap:
    MsgBox "An error occurred."
End Sub
```

function

Line$

Syntax

```
Line$(text$,first[,last])
```

Description

Returns a String containing a single line or a group of lines between first and last.

Comments

- Lines are delimited by carriage return, line feed, or carriage-return/line-feed pairs.
- The Line$ function takes the following parameters:

Parameter	Description
text$	String containing the text from which the lines will be extracted.
first	Integer representing the index of the first line to return. If last is omitted, then this line will be returned. If first is greater than the number of lines in text$, then a zero-length string is returned.
last	Integer representing the index of the last line to return.

Example

```
'This example reads five lines of the autoexec.bat file, extracts the
'third and fourth lines with the Line$ function, and displays them in a
'dialog box.

Const crlf = Chr$(13) + Chr$(10)
Sub Main()
    file$ = "c:\autoexec.bat"
    Open file$ For Input As #1
    For x = 1 To 5
        Line Input #1,lin$
        txt = txt & lin$ & crlf
    Next x
    lines$ = Line$(txt,3,4)
    MsgBox "The 3rd and 4th lines of "' & file$ & "' are:" & crlf & crlf & lines$
End Sub
```

See Also

Item$ (function); ItemCount (function); LineCount (function); Word$ (function); WordCount (function).

function

LineCount

Syntax

```
LineCount(text$)
```

Description

Returns an Integer representing the number of lines in text$.

Comments

- Lines are delimited by carriage return, line feed, or both.

Example

'This example reads your autoexec.bat file into a variable and then 'determines how many lines it is comprised of.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    file$ = "c:\autoexec.bat"
    Open file$ For Input As #1
    Do Until Eof(1)
        Line Input #1,lin$
        txt = txt & lin$ & crlf
    Loop
    lines! = LineCount(txt)
    MsgBox "" & file$ & "' is " & lines! & " lines long!" & crlf & crlf & txt
End Sub
```

See Also

Item$ (function); ItemCount (function); Line$ (function); Word$ (function); WordCount (function).

statement

ListBox

Syntax

```
ListBox X,Y,width,height,ArrayVariable,.Identifier
```

Description

Creates a list box within a dialog box template.

Comments

- When the dialog box is invoked, the list box will be filled with the elements contained in ArrayVariable.
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The ListBox statement requires the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
ArrayVariable	Specifies a single-dimensioned array of strings used to initialize the elements of the list box. If this array has no dimensions, then the list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax: DialogVariable.Identifier

Example

```
'This example creates a dialog box with two list boxes, one
'containing files and the other containing directories.

Sub Main()
    Dim files() As String
    Dim dirs() As String
    Begin Dialog ListBoxTemplate 16,32,184,96,"Sample"
        Text 8,4,24,8,"&Files:"
        ListBox 8,16,60,72,files$,.Files
        Text 76,4,21,8,"&Dirs:"
        ListBox 76,16,56,72,dirs$,.Dirs
        OKButton 140,4,40,14
        CancelButton 140,24,40,14
    End Dialog
    FileList files
    FileDirs dirs

    Dim ListBoxDialog As ListBoxTemplate
    rc% = Dialog(ListBoxDialog)
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

function

ListBoxEnabled

Syntax

```
ListBoxEnabled(name$ | id)
```

Description

Returns True if the given list box is enabled within the active window or dialog box; returns False otherwise.

Comments

- This function is used to determine whether a list box is enabled within the current window or dialog box. If there is no active window, False will be returned.
- The ListBoxEnabled function takes the following parameters:

Parameter	Description
name$	String containing the name of the list box. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
id	Integer specifying the ID of the list box.

Note: The ListBoxEnabled function is used to determine whether a list box is enabled in another application's dialog box. Use the DlgEnable function in dynamic dialog boxes.

Example

'This code fragment checks to see whether the list box is enabled 'before setting the focus to it.

If **ListBoxEnabled(**"Files:"**)** Then ActivateControl "Files:"

See Also

GetListBoxItem$ (function); GetListBoxItemCount (function); ListBoxExists (function); SelectListBoxItem (statement).

function

ListBoxExists

Syntax

```
ListBoxExists(name$ | id)
```

Description

Returns True if the given list box exists within the active window or dialog box; returns False otherwise.

Comments

- This function is used to determine whether a list box exists within the current window or dialog box. If there is no active window, False will be returned.
- The ListBoxExists function takes the following parameters:

Parameter	Description
name$	String containing the name of the list box. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
id	Integer specifying the ID of the list box.

Note: The ListBoxExists function is used to determine whether a list box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

'This code fragment checks to see whether the list box exists and is 'enabled before setting the focus to it.

```
If ListBoxExists("Files:") Then
    If ListBoxEnabled("Files:") Then
        ActivateControl "Files:"
    End If
End If
```

See Also

GetListBoxItem$ (function); GetListBoxItemCount (function); ListBoxEnabled (function); SelectListBoxItem (statement).

topic

Literals

Literals are values of a specific type. The following table shows the different types of literals supported by Delrina Basic:

Literal	Description
10	Integer whose value is 10.
43265	Long whose value is 43,265.
5#	Double whose value is 5.0. A number's type can be explicitly set using any of the following type-declaration characters:
%	Integer
&	Long
#	Double
!	Single
5.5	Double whose value is 5.5. Any number with decimal point is considered a double.
5.4E100	Double expressed in scientific notation.
&HFF	Integer expressed in hexadecimal.
&O47	Integer expressed in octal.
&HFF#	Double expressed in hexadecimal.
"hello"	String of five characters: hello.
"""hello"""	String of seven characters: "hello". Quotation marks can be embedded within strings by using two consecutive quotation marks.
#1/1/1994#	Date value whose internal representation is 34335.0. Any valid date can appear with #'s. Date literals are interpreted at execution time using the locale settings of the host environment. To ensure that date literals are correctly interpreted for all locales, use the international date format: #YYYY-MM-DD HH:MM:SS#

Constant Folding

Delrina Basic supports constant folding where constant expressions are calculated by the compiler at compile time. For example, the expression

```
i% = 10 + 12
```

is the same as:

```
i% = 22
```

Similarly, with strings, the expression

```
s$ = "Hello," + " there" + Chr(46)
```

is the same as:

```
s$ = "Hello, there."
```

function

Loc

Syntax

```
Loc(filenumber)
```

Description

Returns a Long representing the position of the file pointer in the given file.

Comments

- The filenumber parameter is an Integer used by Delrina Basic to refer to the number passed by the Open statement to Delrina Basic.
- The Loc function returns different values depending on the mode in which the file was opened:

File Mode	Returns
Input	Current byte position divided by 128
Output	Current byte position divided by 128
Append	Current byte position divided by 128
Binary	Position of the last byte read or written
Random	Number of the last record read or written

Example

```
'This example reads 5 lines of the autoexec.bat file, determines the
'current location of the file pointer, and displays it in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    file$ = "c:\autoexec.bat"
    Open file$ For Input As #1
    For x = 1 To 5
        If Not EOF(1) Then Line Input #1,lin$
    Next x
    lc% = Loc(1)
    Close
    MsgBox "The file byte location is: " & lc%
End Sub
```

See Also

Seek (function); Seek (statement); FileLen (function).

statement

Lock

Syntax

```
Lock [#] filenumber [,{record | [start] To end}]
```

Description

Locks a section of the specified file, preventing other processes from accessing that section of the file until the Unlock statement is issued.

Comments

- The Lock statement requires the following parameters:

Parameter	Description
filenumber	Integer used by Delrina Basic to refer to the open file-the number passed to the Open statement.
record	Long specifying which record to lock.
start	Long specifying the first record within a range to be locked.
end	Long specifying the last record within a range to be locked.

For sequential files, the record, start, and end parameters are ignored. The entire file is locked.

- The section of the file is specified using one of the following:

Syntax	Description
No parameters	Locks the entire file (no record specification is given).
record	Locks the specified record number (for Random files) or byte (for Binary files).
to end	Locks from the beginning of the file to the specified record (for Random files) or byte (for Binary files).
start to end	Locks the specified range of records (for Random files) or bytes (for Binary files).

- The lock range must be the same as that used to subsequently unlock the file range, and all locked ranges must be unlocked before the file is closed. Ranges within files are not unlocked automatically by Delrina Basic when your script terminates, which can cause file access problems for other processes. It is a good idea to group the Lock and Unlock statements close together in the code, both for readability and so subsequent readers can see that the lock and unlock are performed on the same range. This practice also reduces errors in file locks.

Example

```
'This example creates test.dat and fills it with ten string variable
'records. These are displayed in a dialog box. The file is then reopened
'for read/write, and each record is locked, modified, rewritten, and
'unlocked. The new records are then displayed in a dialog box.

Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
    a$ = "This is record number: "
    b$ = "0"
    rec$ = ""

    msg = ""
    Open "test.dat" For Random Access Write Shared As #1
    For x = 1 To 10
        rec$ = a$ & x
        Lock #1,x
        Put #1,,rec$
        Unlock #1,x
        msg = msg & rec$ & crlf
    Next x
    Close
    MsgBox "The records are:" & crlf & msg

    msg = ""
    Open "test.dat" For Random Access Read Write Shared As #1
    For x = 1 To 10
        rec$ = Mid(rec$,1,23) & (11 - x)
        Lock #1,x
        Put #1,x,rec$
        Unlock #1,x
        msg = msg & rec$ & crlf
    Next x
    MsgBox "The records are: " & crlf & msg
    Close

    Kill "test.dat"
End Sub
```

See Also

Unlock (statement); Open (statement).

function

Lof

Syntax

```
Lof(filenumber)
```

Description

Returns a Long representing the number of bytes in the given file.

Comments

- The filenumber parameter is an Integer used by Delrina Basic to refer to the open filethe number passed to the Open statement.
- The file must currently be open.

Example

```
'This example creates a test file, writes ten records into it,
'then finds the length of the file and displays it in a message box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "This is record number: "

    Open "test.dat" For Random Access Write Shared As #1
    For x = 1 To 10
        rec$ = a$ & x
        put #1,,rec$
        msg = msg & rec$ & crlf
    Next x
    Close

    Open "test.dat" For Random Access Read Write Shared As #1
    r% = Lof(1)
    Close
    MsgBox "The length of 'test.dat' is: " & r%
End Sub
```

See Also

Loc (function); Open (statement); FileLen (function).

function

Log

Syntax

```
Log(number)
```

Description

Returns a Double representing the natural logarithm of a given number.

Comments

- The value of number must be a Double greater than 0.
- The value of e is 2.71828.

Example

```
'This example calculates the natural log of 100 and displays it in
'a message box.

Sub Main()
    x# = Log(100)
    MsgBox "The natural logarithm of 100 is: " & x#
End Sub
```

See Also

Exp (function).

data type

Long

Syntax

Long

Description

Long variables are used to hold numbers (with up to ten digits of precision) within the following range:

```
-2,147,483,648 <= Long <= 2,147,483,647
```

Internally, longs are 4-byte values. Thus, when appearing within a structure, longs require 4 bytes of storage. When used with binary or random files, 4 bytes of storage are required.

The type-declaration character for Long is &.

See Also

Currency (data type); Date (data type); Double (data type); Integer (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CLng (function).

statement

LSet

Syntax 1

```
LSet dest = source
```

Syntax 2

```
LSet dest_variable = source_variable
```

Description

Left-aligns the source string in the destination string or copies one user-defined type to another.

Comments

Syntax 1

- The LSet statement copies the source string source into the destination string dest. The dest parameter must be the name of either a String or Variant variable. The source parameter is any expression convertible to a string.
- If source is shorter in length than dest, then the string is left-aligned within dest, and the remaining characters are padded with spaces. If source$ is longer in length than dest, then source is truncated, copying only the leftmost number of characters that will fit in dest.
- The destvariable parameter specifies a String or Variant variable. If destvariable is a Variant containing Empty, then no characters are copied. If destvariable is not convertible to a String, then a runtime error occurs. A runtime error results if destvariable is Null.

Syntax 2

- The source structure is copied byte for byte into the destination structure. This is useful for copying structures of different types. Only the number of bytes of the smaller of the two structures is copied. Neither the source structure nor the destination structure can contain strings.

Example

```
'This example replaces a 40-character string of asterisks (*) with
'an RSet and LSet string and then displays the result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim msg,tmpstr$
    tmpstr$ = String(40,"*")
    msg = "Here are two strings that have been right-" + crlf
    msg = msg & "and left-justified in a 40-character string."
    msg = msg & crlf & crlf
    RSet tmpstr$ = "Right|"
    msg = msg & tmpstr$ & crlf
    LSet tmpstr$ = "|Left"
    msg = msg & tmpstr$ & crlf
    MsgBox msg
End Sub
```

See Also

RSet (function).

function

LTrim, LTrim$

Syntax

```
LTrim[$](text)
```

Description

Returns text with the leading spaces removed.

Comments

- LTrim$ returns a String, whereas LTrim returns a String variant.
- Null is returned if text is Null.

Example

```
'This example displays a right-justified string and its LTrim result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    txt$ = "        This is text                "
    tr$ = LTrim(txt$)
    MsgBox "Original ->" & txt$ & "<-" & crlf & "Left Trimmed ->" & tr$ & "<-"
End Sub
```

See Also

RTrim, RTrim$ (functions); Trim, Trim$ (functions).

statement

Main

Syntax

```
Sub Main()
End Sub
```

Description

Defines the subroutine where execution begins.

Example

```
Sub Main()
    MsgBox "This is the Main() subroutine and entry point."
End Sub
```

function

Mci

Syntax

```
Mci(command$,result$ [,error$])
```

Description

Runs an Mci command, returning an Integer indicating whether the command was successful.

Comments

- The Mci function takes the following parameters:

Parameter	Description
command$	String containing the command to be run.
result$	String variable into which the result is placed. If the command doesn't return anything, then a zero-length string is returned. To ignore the returned string, pass a zero-length string: r% = Mci("open chimes.wav type waveaudio","")
error$	Optional String variable into which an error string will be placed. A zero-length string will be returned if the function is successful.

Example

```
'This first example plays a wave file. The wave file is played to
'completion before execution can continue.

Sub Main()
    Dim result As String
    Dim ErrorMessage As String
    Dim Filename As String
    Dim rc As Integer

    'Establish name of file in the Windows directory.
    Filename = FileParse$(System.WindowsDirectory$ + "\" + "chimes.wav")

    'Open the file and driver.
    rc = Mci("open " & Filename & " type waveaudio alias CoolSound","",ErrorMessage)
    If (rc) Then
        'Error occurred--display error message to user.
        MsgBox ErrorMessage
        Exit Sub
    End If

    rc = Mci("play CoolSound wait","","")          'Wait for sound to finish.
    rc = Mci("close CoolSound","","")              'Close driver and file.
End Sub
```

Example

```
'This next example shows how to query an Mci device and play an MIDI file in
'the background.

Sub Main()
    Dim result As String
    Dim ErrMsg As String
    Dim Filename As String
    Dim rc As Integer
```

```
    'Check to see whether MIDI device can play for us.
    rc = Mci("capability sequencer can play",result,ErrorMessage)

    'Check for error.
    If rc Then
        MsgBox ErrorMessage
        Exit Sub
    End If

    'Can it play?
    If result <> "true" Then
        MsgBox "MIDI device is not capable of playing."
        Exit Sub
    End If

    'Assemble a filename from the Windows directory.
    Filename = FileParse$(System.WindowsDirectory$ & "\" & "canyon.mid")

    'Open the driver and file.
    rc = Mci("open " & Filename & " type sequencer alias song",result$,ErrMsg)
    If rc Then
        MsgBox ErrMsg
        Exit Sub
    End If

    rc = Mci("play song","","")          'Play in the background.
    MsgBox "Press OK to stop the music.",ebOKOnly
    rc = Mci("close song","","")
End Sub
```

See Also

Beep (statement).

Note

- The Mci function accepts any Mci command as defined in the Multimedia Programmers Reference in the Windows 3.1 SDK.

statement

Menu

Syntax

```
Menu MenuItem$
```

Description

Issues the specified menu command from the active window of the active application.

Comments

- The MenuItem$ parameter specifies the complete menu item name, with each menu level being separated by a period. For example, the "Open" command on the "File" menu is represented by "File.Open". Cascading menu items may have multiple periods, one for each pop-up menu, such as "File.Layout.Vertical". Menu items can also be specified using numeric index values. For example, to select the third menu item from the File menu, use "File.#3". To select the fourth item from the third menu, use "#3.#4".
- Items from an application's system menu can be selected by beginning the menu item specification with a period, such as ".Restore" or ".Minimize".
- A runtime error will result if the menu item specification does not specify a menu item. For example, "File" specifies a menu pop-up rather than a menu item, and "File.Blah Blah" is not a valid menu item.
- When comparing menu item names, this statement removes periods (.), spaces, and the ampersand. Furthermore, all characters after a backspace or tab are removed. Thus, the menu item "&Open...\aCtrl+F12" translates simply to "Open".
- A runtime error is generated if the menu item cannot be found or is not enabled at the time that this statement is encountered.

Examples

```
Sub Main()
    Menu "File.Open"
    Menu "Format.Character.Bold"
    Menu ".Restore"                    'Command from system menu
    Menu "File.#2"
End Sub
```

See Also

MenuItemChecked (function); MenuItemEnabled (function); MenuItemExists (function).

function

MenuItemChecked

Syntax

```
MenuItemChecked(MenuItemName$)
```

Description

Returns True if the given menu item exists and is checked; returns False otherwise.

Comments

- The MenuItemName$ parameter specifies a complete menu item or menu item pop-up following the same format as that used by the Menu statement.

Example

```
'This example turns the ruler off if it is on.

Sub Main()
    If MenuItemChecked("View.Ruler") Then Menu "View.Ruler"
End Sub
```

See Also

Menu (statement); MenuItemEnabled (function); MenuItemExists (function).

function

MenuItemEnabled

Syntax

```
MenuItemEnabled(MenuItemName$)
```

Description

Returns True if the given menu item exists and is enabled; returns False otherwise.

Comments

- The MenuItemName$ parameter specifies a complete menu item or menu item pop-up following the same format as that used by the Menu statement.

Example

```
'This example only pastes if there is something in the Clipboard.

Sub Main()
    If MenuItemEnabled("Edit.Paste") Then
        Menu "Edit.Paste"
    Else
        MsgBox "There is nothing in the Clipboard.",ebOKOnly
    End If
End Sub
```

See Also

Menu (statement); MenuItemChecked (function); MenuItemExists (function).

function

MenuItemExists

Syntax

```
MenuItemExists(MenuItemName$)
```

Description

Returns True if the given menu item exists; returns False otherwise.

Comments

- The MenuItemName$ parameter specifies a complete menu item or menu item pop-up following the same format as that used by the Menu statement.

Examples

```
Sub Main()
    If MenuItemExists("File.Open") Then Beep
    If MenuItemExists("File") Then MsgBox "There is a File menu."
End Sub
```

See Also

Menu (statement); MenuItemChecked (function); MenuItemEnabled (function).

function

Mid, Mid$

Syntax

```
Mid[$](text,start [,length])
```

Description

Returns a substring of the specified string, beginning with start, for length characters.

Comments

- The returned substring starts at character position start and will be length characters long.
- Mid$ returns a String, whereas Mid returns a String variant.
- The Mid/Mid$ functions take the following parameters:

Parameter	Description
text	Any String expression containing the text from which characters are returned.
start	Integer specifying the character position where the substring begins. If start is greater than the length of text$, then a zero-length string is returned.
length	Integer specifying the number of characters to return. If this parameter is omitted, then the entire string is returned, starting at start.

- The Mid function will return Null text is Null.

Example

'This example extracts the left and right halves of a string using 'the Mid functions and displays the text with a message spliced in 'the middle.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "DAVE is a good programmer"
    l$ = Mid(a$,1,7)
    r$ = Mid(a$,16,10)
    MsgBox l$ & " an excellent " & r$
End Sub
```

See Also

InStr (function); Option Compare (statement); Mid, Mid$ (statements).

statement

Mid, Mid$

Syntax

```
Mid[$](variable,start[,length]) = newvalue
```

Description

Replaces one part of a string with another.

Comments

- The Mid/Mid$ statements take the following parameters:

Parameter	Description
variable	String or Variant variable to be changed.
start	Integer specifying the character position within variable where replacement begins. If start is greater than the length of variable, then variable remains unchanged.
length	Integer specifying the number of characters to change. If this parameter is omitted, then the entire string is changed, starting at start.
newvalue	Expression used as the replacement. This expression must be convertible to a String.

- The resultant string is never longer than the original length of variable.
- With Mid, variable must be a Variant variable convertible to a String, and newvalue is any expression convertible to a string. A runtime error is generated if either variant is Null.

Example

```
'This example displays a substring from the middle of a string
'variable using the Mid$ function, replacing the first four characters
'with "NEW " using the Mid$ statement.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "This is the Main string containing text."
    b$ = Mid(a$,14,Len(a$))
    Mid(b$,1) = "NEW"
    MsgBox a$ & crlf & b$
End Sub
```

See Also

Mid, Mid$ (functions); Option Compare (statement).

function

Minute

Syntax

```
Minute(time)
```

Description

Returns the minute of the day encoded in the specified time parameter.

Comments

- The value returned is as an Integer between 0 and 59 inclusive.
- The time parameter is any expression that converts to a Date.

Example

```
'This example takes the current time; extracts the hour, minute,
'and second; and displays them as the current time.

Sub Main()
    Msgbox "It is now minute " & Minute(Time) & " of the hour."
End Sub
```

See Also

Day (function); Second (function); Month (function); Year (function); Hour (function); Weekday (function); DatePart (function).

function

MIRR

Syntax

```
MIRR(ValueArray(),FinanceRate,ReinvestRate)
```

Description

Returns a Double representing the modified internal rate of return for a series of periodic payments and receipts.

Comments

- The modified internal rate of return is the equivalent rate of return on an investment in which payments and receipts are financed at different rates. The interest cost of investment and the rate of interest received on the returns on investment are both factors in the calculations.
- The MIRR function requires the following parameters:

Parameter	Description
ValueArray()	Array of Double numbers representing the payments and receipts. Positive values are payments (invested capital), and negative values are receipts (returns on investment). There must be at least one positive (investment) value and one negative (return) value.
FinanceRate	Double representing the interest rate paid on invested monies (paid out).
ReinvestRate	Double representing the rate of interest received on incomes from the investment (receipts).

- FinanceRate and ReinvestRate should be expressed as percentages. For example, 11 percent should be expressed as 0.11.
- To return the correct value, be sure to order your payments and receipts in the correct sequence.

Example

```
'This example illustrates the purchase of a lemonade stand for $800
'financed with money borrowed at 10%. The returns are estimated to
'accelerate as the stand gains popularity. The proceeds are placed
'in a bank at 9 percent interest. The incomes are estimated (generated)
'over 12 months. This program first generates the income stream array
'in two For...Next loops, and then the modified internal rate of return is
'calculated and displayed. Notice that the annual rates are normalized
'to monthly rates by dividing them by 12.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim valu#(12)
    valu(1) = -800                                  'Initial investment
    msg = valu(1) & ", "
    For x = 2 To 5
        valu(x) = 100 + (x * 2)                     'Incomes months 2-5
        msg = msg & valu(x) & ", "
    Next x
    For x = 6 To 12
        valu(x) = 100 + (x * 10)                    'Incomes months 6-12
        msg = msg & valu(x) & ", "
    Next x
    retrn# = MIRR(valu,.1/12,.09/12)                'Note: normalized annual rates

    msg = "The values: " & crlf & msg & crlf & crlf
    MsgBox msg & "Modified rate: " & Format(retrn#,"Percent")
End Sub
```

See Also

Fv (function); IRR (function); Npv (function); Pv (function).

statement

MkDir

Syntax

```
MkDir dir$
```

Description

Creates a new directory as specified by dir$.

Example

'This example creates a new directory on the default drive. If
'this causes an error, then the error is displayed and the program 'terminates. If no error is generated, the directory is removed with 'the RmDir statement.

```
Sub Main()
    On Error Resume Next
    MkDir "testdir"
    If Err <> 0 Then
        MsgBox "The following error occurred: " & Error(Err)
    Else
        MsgBox "Directory 'testdir' was created and is about to be removed."
        RmDir "testdir"
    End If
End Sub
```

See Also

ChDir (statement); ChDrive (statement); CurDir, CurDir$ (functions); Dir, Dir$ (functions); RmDir (statement).

Notes

This command behaves the same as the DOS "mkdir" command.

operator

Mod

Syntax

```
expression1 Mod expression2
```

Description

Returns the remainder of expression1 / expression2 as a whole number.

Comments

- If both expressions are integers, then the result is an integer. Otherwise, each expression is converted to a Long before performing the operation, returning a Long.
- A runtime error occurs if the result overflows the range of a Long.
- If either expression is Null, then Null is returned. Empty is treated as 0.

Example

```
'This example uses the Mod operator to determine the value of a randomly
'selected card where card 1 is the ace (1) of clubs and card 52 is the
'king (13) of spades. Since the values recur in a sequence of 13 cards
'within 4 suits, we can use the Mod function to determine the value of
'any given card number.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    cval$ = "Ace,Two,Three,Four,Five,Six,Seven,Eight,Nine,Ten,Jack,Queen,King"
    Randomize
    card% = Random(1,52)
    value = card% Mod 13
    If value = 0 Then value = 13
    CardNum$ = Item$(cval,value)
    If card% < 53 Then suit$ = "Spades"
    If card% < 40 Then suit$ = "Hearts"
    If card% < 27 Then suit$ = "Diamonds"
    If card% < 14 Then suit$ = "Clubs"
    msg = "Card number " & card% & " is the "
    msg = msg & CardNum & " of " & suit$
    MsgBox msg
End Sub
```

See Also

/ (operator); \ (operator).

function

Month

Syntax

```
Month(date)
```

Description

Returns the month of the date encoded in the specified date parameter.

Comments

- The value returned is as an Integer between 1 and 12 inclusive.
- The date parameter is any expression that converts to a Date.

Example

```
'This example returns the current month in a dialog box.

Sub Main()
    mons$ = "Jan.,Feb.,Mar.,Apr.,May,Jun.,Jul.,Aug.,Sep.,Oct.,Nov.,Dec."
    tdate$ = Date$
    tmonth! = Month(DateValue(tdate$))
    MsgBox "The current month is: " & Item$(mons$,tmonth!)
End Sub
```

See Also

Day (function); Minute (function); Second (function); Year (function); Hour (function); Weekday (function); DatePart (function).

function

MsgBox

Syntax

```
MsgBox(msg [,[type] [,title]])
```

Description

Displays a message in a dialog box with a set of predefined buttons, returning an Integer representing which button was selected.

Comments

- The MsgBox function takes the following parameters:

Parameter	Description
msg	Message to be displayed-any expression convertible to a String.
	End-of-lines can be used to separate lines (either a carriage return, line feed, or both). If a given line is too long, it will be word-wrapped. If msg contains character 0, then only the characters up to the character 0 will be displayed.
	The width and height of the dialog box are sized to hold the entire contents of msg.
	A runtime error is generated if msg is Null.
type	Integer specifying the type of dialog box (see below).
title	Caption of the dialog box. This parameter is any expression convertible to a String. If it is omitted, then Delrina Basic is used.
	A runtime error is generated if title is Null.

- The MsgBox function returns one of the following values:

Constant	Value	Description
ebOK	1	OK was clicked.
ebCancel	2	Cancel was clicked.
ebAbort	3	Abort was clicked.
ebRetry	4	Retry was clicked.
ebIgnore	5	Ignore was clicked.
ebYes	6	Yes was clicked.
ebNo	7	No was clicked.

- The type parameter is the sub of any of the following values:

Constant	Value	Description
ebOKOnly	0	Displays OK button only.
ebOKCancel	1	Displays OK and Cancel buttons.
ebAbortRetryIgnore	2	Displays Abort, Retry, and Ignore buttons.
ebYesNoCancel	3	Displays Yes, No, and Cancel buttons.
ebYesNo	4	Displays Yes and No buttons.
ebRetryCancel	5	Displays Retry and Cancel buttons.
ebCritical	16	Displays "stop" icon.
ebQuestion	32	Displays "question mark" icon.
ebExclamation	48	Displays "exclamation point" icon.

ebInformation	64	Displays "information" icon.
ebDefaultButton1	0	First button is the default button.
ebDefaultButton2	256	Second button is the default button.
ebDefaultButton3	512	Third button is the default button.
ebApplicationModal	0	Application modal-the current application is suspended until the dialog box is closed.
ebSystemModal	4096	System modal-all applications are suspended until the dialog box is closed.

- The default value for type is 0 (display only the OK button, making it the default).

Breaking Text across Lines

The msg parameter can contain end-of-line characters, forcing the text that follows to start on a new line. The following example shows how to display a string on two lines:

```
MsgBox "This is on" + Chr(13) + Chr(10) + "two lines."
```

The carriage-return or line-feed characters can be used by themselves to designate an end-of-line.

```
r = MsgBox("Hello, World")
r = MsgBox("Hello, World",ebYesNoCancel Or ebDefaultButton1)
r = MsgBox("Hello, World",ebYesNoCancel Or ebDefaultButton1 Or ebCritical)
```

Example

```
Sub Main()
    MsgBox "This is a simple message box."
    MsgBox "This is a message box with a title and an icon.",ebExclamation,"Simple"
    MsgBox "This message box has OK and Cancel buttons.",ebOkCancel,"MsgBox"
    MsgBox "This message box has Abort, Retry, and Ignore buttons.",_
        ebAbortRetryIgnore,"MsgBox"
    MsgBox "This message box has Yes, No, and Cancel buttons.",_
        ebYesNoCancel Or ebDefaultButton2,"MsgBox"
    MsgBox "This message box has Yes and No buttons.",ebYesNo,"MsgBox"
    MsgBox "This message box has Retry and Cancel buttons.",ebRetryCancel,"MsgBox"
    MsgBox "This message box is system modal!",ebSystemModal
End Sub
```

See Also

AskBox$ (function); AskPassword$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SaveFilename$ (function); SelectBox (function); AnswerBox (function).

Note

- MsgBox displays all text in its dialog box in 8-point MS Sans Serif.

statement

MsgBox

Syntax

```
MsgBox msg [,[type] [,title]]
```

Description

This command is the same as the MsgBox function, except that the statement form does not return a value. See MsgBox (function).

Example

```
Sub Main()
    MsgBox "This is text displayed in a message box."      'Display text.
    MsgBox "The result is: " & (10 * 45) 'Display a number.
End Sub
```

See Also

AskBox$ (function); AskPassword$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SaveFilename$ (function); SelectBox (function); AnswerBox (function).

statement

Name

Syntax

```
Name oldfile$ As newfile$
```

Description

Renames a file.

Comments

- Each parameter must specify a single filename. Wildcard characters such as * and ? are not allowed.
- Some platforms let naming of files to different directories on the same physical disk volume. For example, the following rename will work under Windows:

  Name "c:\samples\mydoc.txt" **As** "c:\backup\doc\mydoc.bak"
- You cannot rename files across physical disk volumes. For example, the following will error under Windows:

  Name "c:\samples\mydoc.txt" **As** "a:\mydoc.bak" 'This will error!
- To rename a file to a different physical disk, you must first copy the file, then erase the original:

  ```
  FileCopy "c:\samples\mydoc.txt","a:\mydoc.bak"   'Make a copy
  Kill "c:\samples\mydoc.txt"                      'Delete the original
  ```

Example

```
'This example creates a file called test.dat and then renames
'it to test2.dat.

Sub Main()
    oldfile$ = "test.dat"
    newfile$ = "test2.dat"

    On Error Resume Next
    If FileExists(oldfile$) Then
        Name oldfile$ As newfile$
        If Err <> 0 Then
            msg = "The following error occured: " & Error(Err)
        Else
            msg = """ & oldfile$ & "' was renamed to "' & newfile$ & """
        End If
    Else
        Open oldfile$ For Output As #1
        Close
        Name oldfile$ As newfile$
        If Err <> 0 Then
            msg = """ & oldfile$ & "' not created. The following error occured: " & Error(Err)
        Else
            msg = """ & oldfile$ & "' was created and renamed to "' &   newfile$ & """
        End If
    End If
    MsgBox msg
End Sub
```

See Also

Kill (statement), FileCopy (statement).

method

Net.AddCon

Syntax

```
Net.AddCon netpath$,password$,localname$
```

Description

Redirects a local device (a disk drive or printer queue) to the specified shared device or remote server.

Comments

- The Net.AddCon method takes the following parameters:

Parameter	Description
netpath$	String containing the name of the shared device or the name of a remote server. This parameter can contain the name of a shared printer queue (such as that returned by Net.Browse[1]) or the name of a network path (such as that returned by Net.Browse[0]).
password$	String containing the password for the given device or server. This parameter is mainly used to specify the password on a remote server.
localname$	String containing the name of the local device being redirected, such as "LPT1" or "D:".

- A runtime error will result if no network is present.

Example

```
'This example sets N: so that it refers to the network path SYS:\PUBLIC.

Sub Main()
    Net.AddCon "SYS:\PUBLIC","","N:"
End Sub
```

See Also

Net.CancelCon (method); Net.GetCon$ (method).

method

Net.Browse$

Syntax

```
Net.Browse$(type)
```

Description

Calls the currently installed network's browse dialog box, requesting a particular type of information.

Comments

- The type parameter is an Integer specifying the type of dialog box to display:

Type	Description
0	If type is 0, then this method displays a dialog box that lets the user to browse network volumes and directories. Choosing OK returns the completed pathname as a String.
1	If type is 1, then this function displays a dialog box that lets the user to browse the network's printer queues. Choosing OK returns the complete name of that printer queue as a String. This string is the same format as required by the Net.AddCon method.

- This dialog box differs depending on the type of network installed.
- A runtime error will result if no network is present.

Example

```
'This second example retrieves a valid network path.

Sub Main()
    s$ = Net.Browse$(0)
    If s$ <> "" Then
        MsgBox "The following network path was selected: " & s$
    Else
        MsgBox "Dialog box was canceled."
    End If
End Sub
```

See Also

Net.Dialog (method).

method

Net.CancelCon

Syntax

```
Net.CancelCon connection$ [,isForce]
```

Description

Cancels a network connection.

Comments

- The Net.CancelCon method takes the following parameters:

Parameter	Description
connection$	String containing the name of the device to cancel, such as "LPT1" or "D:".
isForce	Boolean specifying whether to force the cancellation of the connection if there are open files or open print jobs. If this parameter is True, then this method will close all open files and open print jobs before the connection is closed. If this parameter is False, this the method will issue a runtime error if there are any open files or open print jobs.

- A runtime error will result if no network is present.

Example

'This example deletes the drive mapping associated with drive N:.

```
Sub Main()
    Net.CancelCon "N:"
End Sub
```

See Also

Net.AddCon (method); Net.GetCon$ (method).

method

Net.Dialog

Syntax

Net.Dialog

Description

Displays the dialog box that lets configuration of the currently installed network.

Comments

- The displayed dialog box depends on the currently installed network. The dialog box is modal-script execution will be paused until the dialog box is completed.
- A runtime error will result if no network is present.

Example

```
'This example invokes the network driver dialog box.

Sub Main()
    Net.Dialog
End Sub
```

See Also

Net.Browse$ (method).

method

Net.GetCaps

Syntax

```
Net.GetCaps(type)
```

Description

Returns an Integer specifying information about the network and its capabilities.

Comments

- The type parameter specifies what type of information to retrieve:

Value of type	Description
1	Returns the version of the driver specification to which the currently installed network driver conforms. The high byte of the returned value contains the major version number and the low byte contains the minor version number. These values can be retrieved using the following code: MajorVersionNumber = Net.GetCaps(1) \ 256 MinorVersionNumber = Net.GetCaps(1) And &H00FF
2	Returns the type of network. The network type is returned in the high byte and the sub-network type is returned in the low byte. These values can be obtained using the following code: NetType = Net.GetCaps(2) \ 256 SubNetType = Net.GetCaps(2) And &H00FF Using the above values, NetType can be any of the following values: 0 No network is installed. 1 Microsoft Network. 2 Microsoft LAN Manager. 3 Novell NetWare. 4 Banyan Vines. 5 10Net. 6 Locus. 7 SunSoft PC NFS. 8 LanStep. 9 9 Titles. 10 Articom Lantastic. 11 IBM AS/400. 12 FTP Software FTP NFS. 13 DEC Pathworks. If NetType is is 128, then SubNetType is any of the following values (you can test for any of these values using the And operator): 0 None. bit &H0001 Microsoft Network. bit &H0002 Microsoft LAN Manager. bit &H0004 Windows for Workgroups. bit &H0008 Novell NetWare. bit &H0010 Banyan Vines. bit &H0080 Other unspecified network.
3	Returns the network driver version number.
4	Returns 1 if the Net.User$ property is supported, 0 otherwise.
6	Returns any of the following values indicating which connections are supported (you can test for these values using the And operator): bit &H0001 Driver supports Net.AddCon.

bit &H0002 Driver supports Net.CancelCon.
bit &H0004 Driver supports Net.GetCon.
bit &H0008 Driver supports auto connect.
bit &H0010 Driver supports Net.Browse$.

7 Returns a value indicating which printer function are available (you can test for these values using the And operator):

bit &H0002 Driver supports open print job.
bit &H0004 Driver supports close print job.
bit &H0010 Driver supports hold print job.
bit &H0020 Driver supports release print job.
bit &H0040 Driver supports cancel print job.
bit &H0080 Driver supports setting the number of print copies.
bit &H0100 Driver supports watch print queue.
bit &H0200 Driver supports unwatch print queue.
bit &H0400 Driver supports locking queue data.
bit &H0800 Driver supports unlocking queue data.
bit &H1000 Driver supports queue change message.
bit &H2000 Driver supports abort print job.
bit &H4000 Driver supports no arbitrary lock.
bit &H8000 Driver supports write print job.

8 Returns a value indicating which dialog functions are available (you can test for these values using the And operator):

bit &H0001 Driver supports device mode dialog.
bit &H0002 Driver supports the Browse dialog.
bit &H0004 Driver supports the Connect dialog.
bit &H0008 Driver supports the Disconnect dialog.
bit &H0010 Driver supports the View Queue dialog.
bit &H0020 Driver supports the Property dialog.
bit &H0040 Driver supports the Connection dialog.
bit &H0080 Driver supports the Printer Connect dialog.
bit &H0100 Driver supports the Shares dialog.
bit &H0200 Driver supports the Share As dialog.

- A runtime error will result if no network is present.

Examples

```
Sub Main()
    'This example checks the type of network.
    If Net.GetCaps(2) = 768 Then MsgBox "This is a Novell network."

'This checks whether the net supports retrieval of the user name.
If Net.GetCaps(4) And 1 Then MsgBox "User name is: " + Net.User$

'This checks whether this net supports the Browse dialog boxes.
If Net.GetCaps(6) And &H0010 Then MsgBox Net.Browse$(1)
End Sub
```

method

Net.GetCon$

Syntax

```
Net.GetCon$(localname$)
```

Description

Returns the name of the network resource associated with the specified redirected local device.

Comments

- The localname$ parameter specifies the name of the local device, such as "LPT1" or "D:".
- The function returns a zero-length string if the specified local device is not redirected.
- A runtime error will result if no network is present.

Example

```
'This example finds out where drive Z is mapped.

Sub Main()
    NetPath$ = Net.GetCon$("Z:")
    MsgBox "Drive Z is mapped as " & NetPath$
End Sub
```

See Also

Net.CancelCon (method); Net.AddCon (method).

property

Net.User$

Syntax

Net.User$

Description

Returns the name of the user on the network.

Comments

- A runtime error is generated if the network is not installed.

Examples

```
Sub Main()
    'This example tells the user who he or she is.
    MsgBox "Your net user name is: " & Net.User$

    'This example makes sure this capability is supported.
    If Net.GetCaps(4) And 1 Then MsgBox "You are " & Net.User$
End Sub
```

keyword

New

Syntax 1

```
Dim ObjectVariable As New ObjectType
```

Syntax 2

```
Set ObjectVariable = New ObjectType
```

Description

Creates a new instance of the specified object type, assigning it to the specified object variable.

Comments

- The New keyword is used to declare a new instance of the specified data object. This keyword can only be used with data object types.
- At runtime, the application or extension that defines that object type is notified that a new object is being defined. The application responds by creating a new physical object (within the appropriate context) and returning a reference to that object, which is immediately assigned to the variable being declared.
- When that variable goes out of scope (i.e., the Sub or Function procedure in which the variable is declared ends), the application is notified. The application then performs some appropriate action, such as destroying the physical object.

See Also

Dim (statement); Set (statement).

operator

Not

Syntax

```
Not expression
```

Description

Returns either a logical or binary negation of expression.

Comments

- The result is determined as shown in the following table:

If the Expression Is	Then the Result Is
True	False
False	True
Null	Null
Any numeric type	A binary negation of the number. If the number is an Integer, then an Integer is returned. Otherwise, the expression is first converted to a Long, then a binary negation is performed, returning a Long.
Empty	Treated as a Long value 0.

Example

```
'This example demonstrates the use of the Not operator in comparing
'logical expressions and for switching a True/False toggle variable.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a = False
    b = True
    If (Not a and b) Then msg = "a = False, b = True" & crlf

    toggle% = True
    msg = msg & "toggle% is now " & CBool(toggle%) & crlf
    toggle% = Not toggle%
    msg = msg & "toggle% is now " & CBool(toggle%) & crlf
    toggle% = Not toggle%
    msg = msg & "toggle% is now " & CBool(toggle%)
    MsgBox msg
End Sub
```

See Also

Boolean (data type); Comparison Operators (topic).

constant

Nothing

Description

A value indicating that an object variable no longer references a valid object.

Example

```
Sub Main()
    Dim a As Object
    If a Is Nothing Then
        MsgBox "The object variable references no object."
    Else
        MsgBox "The object variable references: " & a.Value
    End If
End Sub
```

See Also

Set (statement); Object (data type).

function

Now

Syntax

Now[()]

Description

Returns a Date variant representing the current date and time.

Example

```
'This example shows how the Now function can be used as an elapsed-
'time counter.

Sub Main()
    t1# = Now
    MsgBox "Wait a while and click OK."
    t2# = Now
    t3# = Second(t2#) - Second(t1#)
    MsgBox "Elapsed time was: " & t3# & " seconds."
End Sub
```

See Also

Date, Date$ (functions); Time, Time$ (functions).

function

NPer

Syntax

```
NPer(Rate,Pmt,Pv,Fv,Due)
```

Description

Returns the number of periods for an annuity based on periodic fixed payments and a constant rate of interest.

Comments

- An annuity is a series of fixed payments paid to or received from an investment over a period of time. Examples of annuities are mortgages, retirement plans, monthly savings plans, and term loans.
- The NPer function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. If the periods are monthly, be sure to normalize annual rates by dividing them by 12.
Pmt	Double representing the amount of each payment or income. Income is represented by positive values, whereas payments are represented by negative values.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, and the future value (see below) would be zero.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be zero, and the present value would be the amount of the loan.
Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.

- Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example

```
'This example calculates the number of $100.00 monthly payments necessary
'to accumulate $10,000.00 at an annual rate of 10%. Payments are made at the
'beginning of the month.

Sub Main()
    ag# = NPer((.10/12),100,0,10000,1)
    MsgBox "The number of monthly periods is: " & Format(ag#,"Standard")
End Sub
```

See Also

IPmt (function); Pmt (function); PPmt (function); Rate (function).

function

Npv

Syntax

```
Npv(Rate,ValueArray())
```

Description

Returns the net present value of an annuity based on periodic payments and receipts, and a discount rate.

Comments

- The Npv function requires the following parameters:

Parameter	Description
Rate	Double that represents the interest rate over the length of the period. If the values are monthly, annual rates must be divided by 12 to normalize them to monthly rates.
ValueArray()	Array of Double numbers representing the payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one negative value.

- Positive numbers represent cash received, whereas negative numbers represent cash paid out.
- For accurate results, be sure to enter your payments and receipts in the correct order because Npv uses the order of the array values to interpret the order of the payments and receipts.
- If your first cash flow occurs at the beginning of the first period, that value must be added to the return value of the Npv function. It should not be included in the array of cash flows.
- Npv differs from the Pv function in that the payments are due at the end of the period and the cash flows are variable. Pv's cash flows are constant, and payment may be made at either the beginning or end of the period.

Example

```
'This example illustrates the purchase of a lemonade stand for $800
'financed with money borrowed at 10%. The returns are estimated to
'accelerate as the stand gains popularity. The incomes are estimated
'(generated) over 12 months. This program first generates the income
'stream array in two For...Next loops, and then the net present value
'(Npv) is calculated and displayed. Note normalization of the annual 10%
'rate.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim valu#(12)
    valu(1) = -800                                  'Initial investment
    msg = valu(1) & ", "
    For x = 2 To 5                                  'Months 2-5
        valu(x) = 100 + (x * 2)
        msg = msg & valu(x) & ", "
    Next x
    For x = 6 To 12                                 'Months 6-12
        valu(x) = 100 + (x * 10)                    'Accelerated income
        msg = msg & valu(x) & ", "
    Next x
    NetVal# = NPV((.10/12),valu)
    msg = "The values:" & crlf & msg & crlf & crlf
    MsgBox msg & "Net present value: " & Format(NetVal#,"Currency")
End Sub
```

See Also

Fv (function); IRR (function); MIRR (function); Pv (function).

constant

Null

Description

Represents a variant of VarType 1.

Comments

- The Null value has special meaning indicating that a variable contains no data.
- Most numeric operators return Null when either of the arguments is Null. This "propagation" of Null makes it especially useful for returning error values through a complex expression. For example, you can write functions that return Null when an error occurs, then call this function within an expression. You can then use the IsNull function to test the final result to see whether an error occurred during calculation.
- Since variants are Empty by default, the only way for Null to appear within a variant is for you to explicitly place it there. Only a few Delrina Basic functions return this value.

Example

```
Sub Main()
    Dim a As Variant
    a = Null
    If IsNull(a) Then MsgBox "The variable is Null."
    MsgBox "The VarType of a is: " & VarType(a) 'Should display 1.
End Sub
```

data type

Object

Syntax

Object

Description

A data type used to declare OLE automation variables.

Comments

- The Object type is used to declare variables that reference objects within an application using OLE automation.
- Each object is a 4-byte (32-bit) value that references the object internally. The value 0 (or Nothing) indicates that the variable does not reference a valid object, as is the case when the object has not yet been given a value. Accessing properties or methods of such Object variables generates a runtime error.

Using Objects

Object variables are declared using the Dim, Public, or Private statement:

```
Dim MyApp As Object
```

Object variables can be assigned values (thereby referencing a real physical object) using the Set statement:

```
Set MyApp = CreateObject("phantom.application")
Set MyApp = Nothing
```

Properties of an Object are accessed using the dot (.) separator:

```
MyApp.Color = 10
i% = MyApp.Color
```

Methods of an Object are also accessed using the dot (.) separator:

```
MyApp.Open "sample.txt"
isSuccess = MyApp.Save("new.txt",15)
```

Automatic Destruction

Delrina Basic keeps track of the number of variables that reference a given object so that the object can be destroyed when there are no longer any references to it:

```
Sub Main()                                  'Number of references to object
    Dim a As Object                                 '0
    Dim b As Object                                 '0
    Set a = CreateObject("phantom.application)  '1
    Set b = a                                           '2
    Set a = Nothing                                 '1
End Sub                                             '0 (object destroyed)
```

Note: An OLE automation object is instructed by Delrina Basic to destroy itself when no variables reference that object. However, it is the responsibility of the OLE automation server to destroy it. Some servers do not destroy their objects-usually when the objects have a visual component and can be destroyed manually by the user.

See Also

Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement).

topic

Objects

Delrina Basic defines two types of objects: data objects and OLE automation objects.

Syntactically, these are referenced in the same way.

What Is an Object

An object in Delrina Basic is an encapsulation of data and routines into a single unit. The use of objects in Delrina Basic has the effect of grouping together a set of functions and data items that apply only to a specific object type.

Objects expose data items for programmability called properties. For example, a sheet object may expose an integer called NumColumns. Usually, properties can be both retrieved (get) and modified (set).

Objects also expose internal routines for programmability called methods. In Delrina Basic, an object method can take the form of a function or a subroutine. For example, a OLE automation object called MyApp may contain a method subroutine called Open that takes a single argument (a filename), as shown below:

```
MyApp.Open "c:\files\sample.txt"
```

Declaring Object Variables

In order to gain access to an object, you must first declare an object variable using either Dim, Public, or Private:

```
Dim o As Object        'OLE automation object
```

Initially, objects are given the value 0 (or Nothing). Before an object can be accessed, it must be associated with a physical object.

Assigning a Value to an Object Variable

An object variable must reference a real physical object before accessing any properties or methods of that object. To instantiate an object, use the Set statement.

```
Dim MyApp As Object
Set MyApp = CreateObject("Server.Application")
```

Accessing Object Properties

Once an object variable has been declared and associated with a physical object, it can be modified using Delrina Basic code. Properties are syntactically accessible using the dot operator, which separates an object name from the property being accessed:

```
MyApp.BackgroundColor = 10
i% = MyApp.DocumentCount
```

Properties are set using Delrina Basic's normal assignment statement:

```
MyApp.BackgroundColor = 10
```

Object properties can be retrieved and used within expressions:

```
i% = MyApp.DocumentCount + 10
MsgBox "Number of documents = " & MyApp.DocumentCount
```

Accessing Object Methods

Like properties, methods are accessed via the dot operator. Object methods that do not return values

behave like subroutines in Delrina Basic (i.e., the arguments are not enclosed within parentheses):

```
MyApp.Open "c:\files\sample.txt",True,15
```

Object methods that return a value behave like function calls in Delrina Basic. Any arguments must be enclosed in parentheses:

```
If MyApp.DocumentCount = 0 Then MsgBox "No open documents."
NumDocs = app.count(4,5)
```

There is no syntactic difference between calling a method function and retrieving a property value, as shown below:

```
variable = object.property(arg1,arg2)
variable = object.method(arg1,arg2)
```

Comparing Object Variables

The values used to represent objects are meaningless to the script in which they are used, with the following exceptions:

Objects can be compared to each other to determine whether they refer to the same object.

Objects can be compared with Nothing to determine whether the object variable refers to a valid object.

Object comparisons are accomplished using the Is operator:

```
If a Is b Then MsgBox "a and b are the same object."
If a Is Nothing Then MsgBox "a is not initialized."
If b Is Not Nothing Then MsgBox "b is in use."
```

Collections

A collection is a set of related object variables. Each element in the set is called a member and is accessed via an index, either numeric or text, as shown below:

```
MyApp.Toolbar.Buttons(0)
MyApp.Toolbar.Buttons("Tuesday")
```

It is typical for collection indexes to begin with 0.

Each element of a collection is itself an object, as shown in the following examples:

```
Dim MyToolbarButton As Object

Set MyToolbarButton = MyApp.Toolbar.Buttons("Save")
MyAppp.Toolbar.Buttons(1).Caption = "Open"
```

The collection itself contains properties that provide you with information about the collection and methods that allow navigation within that collection:

```
Dim MyToolbarButton As Object

NumButtons% = MyApp.Toolbar.Buttons.Count
MyApp.Toolbar.Buttons.MoveNext
MyApp.Toolbar.Buttons.FindNext "Save"

For i = 1 To MyApp.Toolbar.Buttons.Count
    Set MyToolbarButton = MyApp.Toolbar.Buttons(i)
    MyToolbarButton.Caption = "Copy"
Next i
```

Predefined Objects

Delrina Basic predefines a few objects for use in all scripts. These are:

Clipboard	System	Desktop	HWND
Net	Basic	Screen	

Note: Some of these objects are not available on all platforms.

function

Oct, Oct$

Syntax

```
Oct[$](number)
```

Description

Returns a String containing the octal equivalent of the specified number.

Comments

- Oct$ returns a String, whereas Oct returns a String variant.
- The returned string contains only the number of octal digits necessary to represent the number.
- The number parameter is any numeric expression. If this parameter is Null, then Null is returned. Empty is treated as 0. The number parameter is rounded to the nearest whole number before converting to the octal equivalent.

Example

```
'This example accepts a number and displays the decimal and octal 'equivalent until the input number is 0 or invalid.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Do
        xs$ = InputBox("Enter a number to convert:","Octal Convert")
        x = Val(xs$)
        If x <> 0 Then
            MsgBox "Decimal: " & x & "     Octal: " & Oct(x)
        Else
            MsgBox "Goodbye."
        End If
    Loop While x <> 0
End Sub
```

See Also

Hex, Hex$ (functions).

statement

OKButton

Syntax

```
OKButton X,Y,width,height [,.Identifier]
```

Description

Creates an OK button within a dialog box template.

Comments

- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The OKButton statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).

- If the DefaultButton parameter is not specified in the Dialog statement, the OK button will be used as the default button. In this case, the OK button can be selected by pressing Enter on a nonbutton control.
- A dialog box template must contain at least one OKButton, CancelButton, or PushButton statement (otherwise, the dialog box cannot be dismissed).

Example

```
'This example shows how to use the OK and Cancel buttons within a
'dialog box template and how to detect which one closed the dialog box.

Sub Main()
    Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit"
        Text 4,8,108,8,"Are you sure you want to exit?"
        CheckBox 32,24,63,8,"Save Changes",.SaveChanges
        OKButton 12,40,40,14
        CancelButton 60,40,40,14
    End Dialog
    Dim QuitDialog As QuitDialogTemplate
    rc% = Dialog(QuitDialog)
    Select Case rc%
        Case -1
            MsgBox "OK was pressed!"
        Case 1
            MsgBox "Cancel was pressed!"
    End Select
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

statement

On Error

Syntax

```
On Error {Goto label | Resume Next | Goto 0}
```

Description

Defines the action taken when a trappable runtime error occurs.

Comments

- The form On Error Goto label causes execution to transfer to the specified label when a runtime error occurs.
- The form On Error Resume Next causes execution to continue on the line following the line that caused the error.
- The form On Error Goto 0 causes any existing error trap to be removed.
- If an error trap is in effect when the script ends, then an error will be generated.
- An error trap is only active within the subroutine or function in which it appears.
- Once an error trap has gained control, appropriate action should be taken, and then control should be resumed using the Resume statement. The Resume statement resets the error handler and continues execution. If a procedure ends while an error is pending, then an error will be generated. (The Exit Sub or Exit Function statement also resets the error handler, letting a procedure to end without displaying an error message.)

Errors within an Error Handler

If an error occurs within the error handler, then the error handler of the caller (or any procedure in the call stack) will be invoked. If there is no such error handler, then the error is fatal, causing the script to stop executing. The following statements reset the error state (i.e., these statements turn off the fact that an error occurred):

```
Resume
Err=-1
```

The Resume statement forces execution to continue either on the same line or on the line following the line that generated the error. The Err=-1 statement lets explicit resetting of the error state so that the script can continue normal execution without resuming at the statement that caused the error condition.

The On Error statement will not reset the error. Thus, if an On Error statement occurs within an error handler, it has the effect of changing the location of a new error handler for any new errors that may occur once the error has been reset.

Example

```
'This example will demonstrate three types of error handling.
'The first case simply by-passes an expected error and continues
'with program operation. The second case creates an error branch
'that jumps to a common error handling routine that processes
'incoming errors, clears the error (with the Resume statement) and 'resumes program execution. The third case clears all internal error 'handling so that execution will stop when the next error is 'encountered.
```

```
Sub Main()
    Dim x%
    a = 10000
    b = 10000

    On Error Goto Pass          'Branch to this label on error.
    Do
        x% = a * b
    Loop

Pass:
    Err = -1                    'Clear error status.
    MsgBox "Cleared error status and continued."

    On Error Goto Overflow      'Branch to new error routine on any
    x% = 1000                       'subsequent errors.
    x% = a * b
    x% = a / 0

    On Error Resume Next        'Pass by any following errors until
    x% = 1000                       'another On Error statement is
    x% = a * b                  'encountered.

    On Error Goto 0             'Clear error branching.
    x% = a * b                  'Program will stop here.
    Exit Sub                    'Exit before common error routine.

Overflow:                           'Beginning of common error routine.
    If Err = 6 then
        MsgBox "Overflow Branch."
    Else
        MsgBox Error(Err)
    End If

    Resume Next
End Sub
```

See Also

Error Handling (topic); Error (statement); Resume (statement).

statement

Open

Syntax

```
Open filename$ [For mode] [Access accessmode] [lock] As [#] filenumber _
     [Len = reclen]
```

Description

Opens a file for a given mode, assigning the open file to the supplied filenumber.

Comments

- The filename$ parameter is a string expression that contains a valid filename.
- The filenumber parameter is a number between 1 and 255. The FreeFile function can be used to determine an available file number.
- The mode parameter determines the type of operations that can be performed on that file:

File Mode	Description
Input	Opens an existing file for sequential input (filename$ must exist). The value of accessmode, if specified, must be Read.
Output	Opens an existing file for sequential output, truncating its length to zero, or creates a new file. The value of accessmode, if specified, must be Write.
Append	Opens an existing file for sequential output, positioning the file pointer at the end of the file, or creates a new file. The value of accessmode, if specified, must be Read Write.
Binary	Opens an existing file for binary I/O or creates a new file. Existing binary files are never truncated in length. The value of accessmode, if specified, determines how the file can subsequently be accessed.
Random	Opens an existing file for record I/O or creates a new file. Existing random files are truncated only if accessmode is Write. The reclen parameter determines the record length for I/O operations. If the mode parameter is missing, then Random is used.

- The accessmode parameter determines what type of I/O operations can be performed on the file:

Access	Description
Read	Opens the file for reading only. This value is valid only for files opened in Binary, Random, or Input mode.
Write	Opens the file for writing only. This value is valid only for files opened in Binary, Random, or Output mode.
Read Write	Opens the file for both reading and writing. This value is valid only for files opened in Binary, Random, or Append mode.

- If the accessmode parameter is not specified, the following defaults are used:

File Mode	Default Value for accessmode
Input	Read
Output	Write
Append	Read Write
Binary	When the file is initially opened, access is attempted three times in the following order: 1. Read Write 2. Write 3. Read
Random	Same as Binary files

- The lock parameter determines what access rights are granted to other processes that attempt to open the same file. The following table describes the values for lock:

lock Value	Description
Shared	Another process can both read this file and write to it. (Deny none.)
Lock Read	Another process can write to this file but not read it. (Deny read.)
Lock Write	Another process can read this file but not write to it. (Deny write.)
Lock Read Write	Another process is prevented both from reading this file and from writing to it. (Exclusive.)

- If lock is not specified, then the file is opened in Shared mode.
- If the file does not exist and the lock parameter is specified, the file is opened twiceonce to create the file and again to establish the correct sharing mode.
- Files opened in Random mode are divided up into a sequence of records, each of the length specified by the reclen parameter. If this parameter is missing, then 128 is used. For files opened for sequential I/O, the reclen parameter specifies the size of the internal buffer used by Delrina Basic when performing I/O. Larger buffers mean faster file access. For Binary files, the reclen parameter is ignored.

Example

```
'This example opens several files in various configurations.

Sub Main()
    Open "test.dat" For Output Access Write Lock Write As #2
    Close
    Open "test.dat" For Input Access Read Shared As #1
    Close
    Open "test.dat" For Append Access Write Lock Read Write As #3
    Close
    Open "test.dat" For Binary Access Read Write Shared As #4
    Close
    Open "test.dat" For Random Access Read Write Lock Read As #5
    Close
    Open "test.dat" For Input Access Read Shared As #6
    Close
    Kill "test.dat"
End Sub
```

See Also

Close (statement); Reset (statement); FreeFile (function).

function

OpenFilename$

Syntax

```
OpenFilename$[([title$ [,extensions$]])]
```

Description

Displays a dialog box that prompts the user to select from a list of files, returning the full pathname of the file the user selects or a zero-length string if the user selects Cancel.

Comments

- This function displays the standard file open dialog box, which lets the user to select a file. It takes the following parameters:

Parameter	Description
title$	String specifying the title that appears in the dialog box's title bar. If this parameter is omitted, then "Open" is used.
extension$	String specifying the available file types. The format for this string depends on the platform on which Delrina Basic is running. If this parameter is omitted, then all files are displayed.

e$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
f$ = **OpenFilename$(**"Open Picture"**,**e$**)**

Example

```
'This example asks the user for the name of a file, then proceeds to read the
'first line from that file.
Sub Main
    Dim f As String,s As String
    f$ = OpenFilename$("Open Picture","Text Files:*.TXT")
    If f$ <> "" Then
        Open f$ For Input As #1
        Line Input #1,s$
        Close #1
        MsgBox "First line from " & f$ & " is " & s$
    End If
End Sub
```

See Also

MsgBox (statement); AskBox$ (function); AskPassword$ (function); InputBox, InputBox$ (functions); SaveFilename$ (function); SelectBox (function); AnswerBox (function).

Note

- The extensions$ parameter must be in the following format:

```
type:ext[,ext][;type:ext[,ext]]...
```

Placeholder	Description
type	Specifies the name of the grouping of files, such as All Files.
ext	Specifies a valid file extension, such as *.BAT or *.?F?.

For example, the following are valid extensions$ specifications:

```
"All Files:*.*"
"Documents:*.TXT,*.DOC"
"All Files:*.*;Documents:*.TXT,*.DOC"
```

topic

Operator Precedence

The following table shows the precedence of the operators supported by Delrina Basic. Operations involving operators of higher precedence occur before operations involving operators of lower precedence. When operators of equal precedence occur together, they are evaluated from left to right.

Operator	Description	Precedence Order
()	Parentheses	Highest
^	Exponentiation	
-	Unary minus	
/, *	Division and multiplication	
\	Integer division	
Mod	Modulo	
+, -	Addition and subtraction	
&	String concatenation	
=, <>, >, <, <=, >=	Relational	
Like, Is	String and object comparison	
Not	Logical negation	
And	Logical or binary conjunction	
Or	Logical or binary disjunction	
Xor, Eqv, Imp	Logical or binary operators	Lowest

The precedence order can be controlled using parentheses, as shown below:

```
a = 4 + 3 * 2          'a becomes 10.
a = (4 + 3) * 2        'a becomes 14.
```

topic

Operator Precision

When numeric, binary, logical or comparison operators are used, the data type of the result is generally the same as the data type of the more precise operand. For example, adding an Integer and a Long first converts the Integer operand to a Long, then preforms a long addition, overflowing only if the result cannot be contained with a Long. The order of precision is shown in the following table:

Empty	Least precise
Boolean	
Integer	
Long	
Single	
Date	
Double	
Currency	Most precise

There are exceptions noted in the descriptions of each operator.

The rules for operand conversion are further complicated when an operator is used with variant data. In many cases, an overflow causes automatic promotion of the result to the next highest precise data type. For example, adding two Integer variants results in an Integer variant unless it overflows, in which case the result is automatically promoted to a Long variant.

statement

Option Base

Syntax

Option Base {0 | 1}

Description

Sets the lower bound for array declarations.

Comments

- By default, the lower bound used for all array declarations is 0.
- This statement must appear outside of any functions or subroutines.

Example

Option Base 1

```
Sub Main()
    Dim a(10)                    'Contains 10 elements (not 11).
    a(1) = "Hello"
    MsgBox "The first element of the array is: " & a(1)
End Sub
```

See Also

Dim (statement); Public (statement); Private (statement).

statement

Option Compare

Syntax

Option Compare [Binary | Text]

Description

Controls how strings are compared.

Comments

- When Option Compare is set to Binary, then string comparisons are case-sensitive (e.g., "A" does not equal "a"). When it is set to Text, string comparisons are case-insensitive (e.g., "A" is equal to "a").
- The default value for Option Compare is Binary.
- The Option Compare statement affects all string comparisons in any statements that follow the Option Compare statement. Additionally, the setting affects the default behavior of Instr, StrComp, and the Like operator. The following table shows the types of string comparisons affected by this setting:

>	<	<>
<=	>=	Instr
StrComp	Like	

- The Option Compare statement must appear outside the scope of all subroutines and functions. In other words, it cannot appear within a Sub or Function block.

Example

'This example shows the use of Option Compare.

Option Compare Binary

```
Sub CompareBinary
    a$ = "This String Contains UPPERCASE."
    b$ = "this string contains uppercase."
    If a$ = b$ Then
        MsgBox "The two strings were compared case-insensitive."
    Else
        MsgBox "The two strings were compared case-sensitive."
    End If
End Sub
```

Option Compare Text

```
Sub CompareText
    a$ = "This String Contains UPPERCASE."
    b$ = "this string contains uppercase."
    If a$ = b$ Then
        MsgBox "The two strings were compared case-insensitive."
    Else
        MsgBox "The two strings were compared case-sensitive."
    End If
End Sub

Sub Main()
    CompareBinary               'Calls subroutine above.
    CompareText                 'Calls subroutine above.
End Sub
```

See Also

Like (operator); InStr (function); StrComp (function); Comparison Operators (topic).

statement

Option CStrings

Syntax

Option CStrings {On | Off}

Description

Turns on or off the ability to use C-style escape sequences within strings.

Comments

- When Option CStrings On is in effect, the compiler treats the backslash character as an escape character when it appears within strings. An escape character is simply a special character that cannot otherwise be ordinarily typed by the computer keyboard.

Escape	Description	Equivalent Expression
\r	Carriage return	Chr$(13)
\n	Line feed	Chr$(10)
\a	Bell	Chr$(7)
\b	Backspace	Chr$(8)
\f	Form feed	Chr$(12)
\t	Tab	Chr$(9)
\v	Vertical tab	Chr$(11)
\0	Null	Chr$(0)
\"	Double quotation mark	"" or Chr$(34)
\\	Backslash	Chr$(92)
\?	Question mark	?
\'	Single quotation mark	'
\xhh	Hexadecimal number	Chr$(Val("&Hhh))
\ooo	Octal number	Chr$(Val("&Oooo"))
\anycharacter	Any character	anycharacter

- With hexadecimal values, Delrina Basic stops scanning for digits when it encounters a nonhexadecimal digit or two digits, whichever comes first. Similarly, with octal values, Delrina Basic stops scanning when it encounters a nonoctal digit or three digits, whichever comes first.
- When Option CStrings Off is in effect, then the backslash character has no special meaning. This is the default.

Example

Option CStrings On

```
Sub Main()
    MsgBox "They said, \"Watch out for that clump of grass!\""
    MsgBox "First line.\r\nSecond line."
    MsgBox "Char A: \x41 \r\n Char B: \x42"
End Sub
```

statement

OptionButton

Syntax

```
OptionButton X,Y,width,height,title$ [,.Identifier]
```

Description

Defines an option button within a dialog box template.

Comments

- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The OptionButton statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
title$	String containing text that appears within the option button. This text may contain an ampersand character to denote an accelerator letter, such as "&Portrait" for Portrait, which can be selected by pressing the P accelerator.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).

Example

```
'This example creates a group of option buttons.

Sub Main()
    Begin Dialog PowerTemplate 16,31,128,65,"Print"
        GroupBox 8,8,64,52,"Amplifier Output",.Junk
        OptionGroup .Orientation
            OptionButton 16,20,51,8,"10 Watts",.Ten
            OptionButton 16,32,51,8,"50 Watts",.Fifty
            OptionButton 16,44,51,8,"100 Watts",.Hundred
        OKButton 80,8,40,14
    End Dialog
    Dim PowerDialog As PowerTemplate
    Dialog PowerDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Note

- On Windows platforms, accelerators are underlined, and the accelerator combination Alt+letter is used.

function

OptionEnabled

Syntax

```
OptionEnabled(name$ | id)
```

Description

Returns True if the specified option button is enabled within the current window or dialog box; returns False otherwise.

Comments

- This function is used to determine whether a given option button is enabled within the current window or dialog box. If an option button is enabled, then its value can be set using the SetOption statement.
- The OptionEnabled statement takes the following parameters:

Parameter	Description
name$	String containing the name of the option button.
id	Integer specifying the ID of the option button.

Note: The OptionEnabled function is used to determine whether an option button is enabled in another application's dialog box. Use the DlgEnable function with dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                    'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

GetOption (function); OptionExists (function); SetOption (statement).

function

OptionExists

Syntax

```
OptionExists(name$ | id)
```

Description

Returns True if the specified option button exists within the current window or dialog box; returns False otherwise.

Comments

- This function is used to determine whether a given option button exists within the current window or dialog box.
- The OptionExists statement takes the following parameters:

Parameter	Description
name$	String containing the name of the option button.
id	Integer specifying the ID of the option button.

Note: The OptionExists function is used to determine whether an option button exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                    'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

GetOption (function); OptionEnabled (function); SetOption (statement).

statement

OptionGroup

Syntax

```
OptionGroup .Identifier
```

Description

Specifies the start of a group of option buttons within a dialog box template.

Comments

- The .Identifier parameter specifies the name by which the group of option buttons can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the selected option button within the group (0 is the first option button, 1 is the second option button, and so on). This variable can be accessed using the following syntax: DialogVariable.Identifier.
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- When the dialog box is created, the option button specified by .Identifier will be on; all other option buttons in the group will be off. When the dialog box is dismissed, the .Identifier will contain the selected option button.

Example

```
'This example creates a group of option buttons.

Sub Main()
    Begin Dialog PowerTemplate 16,31,128,65,"Print"
        GroupBox 8,8,64,52,"Amplifier Output",.Junk
        OptionGroup .Orientation
            OptionButton 16,20,51,8,"10 Watts",.Ten
            OptionButton 16,32,51,8,"50 Watts",.Fifty
            OptionButton 16,44,51,8,"100 Watts",.Hundred
        OKButton 80,8,40,14
    End Dialog
    Dim PowerDialog As PowerTemplate
    Dialog PowerDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

operator

Or

Syntax

```
expression1 Or expression2
```

Description

Performs a logical or binary disjunction on two expressions.

Comments

- If both expressions are either Boolean, Boolean variants, or Null variants, then a logical disjunction is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

Binary Disjunction

If the two expressions are Integer, then a binary disjunction is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary disjunction is then performed, returning a Long result.

Binary disjunction forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

1	Or	1	=	1	Example:
0	Or	1	=	1	5 10101001
1	Or	0	=	1	6 01101010
0	Or	0	=	0	Or 11101011

Examples

```
'This first example shows the use of logical Or.

Sub Main()
    temperature_alert = True
    pressure_alert = False
    If temperature_alert Or pressure_alert Then
        MsgBox "You had better run!",ebExclamation,"Nuclear Disaster Imminent"
    End If
End Sub


'This second example shows the use of binary Or.

Sub Main()
    Dim w As Integer
```

```
TryAgain:
    s$ = InputBox("Enter a hex number (four digits max).","Binary Or Example")
    If Mid(s$,1,1) <> "&" Then
        s$ = "&H" & s$
    End If
    If Not IsNumeric(s$) Then Goto TryAgain

    w = CInt(s$)
    MsgBox "Your number is &H" & Hex(w)
    w = w Or &H8000
    MsgBox "Your number with the high bit set is &H" & Hex(w)
End Sub
```

See Also

Operator Precedence (topic); Xor (operator); Eqv (operator); Imp (operator); And (operator).

constant

Pi

Syntax

Pi

Description

The Double value 3.14159265358979323846264338327 9.

Comments

- Pi can also be determined using the following formula:

  4 * Atn(1)

Example

```
'This example illustrates the use of the Pi constant.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    dia = InputBox("Enter a circle diameter to compute.","Compute Circle")
    circ# = Pi * dia
    area# = Pi * ((dia / 2) ^ 2)
    msg = "Diameter: " & dia & crlf
    msg = msg & "Circumference: " & Format(circ#,"Standard") & crlf
    msg = msg & "Area: " & Format(area#,"Standard")
    MsgBox msg
End Sub
```

See Also

Tan (function); Atn (function); Cos (function); Sin (function).

statement

Picture

Syntax

```
Picture X,Y,width,height,PictureName$,PictureType [,[.Identifier] [,style]]
```

Description

Creates a picture control in a dialog box template.

Comments

- Picture controls are used for the display of graphics images only. The user cannot interact with these controls.
- The Picture statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
PictureName$	String containing the name of the picture. If PictureType is 0, then this name specifies the name of the file containing the image. If PictureType is 10, then PictureName$ specifies the name of the image within the resource of the picture library. If PictureName$ is empty, then no picture will be associated with the control. A picture can later be placed into the picture control using the DlgSetPicture statement.
PictureType	Integer specifying the source for the image. The following sources are supported:
0	The image is contained in a file on disk.
10	The image is contained in a picture library as specified by the PicName$ parameter on the Begin Dialog statement.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words of PictureName$ are used.
style	Specifies whether the picture is drawn within a 3D frame. It can be any of the following values:
0	Draw the picture control with a normal frame.
1	Draw the picture control with a 3D frame.

- If omitted, then the picture control is drawn with a normal frame.
- The picture control extracts the actual image from either a disk file or a picture library. In the case of bitmaps, both 2- and 16-color bitmaps are supported. In the case of WMFs, Delrina Basic supports the Placeable Windows Metafile.
- If PictureName$ is a zero-length string, then the picture is removed from the picture control, freeing any memory associated with that picture.

Examples

```
'This first example shows how to use a picture from a file.

Sub Main()
    Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"
        OKButton 240,8,40,14
        Picture 8,8,224,64,"c:\bitmaps\logo.bmp",0,.Logo
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

```
'This second example shows how to use a picture from a picture library with
'a 3D frame.

Sub Main()
    Begin Dialog LogoDialogTemplate 16,31,288,76,"Introduction",,"pictures.dll"
        OKButton 240,8,40,14
        Picture 8,8,224,64,"CompanyLogo",10,.Logo,1
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement) , DlgSetPicture (statement).

statement

PictureButton

Syntax

```
PictureButton X,Y,width,height,PictureName$,PictureType [,.Identifier]
```

Description

Creates a picture button control in a dialog box template.

Comments

- Picture button controls behave very much like a push button controls. Visually, picture buttons are different than push buttons in that they contain a graphic image imported either from a file or from a picture library.
- The PictureButton statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
PictureName$	String containing the name of the picture. If PictureType is 0, then this name specifies the name of the file containing the image. If PictureType is 10, then PictureName$ specifies the name of the image within the resource of the picture library. If PictureName$ is empty, then no picture will be associated with the control. A picture can later be placed into the picture control using the DlgSetPicture statement.
PictureType	Integer specifying the source for the image. The following sources are supported: 0 The image is contained in a file on disk. 10 The image is contained in a picture library as specified by the PicName$ parameter on the Begin Dialog statement.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).

- The picture button control extracts the actual image from either a disk file or a picture library, depending on the value of PictureType. The supported picture formats vary from platform to platform.
- If PictureName$ is a zero-length string, then the picture is removed from the picture button control, freeing any memory associated with that picture.

Examples

'This first example shows how to use a picture from a file.

```
Sub Main()
    Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"
        OKButton 240,8,40,14
        PictureButton 8,4,224,64,"c:\bitmaps\logo.bmp",0,.Logo
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

'This second example shows how to use a picture from a picture library.

```
Sub Main()
    Begin Dialog LogoDialogTemplate 16,31,288,76,"Introduction",,"pictures.dll"
        OKButton 240,8,40,14
        PictureButton 8,4,224,64,"CompanyLogo",10,.Logo
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), Picture (statement), DlgSetPicture (statement).

Notes

Picture controls can contain either a bitmap or a WMF (Windows metafile). When extracting images from a picture library, Delrina Basic assumes that the resource type for metafiles is 256.

Picture libraries are implemented as DLLs.

function

Pmt

Syntax

```
Pmt(Rate,NPer,Pv,Fv,Due)
```

Description

Returns the payment for an annuity based on periodic fixed payments and a constant rate of interest.

Comments

- An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.
- The Pmt function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. If the periods are given in months, be sure to normalize annual rates by dividing them by 12.
NPer	Double representing the total number of payments in the annuity.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.

- Rate and NPer must be expressed in the same units. If Rate is expressed in months, then NPer must also be expressed in months.
- Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example

```
'This example calculates the payment necessary to repay a $1,000.00 loan
'over 36 months at an annual rate of 10%. Payments are due at the beginning
'of the period.

Sub Main()
    x = Pmt((.1/12),36,1000.00,0,1)
    msg = "The payment to amortize $1,000 over 36 months @ 10% is: "
    MsgBox msg & Format(x,"Currency")
End Sub
```

See Also

IPmt (function); NPer (function); PPmt (function); Rate (function).

function

PopupMenu

Syntax

```
PopupMenu(MenuItems$())
```

Description

Displays a pop-up menu containing the specified items, returning an Integer representing the index of the selected item.

Comments

- If no item is selected (i.e., the pop-up menu is canceled), then a value of 1 less than the lower bound is returned (normally, -1).
- This function creates a pop-up menu using the string elements in the given array. Each array element is used as a menu item. A zero-length string results in a separator bar in the menu.
- The pop-up menu is created with the upper left corner at the current mouse position.
- A runtime error results if MenuItems$ is not a single-dimension array.
- Only one pop-up menu can be displayed at a time. An error will result if another script runs this function while a pop-up menu is visible.

Example

```
Sub Main()
    Dim a$()
    AppList a$
    w% = PopupMenu(a$)
End Sub
```

See Also

SelectBox (function).

function

PPmt

Syntax

```
PPmt(Rate,Per,NPer,Pv,Fv,Due)
```

Description

Calculates the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Comments

- An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.
- The PPmt function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period.
Per	Double representing the number of payment periods. Per can be no less than 1 and no greater than NPer.
NPer	Double representing the total number of payments in your annuity.
Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
Due	Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period; if it is 1, then payments are due at the start of each period.

- Rate and NPer must be in the same units to calculate correctly. If Rate is expressed in months, then NPer must also be expressed in months.
- Negative values represent payments paid out, whereas positive values represent payments received.

Example

```
'This example calculates the principal paid during each year on a loan of
'$1,000.00 with an annual rate of 10% for a period of 10 years. The result
'is displayed as a table containing the following information: payment,
'principal payment, principal balance.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    pay = Pmt(.1,10,1000.00,0,1)
    msg = "Amortization table for 1,000" & crlf & "at 10% annually for"
    msg = msg & " 10 years: " & crlf & crlf
    bal = 1000.00
    For per = 1 to 10
        prn = PPmt(.1,per,10,1000,0,0)
        bal = bal + prn
        msg = msg & Format(pay,"Currency") & "    " & Format$(Prn,"Currency")
        msg = msg & "    " & Format(bal,"Currency") & crlf
    Next per
    MsgBox msg
End Sub
```

See Also

IPmt (function); NPer (function); Pmt (function); Rate (function).

statement

Print

Syntax

```
Print [[{Spc(n) | Tab(n)}][expressionlist][{; | ,}]]
```

Description

Prints data to an output device.

Comments

- The actual output device depends on the platform on which Delrina Basic is running.
- The following table describes how data of different types is written:

Data Type	Description
String	Printed in its literal form, with no enclosing quotes.
Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
Boolean	Printed as "True" or "False".
Date	Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is consistent with the "general date" format understood by the Format/Format$ functions).
Empty	Nothing is printed.
Null	Prints "Null".
User-defined errors	Printed as "Error code", where code is the value of the user-defined error. The word "Error" is not translated.

- Each expression in expressionlist is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.
- If the last expression in the list is not followed by a comma or a semicolon, then a carriage return is printed to the file. If the last expression ends with a semicolon, no carriage return is printedthe next Print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.
- The Tab and Spc functions provide additional control over the column position. The Tab function moves the file position to the specified column, whereas the Spc function outputs the specified number of spaces.

Examples

```
Sub Main()
    i% = 10
    s$ = "This is a test."
    Print "The value of i=";i%,"the value of s=";s$

    'This example prints the value of i% in print zone 1 and s$ in print
    'zone 3.
    Print i%,,s$

    'This example prints the value of i% and s$ separated by 10 spaces.
    Print i%;Spc(10);s$

    'This example prints the value of i in column 1 and s$ in column 30.
    Print i%;Tab(30);s$
```

```
    'This example prints the value of i% and s$.
    Print i%;s$,
    Print 67
End Sub
```

See Also

ViewportOpen (statement).

Note

- This statement writes data to a viewport window.
- If no viewport window is open, then the statement is ignored. Printing information to a viewport window is a convenient way to output debugging information. To open a viewport window, use the following statement:

  ViewportOpen

statement

Print#

Syntax

```
Print [#]filenumber, [[{Spc(n) | Tab(n)}][expressionlist][{;|,}]]
```

Description

Writes data to a sequential disk file.

Comments

- The filenumber parameter is a number that is used by Delrina Basic to refer to the open file-the number passed to the Open statement.
- The following table describes how data of different types is written:

Data Type	Description
String	Printed in its literal form, with no enclosing quotes.
Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
Boolean	Printed as "True" or "False".
Date	Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is concistent with the "general date" format understood by the Format/Format$ functions).
Empty	Nothing is printed.
Null	Prints "Null".
User-defined errors	Printed to files as "Error code", where code is the value of the user-defined error. The word "Error" is not translated.

- Each expression in expressionlist is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.
- If the last expression in the list is not followed by a comma or a semicolon, then an end-of-line is printed to the file. If the last expression ends with a semicolon, no end-of-line is printedthe next Print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.
- The Write statement always outputs information ending with an end-of-line. Thus, if a Print statement is followed by a Write statement, the file pointer is positioned on a new line.
- The Print statement can only be used with files that are opened in Output or Append mode.
- The Tab and Spc functions provide additional control over the file position. The Tab function moves the file position to the specified column, whereas the Spc function outputs the specified number of spaces.
- In order to correctly read the data using the Input# statement, you should write the data using the Write statement.

Examples

```
Sub Main()
    'This example opens a file and prints some data.
    Open "test.dat" For Output As #1
    i% = 10
    s$ = "This is a test."
    Print #1,"The value of i=";i%,"the value of s=";s$
```

```
    'This example prints the value of i% in print zone 1 and s$ in
    'print zone 3.
    Print #1,i%,,s$

    'This example prints the value of i% and s$ separated by ten spaces.
    Print #1,i%;Spc(10);s$

    'This example prints the value of i in column 1 and s$ in column 30.
    Print #1,i%;Tab(30);s$

    'This example prints the value of i% and s$.
    Print #1,i%;s$,
    Print #1,67

    Close #1
    Kill "test.dat"
End Sub
```

See Also

Open (statement); Put (statement); Write# (statement).

function

PrinterGetOrientation

Syntax

PrinterGetOrientation[()]

Description

Returns an Integer representing the current orientation of paper in the default printer.

Comments

- PrinterGetOrientation returns ebPortrait if the printer orientation is set to portrait; otherwise, it returns ebLandscape.
- This function loads the printer driver and therefore may be slow.

Example

```
'This example toggles the printer orientation.

Sub Main()
    If PrinterGetOrientation = ebLandscape Then
        PrinterSetOrientation ebPortrait
    Else
        PrinterSetOrientation ebLandscape
    End If
End Sub
```

See Also

PrinterSetOrientation (statement).

Note

- The default printer is determined by examining the device= line in the [windows] section of the win.ini file.

statement

PrinterSetOrientation

Syntax

```
PrinterSetOrientation NewSetting
```

Description

Sets the orientation of the default printer to NewSetting.

Comments

- The possible values for NewSetting are as follows:

Setting	Description
ebLandscape	Sets printer orientation to landscape.
ebPortrait	Sets printer orientation to portrait.

- This function loads the printer driver for the default printer and therefore may be slow.

Example

```
'This example toggles the printer orientation.

Sub Main()
    If PrinterGetOrientation = ebLandscape Then
        PrinterSetOrientation ebPortrait
    Else
        PrinterSetOrientation ebLandscape
    End If
End Sub
```

See Also

PrinterGetOrientation (function).

Note

- The default printer is determined by examining the device= line in the [windows] section of the win.ini file.

function

PrintFile

Syntax

```
PrintFile(filename$)
```

Description

Prints the filename$ using the application to which the file belongs.

Comments

- PrintFile returns an Integer indicating success or failure.
- If an error occurs executing the associated application, then PrintFile generates a trappable runtime error, returning 0 for the result. Otherwise, PrintFile returns a value representing that application to the system. This value is suitable for calling the AppActivate statement.

Example

```
'This example asks the user for the name of a text file, then prints it.

Sub Main()
    f$ = OpenFilename$("Print Text File","Text Files:*.txt")
    If f$ <> "" Then
        rc% = PrintFile(f$)
        If rc% > 32 Then
            MsgBox "File is printing."
        End If
    End If
End Sub
```

See Also

Shell (function).

statement

Private

Syntax

```
Private name [(subscripts)] [As type] [,name [(subscripts)] [As type]]...
```

Description

Declares a list of private variables and their corresponding types and sizes.

Comments

- Private variables are global to every Sub and Function within the currently executing script.
- If a type-declaration character is used when specifying name (such as %, @, &, $, or !), the optional [As type] expression is not allowed. For example, the following are allowed:

  Private foo **As** Integer
  Private foo%

- The subscripts parameter lets the declaration of arrays. This parameter uses the following syntax:

  [lower To] upper [,[lower To] upper]...

- The lower and upper parameters are integers specifying the lower and upper bounds of the array. If lower is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.
- The total size of an array (not counting space for strings) is limited to 64K.
- Dynamic arrays are declared by not specifying any bounds:

  Private a()

- The type parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.
- If a variable is seen that has not been explicitly declared with either Dim, Public, or Private, then it will be implicitly declared local to the routine in which it is used.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

```
Private name As String * length
```

where length is a literal number specifying the string's length.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
Integer	0
Long	0
Double	0.0
Single	0.0
Currency	0.0
Object	Nothing
Date	December 31, 1899 00:00:00
Boolean	False
Variant	Empty

String	"" (zero-length string)
User-defined type	Each element of the structure is given a default value, as described above.
Arrays	Each element of the array is given a default value, as described above.

Example

```
'This example sets the value of variable x# in two separate routines
'to show the behavior of private variables.

Private x#

Sub Area()
    x# = 10        'Set this copy of x# to 10 and display
    MsgBox x#
End Sub

Sub Main()
    x# = 100       'Set this copy of x# to 100 and display after calling the Area subroutine
    Area
    MsgBox x#
End Sub
```

See Also

Dim (statement); Redim (statement); Public (statement); Option Base (statement).

statement

Public

Syntax

```
Public name [(subscripts)] [As type] [,name [(subscripts)] [As type]]...
```

Description

Declares a list of public variables and their corresponding types and sizes.

Comments

- Public variables are global to all Subs and Functions in all scripts.
- If a type-declaration character is used when specifying name (such as %, @, &, $, or !), the optional [As type] expression is not allowed. For example, the following are allowed:

  Public foo **As** Integer
  Public foo%

- The subscripts parameter lets the declaration of arrays. This parameter uses the following syntax:
  `[lower To] upper [,[lower To] upper]...`
- The lower and upper parameters are integers specifying the lower and upper bounds of the array. If lower is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.
- The total size of an array (not counting space for strings) is limited to 64K.
- Dynamic arrays are declared by not specifying any bounds:

  Public a()

- The type parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.
- If a variable is seen that has not been explicitly declared with either Dim, Public, or Private, then it will be implicitly declared local to the routine in which it is used.
- For compatibility, the keyword Global is also supported. It has the same meaning as Public.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

```
Public name As String * length
```

where length is a literal number specifying the string's length.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
Integer	0
Long	0
Double	0.0
Single	0.0
Currency	0.0
Date	December 31, 1899 00:00:00
Object	Nothing
Boolean	False
Variant	Empty

String	"" (zero-length string)
User-defined type	Each element of the structure is given a default value, as described above.
Arrays	Each element of the array is given a default value, as described above.

Sharing Variables

When sharing variables, you must ensure that the declarations of the shared variables are the same in each script that uses those variables. If the public variable being shared is a user-defined structure, then the structure definitions must be exactly the same.

Example

```
'This example uses a subroutine to calculate the area of ten circles
'and displays the result in a dialog box. The variables R and Ar are
'declared as Public variables so that they can be used in both Main and Area.

Const crlf = Chr$(13) + Chr$(10)

Public x#,ar#

Sub Area()
    ar# = (x# ^ 2) * Pi
End Sub

Sub Main()
    msg = "The area of the ten circles are:" & crlf & crlf
    For x# = 1 To 10
        Area
        msg = msg & x# & ": " & Format(ar#,"fixed") & Basic.Eoln$
    Next x#
    MsgBox msg
End Sub
```

See Also

Dim (statement); Redim (statement); Private (statement); Option Base (statement).

statement

PushButton

Syntax

```
PushButton X,Y,width,height,title$ [,.Identifier]
```

Description

Defines a push button within a dialog box template.

Comments

- Choosing a push button causes the dialog box to close (unless the dialog function redefines this behavior).
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The PushButton statement accepts the following parameters:

Parameter	Description
X, Y	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer coordinates specifying the dimensions of the control in dialog units.
title$	String containing the text that appears within the push button. This text may contain an ampersand character to denote an accelerator letter, such as "&Save" for Save.
.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).

- If a push button is the default button, it can be selected by pressing Enter on a nonbutton control.
- A dialog box template must contain at least one OKButton, CancelButton, or PushButton statement (otherwise, the dialog box cannot be dismissed).

Example

```
'This example creates a bunch of push buttons and displays
'which button was pushed.

Sub Main()
    Begin Dialog ButtonTemplate 17,33,104,84,"Buttons"
        OKButton 8,4,40,14,.OK
        CancelButton 8,24,40,14,.Cancel
        PushButton 8,44,40,14,"1",.Button1
        PushButton 8,64,40,14,"2",.Button2
        PushButton 56,4,40,14,"3",.Button3
        PushButton 56,24,40,14,"4",.Button4
        PushButton 56,44,40,14,"5",.Button5
        PushButton 56,64,40,14,"6",.Button6
    End Dialog
    Dim ButtonDialog As ButtonTemplate
    WhichButton% = Dialog(ButtonDialog)
    MsgBox "You pushed button " & WhichButton%
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Note

- Accelerators are underlined, and the accelerator combination Alt+letter is used.

statement

Put

Syntax

```
Put [#]filenumber, [recordnumber], variable
```

Description

Writes data from the specified variable to a Random or Binary file.

Comments

- The Put statement accepts the following parameters:

Parameter	Description
filenumber	Integer representing the file to be written to. This is the same value as returned by the Open statement.
recordnumber	Long specifying which record is to be written to the file. For Binary files, this number represents the first byte to be written starting with the beginning of the file (the first byte is 1). For Random files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647. If the recordnumber parameter is omitted, the next record is written to the file (if no records have been written yet, then the first record in the file is written). When recordnumber is omitted, the commas must still appear, as in the following example:
Put **#**1**,,**recvar	If recordlength is specified, it overrides any previous change in file position specified with the Seek statement.

- The variable parameter is the name of any variable of any of the following types:

Variable Type	File Storage Description
Integer	2 bytes are written to the file.
Long	4 bytes are written to the file.
String (variable-length)	In Binary files, variable-length strings are written by first determining the specified string variable's length, then writing that many bytes to the file. In Random files, variable-length strings are written by first writing a 2-byte length, then writing that many characters to the file.
String (fixed-length)	Fixed-length strings are written to Random and Binary files in the same way: the number of characters equal to the string's declared length are written.
Double	8 bytes are written to the file (IEEE format).
Single	4 bytes are written to the file (IEEE format).
Date	8 bytes are written to the file (IEEE double format).
Boolean	2 bytes are written to the file (either -1 for True or 0 for False).
Variant	A 2-byte VarType is written to the file followed by the data as described above. With variants of type 10 (user-defined errors), the 2-byte VarType is followed by a 2-byte unsigned integer (the error value), which is then followed by 2 additional bytes of information. The exception is with strings, which are always preceded by a 2-byte string length.
User-defined types	Each member of a user-defined data type is written individually. In Binary files, variable-length strings within user-defined types are written by first writing a

	2-byte length followed by the string's content. This storage is different than variable-length strings outside of user-defined types. When writing user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.
Arrays	Arrays cannot be written to a file using the Put statement.
Objects	Object variables cannot be written to a file using the Put statement.

- With Random files, a runtime error will occur if the length of the data being written exceeds the record length (specified as the reclen parameter with the Open statement). If the length of the data being written is less than the record length, the entire record is written along with padding (whatever data happens to be in the I/O buffer at that time). With Binary files, the data elements are written contiguously: they are never separated with padding.

Example

```
'This example opens a file for random write, then writes ten
'records into the file with the values 10-50. Then the file
'is closed and reopened in random mode for read, and the
'records are read with the Get statement. The result is displayed
'in a dialog box.

Sub Main()
    Open "test.dat" For Random Access Write As #1
    For x = 1 To 10
        r% = x * 10
        Put #1,x,r%
    Next x
    Close

    Open "test.dat" For Random Access Read As #1
    For x = 1 To 10
        Get #1,x,r%
        msg = msg & "Record " & x & " is: " & r% & Basic.Eoln$
    Next x

    MsgBox msg
    Close
    Kill "test.dat"
End Sub
```

See Also

Open (statement); Put (statement); Write# (statement); Print# (statement).

function

Pv

Syntax

```
Pv(Rate,NPer,Pmt,Fv,Due)
```

Description

Calculates the present value of an annuity based on future periodic fixed payments and a constant rate of interest.

Comments

- The Pv function requires the following parameters:

Parameter	Description
Rate	Double representing the interest rate per period. When used with monthly payments, be sure to normalize annual percentage rates by dividing them by 12.
NPer	Double representing the total number of payments in the annuity.
Pmt	Double representing the amount of each payment per period.
Fv	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be 0.
Due	Integer indicating when the payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.

- Rate and NPer must be expressed in the same units. If Rate is expressed in months, then NPer must also be expressed in months.
- Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example

```
'This example demonstrates the present value (the amount you'd have to pay
'now) for a $100,000 annuity that pays an annual income of $5,000 over 20
'years at an annual interest rate of 10%.

Sub Main()
    pval = Pv(.1,20,-5000,100000,1)
    MsgBox "The present value is: " & Format(pval,"Currency")
End Sub
```

See Also

Fv (function); IRR (function); MIRR (function); Npv (function).

statement

QueEmpty

Syntax

QueEmpty

Description

Empties the current event queue.

Comments

After this statement, QueFlush will do nothing.

Example

```
'This code begins a new queue, then drags a selection over a range of
'characters in Notepad.

Sub Main()
    AppActivate "Notepad"
    QueEmpty            'Make sure the queue is empty.
    QueMouseDn ebLeftButton,1440,1393
    QueMouseUp ebLeftButton,4147,2363
    QueFlush True
End Sub
```

statement

QueFlush

Syntax

```
QueFlush isSaveState
```

Description

Plays back events that are stored in the current event queue.

Comments

- After QueFlush is finished, the queue is empty.
- If isSaveState is True, then QueFlush saves the state of the Caps Lock, Num Lock, Scroll Lock, and Insert and restores the state after the QueFlush is complete. If this parameter is False, these states are not restored.
- The function does not return until the entire queue has been played.

Example

```
'This example pumps some keys into Notepad.

Sub Main()
    AppActivate "Notepad"
    QueKeys "This is a test{Enter}"
    QueFlush True                          'Play back the queue.
End Sub
```

Note

- The QueFlush statement uses the Windows journaling mechanism to replay the mouse and keyboard events stored in the queue. As a result, the mouse position may be changed. Furthermore, events can be played into any Windows application, including DOS applications running in a window.

statement

QueKeyDn

Syntax

```
QueKeyDn KeyString$ [,time]
```

Description

Appends key-down events for the specified keys to the end of the current event queue.

Comments

- The QueKeyDn statement accepts the following parameters:

Parameter	Description
KeyString$	String containing the keys to be sent. The format for KeyString$ is described under the SendKeys statement.
time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString$ parameter. It must be within the following range:

0 <= time <= 32767

- For example, if time is 5000 (5 seconds) and the KeyString$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

'This example plays back a Ctrl + mouse click.

```
Sub Main()
    QueEmpty
    QueKeyDn "^"
    QueMouseClick ebLeftButton 1024,792
    QueKeyUp "^"
    QueFlush True
End Sub
```

See Also

DoKeys (statement); SendKeys (statement); QueKeys (statement); QueKeyUp (statement); QueFlush (statement).

statement

QueKeys

Syntax

```
QueKeys KeyString$ [,time]
```

Description

Appends keystroke information to the current event queue.

Comments

- The QueKeys statement accepts the following parameters:

Parameter	Description
KeyString$	String containing the keys to be sent. The format for KeyString$ is described under the SendKeys statement.
time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString$ parameter. It must be within the following range: `0 <= time <= 32767`

- For example, if time is 5000 (5 seconds) and the KeyString$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

```
Sub Main()
    WinActivate "Notepad"
    QueEmpty
    QueKeys "This is a test.{Enter}This is on a new line.{Enter}"
    QueKeys "{Tab 3}This is indented with three tabs."
    QueKeys "Some special characters: {~}{^}{%}{+}~"
    QueKeys "Invoking the Find dialog.%sf"                    'Alt+s,f
    QueFlush True
End Sub
```

See Also

DoKeys (statement); SendKeys (statement); QueKeyDn (statement); QueKeyUp (statement); QueFlush (statement).

Note

- You cannot send keystrokes to DOS applications running in a window.

statement

QueKeyUp

Syntax

```
QueKeyUp KeyString$ [,time]
```

Description

Appends key-up events for the specified keys to the end of the current event queue.

Comments

- The QueKeyUp statement accepts the following parameters:

Parameter	Description
KeyString$	String containing the keys to be sent. The format for KeyString$ is described under the SendKeys statement.
time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString$ parameter. It must be within the following range: 0 <= time <= 32767

- For example, if time is 5000 (5 seconds) and the KeyString$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

```
'This example plays back a Ctrl + mouse click.

Sub Main()
    QueEmpty
    QueKeyDn "^"
    QueMouseClick ebLeftButton 1024,792
    QueKeyUp "^"
    QueFlush True
End Sub
```

See Also

DoKeys (statement); SendKeys (statement); QueKeys (statement); QueKeyDn (statement); QueFlush (statement).

statement

QueMouseClick

Syntax

```
QueMouseClick button,X,Y [,time]
```

Description

Adds a mouse click to the current event queue.

Comments

- The QueMouseClick statement takes the following parameters:

Parameter	Description	
button	Integer specifying which mouse button to click:	
	ebLeftButton	Click the left mouse button.
	ebRightButton	Click the right mouse button.
X, Y	Integer coordinates, in twips, where the mouse click is to be recorded.	
time	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse click will play back at full speed.	

- A mouse click consists of a mouse button down at position X, Y, immediately followed by a mouse button up.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

```
'This example acvivates Notepad and invokes the Find dialog box. It then
'uses the QueMouseClick command to click the Cancel button.

Sub Main()
    AppActivate "Notepad"       'Activate Notepad.
    QueKeys "%Sf"               'Invoke the Find dialog box.
    QueFlush True               'Play this back (let dialog box to open).
    QueSetRelativeWindow        'Set mouse relative to Find dialog box.
    QueMouseClick ebLeftButton,7059,1486     'Click the Cancel button.
    QueFlush True               'Play back the queue.
End Sub
```

See Also

QueMouseDn (statement); QueMouseUp (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

statement

QueMouseDblClk

Syntax

```
QueMouseDblClk button,X,Y [,time]
```

Description

Adds a mouse double click to the current event queue.

Comments

- The QueMouseDblClk statement takes the following parameters:

Parameter	Description	
button	Integer specifying which mouse button to double-click:	
	ebLeftButton	Double-click the left mouse button.
	ebRightButton	Double-click the right mouse button.
X, Y	Integer coordinates, in twips, where the mouse double click is to be recorded.	
time	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse double click will play back at full speed.	

- A mouse double click consists of a mouse down/up/down/up at position X, Y. The events are queued in such a way that a double click is registered during queue playback.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

'This example double-clicks the left mouse button.

QueMouseDblClk ebLeftButton**,**344**,**360

See Also

QueMouseClick (statement); QueMouseDn (statement); QueMouseUp (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

statement

QueMouseDblDn

Syntax

```
QueMouseDblDn button, X, Y [,time]
```

Description

Adds a mouse double down to the end of the current event queue.

Comments

- The QueMouseDblDn statement takes the following parameters:

Parameter	Description	
button	Integer specifying which mouse button to press:	
	ebLeftButton	Click the left mouse button.
	ebRightButton	Click the right mouse button.
x,y	Integer coordinates, in twips, where the mouse double down is to be recorded.	
time	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse double down will play back at full speed.	

- This statement adds a mouse double down to the current event queue. A double down consists of a mouse down/up/down at position X, Y.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

'This example double-clicks a word, then drags it to a new location.

```
Sub Main()
    QueFlush                                    'Start with empty queue.
    QueMouseDblDn ebLeftButton,356,4931         'Double-click, mouse still down.
    QueMouseMove 600,4931                       'Drag to new spot.
    QueMouseUp ebLeftButton                     'Now release the mouse.
    QueFlush True                               'Play back the queue.
End Sub
```

See Also

QueMouseClick (statement); QueMouseDn (statement); QueMouseUp (statement); QueMouseDblClk (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

statement

QueMouseDn

Syntax

```
QueMouseDn button,X,Y [,time]
```

Description

Adds a mouse down to the current event queue.

Comments

- The QueMouseDn statement takes the following parameters:

Parameter	Description	
button	Integer specifying which mouse button to press:	
	ebLeftButton	Click the left mouse button.
	ebRightButton	Click the right mouse button.
X, Y	Integer coordinates, in twips, where the mouse down is to be recorded.	
time	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse down will play back at full speed.	

- The QueFlush command is used to play back the events stored in the current event queue.

Example

```
'This code begins a new queue, then drags a selection over a range of
'characters in Notepad.

Sub Main()
    AppActivate "Notepad"
    QueEmpty              'Make sure the queue is empty.
    QueMouseDn ebLeftButton,1440,1393
    QueMouseUp ebLeftButton,4147,2363
    QueFlush True
End Sub
```

See Also

QueMouseClick (statement); QueMouseUp (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

statement

QueMouseMove

Syntax

```
QueMouseMove X,Y [,time]
```

Description

Adds a mouse move to the current event queue.

Comments

- The QueMouseMove statement takes the following parameters:

Parameter	Description
X, Y	Integer coordinates, in twips, where the mouse is to be moved.
time	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse move will play back at full speed.

- The QueFlush command is used to play back the events stored in the current event queue.

Example

'This example double-clicks a word, then drags it to a new location.

```
Sub Main()
    QueFlush                                'Start with empty queue.
    QueMouseDblDn ebLeftButton,356,4931     'Double-click, mouse still down.
    QueMouseMove 600,4931                       'Drag to new spot.
    QueMouseUp ebLeftButton                 'Now release the mouse.
    QueFlush True                           'Play back the queue.
End Sub
```

See Also

QueMouseClick (statement); QueMouseDn (statement); QueMouseUp (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMoveBatch (statement); QueFlush (statement).

statement

QueMouseMoveBatch

Syntax

```
QueMouseMoveBatch ManyMoves$
```

Description

Adds a series of mouse-move events to the current event queue.

Comments

- The ManyMoves$ parameter is a string containing positional and timing information in the following format:

```
X,Y,time [,X,Y,time]...
```

- The X and Y parameters specify a mouse position in twips. The time parameter specifies the delay in milliseconds between the current mouse move and the previous event in the queue. If time is 0, then the mouse move will play back as fast as possible.
- The QueMouseMoveBatch command should be used in place of a series of QueMouseMove statements to reduce the number of lines in your script. A further advantage is that, since the mouse-move information is contained within a literal string, the storage for the data is placed in the constant segment instead of the code segment, reducing the size of the code.
- The QueFlush command is used to play back the events stored in the current event queue.

Example

'This example activates PaintBrush, then paints the word "Hi".

```
Sub Main()
    AppActivate "Paintbrush"
    AppMaximize
    QueMouseDn ebLeftButton,2175,3412
    QueMouseMoveBatch "2488,3224,0,2833,2786,0,3114,2347,0,3208,2160,0,3240,2097,0"
    QueMouseMoveBatch "3255,2034,0,3255,1987,0,3255,1956,0,3255,1940,0,3224,1956,0"
    QueMouseMoveBatch "3193,1987,0,3114,2019,0,3036,2066,0,3005,2113,0,2973,2175,0"
    QueMouseMoveBatch "2942,2332,0,2926,2394,0,2926,2582,0,2911,2739,0,2911,2801,0"
    QueMouseMoveBatch "2911,2958,0,2911,3020,0,2911,3052,0,2911,3083,0,2911,3114,0"
    QueMouseMoveBatch "2911,3130,0,2895,3161,0,2895,3193,0,2895,3208,0,2895,3193,0"
    QueMouseMoveBatch "2895,3146,0,2911,3083,0,2926,3020,0,2942,2958,0,2973,2895,0"
    QueMouseMoveBatch "3005,2848,0,3020,2817,0,3036,2801,0,3052,2770,0,3083,2770,0"
    QueMouseMoveBatch "3114,2754,0,3130,2754,0,3146,2770,0,3161,2786,0,3161,2848,0"
    QueMouseMoveBatch "3193,3005,0,3193,3193,0,3208,3255,0,3224,3318,0,3240,3349,0"
    QueMouseMoveBatch "3255,3349,0,3286,3318,0,3380,3271,0,3474,3208,0,3553,3052,0"
    QueMouseMoveBatch "3584,2895,0,3615,2739,0,3631,2692,0,3631,2645,0,3646,2645,0"
    QueMouseMoveBatch "3646,2660,0,3646,2723,0,3646,2880,0,3662,2942,0,3693,2989,0"
    QueMouseMoveBatch "3709,3005,0,3725,3005,0,3756,2989,0,3787,2973,0"
    QueMouseUp ebLeftButton,3787,2973
    QueMouseDn ebLeftButton,3678,2535
    QueMouseMove 3678,2520
    QueMouseMove 3678,2535
    QueMouseUp ebLeftButton,3678,2535
    QueFlush True
End Sub
```

See Also

QueMouseClick (statement); QueMouseDn (statement); QueMouseUp (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueFlush (statement).

statement

QueMouseUp

Syntax

```
QueMouseUp button,X,Y [,time]
```

Description

Adds a mouse up to the current event queue.

Comments

- The QueMouseUp statement takes the following parameters:

Parameter	Description	
button	Integer specifying the mouse buttton to be released:	
	ebLeftButton	Release the left mouse button.
	ebRightButton	Release the right mouse button.
X, Y	Integer coordinates, in twips, where the mouse button is to be released.	
time	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse up will play back at full speed.	

- The QueFlush command is used to play back the events stored in the current event queue.

Example

```
'This code begins a new queue, then drags a selection over a range of
'characters in Notepad.

Sub Main()
    AppActivate "Notepad"
    QueEmpty              'Make sure the queue is empty.
    QueMouseDn ebLeftButton,1440,1393
    QueMouseUp ebLeftButton,4147,2363
    QueFlush True
End Sub
```

See Also

QueMouseClick (statement); QueMouseDn (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

statement

QueSetRelativeWindow

Syntax

```
QueSetRelativeWindow [window_object]
```

Description

Forces all subsequent QueX commands to adjust the mouse positions relative to the specified window.

Comments

- The window_object parameter is an object of type HWND. If window_object is Nothing or omitted, then the window with the focus is used (i.e., the active window).
- The QueFlush command is used to play back the events stored in the current event queue.

Example

```
Sub Main()
    'Adjust mouse coordinates relative to Notepad.
    Dim a As HWND
    Set a = WinFind("Notepad")
    QueSetRelativeWindow a
End Sub
```

function

Random

Syntax

```
Random(min,max)
```

Description

Returns a Long value greater than or equal to min and less than or equal to max.

Comments

- Both the min and max parameters are rounded to Long. A runtime error is generated if min is greater than max.

Example

```
'This example sets the randomize seed then generates six random 'numbers between 1 and 54 for the lottery.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a%(5)
    Randomize

    For x = 0 To 5
        temp = Random(1,54)

        'Elimininate duplicate numbers.
        For y = 0 To 5
            If a(y) = temp Then found = true
        Next

        If found = false Then a(x) = temp Else    x = x - 1

        found = false
    Next

    ArraySort a

    For x = 0 To 5
        msg = msg & a(x) & crlf
    Next x

    MsgBox "Today's winning lottery numbers are: " & crlf & crlf & msg
End Sub
```

See Also

Randomize (statement); Random (function).

statement

Randomize

Syntax

```
Randomize [seed]
```

Description

Initializes the random number generator with a new seed.

Comments

- If seed is not specified, then the current value of the system clock is used.

Example

```
'This example sets the randomize seed then generates six random 'numbers between 1 and 54 for the lottery.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a%(5)
    Randomize  'This sets the random seed.
               'Omitting this line will cause the random numbers to be
               'identical each time the sample is run.

    For x = 0 To 5
        temp = Rnd(1) * 54 + 1

        'Elimininate duplicate numbers.
        For y = 0 To 5
            If a(y) = temp Then found = true
        Next

        If found = false Then a(x) = temp Else    x = x - 1

        found = false
    Next

    ArraySort a

    For x = 0 To 5
        msg = msg & a(x) & crlf
    Next x

    MsgBox "Today's winning lottery numbers are: " & crlf & crlf & msg
End Sub
```

See Also

Random (function); Rnd (function).

function

Rate

Syntax

```
Rate(NPer,Pmt,Pv,Fv,Due,Guess)
```

Description

Returns the rate of interest for each period of an annuity.

Comments

- An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.
- The Rate function requires the following parameters:

Parameter	Description
NPer	Double representing the total number of payments in the annuity.
Pmt	Double representing the amount of each payment per period.
Pv	Double representing the present value of your annuity. In a loan situation, the present value would be the amount of the loan.
Fv	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be zero.
Due	Integer specifying when the payments are due for each payment period. A 0 indicates payment at the end of each period, whereas a 1 indicates payment at the start of each period.
Guess	Double specifying a guess as to the value the Rate function will return. The most common guess is .1 (10 percent).

- Positive numbers represent cash received, whereas negative values represent cash paid out.
- The value of Rate is found by iteration. It starts with the value of Guess and cycles through the calculation adjusting Guess until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, Rate fails, and the user must pick a better guess.

Example

```
'This example calculates the rate of interest necessary to save $8,000
'by paying $200 each year for 48 years. The guess rate is 10%.

Sub Main()
    r# = Rate(48,-200,8000,0,1,.1)
    MsgBox "The rate required is: " & Format(r#,"Percent")
End Sub
```

See Also

IPmt (function); NPer (function); Pmt (function); PPmt (function).

function

ReadIni$

Syntax

```
ReadIni$(section$,item$[,filename$])
```

Description

Returns a String containing the specified item from an ini file.

Comments

- The ReadIni$ function takes the following parameters:

Parameter	Description
section$	String specifying the section that contains the desired variable, such as "windows". Section names are specified without the enclosing brackets.
item$	String specifying the item whose value is to be retrieved.
filename$	String containing the name of the ini file to read.

See Also

WriteIni (statement); ReadIniSection (statement).

Notes:,

Under Windows, if the name of the ini file is not specified, then win.ini is assumed.

If the filename$ parameter does not include a path, then this statement looks for ini files in the Windows directory.

statement

ReadIniSection

Syntax

```
ReadIniSection section$,ArrayOfItems()[,filename$]
```

Description

Fills an array with the item names from a given section of the specified ini file.

Comments

- The ReadIniSection statement takes the following parameters:

Parameter	Description
section$	String specifying the section that contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.
ArrayOfItems()	Specifies either a zero- or a one-dimensioned array of strings or variants. The array can be either dynamic or fixed. If ArrayOfItems() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound, UBound, and ArrayDims functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.
filename$	String containing the name of an ini file.

- On return, the ArrayOfItems() parameter will contain one array element for each variable in the specified ini section.

Example

```
Sub Main()
    Dim items() As String
    ReadIniSection "Windows",items$
    r% = SelectBox("INI Items",,items$)
End Sub
```

See Also

ReadIni$ (function); WriteIni (statement).

Note

- Under Windows, if the name of the ini file is not specified, then win.ini is assumed.
- If the filename$ parameter does not include a path, then this statement looks for ini files in the Windows directory.

statement

Redim

Syntax

```
Redim [Preserve] variablename (subscriptRange) [As type],...
```

Description

Redimensions an array, specifying a new upper and lower bound for each dimension of the array.

Comments

- The variablename parameter specifies the name of an existing array (previously declared using the Dim statement) or the name of a new array variable. If the array variable already exists, then it must previously have been declared with the Dim statement with no dimensions, as shown in the following example:

```
Dim a$() 'Dynamic array of strings (no dimensions yet)
```

- Dynamic arrays can be redimensioned any number of times.
- The subscriptRange parameter specifies the new upper and lower bounds for each dimension of the array using the following syntax:

```
[lower To] upper [,[lower To] upper]...
```

- If lower is not specified, then 0 is used (or the value set using the Option Base statement). A runtime error is generated if lower is less than upper. Array dimensions must be within the following range:

```
-32768 <= lower <= upper <= 32767
```

- The type parameter can be used to specify the array element type. Arrays can be declared using any fundamental data type, user-defined data types, and objects.
- Redimensioning an array erases all elements of that array unless the Preserve keyword is specified. When this keyword is specified, existing data in the array is preserved where possible. If the number of elements in an array dimension is increased, the new elements are initialized to 0 (or empty string). If the number of elements in an array dimension is decreased, then the extra elements will be deleted. If the Preserve keyword is specified, then the number of dimensions of the array being redimensioned must either be zero or the same as the new number of dimensions.

Example

```
'This example uses the FileList statement to redim an array and fill it
'with filename strings. A new array is then redimmed to hold the
'number of elements found by FileList, and the FileList array is
'copied into it and partially displayed.

Sub Main()
    Dim fl$()
    FileList fl$,"*.*"
    count = Ubound(fl$)
    Redim nl$(Lbound(fl$) To Ubound(fl$))
    For x = 1 to count
        nl$(x) = fl(x)
    Next x
    MsgBox "The last element of the new array is: " & nl$(count)
End Sub
```

See Also

Dim (statement); Public (statement); Private (statement); ArrayDims (function); LBound (function); UBound (function).

statement

Rem

Syntax

```
Rem text
```

Description

Causes the compiler to skip all characters on that line.

Example

```
Sub Main()
    Rem This is a line of comments that serves to illustrate the
    Rem workings of the code. You can insert comments to make it more
    Rem readable and maintainable in the future.
End Sub
```

See Also

' (keyword); Comments (topic).

statement

Reset

Syntax

Reset

Description

Closes all open files, writing out all I/O buffers.

Example

```
'This example opens a file for output, closes it with the Reset statement,
'then deletes it with the Kill statement.

Sub Main()
    Open "test.dat" for Output Access Write as # 1
    Reset
    Kill "test.dat"

    If FileExists("test.dat") Then
        MsgBox "The file was not deleted."
    Else
        MsgBox "The file was deleted."
    End If
End Sub
```

See Also

Close (statement); Open (statement).

statement

Resume

Syntax

```
Resume {[0] | Next | label}
```

Description

Ends an error handler and continues execution.

Comments

- The form Resume 0 (or simply Resume by itself) causes execution to continue with the statement that caused the error.
- The form Resume Next causes execution to continue with the statement following the statement that caused the error.
- The form Resume label causes execution to continue at the specified label.
- The Resume statement resets the error state. This means that, after executing this statement, new errors can be generated and trapped as normal.

Example

'This example accepts two integers from the user and attempts to 'multiply the numbers together. If either number is larger than an 'integer, the program processes an error routine and then continues 'program execution at a specific section using 'Resume <label>'. 'Another error trap is then set using 'Resume Next'. The new error 'trap will clear any previous error branching and also 'tell' the 'program to continue execution of the program even if an error is 'encountered.

```
Sub Main()
    Dim a%,b%,x%

Again:
    On Error Goto Overflow
    a% = InputBox("Enter 1st integer to multiply","Enter Number")
    b% = InputBox("Enter 2nd integer to multiply","Enter Number")

    On Error Resume Next          'Continue program execution at next line
    x% = a% * b%                        'if an error (integer overflow) occurs.

    If err = 0 Then
        MsgBox a% & " * " & b% & " = " & x%
    Else
        Msgbox a% & " * " & b% & " cause an integer overflow!"
    End If

    Exit Sub

Overflow:                                   'Error handler.
    MsgBox "You've entered a non-integer value, try again!"
    Resume Again
End Sub
```

See Also

Error Handling (topic); On Error (statement).

statement

Return

Syntax

Return

Description

Transfers execution control to the statement following the most recent GoSub.

Comments

- A runtime error results if a Return statement is encountered without a corresponding GoSub statement.

Example

```
'This example calls a subroutine and then returns execution to the Main
'routine by the Return statement.

Sub Main()
    GoSub SubTrue
    MsgBox "The Main routine continues here."
    Exit Sub

SubTrue:
    MsgBox "This message is generated in the subroutine."
    Return
    Exit Sub
End Sub
```

See Also

GoSub (statement).

function

Right, Right$

Syntax

```
Right[$](text,NumChars)
```

Description

Returns the rightmost NumChars characters from a specified string.

Comments

- Right$ returns a String, whereas Right returns a String variant.
- The Right function takes the following parameters:

Parameter	Description
text	String from which characters are returned. A runtime error is generated if text is Null.
NumChars	Integer specifying the number of characters to return. If NumChars is greater than or equal to the length of the string, then the entire string is returned. If NumChars is 0, then a zero-length string is returned.

Example

```
'This example shows the Right$ function used in a routine to change
'uppercase names to lowercase with an uppercase first letter.

Sub Main()
    lname$ = "WILLIAMS"
    x = Len(lname$)
    rest$ = Right(lname$,x - 1)
    fl$ =   Left(lname$,1)
    lname$ = fl$ & LCase(rest$)
    MsgBox "The converted name is: " & lname$
End Sub
```

See Also

Left, Left$ (functions).

statement

RmDir

Syntax

```
RmDir dir$
```

Comments

- Removes the directory specified by the String contained in dir$.

Example

'This routine creates a directory and then deletes it with RmDir.

```
Sub Main()
    On Error Goto ErrMake
    MkDir("test01")
    On Error Goto ErrRemove
    RmDir("test01")

ErrMake:
    MsgBox "The directory could not be created."
    Exit Sub

ErrRemove:
    MsgBox "The directory could not be removed."
    Exit Sub
End Sub
```

See Also

ChDir (statement); ChDrive (statement); CurDir, CurDir$ (functions); Dir, Dir$ (functions); MkDir (statement).

Note

- This command behaves the same as the DOS "rd" command.

function

Rnd

Syntax

```
Rnd[(number)]
```

Description

Returns a random Single number between 0 and 1.

Comments

- If number is omitted, the next random number is returned. Otherwise, the number parameter has the following meaning:

If	Then
number < 0	Always returns the same number.
number = 0	Returns the last number generated.
number > 0	Returns the next random number.

Example

```
'This example sets the randomize seed then generates six random 'numbers between 1 and 54 for the lottery.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a%(5)
    Randomize

    For x = 0 To 5
        temp = Rnd(1) * 54 + 1

        'Elimininate duplicate numbers.
        For y = 0 To 5
            If a(y) = temp Then found = true
        Next

        If found = false Then a(x) = temp Else    x = x - 1

        found = false
    Next

    ArraySort a

    For x = 0 To 5
        msg = msg & a(x) & crlf
    Next x

    MsgBox "Today's winning lottery numbers are: " & crlf & crlf & msg
End Sub
```

See Also

Randomize (statement); Random (function).

statement

RSet

Syntax

```
RSet destvariable = source
```

Description

Copies the source string source into the destination string destvariable.

Comments

- If source is shorter in length than destvariable, then the string is right-aligned within destvariable and the remaining characters are padded with spaces. If source is longer in length than destvariable, then source is truncated, copying only the leftmost number of characters that will fit in destvariable. A runtime error is generated if source is Null.
- The destvariable parameter specifies a String or Variant variable. If destvariable is a Variant containing Empty, then no characters are copied. If destvariable is not convertible to a String, then a runtime error occurs. A runtime error results if destvariable is Null.

Example

```
'This example replaces a 40-character string of asterisks (*) with
'an RSet and LSet string and then displays the result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim msg,tmpstr$
    tmpstr$ = String(40,"*")
    msg = "Here are two strings that have been right-" + crlf
    msg = msg & "and left-justified in a 40-character string."
    msg = msg & crlf & crlf
    RSet tmpstr$ = "Right|"
    msg = msg & tmpstr$ & crlf
    LSet tmpstr$ = "|Left"
    msg = msg & tmpstr$ & crlf
    MsgBox msg
End Sub
```

See Also

LSet (statement).

function

RTrim, RTrim$

Syntax

```
RTrim[$](text)
```

Description

Returns a string with the trailing spaces removed.

Comments

- RTrim$ returns a String, whereas RTrim returns a String variant.
- Null is returned if text is Null.

Example

```
'This example displays a left-justified string and its RTrim result.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    txt$ = "        This is text                "
    tr$ = RTrim(txt$)
    MsgBox "Original ->" & txt$ & "<-" & crlf & "Right Trimmed ->" & tr$ & "<-"
End Sub
```

See Also

LTrim, LTrim$ (functions); Trim, Trim$ (functions).

function

SaveFilename$

Syntax

```
SaveFilename$[([title$ [,extensions$]])]
```

Description

Displays a dialog box that prompts the user to select from a list of files and returns a String containing the full path of the selected file.

Comments

- The SaveFilename$ function accepts the following parameters:

Parameter	Description
title$	String containing the title that appears on the dialog box's caption. If this string is omitted, then "Save As" is used.
extensions$	String containing the available file types. Its format depends on the platform on which Delrina Basic is running. If this string is omitted, then all files are used.

- The SaveFilename$ function returns a full pathname of the file that the user selects. A zero-length string is returned if the user selects Cancel. If the file already exists, then the user is prompted to overwrite it.

```
e$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
f$ = SaveFilename$("Save Picture",e$)
```

Example

```
'This example creates a save dialog box, giving the user the
'ability to save to several different file types.

Sub Main()
    e$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
    f$ = SaveFilename$("Save Picture",e$)
    If Not f$ = "" Then
        Msgbox "User choose to save file as: " + f$
    Else
        Msgbox "User canceled."
    End IF
End Sub
```

See Also

MsgBox (statement); AskBox$ (function); AskPassword$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SelectBox (function); AnswerBox (function).

Note

- Under Windows the extensions$ parameter must be in the following format:

```
description:ext[,ext][;description:ext[,ext]]...
```

Placeholder	Description
description	Specifies the grouping of files for the user, such as All Files.
ext	Specifies a valid file extension, such as *.BAT or *.?F?.

For example, the following are valid extensions$ specifications:

```
"All Files:*"
"Documents:*.TXT,*.DOC"
"All Files:*;Documents:*.TXT,*.DOC"
```

property

Screen.DlgBaseUnitsX

Syntax

Screen.DlgBaseUnitsX

Description

Returns an Integer used to convert horizontal pixels to and from dialog units.

Comments

- The number returned depends on the name and size of the font used to display dialog boxes.
- To convert from pixels to dialog units in the horizontal direction:

  ((XPixels * 4) + (**Screen.DlgBaseUnitsX** - 1)) / **Screen.DlgBaseUnitsX**
- To convert from dialog units to pixels in the horizontal direction:

  (XDlgUnits * **Screen.DlgBaseUnitsX**) / 4

Example

```
'This example converts the screen width from pixels to dialog units.

Sub Main()
    XPixels = Screen.Width
    conv% = Screen.DlgBaseUnitsX
    XDlgUnits = (XPixels * 4) + (conv% -1) / conv%
    MsgBox "The screen width is " & XDlgUnits & " dialog units."
End Sub
```

See Also

Screen.DlgBaseUnitsY (property).

property

Screen.DlgBaseUnitsY

Syntax

Screen.DlgBaseUnitsY

Description

Returns an Integer used to convert vertical pixels to and from dialog units.

Comments

- The number returned depends on the name and size of the font used to display dialog boxes.
- To convert from pixels to dialog units in the vertical direction:

  (YPixels * 8) + (**Screen.DlgBaseUnitsY** - 1) / **Screen.DlgBaseUnitsY**
- To convert from dialog units to pixels in the vertical direction:

  (YDlgUnits * **Screen.DlgBaseUnitsY**) / 8

Example

```
'This example converts the screen width from pixels to dialog units.

Sub Main()
    YPixels = Screen.Height
    conv% = Screen.DlgBaseUnitsY
    YDlgUnits = (YPixels * 8) + (conv% -1) / conv%
    MsgBox "The screen width is " & YDlgUnits & " dialog units."
End Sub
```

See Also

Screen.DlgBaseUnitsX (property).

property

Screen.Height

Syntax

Screen.Height

Description

Returns the height of the screen in pixels as an Integer.

Comments

- This property is used to retrieve the height of the screen in pixels. This value will differ depending on the display resolution.
- This property is read-only.

Example

```
'This example displays the screen height in pixels.

Sub Main()
    MsgBox "The Screen height is " & Screen.Height & " pixels."
End Sub
```

See Also

Screen.Width (property).

property

Screen.TwipsPerPixelX

Syntax

Screen.TwipsPerPixelX

Description

Returns an Integer representing the number of twips per pixel in the horizontal direction of the installed display driver.

Comments

- This property is read-only.

Example

```
'This example displays the number of twips across the screen horizontally.

Sub Main()
    XScreenTwips = Screen.Width * Screen.TwipsPerPixelX
    MsgBox "Total horizontal screen twips = " & XScreenTwips
End Sub
```

See Also

Screen.TwipsPerPixelY (property).

property

Screen.TwipsPerPixelY

Syntax

Screen.TwipsPerPixelY

Description

Returns an Integer representing the number of twips per pixel in the vertical direction of the installed display driver.

Comments

- This property is read-only.

Example

```
'This example displays the number of twips across the screen vertically.

Sub Main()
    YScreenTwips = Screen.Height * Screen.TwipsPerPixelY
    MsgBox "Total vertical screen twips = " & YScreenTwips
End Sub
```

See Also

Screen.TwipsPerPixelX (property).

property

Screen.Width

Syntax

Screen.Width

Description

Returns the width of the screen in pixels as an Integer.

Comments

- This property is used to retrieve the width of the screen in pixels. This value will differ depending on the display resolution.
- This property is read-only.

Example

```
'This example displays the screen width in pixels
Sub Main()
    MsgBox "The screen width is " & Screen.Width & " pixels."
End Sub
```

See Also

Screen.Height (property).

function

Second

Syntax

```
Second(time)
```

Description

Returns the second of the day encoded in the specified time parameter.

Comments

- The value returned is an Integer between 0 and 59 inclusive.
- The time parameter is any expression that converts to a Date.

Example

'This example fires and event every 10 seconds based on the system 'clock.

```
Sub Main()
    trigger = 10
    Do
        xs% = Second(Now)
        If (xs% Mod trigger = 0) Then
            Beep
            End       'Remove this line to trigger the loop continuously.
            Sleep 1000
        End If
        DoEvents
    Loop
End Sub
```

See Also

Day (function); Minute (function); Month (function); Year (function); Hour (function); Weekday (function); DatePart (function).

function

Seek

Syntax

```
Seek(filenumber)
```

Description

Returns the position of the file pointer in a file relative to the beginning of the file.

Comments

- The filenumber parameter is a number that Delrina Basic uses to refer to the open file-the number passed to the Open statement.
- The value returned depends on the mode in which the file was opened:

File Mode	Returns
Input	Byte position for the next read
Output	Byte position for the next write
Append	Byte position for the next write
Random	Number of the next record to be written or read
Binary	Byte position for the next read or write

- The value returned is a Long between 1 and 2147483647, where the first byte (or first record) in the file is 1.

Example

```
'This example opens a file for random write, then writes ten
'records into the file using the PUT statement. The file position is
'displayed using the Seek Function, and the file is closed.

Sub Main()
    Open "test.dat" For Random Access Write As #1
    For x = 1 To 10
        r% = x * 10
        Put #1,x,r%
    Next x
    y = Seek(1)
    MsgBox "The current file position is: " & y
    Close
End Sub
```

See Also

Seek (statement); Loc (function).

statement

Seek

Syntax

```
Seek [#] filenumber,position
```

Description

Sets the position of the file pointer within a given file such that the next read or write operation will occur at the specified position.

Comments

- The Seek statement accepts the following parameters:

Parameter	Description
filenumber	Integer used by Delrina Basic to refer to the open file-the number passed to the Open statement.
position	Long that specifies the location within the file at which to position the file pointer. The value must be between 1 and 2147483647, where the first byte (or record number) in the file is 1. For files opened in either Binary, Output, Input, or Append mode, position is the byte position within the file. For Random files, position is the record number.

- A file can be extended by seeking beyond the end of the file and writing data there.

Example

```
'This example opens a file for random write, then writes ten
'records into the file using the PUT statement. The file is
'then reopened for read, and the ninth record is read using
'the Seek and Get functions.

Sub Main()
    Open "test.dat" For Random Access Write As #1
    For x = 1 To 10
        rec$ = "Record#: " & x
        Put #1,x,rec$
    Next x
    Close

    Open "test.dat" For Random Access Read As #1
    Seek #1,9
    Get #1,,rec$
    MsgBox "The ninth record = " & x
    Close
    Kill "test.dat"
End Sub
```

See Also

Seek (function); Loc (function).

statement

Select...Case

Syntax

```
Select Case testexpression
[Case expressionlist
  [statement_block]]
[Case expressionlist
  [statement_block]]
     .
     .
[Case Else
  [statement_block]]
End Select
```

Description

Used to run a block of Delrina Basic statements depending on the value of a given expression.

Comments

- The Select Case statement has the following parts:

Part	Description
testexpression	Any numeric or string expression.
statement_block	Any group of Delrina Basic statements. If the testexpression matches any of the expressions contained in expressionlist, then this statement block will be run.
expressionlist	A comma separated list of expressions to be compared against testexpression using any of the following syntaxes: expression [,expression]... expression to expression is relational_operator expression The resultant type of expression in expressionlist must be the same as that of testexpression.

- Multiple expression ranges can be used within a single Case clause. For example:

```
Case 1 to 10,12,15 Is > 40
```

- Only the statement_block associated with the first matching expression will be run. If no matching statement_block is found, then the statements following the Case Else will be run.
- A Select...End Select expression can also be represented with the If...Then expression. The use of the Select statement, however, may be more readable.

Example

```
'This example uses the Select...Case statement to output the
'current operating system.
```

```
Sub Main()
    OpSystem% = Basic.OS
    Select Case OpSystem%
        Case 0,2
            s = "Microsoft Windows"
        Case 1
            s = "DOS"
        Case 3 to 8,12
            s = "UNIX"
        Case 10
            s = "IBM OS/2"
        Case Else
            s = "Other"
    End Select
    MsgBox "This version of Delrina Basic is running on: " & s
End Sub
```

See Also

Choose (function); Switch (function); IIf (function); If...Then...Else (statement).

function

SelectBox

Syntax

```
SelectBox(title,prompt,ArrayOfItems)
```

Description

Displays a dialog box that lets the user to select from a list of choices and returns an Integer containing the index of the item that was selected.

Comments

- The SelectBox statement accepts the following parameters:

Parameter	Description
title	Title of the dialog box. This can be an expression convertible to a String. A runtime error is generated if title is Null.
prompt	Text to appear immediately above the list box containing the items. This can be an expression convertible to a String. A runtime error is generated if prompt is Null.
ArrayOfItems	Single-dimensioned array. Each item from the array will occupy a single entry in the list box. A runtime error is generated if ArrayOfItems is not a single-dimensioned array. ArrayOfItems can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.

- The value returned is an Integer representing the index of the item in the list box that was selected, with 0 being the first item. If the user selects Cancel, -1 is returned.

result% = **SelectBox(**"Picker"**,**"Pick an application:"**,**a$**)**

Example

```
'This example gets the current apps running, puts them in to an array
'and then asks the user to select one from a list.

Sub Main()
    Dim a$()
    AppList a$
    result% = SelectBox("Picker","Pick an application:",a$)
    If Not result% = -1 then
        Msgbox "User selected: " & a$(result%)
    Else
        Msgbox "User canceled"
    End If
End Sub
```

See Also

MsgBox (statement); AskBox$ (function); AskPassword$ (function); InputBox, InputBox$ (functions); OpenFilename$ (function); SaveFilename$ (function); AnswerBox (function).

Note

- SelectBox displays all text in its dialog box in 8-point MS Sans Serif.

statement

SelectButton

Syntax

```
SelectButton name$ | id
```

Description

Simulates a mouse click on the a push button given the push button's name (the name$ parameter) or ID (the id parameter).

Comments

- The SelectButton statement accepts the following parameters:

Parameter	Description
name$	String containing the name of the push button to be selected.
id	Integer representing the ID of the push button to be selected.
	A runtime error is generated if a push button with the given name or ID cannot be found in the active window.

Note: The SelectButton statement is used to select a button in another application's dialog box. This command is not intended for use with built-in or dynamic dialog boxes.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SelectButton "Setup..."
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

ButtonEnabled (function), ButtonExists (function)

statement

SelectComboBoxItem

Syntax

```
SelectComboBoxItem {name$ | id},{ItemName$ | ItemNumber} [,isDoubleClick]
```

Description

Selects an item from a combo box given the name or ID of the combo box and the name or line number of the item.

Comments

- The SelectComboBoxItem statement accepts the following parameters:

Parameter	Description
name$	String indicating the name of the combo box containing the item to be selected. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
id	Integer specifying the ID of the combo box containing the item to be selected.
ItemName$	String specifying which item is to be selected. The string is compared without regard to case. If ItemName$ is a zero-length string, then all currently selected items are deselected. A runtime error results if ItemName$ cannot be found in the combo box.
ItemNumber	Integer containing the index of the item to be selected. A runtime error is generated if ItemNumber is not within the correct range.
isDoubleClick	Boolean value indicating whether a double click of that item is to be simulated.

Note: The SelectComboBoxItem statement is used to set the item of a combo box in another application's dialog box. Use the DlgText statement to change the content of the text box part of a list box in a dynamic dialog box.

Example

```
'This code fragment simulates the selection of a couple of comboboxes

    SelectComboBoxItem "ComboBox1","Item4"
    SelectComboBoxItem 1,2,True
```

See Also

ComboBoxEnabled (function); ComboBoxExists (function); GetComboBoxItem$ (function); GetComboBoxItemCount (function).

statement

SelectListBoxItem

Syntax

```
SelectListBoxItem {name$ | id},{ItemName$ | ItemNumber} [,isDoubleClick]
```

Description

Selects an item from a list box given the name or ID of the list box and the name or line number of the item.

Comments

- The SelectListBoxItem statement accepts the following parameters:

Parameter	Description
name$	String indicating the name of the list box containing the item to be selected. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
id	Integer specifying the ID of the list box containing the item to be selected.
ItemName$	String specifying which item is to be selected. The string is compared without regard to case. If ItemName$ is a zero-length string, then all currently selected items are deselected. A runtime error results if ItemName$ cannot be found in the list box.
ItemNumber	Integer containing the index of the item to be selected. A runtime error is generated if ItemNumber is not within the correct range.
isDoubleClick	Boolean value indicating whether a double click of that item is to be simulated.

The list box must exist within the current window or dialog box; otherwise, a runtime error will be generated.

- For multiselect list boxes, SelectListBoxItem will select additional items (i.e., it will not remove the selection from the currently selected items).

Note: The SelectListBoxItem statement is used to select an item in a list box of another application's dialog box. Use the DlgText statement to change the selected item in a list box within a dynamic dialog box.

Example

```
'This code fragment simulates a double click on the first item in list box 1.

SelectListBoxItem "ListBox1",1,True
```

See Also

GetListBoxItem$ (function); GetListBoxItemCount (function); ListBoxEnabled (function); ListBoxExists (function).

statement

SendKeys

Syntax

```
SendKeys KeyString$ [,[isWait] [,time]]
```

Description

Sends the specified keys to the active application, optionally waiting for the keys to be processed before continuing.

Comments

- The SendKeys statement accepts the following parameters:

Parameter	Description
KeyString$	String containing the keys to be sent. The format for KeyString$ is described below.
isWait	Boolean value. If True (or not specified), then Delrina Basic waits for the keys to be completely processed before continuing.
time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString$ parameter. It must be within the following range: 0 <= time <= 32767 For example, if time is 5000 (5 seconds) and the KeyString$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

Specifying Keys

To specify any key on the keyboard, simply use that key, such as "a" for lowercase a, or "A" for uppercase a.

Sequences of keys are specified by appending them together: "abc" or "dir /w".

Some keys have special meaning and are therefore specified in a special way-by enclosing them within braces. For example, to specify the percent sign, use "{%}". The following table shows the special keys:

Key	Special Meaning	Example	
+	Shift	"+{F1}"	'Shift+F1
^	Ctrl	"^a"	'Ctrl+A
~	Shortcut for Enter	"~"	'Enter
%	Alt	"%F"	'Alt+F
[]	No special meaning	"{[}"	'Open bracket
{}	Used to enclose special keys	"{Up}"	'Up Arrow
()	Used to specify grouping	"^(ab)"	'Ctrl+A, Ctrl+B

Keys that are not displayed when you press them are also specified within braces, such as {Enter} or {Up}. A list of these keys follows:

{BkSp}	{BS}	{Break}	{CapsLock}	{Clear}
{Delete}{Del}	{Down}	{End}	{Enter}	
{Escape}	{Esc}	{Help}	{Home}	{Insert}
{Left}	{NumLock}	{NumPad0}	{NumPad1}	{NumPad2}
{NumPad3}	{NumPad4}	{NumPad5}	{NumPad6}	{NumPad7}
{NumPad8}	{NumPad9}	{NumPad/}	{NumPad*}	{NumPad-}

{NumPad+}	{NumPad.}	{PgDn}	{PgUp}	{PrtSc}
{Right}	{Tab}	{Up}	{F1}	{Scroll Lock}
{F2}	{F3}	{F4}	{F5}	{F6}
{F7}	{F8}	{F9}	{F10}	{F11}
{F12}	{F13}	{F14}	{F15}	{F16}

Keys can be combined with Shift, Ctrl, and Alt using the reserved keys "+", "^", and "%" respectively:

For Key Combination	Use
Shift+Enter	"+{Enter}"
Ctrl+C	"^c"
Alt+F2	"%{F2}"

To specify a modifier key combined with a sequence of consecutive keys, group the key sequence within parentheses, as in the following example:

For Key Combination	Use
Shift+A, Shift+B	"+(abc)"
Ctrl+F1, Ctrl+F2	"^({F1}{F2})"

Use "~" as a shortcut for embedding Enter within a key sequence:

For Key Combination	Use
a, b, Enter, d, e	"ab~de"
Enter, Enter	"~~"

To embed quotation marks, use two quotation marks in a row:

For Key Combination	Use
"Hello"	""Hello""
a"b"c	"a""b""c"

Key sequences can be repeated using a repeat count within braces:

For Key Combination	Use
Ten "a" keys	"{a 10}"
Two Enter keys	"{Enter 2}"

Example

```
'This example runs Notepad, writes to Notepad, and saves the new file using
'the SendKeys statement.

Sub Main()
    id = Shell("Notepad.exe")
    AppActivate "Notepad"
    SendKeys "Hello, Notepad."        'Write some text.
    Sleep 2000
    SendKeys "%fs"                    'Save file (simulate Alt+F,S keys).
    Sleep 2000
    SendKeys "name.txt{ENTER}"        'Enter name of new file to save.
    AppClose "Notepad"
End Sub
```

See Also

DoKeys (statement); QueKeys (statement); QueKeyDn (statement); QueKeyUp (statement).

statement

Set

Syntax 1

```
Set object_var = object_expression
```

Syntax 2

```
Set object_var = New object_type
```

Syntax 3

```
Set object_var = Nothing
```

Description

Assigns a value to an object variable.

Comments

Syntax 1

The first syntax assigns the result of an expression to an object variable. This statement does not duplicate the object being assigned but rather copies a reference of an existing object to an object variable.

The object_expression is any expression that evaluates to an object of the same type as the object_var.

With data objects, Set performs additional processing. When the Set is performed, the object is notified that a reference to it is being made and destroyed. For example, the following statement deletes a reference to object A, then adds a new reference to B.

```
Set a = b
```

In this way, an object that is no longer being referenced can be destroyed.

Syntax 2

In the second syntax, the object variable is being assigned to a new instance of an existing object type. This syntax is valid only for data objects.

When an object created using the New keyword goes out of scope (i.e., the Sub or Function in which the variable is declared ends), the object is destroyed.

Syntax 3

The reserved keyword Nothing is used to make an object variable reference no object. At a later time, the object variable can be compared to Nothing to test whether the object variable has been instantiated:

```
Set a = Nothing
    :
If a Is Nothing Then Beep
```

Example

'This example creates two objects and sets their values.

```
Sub Main()
    Dim document As Object
    Dim page As Object
    Set document = GetObject("c:\resume.doc")
    Set page = Document.ActivePage
    MsgBox page.name
End Sub
```

See Also

= (statement); Let (statement); CreateObject (function); GetObject (function); Nothing (constant).

statement

SetAttr

Syntax

```
SetAttr filename$,attribute
```

Description

Changes the attribute filename$ to the given attribute. A runtime error results if the file cannot be found.

Comments

- The SetAttr statement accepts the following parameters:

Parameter	Description
filename$	String containing the name of the file.
attribute	Integer specifying the new attribute of the file.

- The attribute parameter can contain any combination of the following values:

Constant	Value	Description
ebNormal	0	Turns off all attributes
ebReadOnly	1	Read-only files
ebHidden	2	Hidden files
ebSystem	4	System files
ebVolume	8	Volume label
ebArchive	32	Files that have changed since the last backup
ebNone	64	Turns off all attributes

- The attributes can be combined using the + operator or the binary Or operator.

Example

```
'This example creates a file and sets its attributes to Read-Only and
'System.

Sub Main()
    Open "test.dat" For Output As #1
    Close #1
    MsgBox "The current file attribute is: " & GetAttr("test.dat")
    SetAttr "test.dat",ebReadOnly + ebSystem
    MsgBox "The file attribute was set to: " & GetAttr("test.dat")
    SetAttr "test.dat",ebNormal
    Kill "test.dat"
End Sub
```

See Also

GetAttr (function); FileAttr (function).

Note

- These attributes are the same as those used by DOS.

statement

SetCheckBox

Syntax

```
SetCheckBox {name$ | id},state
```

Description

Sets the state of the check box with the given name or ID.

Comments

- The SetCheckBox statement accepts the following parameters:

Parameter	Description
name$	String containing the name of the check box to be set.
id	Integer specifying the ID of the check box to be set.
state	Integer indicating the new state of the check box. If state is 1, then the box is checked. If state is 0, then the check is removed. If state is 2, then the box is dimmed (only applicable for three-state check boxes).

- A runtime error is generated if a check box with the specified name cannot be found in the active window.
- This statement has the side effect of setting the focus to the given check box.

Note: The SetCheckBox statement is used to set the state of a check box in another application's dialog box. Use the DlgValue statement to modify the state of a check box within a dynamic dialog box.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If
```

```
    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each
Job",False
    End If
End Sub
```

See Also

CheckBoxExists (function); CheckBoxEnabled (function); GetCheckBox (function); DlgValue (statement).

statement

SetEditText

Syntax

```
SetEditText {name$ | id},content$
```

Description

Sets the content of an edit control given its name or ID.

Comments

- The SetEditText statement accepts the following parameters:

Parameter	Description
name$	String containing the name of the text box to be set. The name of a text box control is determined by scanning the window list looking for a text control with the given name that is immediately followed by an edit control. A runtime error is generated if a text box control with that name cannot be found within the active window.
id	Integer specifying the ID of the text box to be set. For text boxes that do not have a preceding text control, the id can be used to absolutely reference the control. The id is determined by examining the dialog box with a resource editor or using an application such as Spy.
content$	String containing the new content for the text box.

- This statement has the side effect of setting the focus to the given text box.

Note: The SetEditText statement is used to set the content of a text box in another application's dialog box. Use the DlgText statement to set the text of a text box within a dynamic dialog box.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If
```

```
    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

EditEnabled (function); EditExists (function); GetEditText$ (function).

statement

SetOption

Syntax

```
SetOption name$ | id
```

Description

Selects the specified option button given its name or ID.

Comments

- The SetOption statement accepts the following parameters:

Parameter	Description
name$	String containing the name of the option button to be selected.
id	Integer containing the ID of the option button to be selected.

- A runtime error is generated if the option button cannot be found within the active window.

Note: The SetOption statement is used to select an option button in another application's dialog box. Use the DlgValue statement to select an option button within a dynamic dialog box.

Example

```
'This example invokes the Windows For Workgroups Print Manager and
'customizes the print output for a common PostScript operation.

Sub Main()
    id = Shell("printman.exe",3)            'Activate Windows Print Manager.
    If MenuItemExists("Options.Printer Setup") And MenuItemEnabled("Options.Printer Setup") Then
        Menu "Options.Printer Setup"    'Enter setup screen.
    Else
        MsgBox "Print Manager Not Ready Or Incorrect Version!",ebExclamation,"Print Manager Error"
        End
    End If

    'Enter Setup|Options.
    If ButtonExists("Setup...") And ButtonEnabled("Setup...") Then
        SendKeys "(%S)(%O)",True
        DoEvents                                'Give window chance to display.
    End If

    'Send output to PostScript file instead of Printer.
    If OptionExists("Encapsulated PostScript File") And OptionEnabled("Encapsulated PostScript File") Then
        SetOption("Encapsulated PostScript File")
    Else
        MsgBox "Cannot Set Print Manager Options!"
    End If

    'Enter file name for output.
    If EditExists("Name:") And EditEnabled("Name:") Then
        If GetEditText$("Name:") <> "myfile.ps" Then SetEditText "Name:","myfile.ps"
    End If

    'Remove custom header with each job.
    If CheckBoxExists("Send Header With Each Job") And CheckBoxEnabled("Send Header With Each Job") Then
        If GetCheckBox("Send Header With Each Job") = 1 Then SetCheckBox "Send Header With Each Job",False
    End If
End Sub
```

See Also

GetOption (function); OptionEnabled (function); OptionExists (function).

function

Sgn

Syntax

```
Sgn(number)
```

Description

Returns an Integer indicating whether a number is less than, greater than, or equal to 0.

Comments

- Returns 1 if number is greater than 0.
- Returns 0 if number is equal to 0.
- Returns -1 if number is less than 0.
- The number parameter is a numeric expression of any type. If number is Null, then a runtime error is generated. Empty is treated as 0.

Example

```
'This example tests the product of two numbers and displays
'a message based on the sign of the result.

Sub Main()
    a% = -100
    b% = 100
    c% = a% * b%
    Select Case Sgn(c%)
        Case -1
            MsgBox "The product is negative " & Sgn(c%)
        Case 0
            MsgBox "The product is 0 " & Sgn(c%)
        Case 1
            MsgBox "The product is positive " & Sgn(c%)
    End Select
End Sub
```

See Also

Abs (function).

function

Shell

Syntax

```
Shell(command$ [,WindowStyle])
```

Description

Runs another application, returning the task ID if successful.

Comments

- The Shell statement accepts the following parameters:

Parameter	Description
command$	String containing the name of the application and any parameters.
WindowStyle	Optional Integer specifying the state of the application window after execution. It can be any of the following values: 1 Normal window with focus 2 Minimized with focus (default) 3 Maximized with focus 4 Normal window without focus 7 Minimized without focus

- An error is generated if unsuccessful running command$.
- The Shell command runs programs asynchronously: the statement following the Shell statement will run before the child application has exited. On some platforms, the next statement will run before the child application has finished loading.
- The Shell function returns a value suitable for activating the application using the AppActivate statement. It is important that this value be placed into a Variant, as its type depends on the platform.

Example

```
'This example displays the Windows Clock, delays awhile, then closes it.

Sub Main()
    id = Shell("clock.exe",1)
    AppActivate "Clock"
    Sleep(2000)
    AppClose "Clock"
End Sub
```

See Also

PrintFile (function); SendKeys (statement); AppActivate (statement).

Note

- This function returns the hInstance of the application. Since this value is only a WORD in size, the upper WORD of the result is always zero.

function

Sin

Syntax

```
Sin(angle)
```

Description

Returns a Double value specifying the sine of angle.

Comments

- The angle parameter is a Double specifying an angle in radians.

Example

```
'This example displays the sine of pi/4 radians (45 degrees).

Sub Main()
    c# = Sin(Pi / 4)
    MsgBox "The sine of 45 degrees is: " & c#
End Sub
```

See Also

Tan (function); Cos (function); Atn (function).

data type

Single

Syntax

Single

Description

A data type used to declare variables capable of holding real numbers with up to seven digits of precision.

Comments

- Single variables are used to hold numbers within the following ranges:

Sign	Range
Negative	-3.402823E38 <= single <= -1.401298E-45
Positive	1.401298E-45 <= single <= 3.402823E38

- The type-declaration character for Single is !.

Storage

Internally, singles are stored as 4-byte (32-bit) IEEE values. Thus, when appearing within a structure, singles require 4 bytes of storage. When used with binary or random files, 4 bytes of storage is required.

Each single consists of the following:

- A 1-bit sign
- An 8-bit exponent
- A 24-bit mantissa

See Also

Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CSng (function).

statement

Sleep

Syntax

```
Sleep milliseconds
```

Description

Causes the script to pause for a specified number of milliseconds.

Comments

- The milliseconds parameter is a Long in the following range:
  `0 <= milliseconds <= 2,147,483,647`

Example

```
'This example displays a message for 2 seconds.

Sub Main()
    MsgOpen "Waiting 2 seconds",0,False,False
    Sleep 2000
    MsgClose
End Sub
```

Note

- The accuracy of the system clock is modulo 55 milliseconds. The value of milliseconds will, in the worst case, be rounded up to the nearest multiple of 55. In other words, if milliseconds is 1, it will be rounded to 55 in the worst case.

function

Sln

Syntax

```
Sln(Cost,Salvage,Life)
```

Description

Returns the straight-line depreciation of an asset assuming constant benefit from the asset.

Comments

- The Sln of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers.
- The formula used to find the Sln of an asset is as follows:

  (Cost - Salvage Value) / Useful Life
- The Sln function requires the following parameters:

Parameter	Description
Cost	Double representing the initial cost of the asset.
Salvage	Double representing the estimated value of the asset at the end of its useful life.
Life	Double representing the length of the asset's useful life.

- The unit of time used to express the useful life of the asset is the same as the unit of time used to express the period for which the depreciation is returned.

Example

```
'This example calculates the straight-line depreciation of an asset
'that cost $10,000.00 and has a salvage value of $500.00 as scrap
'after 10 years of service life.

Sub Main()
    dep# = Sln(10000.00,500.00,10)
    MsgBox "The annual depreciation is: " & Format(dep#,"Currency")
End Sub
```

See Also

SYD (function); DDB (function).

function

Space, Space$

Syntax

```
Space[$](NumSpaces)
```

Description

Returns a string containing the specified number of spaces.

Comments

- Space$ returns a String, whereas Space returns a String variant.
- NumSpaces is an Integer between 0 and 32767.

Example

```
'This example returns a string of ten spaces and displays it.

Sub Main()
    ln$ = Space(10)
    MsgBox "Hello" & ln$ & "over there."
End Sub
```

See Also

String, String$ (functions); Spc (function).

function

Spc

Syntax

```
Spc(numspaces)
```

Description

Prints out the specified number of spaces. This function can only be used with the Print and Print# statements.

Comments

- The numspaces parameter is an Integer specifying the number of spaces to be printed. It can be any value between 0 and 32767.
- If a line width has been specified (using the Width statement), then the number of spaces is adjusted as follows:

  numspaces = numspaces Mod width

- If the resultant number of spaces is greater than width - print_position, then the number of spaces is recalculated as follows:

  numspaces = numspaces - (width - print_position)

- These calculations have the effect of never letting the spaces to overflow the line length. Furthermore, with a large value for column and a small line width, the file pointer will never advance more than one line.

Example

```
'This example displays 20 spaces between the arrows.

Sub Main()
    ViewportOpen
    Print "I am"; Spc(20); "20 spaces apart!"
    Sleep (10000)     'Wait 10 seconds.
    ViewportClose
End Sub
```

See Also

Tab (function); Print (statement); Print# (statement).

function

SQLBind

Syntax

```
SQLBind(ID,array,column)
```

Description

Specifies which fields are returned when results are requested using the SQLRetrieve or SQLRetrieveToFile function.

Comments

- The following table describes the parameters to the SQLBind function:

Parameter	Description
ID	Long parameter specifying a valid connection.
array	Any array of variants. Each call to SQLBind adds a new column number (an Integer) in the appropriate slot in the array. Thus, as you bind additional columns, the array parameter grows, accumulating a sorted list (in ascending order) of bound columns. If array is fixed, then it must be a one-dimensional variant array with sufficient space to hold all the bound column numbers. A runtime error is generated if array is too small. If array is dynamic, then it will be resized to exactly hold all the bound column numbers.
column	Optional Long parameter that specifies the column to which to bind data. If this parameter is omitted, all bindings for the connection are dropped. This function returns the number of bound columns on the connection. If no columns are bound, then 0 is returned. If there are no pending queries, then calling SQLBind will cause an error (queries are initiated using the SQLExecQuery function). If supported by the driver, row numbers can be returned by binding column 0.

- Delrina Basic generates a trappable runtime error if SQLBind fails. Additional error information can then be retrieved using the SQLError function.

Example

```
'This example binds columns to data.

Sub Main()
    Dim columns() As Variant
    id& = SQLOpen("dsn=SAMPLE",,3)
    t& = SQLExecQuery(id&,"Select * From c:\sample.dbf")
    i% = SQLBind(id&,columns,3)
    i% = SQLBind(id&,columns,1)
    i% = SQLBind(id&,columns,2)
    i% = SQLBind(id&,columns,6)
    For x = 0 To (i% - 1)
        MsgBox columns(x)
    Next x
    id& = SQLClose(id&)
End Sub
```

See Also

SQLRetrieve (function); SQLRetrieveToFile (function).

function

SQLClose

Syntax

```
SQLClose(connectionID)
```

Description

Closes the connection to the specified data source.

Comments

- The unique connection ID (connectionID) is a Long value representing a valid connection as returned by SQLOpen. After SQLClose is called, any subsequent calls made with the connectionID will generate runtime errors.
- The SQLClose function returns 0 if successful; otherwise, it returns the passed connection ID and generates a trappable runtime error. Additional error information can then be retrieved using the SQLError function.
- Delrina Basic automatically closes all open SQL connections when either the script or the application terminates. You should use the SQLClose function rather than relying on Delrina Basic to automatically close connections in order to ensure that your connections are closed at the proper time.

Example

```
'This example disconnects the data source sample.

Sub Main()
    Dim s As String
    Dim qry As Long
    id& = SQLOpen("dsn=SAMPLE",s$,3)
    qry = SQLExecQuery(id&,"Select * From c:\sample.dbf")
    MsgBox "There are " & qry & " records in the result set."
    id& = SQLClose(id&)
End Sub
```

See Also

SQLOpen (function).

function

SQLError

Syntax

```
SQLError(ErrArray [, ID])
```

Description

Retrieves driver-specific error information for the most recent SQL functions that failed.

Comments

- This function is called after any other SQL function fails. Error information is returned in a two-dimensional array (ErrArray). The following table describes the parameters to the SQLError function:

Parameter	Description
ErrArray	Two-dimensional Variant array, which can be dynamic or fixed. If the array is fixed, it must be (x,3), where x is the number of errors you want returned. If x is too small to hold all the errors, then the extra error information is discarded. If x is greater than the number of errors available, all errors are returned, and the empty array elements are set to Empty. If the array is dynamic, it will be resized to hold the exact number of errors.
ID	Optional Long parameter specifying a connection ID. If this parameter is omitted, error information is returned for the most recent SQL function call.

- Each array entry in the ErrArray parameter describes one error. The three elements in each array entry contain the following information:

Element	Value
(entry,0)	The ODBC error state, indicated by a Long containing the error class and subclass.
(entry,1)	The ODBC native error code, indicated by a Long.
(entry,2)	The text error message returned by the driver. This field is String type. For example, to retrieve the ODBC text error message of the first returned error, the array is referenced as: ErrArray(0,2) The SQLError function returns the number of errors found.

- Delrina Basic generates a runtime error if SQLError fails. (You cannot use the SQLError function to gather additional error information in this case.)

Example

```
'This example forces a connection error and traps it for use with
'the SQLError function.
```

```
Sub Main()
    Dim a() As Variant
    On Error Goto Trap
    id& = SQLOpen("",,4)
    id& = SQLClose(id&)
    Exit Sub

Trap:
    rc% = SQLError(a)
    If (rc%) Then
        For x = 0 To (rc% - 1)
            MsgBox "The SQL state returned was: " & a(x,0)
            MsgBox "The native error code returned was: " & a(x,1)
            MsgBox a(x,2)
        Next x
    End If
End Sub
```

function

SQLExecQuery

Syntax

```
SQLExecQuery(ID, query$)
```

Description

Runs an SQL statement query on a data source.

Comments

- This function is called after a connection to a data source is established using the SQLOpen function. The SQLExecQuery function may be called multiple times with the same connection ID, each time replacing all results.
- The following table describes the parameters to the SQLExecQuery function:

Parameter	Description
ID	Long identifying a valid connected data source. This parameter is returned by the SQLOpen function.
query$	String specifying an SQL query statement. The SQL syntax of the string must strictly follow that of the driver.

- The return value of this function depends on the result returned by the SQL statement:

SQL Statement	Value
SELECT...FROM	The value returned is the number of columns returned by the SQL statement.
DELETE,INSERT,UPDATE	The value returned is the number of rows affected by the SQL statement.

- Delrina Basic generates a runtime error if SQLExecQuery fails. Additional error information can then be retrieved using the SQLError function.

Example

```
'This example runs a query on the connected data source.

Sub Main()
    Dim s As String
    Dim qry As Long
    id& = SQLOpen("dsn=SAMPLE",s$,3)
    qry = SQLExecQuery(id&,"Select * From c:\sample.dbf")
    MsgBox "There are " & qry & " records in the result set."
    id& = SQLClose(id&)
End Sub
```

See Also

SQLOpen (function); SQLClose (function); SQLRetrieve (function); SQLRetrieveToFile (function).

function

SQLGetSchema

Syntax

```
SQLGetSchema(ID, action, [,[array] [,qualifier$]])
```

Description

Returns information about the data source associated with the specified connection.

Comments

- The following table describes the parameters to the SQLGetSchema function:

Parameter	Description
ID	Long parameter identifying a valid connected data source. This parameter is returned by the SQLOpen function.
action	Integer parameter specifying the results to be returned. The following table lists values for this parameter:

Value	Meaning
1	Returns a one-dimensional array of available data sources. The array is returned in the array parameter.
2	Returns a one-dimensional array of databases (either directory names or database names, depending on the driver) associated with the current connection. The array is returned in the array parameter.
3	Returns a one-dimensional array of owners (user IDs) of the database associated with the current connection. The array is returned in the array parameter.
4	Returns a one-dimensional array of table names for a specified owner and database associated with the current connection. The array is returned in the array parameter.
5	Returns a two-dimentional array (n by 2) containing information about a specified table. The array is configured as follows: (0,0) Zeroth column name (0,1) ODBC SQL data type (Integer) (1,0) First column name (1,1) ODBC SQL data type (Integer) : : (n,0) Nth column name (n,1) ODBC SQL data type (Integer)
6	Returns a string containing the ID of the current user.
7	Returns a string containing the name (either the directory name or the database name, depending on the driver) of the current database.
8	Returns a string containing the name of the data source on the current connection.
9	Returns a string containing the name of the DBMS of the data source on the current connection (e.g., "FoxPro 2.5" or "Excel Files").
10	Returns a string containing the name of the server for the data source.
11	Returns a string containing the owner qualifier used by the data source (e.g., "owner," "Authorization ID," "Schema").
12	Returns a string containing the table qualifier used by the data source (e.g., "table," "file").
13	Returns a string containing the database qualifier used by the data source (e.g., "database," "directory").

14 Returns a string containing the procedure qualifier used by the data source (e.g., "database procedure," "stored procedure," "procedure").

array — Optional Variant array parameter. This parameter is only required for action values 1, 2, 3, 4, and 5. The returned information is put into this array.

If array is fixed and it is not the correct size necessary to hold the requested information, then SQLGetSchema will fail. If the array is larger than required, then any additional elements are erased.

If array is dynamic, then it will be redimensioned to hold the exact number of elements requested.

qualifier — Optional String parameter required for actions 3, 4, or 5. The values are listed in the following table:

Action	Qualifier
3	The qualifier parameter must be the name of the database represented by ID.
4	The qualifier parameter specifies a database name and an owner name. The syntax for this string is: DatabaseName.OwnerName
5	The qualifier parameter specifies the name of a table on the current connection.

- Delrina Basic generates a runtime error if SQLGetSchema fails. Additional error information can then be retrieved using the SQLError function.
- If you want to retrieve the available data sources (where action = 1) before establishing a connection, you can pass 0 as the ID parameter. This is the only action that will run successfully without a valid connection.
- This function calls the ODBC functions SQLGetInfo and SQLTables in order to retrieve the requested information. Some database drivers do not support these calls and will therefore cause the SQLGetSchema function to fail.

Example

```
'This example gets all available data sources.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim dsn() As Variant
    numdims% = SQLGetSchema(0,1,dsn)
    If (numdims%) Then
        msg = "Valid ODBC data sources:" & crlf & crlf
        For x = 0 To numdims% - 1
            msg = msg & dsn(x) & crlf
        Next x
    Else
        msg = "There are no available data sources."
    End If
    MsgBox msg
End Sub
```

See Also

SQLOpen (function).

function

SQLOpen

Syntax

```
SQLOpen(login$ [,[completed$] [,prompt]])
```

Description

Establishes a connection to the specified data source, returning a Long representing the unique connection ID.

Comments

- This function connects to a data source using a login string (login$) and optionally sets the completed login string (completed$) that was used by the driver. The following table describes the parameters to the SQLOpen function:

Parameter	Description
login$	String expression containing information required by the driver to connect to the requested data source. The syntax must strictly follow the driver's SQL syntax.
completed$	Optional String variable that will recieve a completed connection string returned by the driver. If this parameter is missing, then no connection string will be returned.
prompt	Integer expression specifying any of the following values:

Value	Meaning
1	The driver's login dialog box is always displayed.
2	The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior.
3	The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable.
4	The driver's login dialog box is never displayed.

- The SQLOpen function will never return an invalid connection ID. The following example establishes a connection using the driver's login dialog box:

```
id& = SQLOpen("",,1)
```

- Delrina Basic returns 0 and generates a trappable runtime error if SQLOpen fails. Additional error information can then be retrieved using the SQLError function.
- Before you can use any SQL statements, you must set up a data source and relate an existing database to it. This is accomplished using the odbcadm.exe program.

Example

```
'This example connects the data source called "sample," returning the
'completed connction string, and then displays it.

Sub Main()
    Dim s As String
    id& = SQLOpen("dsn=SAMPLE",s$,3)
    MsgBox "The completed connection string is: " & s$
    id& = SQLClose(id&)
End Sub
```

See Also

SQLClose (function).

function

SQLRequest

Syntax

```
SQLRequest(connection$,query$,array [,[output$] [,[prompt]
[,isColumnNames]]])
```

Description

Opens a connection, runs a query, and returns the results as an array.

Comments

- The SQLRequest function takes the following parameters:

Parameter	Description
connection	String specifying the connection information required to connect to the data source.
query	String specifying the query to run. The syntax of this string must strictly follow the syntax of the ODBC driver.
array	Array of variants to be filled with the results of the query. The array parameter must be dynamic: it will be resized to hold the exact number of records and fields.
output	Optional String to receive the completed connection string as returned by the driver.
prompt	Optional Integer specifying the behavior of the driver's dialog box.
isColumnNames	Optional Boolean specifying whether the column names are returned as the first row of results. The default is False.

- Delrina Basic generates a runtime error if SQLRequest fails. Additional error information can then be retrieved using the SQLError function.
- The SQLRequest function performs one of the following actions, depending on the type of query being performed:

Type of Query	Action
SELECT	The SQLRequest function fills array with the results of the query, returning a Long containing the number of results placed in the array.
INSERT, DELETE, UPDATE	The SQLRequest function erases array and returns a Long containing the number of affected rows.

Example

```
'This example opens a data source, runs a select query on it, and
'then displays all the data found in the result set.

Sub Main()
    Dim a() As Variant
    l& = SQLRequest("dsn=SAMPLE;","Select * From c:\sample.dbf",a,,3,True)
    For x = 0 To Ubound(a)
        For y = 0 To l - 1
            MsgBox a(x,y)
        Next y
    Next x
End Sub
```

function

SQLRetrieve

Syntax

```
SQLRetrieve(ID,array[,[maxcolumns] [,[ maxrows] [,[isColumnNames] [,
isFetchFirst]]]])
```

Description

Retrieves the results of a query.

Comments

- This function is called after a connection to a data source is established, a query is run, and the desired columns are bound. The following table describes the parameters to the SQLRetrieve function:

Parameter	Description
ID	Long identifying a valid connected data source with pending query results.
array	Two-dimensional array of variants to receive the results. The array has x rows by y columns. The number of columns is determined by the number of bindings on the connection.
maxcolumns	Optional Integer expression specifying the maximum number of columns to be returned. If maxcolumns is greater than the number of columns bound, the additional columns are set to empty. If maxcolumns is less than the number of bound results, the rightmost result columns are discarded until the result fits.
maxrows	Optional Integer specifying the maximum number of rows to be returned. If maxrows is greater than the number of rows available, all results are returned, and additional rows are set to empty. If maxrows is less than the number of rows available, the array is filled, and additional results are placed in memory for subsequent calls to SQLRetrieve.
isColumnNames	Optional Boolean specifying whether column names should be returned as the first row of results. The default is False.
isFetchFirst	Optional Boolean expression specifying whether results are retrieved from the beginning of the result set. The default is False.

- Before you can retrieve the results from a query, you must (1) initiate a query by calling the SQLExecQuery function and (2) specify the fields to retrieve by calling the SQLBind function.
- This function returns a Long specifying the number of columns available in the array.
- Delrina Basic generates a runtime error if SQLRetrieve fails. Additional error information is placed in memory.

Example

```
'This example runs a query on the connected data source, binds
'columns, and retrieves them.
```

```
Sub Main()
    Dim b() As Variant
    Dim c() As Variant
    id& = SQLOpen("DSN=SAMPLE",,3)
    qry& = SQLExecQuery(id&,"Select * From c:\sample.dbf")
    i% = SQLBind(id&,b,3)
    i% = SQLBind(id&,b,1)
    i% = SQLBind(id&,b,2)
    i% = SQLBind(id&,b,6)
    l& = SQLRetrieve(id&,c)
    For x = 0 To Ubound(c)
        For y = 0 To l& - 1
            MsgBox c(x,y)
        Next y
    Next x
    id& = SQLClose(id&)
End Sub
```

See Also

SQLOpen (function); SQLExecQuery (function); SQLClose (function); SQLBind (function); SQLRetrieveToFile (function).

function

SQLRetrieveToFile

Syntax

```
SQLRetrieveToFile(ID,destination$ [,[isColumnNames] [,delimiter$]])
```

Description

Retrieves the results of a query and writes them to the specified file.

Comments

- The following table describes the parameters to the SQLRetrieveToFile function:

Parameter	Description
ID	Long specifying a valid connection ID.
destination	String specifying the file where the results are written.
isColumnNames	Optional Boolean specifying whether the first row of results returned are the bound column names. By default, the column names are not returned.
delimiter	Optional String specifying the column separator. A tab (Chr$(9)) is used as the default.

- Before you can retrieve the results from a query, you must (1) initiate a query by calling the SQLExecQuery function and (2) specify the fields to retrieve by calling the SQLBind function.
- This function returns the number of rows written to the file. A runtime error is generated if there are no pending results or if Delrina Basic is unable to open the specified file.
- Delrina Basic generates a runtime error if SQLRetrieveToFile fails. Additional error information may be placed in memory for later use with the SQLError function.

Example

```
'This example opens a connection, runs a query, binds columns, and
'writes the results to a file.

Sub Main()
    Dim b() As Variant
    id& = SQLOpen("DSN=SAMPLE;UID=RICH",,4)
    t& = SQLExecQuery(id&,"Select * From c:\sample.dbf")
    i% = SQLBind(id&,b,3)
    i% = SQLBind(id&,b,1)
    i% = SQLBind(id&,b,2)
    i% = SQLBind(id&,b,6)
    l& = SQLRetrieveToFile(id&,"c:\results.txt",True,",")
    id& = SQLClose(id&)
End Sub
```

See Also

SQLOpen (function); SQLExecQuery (function); SQLClose (function); SQLBind (function); SQLRetrieve (function).

function

Sqr

Syntax

```
Sqr(number)
```

Description

Returns a Double representing the square root of number.

Comments

- The number parameter is a Double greater than or equal to 0.

Example

```
'This example calculates the square root of the numbers from 1 to 10
'and displays them.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    For x = 1 To 10
        sx# = Sqr(x)
        msg = msg & "The square root of " & x & " is " & Format(sx#,"Fixed") & crlf
    Next x
    MsgBox msg
End Sub
```

statement

Stop

Syntax

Stop

Description

Suspends execution of the current script, returning control to a debugger if one is present. If a debugger is not present, this command will have the same effect as End.

Example

```
'The Stop statement can be used for debugging. In this example, it is used
'to stop execution when Z is randomly set to 0.

Sub Main()
    For x = 1 To 10
        z = Random(0,10)
        If z = 0 Then Stop
        y = x / z
    Next x
End Sub
```

See Also

Exit For (statement); Exit Do (statement); Exit Function (statement); Exit Sub (statement); End (statement).

function

Str, Str$

Syntax

```
Str[$](number)
```

Description

Returns a string representation of the given number.

Comments

- The number parameter is any numeric expression or expression convertible to a number. If number is negative, then the returned string will contain a leading minus sign. If number is positive, then the returned string will contain a leading space.
- Singles are printed using only 7 significant digits. Doubles are printed using 15-16 significant digits.
- These functions only output the period as the decimal separator and do not output thousands separators. Use the CStr, Format, or Format$ function for this purpose.

Example

```
'In this example, the Str$ function is used to display the value of a
'numeric variable.

Sub Main()
    x# = 100.22
    MsgBox "The string value is: " + Str(x#)
End Sub
```

See Also

Format, Format$ (functions); CStr (function).

function

StrComp

Syntax

```
StrComp(string1,string2 [,compare])
```

Description

Returns an Integer indicating the result of comparing the two string arguments.

Comments

- Any of the following values are returned:

0	string1 = string2
1	string1 > string2
-1	string1 < string2
Null	string1 or string2 is Null

- The StrComp function accepts the following parameters:

Parameter	Description
string1	First string to be compared, which can be any expression convertible to a String.
string2	Second string to be compared, which can be any expression convertible to a String.
compare	Optional Integer specifying how the comparison is to be performed. It can be either of the following values:
0	Case-sensitive comparison
1	Case-insensitive comparison

- If compare is not specified, then the current Option Compare setting is used. If no Option Compare statement has been encountered, then Binary is used (i.e., string comparison is case-sensitive).

Example

```
'This example compares two strings and displays the results.
'It illustrates that the function compares two strings to the
'length of the shorter string in determining equivalency.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    dim abc as boolean
    dim abi as boolean
    dim cdc as boolean
    dim cdi as boolean

    a$ = "This string is UPPERCASE and lowercase"
    b$ = "This string is uppercase and lowercase"
    c$ = "This string"
    d$ = "This string is uppercase and lowercase characters"
    msg = "a = " & a & crlf
    msg = msg & "b = " & b & crlf
    msg = msg & "c = " & c & crlf
    msg = msg & "d = " & d & crlf & crlf
```

```
    abc = StrComp(a$,b$,0)
    msg = msg & "a and c (insensitive)   : " & abc & crlf
    abi = StrComp(a$,b$,1)
    msg = msg & "a and c (sensitive): " & abi & crlf
    cdc = StrComp(c$,d$,1)
    msg = msg & "c and d (insensitive): " & cdc & crlf
    cdi = StrComp(c$,d$,1)
    msg = msg & "c and d (sensitive)   : " & cdi & crlf

    MsgBox msg
End Sub
```

See Also

Comparison Operators (topic); Like (operator); Option Compare (statement).

data type

String

Syntax

String

Description

A data type capable of holding a number of characters.

Comments

- Strings are used to hold sequences of characters, each character having a value between 0 and 255. Strings can be any length up to a maximum length of 32767 characters.
- Strings can contain embedded nulls, as shown in the following example:

```
s$ = "Hello" + Chr$(0) + "there"        'String with embedded null
```

- The length of a string can be determined using the Len function. This function returns the number of characters that have been stored in the string, including unprintable characters.
- The type-declaration character for String is $.
- String variables that have not yet been assigned are set to zero-length by default.
- Strings are normally declared as variable-length, meaning that the memory required for storage of the string depends on the size of its content. The following Delrina Basic statements declare a variable-length string and assign it a value of length 5:

```
Dim s As String
s = "Hello"                 'String has length 5.
```

- Fixed-length strings are given a length in their declaration:

```
Dim s As String * 20
s = "Hello"                 'String has length 20 (internally pads with spaces).
```

- When a string expression is assigned to a fixed-length string, the following rules apply:
- If the string expression is less than the length of the fixed-length string, then the fixed-length string is padded with spaces up to its declared length.
- If the string expression is greater than the length of the fixed-length string, then the string expression is truncated to the length of the fixed-length string.
- Fixed-length strings are useful within structures when a fixed size is required, such as when passing structures to external routines.
- The storage for a fixed-length string depends on where the string is declared, as described in the following table:

Strings Declared	Are Stored
In structures	In the same data area as that of the structure. Local structures are on the stack; public structures are stored in the public data space; and private structures are stored in the private data space. Local structures should be used sparingly as stack space is limited.
In arrays	In the global string space along with all the other array elements.
Local routines	On the stack. The stack is limited in size, so local fixed-length strings should be used sparingly.

See Also

Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); Variant (data type); Boolean (data type); DefType (statement); CStr (function).

function

String, String$

Syntax

```
String[$](number,[CharCode | text$])
```

Description

Returns a string of length number consisting of a repetition of the specified filler character.

Comments

- String$ returns a String, whereas String returns a String variant.
- These functions take the following parameters:

Parameter	Description
number	Integer specifying the number of repetitions.
CharCode	Integer specifying the character code to be used as the filler character. If CharCode is greater than 255 (the largest character value), then Delrina Basic converts it to a valid character using the following formula: CharCode Mod 256
text$	Any String expression, the first character of which is used as the filler character.

Example

```
'This example uses the String function to create a line of "=" signs
'the length of another string and then displays the character string
'underlined with the generated string.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "This string will appear underlined."
    b$ = String(Len(a$),"_")
    MsgBox a$ & crlf & b$
End Sub
```

See Also

Space, Space$ (functions).

statement

Sub...End Sub

Syntax

```
[Private | Public] [Static] Sub name[(arglist)]
     [statements]
End Sub
```

Where arglist is a comma-separated list of the following (up to 30 arguments are allowed):

```
[Optional] [ByVal | ByRef] parameter[()] [As type]
```

Description

Declares a subroutine.

Comments

- The Sub statement has the following parts:

Part	Description
Private	Indicates that the subroutine being defined cannot be called from other scripts.
Public	Indicates that the subroutine being defined can be called from other scripts. If the Private and Public keywords are both missing, then Public is assumed.
Static	Recognized by the compiler but currently has no effect.
name	Name of the subroutine, which must follow Delrina Basic naming conventions: 1. Must start with a letter. 2. May contain letters, digits, and the underscore character (_). Punctuation and type-declaration characters are not allowed. The exclamation point (!) can appear within the name as long as it is not the last character. 3. Must not exceed 80 characters in length.
Optional	Keyword indicating that the parameter is optional. All optional parameters must be of type Variant. Furthermore, all parameters that follow the first optional parameter must also be optional. If this keyword is omitted, then the parameter is required.

Note: You can use the IsMissing function to determine if an optional parameter was actually passed by the caller.

ByVal	Keyword indicating that the parameter is passed by value.
ByRef	Keyword indicating that the parameter is passed by reference. If neither the ByVal nor the ByRef keyword is given, then ByRef is assumed.
parameter	Name of the parameter, which must follow the same naming conventions as those used by variables. This name can include a type-declaration character, appearing in place of As type.
type	Type of the parameter (i.e., Integer, String, and so on). Arrays are indicated with parentheses. For example, an array of integers would be declared as follows: Sub **Test**(**a() As Integer**) End Sub

- A subroutine terminates when one of the following statements is encountered:

End Sub
Exit Sub

- Subroutines can be recursive.

Passing Parameters to Subroutines

Parameters are passed to a subroutine either by value or by reference, depending on the declaration of that parameter in arglist. If the parameter is declared using the ByRef keyword, then any modifications to that passed parameter within the subroutine change the value of that variable in the caller. If the parameter is declared using the ByVal keyword, then the value of that variable cannot be changed in the called subroutine. If neither the ByRef or ByVal keywords are specified, then the parameter is passed by reference.

You can override passing a parameter by reference by enclosing that parameter within parentheses. For instance, the following example passes the variable j by reference, regardless of how the third parameter is declared in the arglist of UserSub:

```
UserSub 10,12,(j)
```

Optional Parameters

Delrina Basic lets you to skip parameters when calling subroutines, as shown in the following example:

```
Sub Test(a%,b%,c%)
End Sub

Sub Main
    Test 1,,4       'Parameter 2 was skipped.
End Sub
```

You can skip any parameter with the following restrictions:

1. The call cannot end with a comma. For instance, using the above example, the following is not valid:

   ```
   Test 1,,
   ```

2. The call must contain the minimum number of parameters as requred by the called subroutine. For instance, using the above example, the following are invalid:

   ```
   Test ,1             'Only passes two out of three required parameters.
   Test 1,2        'Only passes two out of three required parameters.
   ```

When you skip a parameter in this manner, Delrina Basic creates a temporary variable and passes this variable instead. The value of this temporary variable depends on the data type of the corresponding parameter in the argument list of the called subroutine, as described in the following table:

Value	Data type
0	Integer, Long, Single, Double, Currency
Zero-length string	String
Nothing	Object (or any data object)
Error	Variant
December 30, 1899	Date
False	Boolean

Within the called subroutine, you will be unable to determine if a parameter was skipped unless the parameter was declared as a variant in the argument list of the subroutine. In this case, you can use the IsMissing function to determine if the parameter was skipped:

```
Sub Test(a,b,c)
    If IsMissing(a) Or IsMissing(b) Then Exit Sub
End Sub
```

Example

'This example uses a subroutine to calculate the area of a circle.

```
Sub Main()
    r = inputbox("Enter a circle radius to be converted to area","Radius -> Area")
    PrintArea r
End Sub

Sub PrintArea(r)
    area! = (r ^ 2) * Pi
    MsgBox "The area of a circle with radius " & r & " = " & area!
End Sub
```

See Also

Main (keyword); Function...End Function (statement).

function

Switch

Syntax

```
Switch(condition1,expression1 [,condition2,expression2 ...
[,condition7,expression7]])
```

Description

Returns the expression corresponding to the first True condition.

Comments

- The Switch function evaluates each condition and expression, returning the expression that corresponds to the first condition (starting from the left) that evaluates to True. Up to seven condition/expression pairs can be specified.
- A runtime error is generated it there is an odd number of parameters (i.e., there is a condition without a corresponding expression).
- The Switch function returns Null if no condition evaluates to True.

Example

```
'The following code fragment displays the current operating platform. If the
'platform is unknown, then the word "Unknown" is displayed.

Sub Main()
    Dim a As Variant
    a = Switch(Basic.OS = 0,"Windows 3.1",Basic.OS = 2,"Win32",Basic.OS = 11,"OS/2")
    MsgBox "The current platform is: " & IIf(IsNull(a),"Unknown",a)
End Sub
```

See Also

Choose (function); IIf (function); If...Then...Else (statement); Select...Case (statement).

function

SYD

Syntax

```
SYD(Cost,Salvage,Life,Period)
```

Description

Returns the sum of years' digits depreciation of an asset over a specific period of time.

Comments

- The SYD of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers.
- The formula used to find the SYD of an asset is as follows:

  (Cost - Salvage_Value) * Remaining_Useful_Life / SYD
- The SYD function requires the following parameters:

Parameter	Description
Cost	Double representing the initial cost of the asset.
Salvage	Double representing the estimated value of the asset at the end of its useful life.
Life	Double representing the length of the asset's useful life.
Period	Double representing the period for which the depreciation is to be calculated. It cannot exceed the life of the asset.

- To receive accurate results, the parameters Life and Period must be expressed in the same units. If Life is expressed in terms of months, for example, then Period must also be expressed in terms of months.

Example

```
'In this example, an asset that cost $1,000.00 is depreciated over ten years.
'The salvage value is $100.00, and the sum of the years' digits
'depreciation is shown for each year.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    For x = 1 To 10
        dep# = SYD(1000,100,10,x)
        msg = msg & "Year: " & x & "    Dep: " & Format(dep#,"Currency") & crlf
    Next x
    MsgBox msg
End Sub
```

See Also

Sln (function); DDB (function).

method

System.Exit

Syntax

System.Exit

Description

Exits the operating environment.

Example

```
'This example asks whether the user would like to restart Windows
'after exiting.

Sub Main
    button = MsgBox("Restart Windows on exit?",ebYesNo,"Exit Windows")
    If button = ebYes Then System.Restart          'Yes button selected.
    If button = ebNo Then System.Exit              'No button selected.
End Sub
```

See Also

System.Restart (method).

property

System.FreeMemory

Syntax

System.FreeMemory

Description

Returns a Long indicating the number of bytes of free memory.

Example

```
'The following example gets the free memory and converts it to
'kilobytes

Sub Main()
    FreeMem& = System.FreeMemory
    FreeKBytes$ = Format(FreeMem& / 1000,"##,###")
    MsgBox FreeKbytes$ & " Kbytes of free memory"
End Sub
```

See Also

System.TotalMemory (property); System.FreeResources (property); Basic.FreeMemory (property).

property

System.FreeResources

Syntax

System.FreeResources

Description

Returns an Integer representing the percentage of free system resources.

Comments

- The returned value is between 0 and 100.

Example

```
'This example gets the percentage of free resources.

Sub Main()
    FreeRes% = System.FreeResources
    MsgBox FreeRes% & "% of memory resources available."
End Sub
```

See Also

System.TotalMemory (property); System.FreeMemory (property); Basic.FreeMemory (property).

method

System.MouseTrails

Syntax

```
System.MouseTrails isOn
```

Description

Toggles mouse trails on or off.

Comments

- If isOn is True, then mouse trails are turned on; otherwise, mouse trails are turned off.
- A runtime error is generated if mouse trails is not supported on your system.

Example

```
'This example turns on mouse trails.
Sub Main
    On Error Resume Next
    System.MouseTrails 1
    If Err <> 0 Then MsgBox "This system or O/S will not support
mouse trails!",ebInformation,"Error"
End Sub
```

Note

- The setting is saved in the INI file permanently.

method

System.Restart

Syntax

System.Restart

Description

Restarts the operating environment.

Example

```
'This example asks whether the user would like to restart Windows after
'exiting.

Sub Main
    button = MsgBox ("Restart Windows on exit?",ebYesNo,"Exit Windows")
    If button = ebYes Then System.Restart          'Yes button selected.
    If button = ebNo Then System.Exit              'No button selected.
End Sub
```

See Also

System.Exit (method).

property

System.TotalMemory

Syntax

System.TotalMemory

Description

Returns a Long representing the number of bytes of available free memory in Windows.

Example

```
'This example displays the total system memory.

Sub Main()
    TotMem& = System.TotalMemory
    TotKBytes$ = Format(TotMem& / 1000,"##,###")
    MsgBox TotKbytes$ & " Kbytes of total system memory exist"
End Sub
```

See Also

System.FreeMemory (property); System.FreeResources (property); Basic.FreeMemory (property).

property

System.WindowsDirectory$

Syntax

System.WindowsDirectory$

Description

Returns the home directory of the operating environment.

Example

'This example displays the windows directory.

```
Sub Main
    MsgBox "Windows directory = " & System.WindowsDirectory$
End Sub
```

See Also

Basic.HomeDir$ (property).

property

System.WindowsVersion$

Syntax

System.WindowsVersion$

Description

Returns the version of the operating environment, such as "3.0" or "3.1."

Example

```
'This example sets the UseWin31 variable to True if the Windows version is
'greater than or equal to 3.1; otherwise, it sets the UseWin31 variable
'to False.

Sub Main()
    If Val(System.WindowsVersion$) > 3.1 Then
        MsgBox "You are running a Windows version later than 3.1"
    Else
        MsgBox "You are running Windows version 3.1 or earlier"
    End If
End Sub
```

See Also

Basic.Version$ (property).

function

Tab

Syntax

```
Tab(column)
```

Description

Prints the number of spaces necessary to reach a given column position.

Comments

- This function can only be used with the Print and Print# statements.
- The column parameter is an Integer specifying the desired column position to which to advance. It can be any value between 0 and 32767 inclusive.

Rule 1: If the current print position is less than or equal to column, then the number of spaces is calculated as:

```
column - print_position
```

Rule 2: If the current print position is greater than column, then column - 1 spaces are printed on the next line.

If a line width is specified (using the Width statement), then the column position is adjusted as follows before applying the above two rules:

```
column = column Mod width
```

- The Tab function is useful for making sure that output begins at a given column position, regardless of the length of the data already printed on that line.

Example

```
'This example prints three column headers and three numbers
'alligned below the column headers.

Sub Main()
    ViewportOpen
    Print "Column1";Tab(10);"Column2";Tab(20);"Column3"
    Print Tab(3);"1";Tab(14);"2";Tab(24);"3"
    Sleep(10000)            'Wait 10 seconds.
    ViewportClose
End Sub
```

See Also

Spc (function); Print (statement); Print# (statement).

function

Tan

Syntax

```
Tan(angle)
```

Description

Returns a Double representing the tangent of angle.

Comments

- The angle parameter is a Double value given in radians.

Example

```
'This example computes the tangent of pi/4 radians (45 degrees).

Sub Main()
    c# = Tan(Pi / 4)
    MsgBox "The tangent of 45 degrees is: " & c#
End Sub
```

See Also

Sin (function); Cos (function); Atn (function).

statement

Text

Syntax

```
Text x,y,width,height,title$ [,[.Identifier] [,[FontName$] [,[size]
[,style]]]]
```

Description

Defines a text control within a dialog box template. The text control only displays text; the user cannot set the focus to a text control or otherwise interact with it.

Comments

- The text within a text control word-wraps. Text controls can be used to display up to 32K of text.
- The Text statement accepts the following parameters:

Parameter	Description
x, y	Integer positions of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer dimensions of the control in dialog units.
title$	String containing the text that appears within the text control. This text may contain an ampersand character to denote an accelerator letter, such as "&Save" for Save. Pressing this accelerator letter sets the focus to the control following the Text statement in the dialog box template.
Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words from title$ are used.
FontName$	Name of the font used for display of the text within the text control. If omitted, then the default font for the dialog is used.
size	Size of the font used for display of the text within the text control. If omitted, then the default size for the default font of the dialog is used.
style	Style of the font used for display of the text within the text control. This can be any of the following values:
ebRegular	Normal font (i.e., neither bold nor italic)
ebBold	Bold font
ebItalic	Italic font
ebBoldItalic	Bold-italic font
	If omitted, then ebRegular is used.

Example

```
Sub Main()
    Begin Dialog UserDialog 81,64,128,60,"Untitled"
        CancelButton 80,32,40,14
        OKButton 80,8,40,14
        Text 4,8,68,44,"This text is displayed in the dialog box."
    End Dialog
    Dim d As UserDialog
    Dialog d
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton

(statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

statement

TextBox

Syntax

```
TextBox x,y,width,height,.Identifier [,[isMultiline] [,[FontName$] [,[size]
[,style]]]]
```

Description

Defines a single or multiline text-entry field within a dialog box template.

Comments

- If isMultiline is 1, the TextBox statement creates a multiline text-entry field. When the user types into a multiline field, pressing the Enter key creates a new line rather than selecting the default button.
- This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).
- The TextBox statement requires the following parameters:

Parameter	Description
x, y	Integer position of the control (in dialog units) relative to the upper left corner of the dialog box.
width, height	Integer dimensions of the control in dialog units.
Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates a string variable whose value corresponds to the content of the text box. This variable can be accessed using the syntax:
DialogVariable.Identifier	
isMultiline	Specifies whether the text box can contain more than a single line (0 = single-line; 1 = multiline).
FontName$	Name of the font used for display of the text within the text box control. If omitted, then the default font for the dialog is used.
size	Size of the font used for display of the text within the text box control. If omitted, then the default size for the default font of the dialog is used.
style	Style of the font used for display of the text within the text box control. This can be any of the following values:
ebRegular	Normal font (i.e., neither bold nor italic)
ebBold	Bold font
ebItalic	Italic font
ebBoldItalic	Bold-italic font
	If omitted, then ebRegular is used.

- When the dialog box is created, the Identifier variable is used to set the initial content of the text box. When the dialog box is dismissed, the variable will contain the new content of the text box.
- A single-line text box can contain up to 256 characters. The length of text in a multiline text box is not limited by Delrina Basic; the default memory limit specified by the given platform is used instead.

Example

```
Sub Main()
    Begin Dialog UserDialog 81,64,128,60,"Untitled"
        CancelButton 80,32,40,14
        OKButton 80,8,40,14
        TextBox 4,8,68,44,.TextBox1,1
    End Dialog
    Dim d As UserDialog
    d.TextBox1 = "Enter text before invoking"      'Display text in the Textbox by setting the default value of the
TextBox before showing it.
    Dialog d
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); Begin Dialog (statement), PictureButton (statement).

Note

- Under Windows, 8-point MS Sans Serif is the default font used within user dialogs.

function

Time, Time$

Syntax

Time[$][()]

Description

Returns the system time as a String or as a Date variant.

Comments

- The Time$ function returns a String contains the time in 24-hour time format, whereas Time returns a Date variant.
- To set the time, use the Time/Time$ statements.

Example

```
'This example returns the system time and displays it in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    oldtime$ = Time
    msg = "Time was: " & oldtime$ & crlf
    Time = "10:30:54"
    msg = msg & "Time set to: " & Time & crlf
    Time = oldtime$
    msg = msg & "Time restored to: " & Time
    MsgBox msg
End Sub
```

See Also

Time, Time$ (statements); Date, Date$ (functions); Date, Date$ (statements); Now (function).

statement

Time, Time$

Syntax

```
Time[$] = newtime
```

Description

Sets the system time to the time contained in the specified string.

Comments

- The Time$ statement requres a string variable in one of the following formats:

  ```
  HH
  HH:MM
  HH:MM:SS
  ```

  where HH is between 0 and 23, MM is between 0 and 59, and SS is between 0 and 59.

- The Time statement converts any valid expression to a time, including string and numeric values. Unlike the Time$ statement, Time recognizes many different time formats, including 12-hour times.

Example

```
'This example returns the system time and displays it in a dialog box.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    oldtime$ = Time
    msg = "Time was: " & oldtime$ & crlf
    Time = "10:30:54"
    msg = msg & "Time set to: " & Time & crlf
    Time = oldtime$
    msg = msg & "Time restored to: " & Time
    MsgBox msg
End Sub
```

See Also

Time, Time$ (functions); Date, Date$ (functions); Date, Date$ (statements).

function

Timer

Syntax

Timer

Description

Returns a Single representing the number of seconds that have elapsed since midnight.

Example

```
'This example displays the elapsed time between execution start and
'the time you clicked the OK button on the first message.

Sub Main()
    start& = Timer
    MsgBox "Click the OK button, please."
    total& = Timer - start&
    MsgBox "The elapsed time was: " & total& & " seconds."
End Sub
```

See Also

Time, Time$ (functions); Now (function).

function

TimeSerial

Syntax

```
TimeSerial(hour,minute,second)
```

Description

Returns a Date variant representing the given time with a date of zero.

Comments

- The TimeSerial function requires the following parameters:

Parameter	Description
hour	Integer between 0 and 23.
minute	Integer between 0 and 59.
second	Integer between 0 and 59.

Example

```
Sub Main()
    start# = TimeSerial(10,22,30)
    finish# = TimeSerial(10,35,27)
    dif# = Abs(start# - finish#)
    MsgBox "The time difference is: " & Format(dif#,"hh:mm:ss")
End Sub
```

See Also

DateValue (function); TimeValue (function); DateSerial (function).

function

TimeValue

Syntax

```
TimeValue(time_string$)
```

Description

Returns a Date variant representing the time contained in the specified string argument.

Comments

- This function interprets the passed time_string$ parameter looking for a valid time specification.
- The time_string$ parameter can contain valid time items separated by time separators such as colon (:) or period (.).
- Time strings can contain an optional date specification, but this is not used in the formation of the returned value.
- If a particular time item is missing, then it is set to 0. For example, the string "10 pm" would be interpreted as "22:00:00."

Example

```
'This example calculates the TimeValue of the current time and
'displays it in a dialog box.

Sub Main()
    t1$ = "10:15"
    t2# = TimeValue(t1$)
    MsgBox "The TimeValue of " & t1$ & " is: " & t2#
End Sub
```

See Also

DateValue (function); TimeSerial (function); DateSerial (function).

Notes

Under Windows, time specifications vary, depending on the international settings contained in the [intl] section of the win.ini file.

function

Trim, Trim$

Syntax

```
Trim[$](text)
```

Description

Returns a copy of the passed string expression (text) with leading and trailing spaces removed.

Comments

- Trim$ returns a String, whereas Trim returns a String variant.
- Null is returned if text is Null.

Example

```
'This example uses the Trim$ function to extract the nonblank part
'of a string and display it.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    txt$ = "        This is text                "
    tr$ = Trim(txt$)
    MsgBox "Original ->" & txt$ & "<-" & crlf & "Trimmed ->" & tr$ & "<-"
End Sub
```

See Also

LTrim, LTrim$ (functions); RTrim, RTrim$ (functions).

constant

True

Description

Boolean constant whose value is True.

Comments

- Used in conditionals and Boolean expressions.

Example

```
'This example sets variable a to True and then tests to see whether
'(1) A is True; (2) the True constant = -1; and (3) A is
'equal to -1 (True).

Sub Main()
    a = True
    If ((a = True) and (True = -1) and (a = -1)) then
        MsgBox "a is True."
    Else
        MsgBox "a is False."
    End If
End Sub
```

See Also

False (constant); Constants (topic); Boolean (data type).

statement

Type

Syntax

```
Type username
     variable As type
     variable As type
     variable As type
     :
End Type
```

Description

The Type statement creates a structure definition that can then be used with the Dim statement to declare variables of that type. The username field specifies the name of the structure that is used later with the Dim statement.

Comments

- Within a structure definition appear field descriptions in the format:
  `variable As type`

  where variable is the name of a field of the structure, and type is the data type for that variable. Any fundamental data type or previously declared user-defined data type can be used within the structure definition (structures within structures are allowed). Only fixed arrays can appear within structure definitions.
- The Type statement can only appear outside of subroutine and function declarations.
- When declaring strings within fixed-size types, it is useful to declare the strings as fixed-length.

Fixed-length strings are stored within the structure itself rather than in the string space. For example, the following structure will always require 62 bytes of storage:

```
Type Person
    FirstName As String * 20
    LastName As String * 40
    Age As Integer
End Type
```

Note: Fixed-length strings within structures are size-adjusted upward to an even byte boundary. Thus, a fixed-length string of length 5 will occupy 6 bytes of storage within the structure.

Example

```
'This example displays the use of the Type statement to create a
'structure representing the parts of a circle and assign values
'to them.

Type Circ
    msg As String
    rad As Integer
    dia As Integer
    are As Double
    cir As Double
End Type
```

```
Sub Main()
    Dim circle As Circ
    circle.rad = 5
    circle.dia = circle.rad * 2
    circle.are = (circle.rad ^ 2) * Pi
    circle.cir = circle.dia * Pi
    circle.msg = "The area of this circle is: " & circle.are
    MsgBox circle.msg
End Sub
```

See Also

Dim (statement); Public (statement); Private (statement).

function

UBound

Syntax

```
UBound(ArrayVariable() [,dimension])
```

Description

Returns an Integer containing the upper bound of the specified dimension of the specified array variable.

Comments

- The dimension parameter is an integer that specifies the desired dimension. If not specified, then the upper bound of the first dimension is returned.
- The UBound function can be used to find the upper bound of a dimension of an array returned by an OLE automation method or property:

```
UBound(object.property [,dimension])

UBound(object.method [,dimension])
```

Example

```
'This example dimensions two arrays and displays their upper bounds.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a(5 To 12)
    Dim b(2 To 100,9 To 20)
    uba = UBound(a)
    ubb = UBound(b,2)
    MsgBox "The upper bound of a is: " & uba & crlf & " The upper bound of b is: " & ubb

'This example uses Lbound and Ubound to dimension a dynamic array to
'hold a copy of an array redimmed by the FileList statement.

    Dim fl$()
    FileList fl$,"*"
    count = Ubound(fl$)
    If ArrayDims(a) Then
        Redim nl$(Lbound(fl$) To Ubound(fl$))
        For x = 1 To count
            nl$(x) = fl$(x)
        Next x
        MsgBox "The last element of the new array is: " & nl$(count)
    End If
End Sub
```

See Also

LBound (function); ArrayDims (function); Arrays (topic).

function

UCase, UCase$

Syntax

```
UCase[$](text)
```

Description

Returns the uppercase equivalent of the specified string.

Comments

- UCase$ returns a String, whereas UCase returns a String variant.
- Null is returned if text is Null.

Example

```
'This example uses the UCase$ function to change a string from
'lowercase to uppercase.

Sub Main()
    a1$ = "this string was lowercase, but was converted."
    a2$ = UCase(a1$)
    MsgBox a2$
End Sub
```

See Also

LCase, LCase$ (functions).

statement

Unlock

Syntax

```
Unlock [#] filenumber [,{record | [start] To end}]
```

Description

Unlocks a section of the specified file, letting other processes access to that section of the file.

Comments

- The Unlock statement requires the following parameters:

Parameter	Description
filenumber	Integer used by Delrina Basic to refer to the open file-the number passed to the Open statement.
record	Long specifying which record to unlock.
start	Long specifying the first record within a range to be unlocked.
end	Long specifying the last record within a range to be unlocked.

- For sequential files, the record, start, and end parameters are ignored: the entire file is unlocked.
- The section of the file is specified using one of the following:

Syntax	Description
No record specification	Unlock the entire file.
record	Unlock the specified record number (for Random files) or byte (for Binary files).
to end	Unlock from the beginning of the file to the specified record (for Random files) or byte (for Binary files).
start to end	Unlock the specified range of records (for Random files) or bytes (for Binary files).

- The unlock range must be the same as that used by the Lock statement.

Example

'This example creates a file named test.dat and fills it with ten 'string variable records. These are displayed in a dialog box. The 'file is then reopened for read/write, and each record is locked, 'modified, rewritten, and unlocked. The new records are then 'displayed in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
    a$ = "This is record number: "
    b$ = "0"
    rec$ = ""

    msg = ""
    Open "test.dat" For Random Access Write Shared As #1
    For x = 1 To 10
        rec$ = a$ & x
        Lock #1,x
        Put #1,,rec$
        Unlock #1,x
        msg = msg & rec$ & crlf
    Next x
    Close
    MsgBox "The records are: " & crlf & msg

    msg = ""
    Open "test.dat" For Random Access Read Write Shared As #1
    For x = 1 to 10
        rec$ = Mid(rec$,1,23) & (11 - x)
        Lock #1,x                       'Lock it for our use.
        Put #1,x,rec$           'Nobody's changed it.
        UnLock #1,x
        msg = msg & rec$ & crlf
    Next x
    MsgBox "The records are: " & crlf & msg
    Close

    Kill "test.dat"
End Sub
```

See Also

Lock (statement); Open (statement).

topic

User-Defined Types

User-defined types (UDTs) are structure definitions created using the Type statement. UDTs are equivalent to C language structures.

Declaring Structures

The Type statement is used to create a structure definition. Type declarations must appear outside the body of all subroutines and functions within a script and are therefore global to an entire script.

Once defined, a UDT can be used to declare variables of that type using the Dim, Public, or Private statement. The following example defines a rectangle structure:

```
Type Rect
    left As Integer
    top As Integer
    right As Integer
    bottom As Integer
End Type
    :
Sub Main()
    Dim r As Rect
        :
    r.left = 10
End Sub
```

Any fundamental data type can be used as a structure member, including other user-defined types. Only fixed arrays can be used within structures.

Copying Structures

UDTs of the same type can be assigned to each other, copying the contents. No other standard operators can be applied to UDTs.

```
Dim r1 As Rect
Dim r2 As Rect
    :
r1 = r2
```

When copying structures of the same type, all strings in the source UDT are duplicated and references are placed into the target UDT.

The LSet statement can be used to copy a UDT variable of one type to another:

```
LSet variable1 = variable2
```

LSet cannot be used with UDTs containing variable-length strings. The smaller of the two structures determines how many bytes get copied.

Passing Structures

UDTs can be passed both to user-defined routines and to external routines, and they can be assigned. UDTs are always passed by reference.

Since structures are always passed by reference, the ByVal keyword cannot be used when defining structure arguments passed to external routines (using Declare). The ByVal keyword can only be used with fundamental data types such as Integer and String.

Passing structures to external routines actually passes a far pointer to the data structure.

Size of Structures

The Len function can be used to determine the number of bytes occupied by a UDT:

```
Len(udt_variable_name)
```

Since strings are stored in Delrina Basic's data space, only a reference (currently, 2 bytes) is stored within a structure. Thus, the Len function may seem to return incorrect information for structures containing strings.

function

Val

Syntax

```
Val(string_expression)
```

Description

Converts a given string expression to a number.

Comments

- The number parameter can contain any of the following:
- Leading minus sign (for nonhex or octal numbers only)
- Hexadecimal number in the format &Hhexdigits
- Octal number in the format &Ooctaldigits
- Floating-point number, which can contain a decimal point and an optional exponent
- Spaces, tabs, and line feeds are ignored.
- If number does not contain a number, then 0 is returned.
- The Val function continues to read characters from the string up to the first nonnumeric character.
- The Val function always returns a double-precision floating-point value. This value is forced to the data type of the assigned variable.

Example

```
'This example inputs a number string from an InputBox and converts it
'to a number variable.

Sub Main()
    a$ = InputBox("Enter anything containing a number","Enter Number")
    b# = Val(a$)
    MsgBox "The value is: " & b#
End Sub

'The following table shows valid strings and their numeric equivalents:
'    "1   2        3"          123
'    "12.3"                       12.3
'    "&HFFFF"                     -1
'    "&O77"                       63
'    "12.345E-02"            .12345
```

See Also

CDbl (function); Str, Str$ (functions).

data type

Variant

Syntax

Variant

Description

A data type used to declare variables that can hold one of many different types of data.

Comments

- During a variant's existence, the type of data contained within it can change. Variants can contain any of the following types of data:

Type of Data	Delrina Basic Data Types
Numeric	Integer, Long, Single, Double, Boolean, Date, Currency.
Logical	Boolean.
Dates and times	Date.
String	String.
Object	Object.
No valid data	A variant with no valid data is considered Null.
Uninitialized	An uninitialized variant is considered Empty.

- There is no type-declaration character for variants.
- The number of significant digits representable by a variant depends on the type of data contained within the variant.
- Variant is the default data type for Delrina Basic. If a variable is not explicitly declared with Dim, Public, or Private, and there is no type-declaration character (i.e., #, @, !, %, or &), then the variable is assumed to be Variant.

Determining the Subtype of a Variant

The following functions are used to query the type of data contained within a variant:

Function	Description
VarType	Returns a number representing the type of data contained within the variant.
IsNumeric	Returns True if a variant contains numeric data. The following are considered numeric: Integer, Long, Single, Double, Date, Boolean,Currency If a variant contains a string, this function returns True if the string can be converted to a number. If a variant contains an Object whose default property is numeric, then IsNumeric returns True.
IsObject	Returns True if a variant contains an object.
IsNull	Returns True if a variant contains no valid data.
IsEmpty	Returns True if a variant is uninitialized.
IsDate	Returns True if a variant contains a date. If the variant contains a string, then this function returns True if the string can be converted to a date. If the variant contains an Object, then this function returns True if the default property of that object can be converted to a date.

Assigning to Variants

Before a Variant has been assigned a value, it is considered empty. Thus, immediately after

declaration, the VarType function will return ebEmpty. An uninitialized variant is 0 when used in numeric expressions and is a zero-length string when used within string expressions.

A Variant is Empty only after declaration and before assigning it a value. The only way for a Variant to become Empty after having received a value is for that variant to be assigned to another Variant containing Empty, for it to be assigned explicitly to the constant Empty, or for it to be erased using the Erase statement.

When a variant is assigned a value, it is also assigned that value's type. Thus, in all subsequent operations involving that variant, the variant will behave like the type of data it contains.

Operations on Variants

Normally, a Variant behaves just like the data it contains. One exception to this rule is that, in arithmetic operations, variants are automatically promoted when an overflow occurs. Consider the following statements:

```
Dim a As Integer,b As Integer,c As Integer
Dim x As Variant,y As Variant,z As Variant

a% = 32767
b% = 1
c% = a% + b%                'This will overflow.

x = 32767
y = 1
z = x + y                   'z becomes a Long because of Integer overflow.
```

In the above example, the addition involving Integer variables overflows because the result (32768) overflows the legal range for integers. With Variant variables, on the other hand, the addition operator recognizes the overflow and automatically promotes the result to a Long.

Adding Variants

The + operator is defined as performing two functions: when passed strings, it concatenates them; when passed numbers, it adds the numbers.

With variants, the rules are complicated because the types of the variants are not known until execution time. If you use +, you may unintentionally perform the wrong operation.

It is recommended that you use the & operator if you intend to concatenate two String variants. This guarantees that string concatenation will be performed and not addition.

Variants That Contain No Data

A Variant can be set to a special value indicating that it contains no valid data by assigning the Variant to Null:

```
Dim a As Variant
a = Null
```

The only way that a Variant becomes Null is if you assign it as shown above.

The Null value can be useful for catching errors since its value propagates through an expression.

Variant Storage

Variants require 16 bytes of storage internally:

A 2-byte type

A 2-byte extended type for data objects

4 bytes of padding for alignment

An 8-byte value

Unlike other data types, writing variants to Binary or Random files does not write 16 bytes. With variants, a 2-byte type is written, followed by the data (2 bytes for Integer and so on).

Disadvantages of Variants

The following list describes some disadvantages of variants:

1. Using variants is slower than using the other fundamental data types (i.e., Integer, Long, Single, Double, Date, Object, String, Currency, and Boolean). Each operation involving a Variant requires examination of the variant's type.
2. Variants require more storage than other data types (16 bytes as opposed to 8 bytes for a Double, 2 bytes for an Integer, and so on).
3. Unpredictable behavior. You may write code to expect an Integer variant. At runtime, the variant may be automatically promoted to a Long variant, causing your code to break.

Passing Nonvariant Data to Routines Taking Variants

Passing nonvariant data to a routine that is declared to receive a variant by reference prevents that variant from changing type within that routine. For example:

```
Sub Foo(v As Variant)
    v = 50                      'OK.
    v = "Hello, world."         'Get a type-mismatch error here!
End Sub

Sub Main()
    Dim i As Integer
    Foo i                       'Pass an integer by reference.
End Sub
```

In the above example, since an Integer is passed by reference (meaning that the caller can change the original value of the Integer), the caller must ensure that no attempt is made to change the variant's type.

Passing Variants to Routines Taking Nonvariants

Variant variables cannot be passed to routines that accept nonvariant data by reference, as demonstrated in the following example:

```
Sub Foo(i As Integer)
End Sub

Sub Main()
    Dim a As Variant
    Foo a                       'Compiler gives type-mismatch error here.
End Sub
```

See Also

Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Boolean (data type); DefType (statement); CVar (function); Empty (constant); Null (constant); VarType (function).

function

VarType

Syntax

```
VarType(variable)
```

Description

Returns an Integer representing the type of data in variable.

Comments

- The variable parameter is the name of any Variant.
- The following table shows the different values that can be returned by VarType:

Value	Constant	Data Type
0	ebEmpty	Uninitialized
1	ebNull	No valid data
2	ebInteger	Integer
3	ebLong	Long
4	ebSingle	Single
5	ebDouble	Double
6	ebCurrency	Currency
7	ebDate	Date
8	ebString	String
9	ebObject	Object (OLE automation object)
10	ebError	User-defined error
11	ebBoolean	Boolean
12	ebVariant	Variant (not returned by this function)
13	ebDataObject	Non-OLE automation object

Comments

When passed an cbject, the VarType function returns the type of the default property of that object. If the object has no default property, then either ebObject or ebDataObject is returned, depending on the type of variable.

Example

```
Sub Main()
    Dim v As Variant
    v = 5&                      'Set v to a Long.

    If VarType(v) = ebInteger Then
        Msgbox "v is an Integer."
    ElseIf VarType(v) = ebLong Then
        Msgbox "v is a Long."
    End If
End Sub
```

See Also

Empty (constant); Null (constant); Variant (data type).

statement

ViewportClear

Syntax

ViewportClear

Description

Clears the open viewport window.

Comments

- The statement has no effect if no viewport is open.

Example

```
Sub Main()
    ViewportOpen
    Print "This will be displayed in the viewport window."
    Sleep 2000
    ViewportClear
    Print "This will replace the previous text."
    Sleep 2000
    ViewportClose
End Sub
```

See Also

ViewportClose (statement); ViewportOpen (statement).

statement

ViewportClose

Syntax

ViewportClose

Description

This statement closes an open viewport window.

Comments

The statement has no effect if no viewport is opened.

Example

```
Sub Main()
    ViewportOpen
    Print "This will be displayed in the viewport window."
    Sleep 2000
    ViewportClose
End Sub
```

See Also

ViewportOpen (statement).

statement

ViewportOpen

Syntax

```
ViewportOpen [title$ [,x,y [,width,height]]]
```

Description

Opens a new viewport window or switches the focus to the existing viewport window.

Comments

- The ViewportOpen statement requires the following parameters:

Parameter	Description
title$	Specifies a String containing the text to appear in the viewport's caption.
x,y	Specifies Integer coordinates given in twips indicating the initial position of the upper left corner of the viewport.
width,height	Specifies Integer values indicating the initial width and height of the viewport.

- This statement has no effect if a viewport window is already open.
- Combined with the Print statement, a viewport window is a convenient place to output debugging information.
- The viewport window is closed when the Delrina Basic host application is terminated.
- The buffer size for the viewport is 32K. Information from the start of the buffer is removed to make room for additional information being appended to the end of the buffer.
- The following keys work within a viewport window:

Up	Scrolls up by one line.
Down	Scrolls down by one line.
Home	Scrolls to the first line in the viewport window.
End	Scrolls to the last line in the viewport window.
PgDn	Scrolls the viewport window down by one page.
PgUp	Scrolls the viewport window up by one page.
Ctrl+PgUp	Scrolls the viewport window left by one page.
Ctrl+PgDn	Scrolls the viewport window right by one page.

- Only one viewport window can be open at any given time. Any scripts with Print statements will output information into the same viewport window.
- When printing to viewports, the end-of-line character can be any of the following: a carriage return, a line feed, or a carriage-return/line-feed pair.

Example

```
Sub Main()
    ViewportOpen "Delrina Basic Viewport",100,100,500,500
    Print "This will be displayed in the viewport window."
    Sleep 2000
    ViewportClose
End Sub
```

See Also

ViewportClose (statement).

statement

VLine

Syntax

```
VLine [lines]
```

Description

Scrolls the window with the focus up or down by the specified number of lines.

Comments

- The lines parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled down by one line.

Example

'This example prints a series of lines to the viewport, then 'scrolls back up the lines to the top using VLine.

```
Sub Main()
    If AppFind$("Notepad") = "" Then
        id = Shell("notepad.exe",3)
    Else
        AppActivate "Notepad"
        AppMaximize
    End If
    VLine -40          'Scroll 40 lines upward in Notepad.
End Sub
```

See Also

VPage (statement); VScroll (statement).

statement

VPage

Syntax

```
VPage [pages]
```

Description

Scrolls the window with the focus up or down by the specified number of pages.

Comments

- The pages parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled down by one page.

Example

```
'This example scrolls the viewport window up five pages.

Sub Main()
    If AppFind$("Notepad") = "" Then
        id = Shell("notepad.exe",3)
    Else
        AppActivate "Notepad"
        AppMaximize
    End If
    VPage -5        'Jump 5 pages backward in Notepad.
End Sub
```

See Also

VLine (statement); VScroll (statement).

statement

VScroll

Syntax

```
VScroll percentage
```

Description

Sets the thumb mark on the vertical scroll bar attached to the current window.

Comments

- The position is given as a percentage of the total range associated with that scroll bar. For example, if the percentage parameter is 50, then the thumb mark is positioned in the middle of the scroll bar.

Example

```
'This example prints a bunch of lines to the viewport, then
'scrolls back to the top using VScroll.

Sub Main()
    If AppFind$("Notepad") = "" Then
        id = Shell("notepad.exe",3)
    Else
        AppActivate "Notepad"
        AppMaximize
    End If
    VScroll 0                'Jump to top of Notepad document.
End Sub
```

See Also

VLine (statement); VPage (statement).

function

Weekday

Syntax

```
Weekday(date)
```

Description

Returns an Integer value representing the day of the week given by date. Sunday is 1, Monday is 2, and so on.

The date parameter is any expression representing a valid date.

Example

```
'This example gets a date in an input box and displays
'the day of the week and its name for the date entered.

Sub Main()
    Dim a$(7)
    a$(1) = "Sunday"
    a$(2) = "Monday"
    a$(3) = "Tuesday"
    a$(4) = "Wednesday"
    a$(5) = "Thursday"
    a$(6) = "Friday"
    a$(7) = "Saturday"

Reprompt:
    bd = InputBox("Please enter your birthday.","Enter Birthday")
    If Not(IsDate(bd)) Then Goto Reprompt

    dt = DateValue(bd)
    dw = WeekDay(dt)
    Msgbox "You were born on day " & dw & ", which was a " & a$(dw)
End Sub
```

See Also

Day (function); Minute (function); Second (function); Month (function); Year (function); Hour (function); DatePart (function).

statement

While...Wend

Syntax

```
While condition
  [statements]
Wend
```

Description

Repeats a statement or group of statements while a condition is True.

Comments

- The condition is initially and then checked at the top of each iteration through the loop.

Example

```
'This example runs a While loop until the random number generator
'returns a value of 1.

Sub Main()
    x% = 0
    count% = 0
    While x% <> 1 And count% < 500
        x% = Rnd(1)
        If count% > 1000 Then
            Exit Sub
        Else
            count% = count% + 1
        End If
    Wend
    MsgBox "The loop run " & count% & " times."
End Sub
```

See Also

Do...Loop (statement); For...Next (statement).

Notes

Due to errors in program logic, you can inadvertantly create infinite loops in your code. Under Windows, you can break out of infinite loops using Ctrl+Break.

statement

Width#

Syntax

```
Width# filenumber,newwidth
```

Description

Specifies the line width for sequential files opened in either Output or Append mode.

Comments

- The Width# statement requires the following parameters:

Parameter	Description
filenumber	Integer used by Delrina Basic to refer to the open file-the number passed to the Open statement.
newwidth	Integer between 0 to 255 inclusive specifying the new width. If newwidth is 0, then no maximum line length is used.

- When a file is initially opened, there is no limit to line length. This command forces all subsequent output to the specified file to use the specified value as the maximum line length.
- The Width statement affects output in the following manner: if the column position is greater than 1 and the length of the text to be written to the file causes the column position to exceed the current line width, then the data is written on the next line.
- The Width statement also affects output of the Print command when used with the Tab and Spc functions.

Example

```
'This statement sets the maximum line width for file number 1 to 80
'columns.

Const crlf$ = Chr$(13) + Chr$(10)

Sub Main()
    Dim i,msg,newline$

    Open "test.dat" For Output As #1     'Create data file.
    For i = 0 To 9
        Print #1,Chr(48 + i);    'Print 0-9 to test file all on same line.
    Next i
    Print #1,crlf   'New line.
    Width #1,5   'Change line width to 5.

    For i = 0 To 9 'Print 0-9 again. This time, five characters print before line wraps.
        Print #1,Chr(48 + i);
    Next i
    Close #1

    msg = "The effect of the Width statement is as shown below: " & crlf
    Open "test.dat" For Input As #1 'Read new file.
    Do While Not Eof(1)
        Input #1,newline$
        msg = msg & crlf$ & newline$
    Loop
    Close #1
    msg = msg & crlf$ & crlf$ & "Choose OK to remove the test file."

    MsgBox msg 'Display effects of Width.
    Kill "test.dat"
End Sub
```

See Also

Print (statement); Print# (statement); Tab (function); Spc (function).

statement

WinActivate

Syntax

```
WinActivate [window_name$ | window_object] [,timeout]
```

Description

Activates the window with the given name or object value.

Comments

- The WinActivate statement requires the following parameters:

Parameter	Description
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinActivate** "Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.
timeout	Integer specifying the number of milliseconds for which to attempt activation of the specified window. If not specified (or 0), then only one attempt will be made to activate the window. This value is handy when you are not certain that the window you are attempting to activate has been created.

- If window_name$ and window_object are omitted, then no action is performed.

Example

```
'This example runs the clock.exe program by activating the Run File
'dialog box from within Program Manager.

Sub Main()
    WinActivate "Program Manager"
    Menu "File.Run"
    WinActivate "Program Manager|Run"
    SendKeys "clock.exe{ENTER}"
End Sub
```

See Also

AppActivate (statement).

statement

WinClose

Syntax

```
WinClose [window_name$ | window_object]
```

Description

Closes the given window.

Comments

- The WinClose statement requires the following parameters:

Parameter	Description
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinClose** "Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.

- If window_name$ and window_object are omitted, then the window with the focus is closed.
- This command differs from the AppClose command in that this command operates on the current window rather than the current top-level window (or application).

Example

```
'This example closes Microsoft Word if its object reference is found.

Sub Main()
    Dim WordHandle As HWND
    Set WordHandle = WinFind("Word")
    If (WordHandle Is Not Nothing) Then WinClose WordHandle
End Sub
```

See Also

WinFind (function)

Notes

The current window can be an MDI child window, a pop-up window, or a top-level window.

function

WinFind

Syntax

```
WinFind(name$) As HWND
```

Description

Returns an object variable referencing the window having the given name.

Comments

- The name$ parameter is specified using the same format as that used by the WinActivate statement.

Example

```
'This example closes Microsoft Word if its object reference is found.

Sub Main()
    Dim WordHandle As HWND
    Set WordHandle = WinFind("Word")
    If (WordHandle Is Not Nothing) Then WinClose WordHandle
End Sub
```

See Also

WinActivate (statement).

statement

WinList

Syntax

```
WinList ArrayOfWindows()
```

Description

Fills the passed array with references to all the top-level windows.

Comments

- The passed array must be declared as an array of HWND objects.
- The ArrayOfWindows parameter must specify either a zero- or one-dimensioned dynamic array or a single-dimensioned fixed array. If the array is dynamic, then it will be redimensioned to exactly hold the new number of elements. For fixed arrays, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are unused. A runtime error results if the array is too small to hold the new elements.
- After calling this function, use the LBound and UBound functions to determine the new size of the array.

Examples

```
'This example minimizes all top-level windows.

Sub Main()
    Dim a() As HWND
    WinList a
    For i = 1 To UBound(a)
        WinMinimize a(i)
    Next i
End Sub
```

See Also

WinFind (function)

statement

WinMaximize

Syntax

```
WinMaximize [window_name$ | window_object]
```

Description

Maximizes the given window.

Comments

- The WinMaximize statement requires the following parameters:

Parameter	Description
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinMaximize** "Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.

- If window_name$ and window_object are omitted, then the window with the focus is maximized.
- This command differs from the AppMaximize command in that this command operates on the current window rather than the current top-level window.

Example

```
'This example maximizes all top-level windows.

Sub Main()
    Dim a() As HWND
    WinList a
    For i = 1 To UBound(a)
        WinMaximize a(i)
    Next i
End Sub
```

See Also

WinMinimize (statement); WinRestore (statement).

Note

- The current window can be an MDI child window, a pop-up window, or a top-level window.

statement

WinMinimize

Syntax

```
WinMinimize [window_name$ | window_object]
```

Description

Minimizes the given window.

Comments

- The WinMinimize statement requires the following parameters:

Parameter	Description
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinMinimize** "Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.

- If window_name$ and window_object are omitted, then the window with the focus is minimized.
- This command differs from the AppMinimize command in that this command operates on the current window rather than the current top-level window.

Example

```
Sub Main()
    Dim a() As HWND
    WinList a
    For i = 1 To UBound(a)
        WinMinimize a(i)
    Next i
End Sub
```

See Also

WinMaximize (statement); WinRestore (statement).

Note

- The current window can be an MDI child window, a pop-up window, or a top-level window.

statement

WinMove

Syntax

```
WinMove x,y [,window_name$ | window_object]
```

Description

Moves the given window to the given x,y position.

Comments

- The WinMove statement requires the following parameters:

Parameter	Description
x,y	Integer coordinates given in twips that specify the new location for the window.
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinMove** 100**,**100**,**"Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.

- If window_name$ and window_object are omitted, then the window with the focus is moved.
- This command differs from the AppMove command in that this command operates on the current window rather than the current top-level window. When moving child windows, remember that the x and y coordinates are relative to the client area of the parent window.

Example

```
'This example moves Program Manager to upper left corner of the screen.

Sub Main()
    If AppFind$("Program Manager") = "" Then
        MsgBox "Program Manager not found!",ebExclamation,"Window Not Found"
    Else
        AppActivate "Program Manager"
        AppRestore
        WinMove 0,0,"Program Manager"
    End If
End Sub
```

See Also

WinSize (statement).

Note

- The current window can be an MDI child window, a pop-up window, or a top-level window.

statement

WinRestore

Syntax

```
WinRestore [window_name$ | window_object]
```

Description

Restores the specified window to its restore state.

Comments

- Restoring a minimized window restores that window to it screen position before it was minimized. Restoring a maximized window resizes the window to its size previous to maximizing.
- The WinRestore statement requires the following parameters:

Parameter	Description
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinRestore** "Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.

- If window_name$ and window_object are omitted, then the window with the focus is restored.
- This command differs from the AppRestore command in that this command operates on the current window rather than the current top-level window.

Example

```
'This example minimizes all top-level windows except for Program
'Manager.

Sub Main()
    Dim a() As HWND
    WinList a
    For i = 0 To UBound(a)
      WinMinimize a(i)
    Next I
    WinRestore "Program Manager"
End Sub
```

See Also

WinMaximize (statement); WinMinimize (statement.

Note

- The current window can be an MDI child window, a pop-up window, or a top-level window.

statement

WinSize

Syntax

```
WinSize width,height [,window_name$ | window_object]
```

Description

Resizes the given window to the specified width and height.

Comments

- The WinSize statement requires the following parameters:

Parameter	Description
width,height	Integer coordinates given in twips that specify the new size of the window.
window_name$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (|), as in the following example: **WinSize** 100**,**100**,**"Notepad|Find" In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name$ parameter to indicate a specific window to activate.

- If window_name$ and window_object are omitted, then the window with the focus is resized.
- This command differs from the AppSize command in that this command operates on the current window rather than the current top-level window.

Example

```
'This example runs and resizes Notepad.

Sub Main()
    id = Shell("notepad.exe",3)
    WinSize 4400,8500,"Notepad"
End Sub
```

See Also

WinMove (statement).

Note

- The current window can be an MDI child window, a pop-up window, or a top-level window.

function

Word$

Syntax

```
Word$(text$,first[,last])
```

Description

Returns a String containing a single word or sequence of words between first and last.

Comments

- The Word$ function requires the following parameters:

Parameter	Description
text$	String from which the sequence of words will be extracted.
first	Integer specifing the index of the first word in the sequence to return. If last is not specified, then only that word is returned.
last	Integer specifying the index of the last word in the sequence to return. If last is specified, then all words between first and last will be returned, including all spaces, tabs, and end-of-lines that occur between those words.

- Words are separated by any nonalphanumeric characters such as spaces, tabs, end-of-lines, and punctuation.
- If first is greater than the number of words in text$, then a zero-length string is returned.
- If last is greater than the number of words in text$, then all words from first to the end of the text are returned.

Example

```
'This example finds the name "Stuart" in a string and then
'extracts two words from the string.

Sub Main()
    s$ = "My last name is Williams; Stuart is my surname."
    c$ = Word$(s$,5,6)
    MsgBox "The extracted name is: " & c$
End Sub
```

See Also

Item$ (function); ItemCount (function); Line$ (function); LineCount (function); WordCount (function).

function

WordCount

Syntax

```
WordCount(text$)
```

Description

Returns an Integer representing the number of words in the specified text.

Comments

- Words are separated by spaces, tabs, and end-of-lines.

Example

```
'This example counts the number of words in a particular string.

Sub Main()
    s$ = "My last name is Williams; Stuart is my surname."
    i% = WordCount(s$)
    MsgBox """ & s$ & "" has " & i% & " words."
End Sub
```

See Also

Item$ (function); ItemCount (function); Line$ (function); LineCount (function); Word$ (function).

statement

Write#

Syntax

```
Write [#]filenumber [,expressionlist]
```

Description

Writes a list of expressions to a given sequential file.

Comments

- The file referenced by filenumber must be opened in either Output or Append mode.
- The filenumber parameter is an Integer used by Delrina Basic to refer to the open file-the number passed to the Open statement.
- The following table summarizes how variables of different types are written:

Data Type	Description
Any numeric type	Written as text. There is no leading space, and the period is always used as the decimal separator.
String	Written as text, enclosed within quotes.
Empty	No data is written.
Null	Written as #NULL#.
Boolean	Written as #TRUE# or #FALSE#.
Date	Written using the universal date format: #YYYY-MM-DD HH:MM:SS#
user-defined errors	Written as #ERROR ErrorNumber#, where ErrorNumber is the value of the user-defined error. The word ERROR is not translated.

- The Write statement outputs variables separated with commas. After writing each expression in the list, Write outputs an end-of-line.
- The Write statement can only be used with files opened in Output or Append mode.

Example

```
'This example opens a file for sequential write, then writes ten
'records into the file with the values 10...50. Then the file
'is closed and reopened for read, and the records are read with the
'Input statement. The results are displayed in a dialog box.

Sub Main()
    Open "test.dat" For Output Access Write As #1
    For x = 1 To 10
        r% = x * 10
        Write #1,x,r%
    Next x
    Close

    Open "test.dat" For Input Access Read As #1
    For x = 1 To 10
        Input #1,a%,b%
        msg = msg & "Record " & a% & ": " & b% & Basic.Eoln$
    Next x

    MsgBox msg
    Close
End Sub
```

See Also

Open (statement); Put (statement); Print# (statement).

statement

WriteIni

Syntax

```
WriteIni section$,ItemName$,value$[,filename$]
```

Description

Writes a new value into an ini file.

Comments

- The WriteIni statement requires the following parameters:

Parameter	Description
section$	String specifying the section that contains the desired variables, such as "windows." Section names are specified without the enclosing brackets.
ItemName$	String specifying which item from within the given section you want to change. If ItemName$ is a zero-length string (""), then the entire section specified by section$ is deleted.
value$	String specifying the new value for the given item. If value$ is a zero-length string (""), then the item specified by ItemName$ is deleted from the ini file.
filename$	String specifying the name of the ini file.

Example

```
'This example sets the txt extension to be associated with Notepad.

Sub Main()
    WriteIni "Extensions","txt","c:\windows\notepad.exe ^.txt","win.ini"
End Sub
```

See Also

ReadIni$ (function); ReadIniSection (statement).

Notes

Under Windows, if filename$ is not specified, the win.ini file is used.

If the filename$ parameter does not include a path, then this statement looks for ini files in the Windows directory.

operator

Xor

Syntax

```
expression1 Xor expression2
```

Description

Performs a logical or binary exclusion on two expressions.

Comments

- If both expressions are either Boolean, Boolean variants, or Null variants, then a logical exclusion is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	False
True	False	True
False	True	True
False	False	False

- If either expression is Null, then Null is returned.

Binary Exclusion

If the two expressions are Integer, then a binary exclusion is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long, and a binary exclusion is then performed, returning a Long result.

Binary exclusion forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

1	Xor	1	=	0	Example:
0	Xor	1	=	1	5 01101001
1	Xor	0	=	1	6 10101010
0	Xor	0	=	0	Xor 11000011

Example

```
'This example builds a logic table for the XOR function and
'displays it.

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    msg = "Logic table for Xor:" & crlf & crlf
    For x = -1 To 0
        For y = -1 To 0
            z = x Xor y
            msg = msg & CBool(x) & " Xor "
            msg = msg & CBool(y) & " = "
            msg = msg & CBool(z) & crlf
        Next y
    Next x
    MsgBox msg
End Sub
```

See Also

Operator Precedence (topic); Or (operator); Eqv (operator); Imp (operator); And (operator).

function

Year

Syntax

```
Year(date)
```

Description

Returns the year of the date encoded in the specified date parameter. The value returned is between 100 and 9999 inclusive.

The date parameter is any expression representing a valid date.

Example

```
'This example returns the current year in a dialog box.

Sub Main()
    tdate$ = Date$
    tyear! = Year(DateValue(tdate$))
    MsgBox "The current year is " & tyear!
End Sub
```

See Also

Day (function); Minute (function); Second (function); Month (function); Hour (function); Weekday (function); DatePart (function).

Get | Set

phoneBookView

Description

Sets how the phonebook displays entries.

MenuString()

Syntax

```
MenuString <charStr>

application.MenuString <charStr>
```

Parameter	Description
charStr	A string of characters representing menu actions. For example “FX” represents the menu selection “File” then “Exit”.

Description

Performs Wincomm menu actions.

DCAPI Function

dcMenuString

Command

SizeWinComm()

Syntax

```
SizeWinComm <mode>
```

Parameter	Description	
Mode	Sets the window size. Mode can have one of the following parameters:	
	DC_S_MAX	Maximizes the window.
	DC_S_MIN	Minimizes the window.
	DC_S_RSTR	Restores the window to its previous size.
	DC_S_HIDE	Hides the window.

Description

Sets the size of the WinComm application window.

DCAPI Function

dcSizeWinCommPRO

Function

nameAddress

Syntax

```
<string> = nameAddress
```

Description

Gets the strings entered during WimComm installation.

Name, company, street, city, state/province, zip.

Separated by cr/lf.

DCAPI Function

dcGetNameString

Command

SetNameAddress

Syntax

```
SetNameAddress <string>
```

Description

Sets the strings entered during WimComm installation.

Name, company, street, city, state/province, zip.

Separated by cr/lf.

DCAPI Function

dcSetNameString

Function

serialNumber

Syntax

```
<string> = serialNumber
```

Description

Gets the WinComm serial number string.

DCAPI Function

```
dcGetSerNumString
```

Command

SetSerialNumber

Syntax

```
SetSerialNumber <string>
```

Description

Sets the WinComm serial number string.

DCAPI Function

dcSetSerNumString

Function

winCommVersion

Syntax

```
<string> = winCommVersion
```

Description

Gets the WinComm version number.

DCAPI Function

dcGetVersion

Command

+>ReloadPhonebook()

Syntax

```
ReloadPhonebook
```

Description

Instructs WinComm to rescan for phonebook files.

DCAPI Function

dcReloadPhonebook

Function

RunScript

Syntax

```
<returnCode> = RunScript <scriptFileName>
```

Parameter	Description
returnCode	O is ok. The following error messages are possible: DC_ERR_CANNOT_ACCESS_FILE the specified file either does not exist or does not have read permission. DC_ERR_WRONG_FILE_TYPE the specified file is not a .dbp file.
ScriptFileName	Path and name of the file to run.

Description

Runs a script concurrently. Note that any file that can be run from menu Script | Run can be run from this extension.

Command

SavePhonebook()

Syntax

```
SavePhonebook
```

Description

Writes the WinComm preferences file to disk.

DCAPI Function

dcSavePhonebook

Command

WinCommSleep()

Syntax

```
WinCommSleep <timeFactor>
```

Parameter	Description
timeFactor	Sleep time in milliseconds.

Description

Puts the script thread to sleep for a given time.

DCAPI Function

dcSleep

Command

+>OpenSession()

Syntax

```
OpenSession <sessionFile>
```

Parameter	Description
sessionFile	The path and name of the session file to open.

Description

Opens an existing session from a session file.

DCAPI Function

dcOpenSession

Get

application

Syntax

```
session.application
```

Description

Points back to the application domain.

Get

connectionStatus

Syntax

```
<status> = session.connectionStatus
```

Parameter	Description
status	Status can be one of the following options: DC_NOTSTARTED DC_CONNECTED DC_DISCONNECTED DC_CONNECTING DC_DISCONNECTING DC_WAITING DC_SURENDERED

Description

Returns a connection status code for this session.

DCAPI Function

dcGetConnectionStatus

Get

connectionTime

Syntax

```
<connectTime> = session.connectionTime
```

Parameter	Description
connectTime	Number of seconds the session has been connected.

Description

Returns the number of seconds the session has been connected.

DCAPI Function

dcGetConnectTime

Get

+>error

Syntax

```
<err> = session.error
```

Parameter	Description
err	If 0, no error, o/w error code.

Description

Returns an error code from the session object. Each call to an extension clears out the error, so any error is the result of the last session object call.

DCAPI Function

Replacement for returning error values with functions.

Get

+>frameState

Syntax

```
<state> = session.frameState
```

Parameter	Description
state	State can have one of the following options: DC_FRAME_IS_ACTIVE DC_FRAME_IS_ICONIC DC_SESSION_IS_ACTIVE DC_SESSION_IS_ICONIC

Description

Flag describing the state of WinComm and session frame.

DCAPI Function

dcGetWindosStateBits

Set

messageTimer

Syntax

```
session.messageTimer = <timeout>
```

Parameter	Description
timeout	Nmber of seconds before timeout: 0 = no message at all. -1 = no timeout.

Description

Timeout for error and informational message boxes.

DCAPI Function

dcSetMessageTimer

Get | Set

sessionSize

Syntax

```
session.sessionSize = <mode>
```

Parameter	Description
mode	Sets the session window size. Mode can have one of the following values: DC_S_MAX DC_S_MIN DC_S_RSTR

Description

The size of the current active session.

DAPI Function

dcSizeSession

Command

ComDriverSpecial()

Syntax

```
session.ComDriverSpecial <task>
```

Parameter	Description
task	string providing specific instructions on what task a driver should carry out.

Description

Lets access to special features of specific com device drivers using a common API.

DCAPI Function

dcComDriverSpecial

Command

SkipConnection()

Syntax

```
session.SkipConnection
```

Description

Clears the preconnection program flag.

DCAPI Function

dcSkipConnection

Command

BlockRemoteInput()

Syntax

```
session.BlockRemoteInput
```

Description

Increments the session's character receive blocking counter.

DCAPI Function

dcBlockRemoteInput

Command

CaptureToPrinterBegin()

Syntax

```
session.CaptureToPrinterBegin <mode>, <method>
```

Parameter	Description
mode	Mode can have one of the following parameters:
	DC_CP_LINES
	DC_CP_CDCR
	DC_CP_SCREEN
method	Method can have one of the following parameters:
	C_CP_PAGE
	DC_CP_SESSION

Description

Turns Capture to Printer on for the current active session.

DCAPI Function

dcCaptureToPrinterBegin

Command

CaptureToPrinterControl()

Syntax

```
session. CaptureToPrinterControl <mode>
```

Parameter	Description
mode	Mode can have one of the following parameters: DC_CP_END DC_CP_PAUSE DC_CP_RESUME

Description

Changes printer capture functionality after a call to CaptureToPrinterBegin().

DCAPI Function

dcCaptureToPrinterControl

Command

Close()

Syntax

```
session.Close
```

Description

Closes the current session.

DCAPI Function

dcCloseSession

Command

Connect()

Syntax

```
session.Connect <connectMode>
```

Parameter	Description
connectMode	ConnectMode can have one of the following parameters: DC_CNCT_STANDARD DC_CNCT_LEARN_LOGIN DC_CNCT_DO_NOT_DIAL DC_CNCT_DO_NOT_LOGIN DC_CNCT_ANSWER_MODE DC_CNCT_ANSWER_HOLD

Description

Starts the connection within the current session.

DCAPI Function

dcConnectSession

Command

ConnectAndDial()

Syntax

```
session.ConnectAndDial <connectMode>, <phoneNumber>
```

Parameter	Description
connectMode	ConnectMode can have one of the following parameters: DC_CNCT_STANDARD DC_CNCT_LEARN_LOGIN DC_CNCT_DO_NOT_LOGIN
phoneNumber	A string containing the number to be dialed.

Description

Starts connection and dials the supplied telephone number rather than the number in the session settings file.

DCAPI Function

dcConnectAndDial

Command

Disconnect()

Syntax

```
session.Disconnect
```

Description

Disconnects a session.

DCAPI Function

dcDisconnectSession

Function

GetDataString()

Syntax

```
<string> = session.GetDataString( <index> )
```

Parameter	Description
string	New data string.
index	Which string to return.

Description

Returns one of the 20 data strings associated with the session.

DCAPI Function

dcDisconnectSession

Command

+>ReleaseRemoteInput()

Syntax

```
session.ReleaseRemoteInput
```

Description

Decrements the session's character receive blocking counter.

DCAPI Function

dcReleaseRemoteInput

Command

SetDataString()

Syntax

```
session.SetDataString <index>, <string>
```

Parameter	Description
index	Index of the string to be set. 1-20 (Values 1-10 are not stored with the session file.)
String	New data string.

Description

Sets one of the twenty data strings stored within each session.

DCAPI Function

dcSetSessionDataString

Command

WaitForConnection()

Syntax

```
session.WaitForConnection < status>, < timeFactor>
```

Parameter	Description
status	Status can be one of the following parameters: DC_CONNECTED DC_DISCONNECTED DC_CONNECTING
timeFactor	How long to wait for the state before timing out, measured in milliseconds.

Description

Waits for the connection status to match the supplied connection status flag.

DCAPI Function

dcWaitForConnection

Get

Transfer

Syntax

```
Dim <parameter> As Transfer
```

Parameter	Description
parameter	session.Transfer or session.Transfer.<foo>

Description

Provides the transfer object that is associated with the current session.

Get | Set

captureFileName

Syntax

```
<filename> = session.captureFileName
```

session.captureFileName = <filename>

Parameter	Description
filename	The filename string.

Description

Sets the name of the capture file for the currently active session.

DCAPI Function

dcGetCaptureFileName

dcSetCaptureFileName

Get | Set

logFileName

Syntax

```
<filename> = session.logFileName

session.logFileName= <filename>
```

Parameter	Description
filename	The filename string.

Description

Sets the name of the current log file.

DCAPI Function

dcGetLogFileName

dcSetLogFileName

Set

logonTask

Syntax

```
session.logonTask = <filename>
```

Parameter	Description
filename	The filename string.

Description

Sets the task by file name that is run when the current active session successfully completes a connection.

DCAPI Function

dcSetLogonTask

Command

CaptureToFileBegin()

Syntax

```
session.CaptureToFileBegin <mode>, <append>
```

Parameter	Description
mode	Mode can have one of the following parameters: DC_C_LINES DC_C_CHAR DC_C_RAW DC_C_SCREEN
append	Boolean flag. True = append.

Description

Turns the Capture to File function on for the current session.

DCAPI Function

dcCaptureBegin

Command

CaptureToFileControl()

Syntax

```
session.CaptureToFileControl <changeCode>
```

Parameter	Description
changeCode	ChangeCode can have one of the following parameters: DC_C_END DC_C_PAUSE DC_C_RESUME

Description

Changes a capture operation after CaptureBegin().

DCAPI Function

dcCaptureControl

Command

WriteLogEntry()

Syntax

```
session.WriteLogEntry <entry>
```

Parameter	Description
entry	The string that is added to the log file.

Description

Appends a timed and dated entry to the log file for the current active session.

DCAPI Function

dcWriteLogEntry

Get | Set

asciiInputWaitChar

Syntax

```
<setting> = session.asciiInputWaitChar

session.asciiInputWaitChar = <setting>
```

Parameter	Description
setting	Character to wait for.

Description

Waits for a character to be entered.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

+>asciiOutputTabValue

Syntax

```
<setting> = session.asciiOutputTabValue
```

```
session.asciiOutputTabValue = <setting>
```

Parameter	Description
setting	Tab expansion integer value.

Description

Sets the output tab expansion value.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiCharDelayValue

Syntax

```
<setting> = session.asciiCharDelayValue
```

```
session.asciiCharDelayValue = <setting>
```

Parameter	Description
setting	Char delay integer value.

Description

Sets the character delay value.

DCAPI Function

```
dcGetAsciiSettings
```

```
dc SetAsciiSettings
```

Get | Set

asciiLineDelayValue

Syntax

```
<setting> = session.asciiLineDelayValue
```

```
session.asciiLineDelayValue = <setting>
```

Parameter	Description
setting	Line delay integer value.

Description

Sets the line delay value.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiInputTabValue

Syntax

```
<setting> = session.asciiInputTabValue
```

```
session.asciiInputTabValue = <setting>
```

Parameter	Description
setting	Input tab integer value.

Description

Sets the input tab conversion value.

DCAPI Function

```
dcGetAsciiSettings
```

```
dcSetAsciiSettings
```

Get | Set

asciiAppendLFSend

Syntax

```
<setting> = session.asciiAppendLFSend
session.asciiAppendLFSend = <setting>
```

Parameter	Description
setting	Boolean.

Description

Appends a line feed to send line ends.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

+>asciiExpandBlankToSpace

Syntax

```
<setting> = session.asciiExpandBlankToSpace

session.asciiExpandBlankToSpace = <setting>
```

Parameter	Description
setting	Boolean.

Description

Expands blank lines sent to include a space.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiLocalEcho

Syntax

```
<setting> = session.asciiLocalEcho
```

```
session.asciiLocalEcho = <setting>
```

Parameter	Description
setting	Boolean.

Description

Echos typed characters locally.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiWaitForLineEnd

Syntax

```
<setting> = session.asciiWaitForLineEnd
session.asciiWaitForLineEnd = <setting>
```

Parameter	Description
setting	Boolean.

Description

Enables wait for line end character.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiTabExpandSend

Description

Uses asciiOutputTabValue and asciiInputTabValue.

Enables tab expansion for sent characters.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiAppendLFReceive

Syntax

```
<setting> = session.asciiAppendLFReceive

session.asciiAppendLFReceive = <setting>
```

Parameter	Description
setting	Boolean.

Description

Appends line feeds to received line ends.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

ascii7BitReceive

Syntax

```
<setting> = session.ascii7BitReceive

session.ascii7BitReceive = <setting>
```

Parameter	Description
setting	Boolean.

Description

Forces incoming characters to seven bits.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiRemoteEcho

Syntax

```
<setting> = session.asciiRemoteEcho

session.asciiRemoteEcho = <setting>
```

Parameter	Description
setting	Boolean.

Description

Echos received characters to sender.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiWrapLines

Syntax

```
<setting> = session.asciiWrapLines

session.asciiWrapLines = <setting>
```

Parameter	Description
setting	Boolean.

Description

Wraps lines that exceed the terminal width.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

asciiShowHex

Syntax

```
<setting> = session.asciiShowHex
```

```
session.asciiShowHex = <setting>
```

Parameter	Description
setting	Boolean.

Description

Shows the hex value of non-printing characters.

DCAPI Function

dcGetAsciiSettings

dcSetAsciiSettings

Get | Set

baudRate

Syntax

```
<setting> = session.baudRate

session.baudRate = <setting>
```

Parameter	Description
setting	Long.

Description

Sets the baud rate for the current connection, if the port type in use for the currently active session supports baud rates.

DCAPI Function

dcGetBaudRate

dcSetBaudRate

Get | Set

+>bitsPerCharacter

Syntax

```
<setting> = session.bitsPerCharacter
```

```
session.bitsPerCharacter = <setting>
```

Parameter	Description
setting	Setting can have one of the following parameters: DC_M_7_BITS DC_M_8_BITS

Description

Sets the current bits per character for this session.

DCAPI Function

dcGetPortMode

dcSetPortMode

Get | Set

parity

Syntax

```
<setting> = session.parity

session.parity  = <setting>
```

Parameter	Description
setting	Setting can have one of the following paramters: DC_M_E_PRTY DC_M_O_PRTY DC_M_N_PRTY DC_M_M_PRTY DC_M_S_PRTY

Description

Sets the current parity for this session.

DCAPI Function

dcGetPortMode

dcSetPortMode

Get | Set

stopBits

Syntax

```
<setting> = session.stopBits
```

```
session.stopBits = <setting>
```

Parameter	Description
setting	setting can have one of the following parameters: DC_M_1_STOP DC_M_2_STOP

Description

Sets the current stop bits for the current session.

DCAPI Function

dcGetPortMode

dcSetPortMode

Get | Set

dialingMethod

Syntax

```
<setting> = session.dialingMethod
```

```
session.dialingMethod = <setting>
```

Parameter	Description
setting	setting can have one of the following parameters: DC_PT_PULSE DC_PT_TONE

Description

Sets the device dialing method for pulse or tone dialing.

DCAPI Function

dcGetPulseTone

dcSetPulseTone

Get | Set

emulator

Syntax

```
<name> = session.emulator

session.emulator = <name>
```

Parameter	Description
name	The string which contains the emulator name.

Description

Sets the name of the emulator for the currently active session.

DCAPI Function

dcGetEmulator

dcSetEmulator

Get | Set

portDevice

Syntax

```
<name> = session.portDevice

session.portDevice = <name>
```

Parameter	Description
name	The string which contains the communications port device name.

Description

Sets the port device for the currently active session.

DCAPI Function

dcGetPortPrefs

dcSetPortPrefs

Get | Set

portName

Syntax

```
<name> = session.portName

session.portName = <name>
```

Parameter	Description
name	The string which contains the port name. COM1, etc.

Description

Sets the port name for the current session.

DCAPI Function

dcGetPortName

dcSetPortName

Get | Set

portType

Syntax

```
<name> = session.portType
```

```
session.portType = <name>
```

Parameter	Description
name	The string which contains the port type.

Description

Sets the type of communications port for the current session.

DCAPI Function

dcGetPortName

dcSetPortName

Get | Set

ringsForAnswer

Syntax

```
<rings> = session.ringsForAnswer

session.ringsForAnswer = <rings>
```

Parameter	Description
rings	An integer value for the number of rings until an incoming call before it is answered.

Description

Sets the number of rings for an incoming call before it is answered.

DCAPI Function

dcGetRingsForAnswer

dcSetRingsForAnswer

Get

numberReceived

Syntax

```
<number> = session.numberReceived
```

Parameter	Description
number	Integer containing the number of characters received on the last successful call to this session object's GetInput function.

Description

To keep the syntax of this function consistent between OLE (Visual Basic) and Delrina Basics, the number of characters received is retrieved by this get method. Visual Basic cannot pass a reference parameter to OLE and the GetInput function returns a string..

DCAPI Function

dcGetInput

Command

EnableEmulatorDisplay

Syntax

```
session.EnableEmulatorDisplay
```

Description

Enables the emulator display.

DCAPI Function

dcHideInput

Command

DisableEmulatorDisplay

Syntax

```
session.DisableEmulatorDisplay
```

Description

Disables the emulator display.

DCAPI Function

dcHideInput

Command

EnableLocalEcho

Syntax

```
session.EnableLocalEcho
```

Description

Enables the local display of transmitted characters.

DCAPI Function

```
dcSetLocalEcho
```

Command

+>DisableLocalEcho

Syntax

```
session.DisableLocalEcho
```

Description

Disables the local display of transmitted characters.

DCAPI Function

dcSetLocalEcho

Command

EnableRemoteEcho

Syntax

```
session.EnableRemoteEcho
```

Description

Enables the remote echo of received characters.

DCAPI Function

dcSetEcho

Command

DisableRemoteEcho

Syntax

```
session.DisableRemoteEcho
```

Description

Disables the remote echo of received characters.

DCAPI Function

dcSetEcho

Command

Enable7BitStrip

Syntax

```
session.Enable7BitStrip
```

Description

Enables the stripping of received characters to seven bits.

DCAPI Function

dcSetStripTo7Bits

Command

Disable7BitStrip

Syntax

```
session.Disable7BitStrip
```

Description

Disables stripping of received characters to seven bits.

DCAPI Function

dcSetStripTo7Bits

Command

ClearOutputBuffer()

Syntax

```
session.ClearOutputBuffer
```

Description

Clears WinComm's character output buffer.

DCAPI Function

dcClearOutputBuffer

Function

GetInput()

Syntax

```
<buffer> = session.GetInput( <mode>, <count>, <startTimeout>, <charTimeout>
)
```

Parameter	Description
Buffer	A string returned containing the data. This string is allocated within the extension to hold ‘count’ characters.
Mode	Mode can have one of the following paramters:
	If DC_GI_BACKSPACE, backspace characters are processed before the buffer is returned.
	If DC_GI_EOL, return after a carriage return. ‘count’ is still the size of the return buffer to allocate.
	The two mode values may be ORed together, otherwise send 0 if specifying neither.
Count	umber of characters to wait for AND the size of the return buffer to allocate. If the mode includes DC_GI_EOL, this value only specifies the size of the return buffer to allocate.
StartTimeout	he number of milliseconds to wait for the first character (in a long) to wait before returning DC_ERR_TIMEOUT.
CharTimeout	he number of milliseconds to wait (in a long) between characters before returning.

Description

Get data received from the remote system and fills a string buffer with it.

DCAPI Function

```
dcGetInput
```

Function

GetRuntimeValue()

Syntax

```
<runtimeString> = session.GetRuntimeValue( <value>, <prompt> )
```

Parameter	Description
runtimeString	String of text returned.
Value	Which value is requested. Value can have one of the following parameters:
	DC_RV_USERNAME
	DC_RV_USERID
	DC_RV_PASSWORD
prompt	Boolean. If true, prompt the user for a value if none has been stored in the session.

Description

Returns one of the runtime strings stored with each session.

DCAPI Function

dcGetRuntimeValue

Function

GetTextFromScreen()

Syntax

```
<text> = session.GetTextFromScreen( <row>, <col>, <#ofchars> )
```

Parameter	Description
text	String of text returned.
Row	Integer screen row.
Col	Integer screen column.
#ofchars	Integer number of characters to return.

Description

Returns text from the emulator screen.

DCAPI Function

dcGetTextFromScreen

Command

TypeLocalText()

Syntax

```
session.TypeLocalText <textString>
```

Parameter	Description
textString	String of text.

Description

Displays text on screen using the current emulator.

DCAPI Function

dcTypeLocalText

Command

TypeText()

Syntax

```
session.TypeText <textString>
```

Parameter	Description
textString	String of text.

Description

Transmits a string of characters through a connected session's port.

DCAPI Function

dcTypeText

Command

+>WaitForActivity()

Syntax

```
session.WaitForActivity <timeout>
```

Parameter	Description
timeout	How long to wait in a long before setting DC_ERR_TIMED_OUT.

Description

Waits for any characters to be received from the remote system.

DCAPI Function

dcWaitForActivity

Command

WaitForLines()

Syntax

```
session.WaitForLines <lines> <timeout>
```

Parameter	Description
Lines	The number of lines to wait for in a long.
Timeout	How long to wait in a long before setting DC_ERR_TIMED_OUT.

Description

Waits for a given number of lines to be received.

DCAPI Function

dcWaitForLines

Command

WaitForLull()

Syntax

```
session.WaitForLull <lull> <timeout>
```

Parameter	Description
lull	The number of millisecond lull between characters in a long.
Timeout	How long to wait in a long before setting DC_ERR_TIMED_OUT.

Description

Waits for a gap (pause) in the input data stream of the given length.

DCAPI Function

dcWaitForLull

Command

WaitForOutputDone()

Syntax

```
session.WaitForOutputDone <timeout>
```

Parameter	Description
timeout	How long to wait in a long before setting DC_ERR_TIMED_OUT.

Description

Waits for a session to finish sending characters which may be in the buffer.

DCAPI Function

dcWaitForOutputDone

Function

WaitForPrompt()

Syntax

```
<index> = session.WaitForPrompt <strings> <pause> <timeout>
```

Parameter	Description
index	Returns the index into your array of strings of the string that was matched if the timeout did not occur beforehand.
Strings	A string with embedded nulls separating each string to wait for with an explicit null at the end.
Pause	Number of milliseconds of a pause which must occur after one of the strings is matched before a successful return.
Timeout	How long to wait in a long before setting DC_ERR_TIMED_OUT.

Description

Checks for a match between received characters and given strings. Requires a pause gap after a match.

Example

```
st = "Hello" + Chr$(0) + "there" + Chr$(0) + "end." + Chr$(0)
i = S.WaitForPrompt( st, 1000, 1000 )
```

DCAPI Function

dcWaitForPrompt

Function

WaitForString()

Syntax

```
<index> = session.WaitForString <strings> <timeout>
```

Parameter	Description
index	Returns the index into your array of strings of the string that was matched if the timeout did not occur beforehand.
Strings	A string with embedded nulls separating each string to wait for with an explicit null at the end.
Timeout	How long to wait in a long before setting DC_ERR_TIMED_OUT.

Description

Checks for a match between received characters and given strings but does not require a pause after a string match.

Example:

st = "Hello" + Chr$(0) + "there" + Chr$(0) + "end." + Chr$(0)

i = S.WaitForString(st, 1000)

DCAPI Function

dcWaitForString

ClearDropList()

Syntax

```
transfer.ClearDropList
```

Description

Clears the internal list of files dropped on an open session.

ClearSendList()

Syntax

```
transfer.ClearSendList
```

Description

Clears the internal list of files queued up to be transmitted.

DropSend()

syntax

```
transfer.DropSend <waitFlag>
```

Parameter	Description
waitFlag	Boolean. If TRUE, wait for the transfer to have been completed before returning. If FALSE, return as soon as the transfer has started.

Description

Sends files that have been placed on the internal dropped files list.

GetDropList()

syntax

```
<string> = transfer.GetDropList( <index> )
```

Parameter	Description
index	Select which item to return. Selections start at integer 0.

Description

Return the path and filename of files dropped on the session.

DCAPI Function

dcGetXferDropList

Get

finalStatus

Syntax

<status> = session.finalStatus

Parameter	Description
status	Status can have one of the following values: DC_XFS_OK DC_XFS_RMT_CANNED DC_XFS_USER_CANNED DC_XFS_LOST_CARRIER DC_XFS_ERROR_LIMIT DC_XFS_NO_RESPONSE DC_XFS_OUT_OF_SEQ DC_XFS_BAD_FORMAT DC_XFS_TOO_MANY DC_XFS_DISK_FULL DC_XFS_CANT_OPEN DC_XFS_DISK_ERROR DC_XFS_NO_MEM DC_XFS_FILE_EXISTS DC_XFS_COMPLETE DC_XFS_CANT_START DC_XFS_OLDER_FILE DC_XFS_NO_FILETIME DC_XFS_WONT_CANCEL DC_XFS_GEN_FAILURE DC_XFS_NO_VSCAN DC_XFS_VIRUS_DETECT DC_XFS_USER_SKIP DC_XFS_REFUSE

Description

Returns the final status of the most recent file transfer.

DCAPI Function

dcGetXferFinalStatus

Get | Set

parameters

Syntax

```
<status> = session.parameters

session.parameters = <status>
```

Parameter	Description
status	These are flag bits in a long format: XF_DN_MASK XF_DN_APPEND XF_DN_OVERWRT XF_DN_REFUSE XF_DN_NEWER XF_DN_DATE XF_DN_SEQ XF_CHECK_VIRUS XF_USE_FILENAME XF_USE_DIRECTORY XF_SAVE_PARTIAL XF_USE_DATETIME

Description

File transfer operation flags.

DCAPI Function

dcGetXferParameters

dcSetXferParameters

Get

+>percentComplete

Syntax

```
<status> = session.percentComplete
```

Parameter	Description
status	An integer percentage value that indicates how much of a transfer has completed (0-99).

Description

Indicates how much of a file transfer is completed (in percent).

DCAPI Function

```
dcGetXferStatus
```

Set | Get

receivingDirectory

Syntax

```
<dir> = session.receivingDirectory

session.receivingDirectory = <dir>
```

Parameter	Description
dir	The name of the transfer directory in a string.

Description

Displays the receiving file transfer directory.

DCAPI Function

dcGetXferDirectory

dcSetXferDirectory

Get | Set

+>sendingDirectory

Syntax

```
<dir> = session.sendingDirectory

session.sendingDirectory = <dir>
```

Parameter	Description
dir	The name of the transfer directory in a string.

Description

Displays the sending file transfer directory.

DCAPI Function

dcGetXferDirectory

dcSetXferDirectory

Get | Set

sendProtocol

Syntax

```
<protocol> = session.sendProtocol

session.sendProtocol = <protocol >
```

Parameter	Description
protocol	Protocol can have one of the following parameters: DC_HYPERP DC_COMPUSERV_B DC_KERMIT DC_XMODEM DC_XMODEM_1K DC_YMODEM DC_YMODEM_G DC_ZMODEM

Description

Sets the sending file transfer protocol.

DCAPI Function

dcGetXferProtocol

dcSetXferProtocol

Get | Set

receiveProtocol

Syntax

```
<protocol> = session.receiveProtocol

session.receiveProtocol = <protocol >
```

Parameter	Description
protocol	Protocol can have one of the following parameters: DC_HYPERP DC_COMPUSERV_B DC_KERMIT DC_XMODEM DC_XMODEM_1K DC_YMODEM DC_YMODEM_G DC_ZMODEM

Description

Sets the receiving file transfer protocol.

DCAPI Function

dcGetXferProtocol

dcSetXferProtocol

Command

AddToSendList()

Syntax

```
session.AddToSendList <filename>
```

Parameter	Description
filename	The name and path of the file in a string.

Description

Adds a path and filename to the internal list of files queued for sending.

DCAPI Function

dcXferAddToSendList

Command

Receive()

Syntax

```
session.Receive <filedir>, <waitFlag>
```

Parameter	Description
filedir	The filename or directory in a string.
WaitFlag	Boolean flag. TRUE = don't return until complete.

Description

Begins receiving a file or group of files sent remotely.

DCAPI Function

dcXferReceive

Command

Send()

Syntax

```
session.Send <filename>, <waitFlag>
```

Parameter	Description
filename	The filename in a string.
WaitFlag	Boolean flag. TRUE = don't return until complete.

Description

Adds a single file to the internal list of files to transfer and begins the transfer.

DCAPI Function

dcXferSend

Command

SendBatch()

Syntax

```
transfer.SendBatch <filename>, <waitFlag>
```

Parameter	Description
filename	The filename of the list of files in a string.
WaitFlag	Boolean flag. TRUE = don't return until complete.

Description

Names a batch file to the transfer queue and begins transfer. The ASCII text file contains one file description per line.

DCAPI Function

dcXferSendBatch

Command

SendFromList()

Syntax

```
session.SendFromList <waitFlag>
```

Parameter	Description
waitFlag	Boolean flag. TRUE = don't return until complete.

Description

Sends files on the internal list of files to transfer using the current default protocol.

DCAPI Function

```
dcXferSendFromList
```

Command

TextSend()

Syntax

```
session.TextSend <filename>
```

Parameter	Description
filename	Path and filename of the file to be sent in a string.

Description

Queues a text file for transmission by the connected session.

DCAPI Function

dcTextSend

Command

WaitForTransfer()

Syntax

```
session.WaitForTransfer <timeout>
```

Parameter	Description
timeout	The number of milliseconds to wait in a long.

Description

Waits a specified amount of time for a transfer to complete.

DCAPI Function

dcWaitForTransfer

Get | Set

sizePhonebook

Syntax

```
phonebook.SizePhonebook <mode>
```

Parameter	Description
mode	Sets the phonebook size. Mode can have one of the following values: DC_S_MAX DC_S_MIN DC_S_RSTR

Description

Changes the size of the phonebook.

+>GetEntry()

Syntax

```
filename = phonebook.GetEntry( <index> )
```

Parameter	Description
index	The index into the phonebook that selects the session filename desired.
Filename	The filename for the entry specified by the index parameter.

Description

Returns the session filename of the given session index.

DCAPI Function

dcGetPhonebookEntry

GetSelectedEntry()

Syntax

```
filename = phonebook.GetSelectedEntry( <index> )
```

Parameter	Description
index	The index of the filename to return from the list of selected entries.
filename	The filename for the selected entry specified by the index parameter.

Description

Returns the filename of a selected entry in the phonebook.

DCAPI Function

dcGetSelectedPhonebookEntry

Get

portDeviceName

Syntax

```
<name> = session.portDeviceName

session.portDeviceName = <name>
```

Parameter	Description
name	String which contains the communications port device name. For example, Hayes Accura 144 + Fax 144.

Description

Sets the port device name for the current session.

DCAPI Command

dcGetPortPrefs

+>Command

SetPhoneNumber()

Syntax

```
session.SetPhoneNumber <session file name>, <countryCode>, <areaCode>,
<phoneNumber>, <long distance flag>
```

Parameter	Description
session file name	A string providing specific session file name with a possible path.
CountryCode	The country code in a string.
AreaCode	The area code in a string.
PhoneNumber	The phone number to set in a string.
long distance flag	Boolean. TRUE for log distance, FALSE for local.

Description

Sets the phone number and long distance flag for a TAPI session file. This must be done on a session file that is not open. The change to this data will be present when the session is opened. If the session is not opened, ReloadPhonebook can be called to update the session properties. Will return DC_ERR_NO_SESSION if not a TAPI session and no changes will be performed.

Example

sess.SetPhoneNumber “mysess.wcs”, “1”, “716”, “555-9999”, FALSE

Function

SetRuntimeValue()

Syntax

```
session.SetRuntimeValue <whichvalue>, <runtimestring>
```

Value

Parameter	Description	
whichvalue	Determines which runtime value is set. The options are:	
	DC_RV_USERNAME	Sets the full name.
	DC_RV_USERID	Sets the user ID.
	DC_RV_PASSWORD	Sets the password.
runtimeString	String of text to set.	

Description

Sets one of the strings in the session.