

Command Reference

This chapter describes each element (statement, function, constant, operator, etc.) provided by the Desktop Control Language (DCL). The language elements are listed alphabetically. For most elements, the following information is provided:

- Description.
- Syntax.
- Comments offering further explanation.
- Example. This is either a code sample or a reference to a topic containing a code sample.
- See Also. This is a reference to groups of related language elements in Chapter 7, “Related Commands” and to additional elements in this chapter.

The following symbols are used in this chapter:

Indicates that a statement or function cannot be used in a DOS environment. (It can only be used under Windows.)



A single DCL command cannot be broken into multiple lines separated by carriage return/line feeds ((Chr\$(13)+Chr\$(10))). Because of the width of the physical page, long commands used in code samples must be broken. In these cases, the symbol indicates that the current line continues without a carriage return/line feed.

'

Description	Begins a comment line or comments the remainder of the current line.
Syntax	<code>' text</code>
Comments	Causes the compiler to skip all characters between this character and the end of the current line.
See Also	Comments (Chapter 6) REM Statement

Description	Multiplication operator.
Syntax	<code>expression1 * expression2</code>
Returns	Returns the product of <code>expression1</code> and <code>expression2</code> .
See Also	Operators (Chapter 7)

+

Description	Addition operator.
Syntax	<code>expression1 + expression2</code>
Returns	Returns the sum of <code>expression1</code> and <code>expression2</code> .
See Also	Operators (Chapter 7)

-

Description	Subtraction operator.
Syntax	<code>expression1 - expression2</code>

Returns	Returns the difference between <code>expression1</code> and <code>expression2</code> .
See Also	Operators (Chapter 7)

/

Description	Long Division operator.
Syntax	<code>expression1 / expression2</code>
Returns	Returns the quotient of <code>expression1</code> and <code>expression2</code> .
See Also	Operators (Chapter 7)

<

Description	Less Than operator.
Syntax	<code>expression1 < expression2</code>
Returns	TRUE if <code>expression1</code> is less than <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is less than <code>expression2</code> .
See Also	Operators (Chapter 7) Strings (Chapter 7)

<=

Description	Less Than or Equal To operator.
Syntax	<code>expression1 <= expression2</code>
Returns	TRUE if <code>expression1</code> is less than or equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is less than or equal to <code>expression2</code> .
See Also	Operators (Chapter 7) Strings (Chapter 7)



Description	Not Equal operator.
Syntax	<code>expression1 <> expression2</code>
Returns	TRUE if <code>expression1</code> is not equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is not equal to <code>expression2</code> .
See Also	Operators (Chapter 7) Strings (Chapter 7)



Description	Equal To operator.
Syntax	<code>expression1 = expression2</code>
Returns	TRUE if <code>expression1</code> is equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is equal <code>expression2</code> .
See Also	Operators (Chapter 7) Strings (Chapter 7)



Description	Greater Than operator.
Syntax	<code>expression1 > expression2</code>
Returns	TRUE if <code>expression1</code> is greater than <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is greater than <code>expression2</code> .
See Also	Operators (Chapter 7) Strings (Chapter 7)

>=

Description	Greater Than or Equal To operator.
Syntax	<code>expression1 >= expression2</code>
Returns	TRUE if <code>expression1</code> is greater than or equal to <code>expression2</code> , otherwise FALSE. If the two expressions are strings, then the operator performs a case-sensitive comparison between the two expressions, returning TRUE if <code>expression1</code> is greater than or equal to <code>expression2</code> .
See Also	Operators (Chapter 7) Strings (Chapter 7)

\

Description	Integer Division operator.
Syntax	<code>expression1 \ expression2</code>
Returns	Returns the integer portion of the quotient of <code>expression1</code> divided by <code>expression2</code> . Any fractional part of the answer is dropped. For example <code>3 \ 1</code> equals 1 rather than 1.5.
See Also	Operators (Chapter 7)

^

Description	Exponentiation operator.
Syntax	<code>expression1 ^ expression2</code>
Returns	Returns <code>expression1</code> raised to the power specified by <code>expression2</code> .
See Also	Operators (Chapter 7)

Abs Function

Description	Returns the absolute value of a given number.
Syntax	<code>abs (number)</code>

Example

```

sub main()
    'ABS() - absolute value
    dim a as single
    dim b as integer

    a = 2.34
    b = abs(a)
    msgbox str$(b)
end sub

```

See Also Math Statements and Functions (Chapter 7)

ActivateControl Statement

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

Syntax 1 `ActivateControl ControlName$`

Syntax 2 `ActivateControl ControlID%`

Comments The control can be referenced using either the `ControlName$` or the `ControlID%`.

For push buttons, radio buttons, or check boxes, `ControlName$` is the name of the actual button. For listboxes, comboboxes, and edit boxes, `ControlName$` is the name that appears within the static text control that immediately precedes it in the window manager list.

A runtime error is generated if a control with `ControlName$` or `ControlID%` cannot be found.

This statement is used primarily to set the focus to a custom control within a dialog box. This is accomplished by setting the focus first to a known control that immediately precedes the custom control, then simulating a TAB key press:

```

ActivateControl "Portrait"
DoKeys "{TAB}"

```

Example Dialog Examples

See Also Dialog Manipulation (Chapter 7)

AddIni Function

Description Reads the .INI settings of the source .INI file and adds them to the destination .INI file.

Syntax	<code>AddIni(srcfile\$, destfile\$)</code>
Comments	<p>If the <code>destfile\$</code> is not provided, WIN.INI is used. If the specified destination file does not exist, it is created. If an .INI setting is not present in the destination file, it is added. The function returns TRUE if it completes successfully or FALSE if it doesn't.</p> <p>It is assumed that all entries in the <code>srcfile\$</code> .INI file are in standard .INI format. Any entries that do not follow this Windows standard are ignored. If the specified source file does not exist, the function ends.</p>
Example	<pre>sub main() 'Example of AddIni 'add the contents of myini.ini to win.ini a% = AddIni("myini.ini", "win.ini") if a% then msgbox "INI settings added to Win.INI" else msgbox "AddIni failed." end if end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

And

Description	And operator.
Syntax	expression1 And expression2
Returns	TRUE if expression1 is TRUE and expression2 is TRUE, otherwise FALSE. If the two operands are numeric, then the result is the bitwise AND of the two arguments.
Notes	If either of the two operands is a floating point number, then the two operands are first converted to longs, then a bitwise AND is performed.
Example	<pre>sub main() 'AND statement dim a as integer dim b as integer a = 5 b = 9 if (a < 6) AND (b > 8) then msgbox "Both conditions were true." else msgbox "Both conditions were not true." end if if (a < 6) AND (b > 9) then msgbox "Both conditions were true." else msgbox "Both conditions were not true." end if end sub</pre>
See Also	Operators (Chapter 7)

AnswerBox Function

Description	Displays a box that prompts the user for a response and indicates which button the user pressed.
Syntax	AnswerBox(prompt\$ [,button1\$ [,button2\$ [,button3\$]]])
Returns	Returns an integer indicating which button was pushed (1 for the first button, 2 for the second, and so on). 0 is returned if the user cancels the dialog box (by pressing Escape).

Comments	<p>The dialog box is sized to hold the entire contents of <code>prompt\$</code>. The <code>prompt\$</code> string can contain CR/LF, <code>Chr\$(13)+Chr\$(10)</code>, to separate lines.</p> <p>The maximum size of the dialog box is 5/8 the width of the screen and 3/4 the height of the screen. If a given line is too long, it will be word wrapped.</p> <p>The <code>button\$</code> parameters specify one or more buttons to appear below the displayed <code>prompt\$</code>. If no buttons are specified, then "OK" and "Cancel" are used. Up to three buttons can be specified. The width of each button is determined by the width of the widest button.</p> <p>You can use the '&' symbol to specify an accelerator key in the label of the button.</p> <p>The dialog box uses the 8 point Helvetica font.</p>
Example	<pre>sub main() 'AnswerBox example 'This is a default answer box without 'specifying the buttons r% = AnswerBox("Example prompt?") msgbox str\$(r%) 'display the result r% = AnswerBox("Example prompt 2?", "&Maybe", "&OK", "Maybe &Not") msgbox str\$(r%) 'display the result end sub</pre>
See Also	Dialog Display (Chapter 7)

AppActivate Statement

Description	Activates the specified top-level window.
Syntax	<code>AppActivate WindowName\$</code>
Comments	<p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If the application is minimized, it will be restored by this command. If the window being activated is a full-screen DOS application, that application will be restored to full screen.</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box) or if the window is not found.</p>
Example	<pre>sub main() 'AppActivate example 'activate Applications Manager AppActivate "Applications Manager" end sub</pre>

See Also Window Manipulation (Chapter 7)

AppClose Statement

Description Closes the specified top-level application.

Syntax `AppClose [WindowName$]`

Comments The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If `WindowName$` parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example

```
sub main()
    'AppClose example

    'search for a copy of Notepad that is running
    appname$ = AppFind$("Notepad")

    'close the copy of Notepad that was found
    AppClose appname$
end sub
```

See Also Window Manipulation (Chapter 7)

AppFileName\$ Function

Description Returns the filename of the program that owns the top-level window of the given title.

Syntax `AppFileName$(WindowName$)`

Comments The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

For DOS applications, the filename returned is that of the actual executable program, not WINOLDAP.EXE.

Example

```
sub main()
```

```

'example of AppFileName$

'find a copy of Notepad that is running
appname$ = AppFind$("Notepad")

'get the name of the program it was executed from
appfile$ = AppFileName$(appname$)

'display the file name
MsgBox appfile$
end sub

```

See Also Window Manipulation (Chapter 7)

AppFind Function

Description Returns the full name of the top-level window, given a partial window name.

Syntax `AppFind$(partial_name$)`

Comments The name is specified using the same format as that used by the `WinActivate` statement.

Example

```

sub main()
    'example of AppFind$

    appname$ = AppFind$("Notepad")
    MsgBox appname$
end sub

```

See Also Window Manipulation (Chapter 7)

AppGetActive\$ Function

Description Returns the title of the active window.

Syntax `AppGetActive$()`

Comments This function is used to retrieve the title of the active top-level window. The returned value can be used in subsequent calls to routines that require a title to a window.

If "" is returned, no window is active. This is a rare occurrence, it indicates that Windows may be in an unstable state.

Example 1

```

sub main()
    'example of AppGetActive$

    'activate a copy of Notepad
    appname$ = AppFind$("Notepad")
    AppActivate appname$

    'now see if it is active
    appname$ = AppGetActive$()
    MsgBox appname$
end sub

```

Example 2

```

n$ = AppGetActivate$
AppMinimize n$

```

See Also

Window Manipulation (Chapter 7)

AppGetPosition Statement

Description

Gets the position of a specified top-level window.

Syntax

```
AppGetPosition x%,y%,width%,height% [,WindowName$]
```

Comments

The numeric arguments are filled with the pixel position of the window on the display. If an argument is not a variable reference, then the argument is ignored, as in the following example which only retrieves the position and ignores the width and height:

```

dim x as integer,y as integer
AppGetPosition x,y,0,0,"Program Manager"

```

The WindowName\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If WindowName\$ parameter is missing, then the window with the focus is used (i.e., the active window).

Example

```

sub main()
    'example of AppGetPosition
    dim x, y, w, h as integer

    'get the position, width and height of Notepad
    appname$ = AppFind$("Notepad")
    AppGetPosition x, y, w, h, appname$

    'display them
    MsgBox appname$+" is at "+str$(x)+",""+str$(y)+ " with width of "+ ←
        str$(w)+" and height of "+str$(h)

```

```
end sub
```

See Also Window Manipulation (Chapter 7)

AppGetState Function

Description	Returns an integer representing the state of the top-level window.						
Syntax	AppGetState (WindowName\$)						
Returns	<table><tr><td>WS_MAXIMIZED</td><td>window is maximized</td></tr><tr><td>WS_MINIMIZED</td><td>window is minimized</td></tr><tr><td>WS_RESTORED</td><td>window is restored</td></tr></table>	WS_MAXIMIZED	window is maximized	WS_MINIMIZED	window is minimized	WS_RESTORED	window is restored
WS_MAXIMIZED	window is maximized						
WS_MINIMIZED	window is minimized						
WS_RESTORED	window is restored						
Comments	<p>The WindowName\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If WindowName\$ parameter is missing, then the window with the focus is used (i.e., the active window).</p>						
Example	<pre>sub main() 'example of AppGetState appname\$ = AppFind\$("Notepad") appstate% = AppGetState(appname\$) if appstate% = WS_MAXIMIZED then MsgBox appname\$ + " is Maximized - " + str\$(appstate%) else if appstate% = WS_MINIMIZED then MsgBox appname\$ + " is Minimized - " + str\$(appstate%) else if appstate% = WS_RESTORED then MsgBox appname\$ + " is Restored - " + str\$(appstate%) else MsgBox appname\$ + " is in an unknown state - " + str\$(appstate%) end if end if end if end sub</pre>						
See Also	Window Manipulation (Chapter 7)						

AppHide Statement

Description	Hides the specified top-level window.
Syntax	<code>AppHide [WindowName\$]</code>
Comments	<p>Nothing happens if the window is already hidden.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	<pre>sub main() 'example of AppHide and AppShow appname\$ = AppFind\$("Notepad") AppHide appname\$ MsgBox appname\$+" is now hidden." AppShow appname\$ MsgBox appname\$+" is no longer hidden." end sub</pre>
See Also	Window Manipulation (Chapter 7)

AppList Statement

Description	Fills the specified string array with the names of all the active applications.
Syntax	<code>AppList ArrayofAppNames\$()</code>
Comments	<p>Before calling this function, you must declare the array to contain the names of the application. Use the <code>Dim</code> command.</p> <p>After calling this function, use the <code>lbound()</code> and <code>ubound()</code> functions to determine the new size of the array.</p>
Example	<pre>sub main() 'example of AppList dim appnames(1) as string AppList appnames</pre>

```

        for i% = lbound(appnames) to ubound(appnames)
            MsgBox appnames(i%)
        next i%
    end sub

```

See Also Window Manipulation (Chapter 7)

AppMaximize Statement

Description Maximizes the specified top-level window.

Syntax AppMaximize [WindowName\$]

Comments Nothing happens if the window is already maximized or is hidden.

The WindowName\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If WindowName\$ parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example

```

sub main()
    'example of AppMaximize, AppRestore, and
    'AppMinimize

    appname$ = AppFind$("Notepad")
    AppMaximize appname$
    msgbox appname$+" is maximized."
    AppRestore appname$
    msgbox appname$+" is restored."
    AppMinimize appname$
    msgbox appname$+" is minimized."
end sub

```

See Also Window Manipulation (Chapter 7)

AppMinimize Statement

Description Minimizes the specified top-level window.

Syntax	<code>AppMinimize [WindowName\$]</code>
Comments	<p>Nothing happens if the window is already minimized or is hidden.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	<pre>sub main() 'example of AppMaximize, AppRestore, and 'AppMinimize appname\$ = AppFind\$("Notepad") AppMaximize appname\$ msgbox appname\$+" is maximized." AppRestore appname\$ msgbox appname\$+" is restored." AppMinimize appname\$ msgbox appname\$+" is minimized." end sub</pre>
See Also	Window Manipulation (Chapter 7)

AppMove Statement

Description	Sets the position of the specified top-level window to a given <code>x, y</code> pixel location.
Syntax	<code>AppMove x%, y%[, WindowName\$]</code>
Comments	<p>This statement has no effect if the top-level window is maximized.</p> <p>It is valid to specify an <code>x, y</code> location that is not visible.</p> <p>The <code>WindowName\$</code> parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If <code>WindowName\$</code> parameter is missing, the window with the focus is used (i.e., the active window).</p> <p>A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).</p>
Example	<pre>sub main() 'example of AppMove</pre>


```

dim ax, ay as integer

' get current position of Notepad to save
appname$ = AppFind$("Notepad")
AppGetPosition ax, ay, 0, 0, appname$

' move Notepad around a little
for i% = 0 to 200
    AppMove i%, i%, appname$
next i%

AppMove ax, ay, appname$
end sub

```

See Also Window Manipulation (Chapter 7)

AppRestore Statement

Description Restores the specified top-level window.

Syntax AppRestore [WindowName\$]

Comments This statement has an effect only if the specified window is either maximized or minimized. AppRestore will do nothing if the specified window is hidden.

The WindowName\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If WindowName\$ parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example

```

sub main()
    'example of AppMaximize, AppRestore, and
    'AppMinimize

    appname$ = AppFind$("Notepad")
    AppMaximize appname$
    msgbox appname$+" is maximized."
    AppRestore appname$
    msgbox appname$+" is restored."
    AppMinimize appname$
    msgbox appname$+" is minimized."
end sub

```

See Also Window Manipulation (Chapter 7)

AppSetState Statement

Description Sets the state of the specified top-level window.

Syntax `AppSetState [WindowName$]`

Comments This statement sets the state of the specified top-level window to one of the following values:

<code>WS_MAXIMIZED</code>	maximize the window
<code>WS_MINIMIZED</code>	minimize the window
<code>WS_RESTORED</code>	restore the window

The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If `WindowName$` parameter is missing, the window with the focus is used (i.e., the active window).

Example

```
sub main()
    'example of AppSetState

    appname$ = AppFind$("Notepad")
    AppSetState WS_MAXIMIZED, appname$
    msgbox "Notepad is maximized"
    AppSetState WS_RESTORED, appname$
    msgbox "Notepad is restored"
    AppSetState WS_MINIMIZED, appname$
    msgbox "Notepad is minimized"
end sub
```

See Also Window Manipulation (Chapter 7)

AppShow Statement

Description Shows the specified top-level window.

Syntax `AppShow [WindowName$]`

Comments Nothing happens if the window is already displayed.

The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If `WindowName$` parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example

```
sub main()
    'example of AppHide and AppShow

    appname$ = AppFind$("Notepad")
    AppHide appname$
    MsgBox appname$+" is now hidden."
    AppShow appname$
    MsgBox appname$+" is no longer hidden."
end sub
```

See Also

Window Manipulation (Chapter 7)

AppSize Statement

Description

Sets the width and height of the specified top-level window. It lets you size sizable and non-sizable windows.

Syntax

```
AppSize width%,height%[,WindowName$]
```

Comments

This statement works only if the specified application is restored (i.e., not minimized or maximized).

The `WindowName$` parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.

If `WindowName$` parameter is missing, the window with the focus is used (i.e., the active window).

A runtime error results if the window being activated is not enabled (which is the case if that application currently is displaying a modal dialog box).

Example

```
sub main()
    'example of AppSize
    dim ax, ay, aw, ah as integer

    appname$ = AppFind$("Notepad")
    AppSetState WS_RESTORED, appname$
    AppGetPosition ax, ay, aw, ah, appname$
```

```

        for i% = 10 to 200
            AppSize i%, i%, appname$
        next i%
        AppSize aw, ah, appname$
        AppMove ax, ay, appname$
        AppSetState WS_MINIMIZED, appname$
    end sub

```

See Also Window Manipulation (Chapter 7)

AppType Function

Description	Returns an integer representing the type of application owning the specified window.
Syntax	AppType[(WindowName\$)]
Returns	<div>TYPE_DOS DOS executable</div> <div>TYPE_WINDOWS Windows executable</div>
Comments	<p>The WindowName\$ parameter is the title of a top-level window. The complete text of the window title is required. The title is not case-sensitive.</p> <p>If WindowName\$ parameter is missing, the window with the focus is used (i.e., the active window).</p>
Example	<pre> sub main() 'example of AppType msgbox "After you press OK, you have 10 seconds to activate an app to ↵ check." sleep 10000 'wait for 10 seconds appname\$ = AppGetActive\$() at% = AppType(appname\$) select case at% case TYPE_DOS msgbox "DOS Application" case TYPE_WINDOWS msgbox "WINDOWS Application" end select end sub </pre>

See Also Window Manipulation (Chapter 7)

ArrayDims Function

Description	Returns an integer representing the number of dimensions in the specified array.
Syntax	<code>ArrayDims(arrayvariable)</code>
Comments	If the function indicates zero dimensions, the array was declared as an empty array (e.g. <code>dim x\$()</code>).
Example 1	<pre>sub main() 'example of ArrayDims dim apps(1) as string dim oapps(1,1) as string 'ArrayDims shows how many dimensions an array 'was declared with msgbox "apps(1) has "+str\$(arraydims(apps))+" dimension(s)."</pre>
Example 2	<pre>sub main() dim f\$() files f\$, "C:*.BAT" if dims(f\$) = 0 then exit sub end sub</pre> <p>'allocate empty array 'fill the array 'exit if no 'elements</p>
See Also	Arrays (Chapter 7)

ArraySort Statement

Description	Sorts a single-dimensioned array.
Syntax 1	<code>ArraySort s\$()</code>
Syntax 2	<code>ArraySort a%()</code>
Syntax 3	<code>ArraySort a&()</code>
Syntax 4	<code>ArraySort a!()</code>
Syntax 5	<code>ArraySort a#()</code>
Comments	<p>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 routine sorts lower numbers to the lowest array index locations.</p> <p>A runtime error results if an array with more than one dimension is specified.</p>
Example 1	<pre>sub main() 'example of ArraySort</pre>

```

        dim apps(1) as string

        'get list of running applications
        AppList apps
        'sort it
        ArraySort apps
        'display them
        for i% = lbound(apps) to ubound(apps)
            msgbox apps(i%)
        next i%
    end sub

```

Example 2

```

sub main()
    dim a$(100)
    ArraySort a$
    result = SelectBox("Title", "Prompt:", a$)
end sub

```

See Also

Arrays (Chapter 7)

Asc Function

Description Returns the numeric ASCII code for the first character of the specified string.

Syntax `asc(text$)`

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

Example

```

sub main()
    'example of ASC()
    dim acode as integer
    dim astr as string

    astr = "This is a string."
    'the ASC function returns the ASCII code for
    'the first character of the given string
    acode = asc(astr)
    msgbox "The first character of the string (" + astr + ") has an ASCII code ←
        of " + str$(acode)
end sub

```

See Also

Conversions (Chapter 7)

Strings (Chapter 7)

AskBox\$ Function

Description	Displays a dialog box that asks the user to enter a string in an edit box, and returns the string the user enters.
Syntax	<code>AskBox\$(prompt\$ [,default\$])</code>
Returns	Returns a string that the user typed, or returns an empty string indicating that the user canceled the dialog box.
Comments	<p>The dialog box is sized to the appropriate width depending on the width of <code>prompt</code>.</p> <p>When the dialog box is displayed, the edit box has the focus.</p> <p>If <code>default</code> is specified, then this is used as the initial contents of the edit field.</p> <p>The dialog box has OK and Cancel buttons.</p> <p>The dialog box uses the 8 point Helvetica font.</p> <p>The maximum number of characters that can be typed into the edit box is 255.</p>
Example	<pre>sub main() 'example of AskBox\$ userinput\$ = AskBox\$("Key in something below:","This is the default ← value.") msgbox "You entered -> "+userinput\$ end sub</pre>
See Also	Dialog Display (Chapter 7)

AskPassword\$ Function

Description	Displays a dialog box that asks the user to enter his or her password in an edit box, and returns the string the user enters.
Syntax	<code>AskPassword\$(prompt\$)</code>
Returns	Returns the string that the user typed, or returns an empty string indicating that the user canceled the dialog box.
Comments	<p>Unlike the <code>AskBox</code> command, the user sees asterisks in place of the characters that are actually typed. This allows the input of passwords.</p> <p>When the dialog box is displayed, the edit box has the focus.</p> <p>The dialog box is sized to the appropriate width depending on the width of <code>prompt\$</code>.</p>

The dialog box has OK and Cancel buttons.

The dialog box uses the 8 point Helvetica font.

The maximum number of characters that can be typed into the edit box is 255.

Example

```
sub main()
    'example of AskPassword$

    mypassword$ = AskPassword$("Type in a fake password:")
    msgbox "The password you typed was -> "+mypassword$
end sub
```

See Also

Dialog Display (Chapter 7)

Atn Function

Description Returns a double-precision number representing the arctangent of a given number.

Syntax `atn(number#)`

Comments To calculate the value of PI, use:

$$4 * \text{ATN}(1)$$
Example

```
sub main()
    'example of ATN (arctangent)
    dim mypi as double

    mypi = 4 * atn(1)
    msgbox str$(mypi)
end sub
```

See Also

Math Statements and Functions (Chapter 7)

ATTR_ARCHIVE

Description Constant.

Value 32

Comments Bit position of a file attribute indicating that a file hasn't been backed up. This value is used by `GetAttr`, `SetAttr`, and `FileList`.

ATTR_DIRECTORY

Description	Constant.
Value	16
Comments	Bit position of a file attribute indicating that a file is a directory entry. This value is used by <code>GetAttr</code> , <code>SetAttr</code> , and <code>FileList</code> .

ATTR_HIDDEN

Description	Constant.
Value	2
Comments	Bit position of a file attribute indicating that a file is hidden. This value is used by <code>GetAttr</code> , <code>SetAttr</code> , and <code>FileList</code> .

ATTR_NONE

Description	Constant.
Value	64
Comments	Bit position of a file attribute indicating that a file has no other attributes set. This value is used by <code>GetAttr</code> , <code>SetAttr</code> , and <code>FileList</code> .

ATTR_NORMAL

Description	Constant.
Value	0
Comments	Bit position of a file attribute indicating that a file has no other attributes set. This value is used by <code>GetAttr</code> , <code>SetAttr</code> , and <code>FileList</code> .

ATTR_READONLY

Description	Constant.
Value	1
Comments	Bit position of a file attribute indicating that a file is read-only. This value is used by GetAttr, SetAttr, and FileList.

ATTR_SYSTEM

Description	Constant.
Value	4
Comments	Bit position of a file attribute indicating that a file is a system file. This value is used by GetAttr, SetAttr, and FileList.

ATTR_VOLUME

Description	Constant.
Value	8
Comments	Bit position of a file attribute indicating that a file is the volume label. This value is used by GetAttr, SetAttr, and FileList.

Beep Statement

Description	Makes a single system beep.
Syntax	beep
Example	<pre>sub main() 'example of BEEP beep end sub</pre>

Begin Dialog...End Dialog Statement

Description	The <code>Begin Dialog...End Dialog</code> block defines a dialog box template.
Syntax	<pre>begin dialog DialogName\$,x%,y%,width%,height% end dialog</pre>
Comments	<p>The <code>x</code>, <code>y</code> parameters are the dialog coordinates of the upper left hand corner of the dialog box relative to the parent window.</p> <p>The <code>width</code>, <code>height</code> parameters specify the width and height dimensions of the dialog box in dialog units.</p> <p>The <code>DialogName</code> parameter specifies the name used to dimension a variable of this type (i.e., a variable that refers to this dialog box template). Once a dialog template has been defined, a variable can be dimensioned using this name:</p> <pre>dim MyDlg as DialogName</pre> <p>An error is generated if the dialog box template is empty.</p> <p>A dialog template must have at least one <code>PushButton</code>, <code>OKButton</code>, or <code>CancelButton</code>. Otherwise, there will be no way to close the dialog box.</p> <p>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. All dialogs created by DCL use an 8 point Helvetica font.</p>
Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

ButtonEnabled Function

Description	Determines whether the specified button within the current window is enabled.
Syntax 1	<code>ButtonEnabled (ButtonName\$)</code>
Syntax 2	<code>ButtonEnabled (ButtonID%)</code>
Returns	Returns the integer <code>TRUE</code> if the specified button within the current window is enabled, otherwise this function returns <code>FALSE</code> .

Comments	<p>The button can be specified either by its name (ButtonName\$) or using its ID (ButtonID%). ButtonName\$ is the text on the button's label.</p> <p>When a button is enabled, it can be pressed using the <code>SelectButton</code> statement.</p>
Example	<pre> sub main() 'Example of ButtonEnabled appn\$ = AppFind\$("Notepad") if appn\$ <> "" then AppActivate appn\$ Menu "File.Page Setup" if ButtonEnabled("OK") then msgbox "OK button is enabled." else msgbox "OK button is not enabled." end if end if end sub </pre>
See Also	Dialog Manipulation (Chapter 7)

ButtonExists Function

Description	Determines whether the specified button exists within the current window.
Syntax 1	<code>ButtonExists (ButtonName\$)</code>
Syntax 2	<code>ButtonExists (ButtonID%)</code>
Returns	Returns the integer TRUE if the specified button exists within the current window, otherwise this function returns FALSE.
Comments	The button can be specified either by its name (ButtonName\$) or using its id (ButtonID%). ButtonName\$ is the text on the button's label.
Example	<pre> sub main() 'Example of ButtonExists appn\$ = AppFind\$("Notepad") if appn\$ <> "" then AppActivate appn\$ Menu "File.Page Setup" if ButtonExists("Cancel") then msgbox "Cancel button exists." else msgbox "Cancel button does not exist." end if end if end sub </pre>

```
        end if  
    end if  
end sub
```

See Also Dialog Manipulation (Chapter 7)

Call Statement

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

Syntax `call subroutine_name [(arguments)]`

Comments Using this statement is equivalent to:
`subroutine_name [arguments]`

Use of the `call` statement is never required.

Example

```
sub doubleit (a as integer)
    msgbox str$(a * 2)
end sub

sub main()
    'Example of the Call statement

    'the following statements do the same thing
    Call doubleit (3)
    doubleit(3)
end sub
```

See Also Procedure Statements (Chapter 7)

CancelButton Statement

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

Syntax `CancelButton x%,y%,width%,height%`

Comments This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x,y,width,height` parameters are specified in dialog coordinates. The `x,y` position is relative to the upper left corner of the dialog box.

Example Dialog Examples

See Also Dialog Creation (Chapter 7)

CDbl Function

Description	Returns the double-precision equivalent of the passed numeric expression.
Syntax	<code>cdbl (number#)</code>
Comments	This function has the same effect as assigning the numeric expression to a double-precision variable.
Example	<pre>sub main() 'example of CDbl dim adouble as double dim asingle as single dim ainteger as integer dim along as long asingle = pi adouble = pi ainteger = 30000 along = 88000 adouble = cdbl(ainteger) msgbox str\$(adouble) adouble = cdbl(along) msgbox str\$(adouble) msgbox str\$(asingle) adouble = pi msgbox str\$(adouble) 'Now convert the single to double. You'll see a 'change in the value due to conversion of a 'smaller precision to double-precision. adouble = cdbl(asingle) msgbox str\$(adouble) end sub</pre>
See Also	Conversions (Chapter 7)

ChDir Statement

Description	Changes the current directory of the specified drive.
Syntax	<code>chdir newdir\$</code>
Comments	This statement will not change to the specified drive.

If you do not include a drive in the `newdir$` parameter, the current drive is assumed.

This statement behaves the same as the DOS "cd" command.

Example

```
sub main()
    'example of ChDir

    msgbox CurDir$("C")
    ChDir "C:\"
    msgbox CurDir$("C")
end sub
```

See Also

File Input and Output (Chapter 7)

ChDrive Statement

Description

Changes the default drive.

Syntax

`chdrive DriveLetter$`

Comments

Only the first character of `DriveLetter$` is used. You can use the same string for the `ChDrive` and `ChDir` commands.

`DriveLetter$` is case insensitive.

If `DriveLetter$` is empty, then the current drive is not changed.

Example

```
sub main()
    'example of ChDrive

    msgbox CurDir$()
    chdrive "D"
    msgbox CurDir$()
end sub
```

See Also

File Input and Output (Chapter 7)

CheckBox Statement

Description

Defines a checkbox within a dialog box template.

Syntax

`CheckBox x%,y%,width%,height%,title$, .Field`

Comments

Checkboxes can be either on or off, depending on the value of `.Field`.

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x`, `y`, `width`, `height` parameters are specified in dialog coordinates. The `x`, `y` position is relative to the upper left corner of the dialog box. The `width` and `height` parameters specify the dimensions of the checkbox and label.

On entry to the `Dialog` statement, the `.Field` variable is used to set the initial state of the checkbox. On exit from the `Dialog` statement, the `.Field` variable is used to determine the final state of the checkbox. If the value is 0, the checkbox is unchecked; 1 indicates that the checkbox is checked.

The `title$` parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for Font.

Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

CheckboxEnabled Function

Description	Determines whether the specified checkbox in the current window is enabled.
Syntax 1	<code>CheckboxEnabled (CheckboxName\$)</code>
Syntax 2	<code>CheckboxEnabled (CheckboxID%)</code>
Returns	Returns the integer TRUE if the specified checkbox within the current window is enabled, otherwise this function returns FALSE.
Comments	<p>The checkbox can be specified either by its name (<code>CheckboxName\$</code>) or using its id (<code>CheckboxID%</code>). <code>CheckboxName\$</code> is the text of the checkbox label.</p> <p>When a checkbox is enabled, its state can be set using the <code>SetCheckbox</code> statement.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

CheckboxExists Function

Description	Determines whether the specified checkbox exists within the current window.
Syntax 1	<code>CheckboxExists (CheckboxName\$)</code>

Syntax 2	<code>CheckboxExists (CheckboxID%)</code>
Returns	Returns the integer TRUE if the specified checkbox exists within the current window, otherwise this function returns FALSE.
Comments	The checkbox can be specified either by its name (<code>CheckboxName\$</code>) or using its ID (<code>CheckboxID%</code>). <code>CheckboxName\$</code> is the text of the checkbox label.
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

Chr\$ Function

Description	Returns the character for the specified ASCII code.
Syntax	<code>chr\$ (AsciiCode%)</code>
Comments	<p><code>AsciiCode</code> must be an integer between 0 and 255.</p> <p>The <code>Chr\$()</code> function can be used within constant declarations, as in the following example:</p> <pre>const crlf\$ = chr\$(13) + chr\$(10)</pre>
Example	<pre>sub main() 'example of Chr\$() 'List the letters of the alphabet startcode = asc("A") alphstr\$ = "" msgbox "The alphabet starts at code "+str\$(startcode) for i% = startcode to startcode + 25 alphstr\$ = alphstr\$ + chr\$(i%) next i% msgbox alphstr\$ end sub</pre>
See Also	<p>Conversions (Chapter 7)</p> <p>Strings (Chapter 7)</p>

CInt Function

Description	Returns the integer portion of the given expression. In cases of fractions, this function rounds to the nearest integer.
--------------------	--

Syntax	<code>cint(number#)</code>
Comments	<p>The passed numeric expression must be within the following range:</p> <pre>-32768 <= number <= 32767</pre> <p>A runtime error results if the passed expression is not within the above range.</p> <p>This function has the same effect as assigning a numeric expression to a variable of type integer.</p>
Example	<pre>sub main() 'Example of CINT dim a as integer a = cint(pi) 'convert PI to an integer msgbox str\$(a) end sub</pre>
See Also	<p>Conversions (Chapter 7)</p> <p>Fix Function</p> <p>Int Function</p>

Clipboard\$ Statement and Function

Description	<p>The <code>Clipboard\$</code> statement puts a string in the clipboard.</p> <p>The <code>Clipboard\$</code> function returns the text in the clipboard.</p>
Statement Syntax	<code>clipboard\$ NewContent\$</code>
Comments	This statement puts <code>NewContent\$</code> into the clipboard.
Function Syntax	<code>clipboard\$()</code>
Returns	Text contained in the clipboard.
Comments	If the clipboard doesn't contain text, or the clipboard is empty, then an empty string is returned.
Example	<pre>sub main() 'Example of Clipboard\$(), Clipboard\$, and 'ClipboardClear 'WARNING: after you run this, your clipboard 'contents will be gone msgbox Clipboard\$()</pre>

```

ClipboardClear
msgbox Clipboard$()
Clipboard$ "I was here!"
msgbox Clipboard$()
ClipboardClear
end sub

```

See Also Clipboard Manipulation (Chapter 7)

ClipboardClear Statement

Description Clears (removes the contents of) the clipboard.

Syntax ClipboardClear

Example

```

sub main()
'Example of Clipboard$(), Clipboard$, and
'ClipboardClear
'WARNING: after you run this, your clipboard
'contents will be gone

msgbox Clipboard$()
ClipboardClear
msgbox Clipboard$()
Clipboard$ "I was here!"
msgbox Clipboard$()
ClipboardClear
end sub

```

See Also Clipboard Manipulation (Chapter 7)

CLng Function

Description Returns a long integer representing the result of the given numeric expression.

Syntax CLng (number#)

Comments The passed numeric expression must be within the following range:

-2147483648 <= number <= 2147483647

A runtime error results if the passed expression is not within the above range.

This function has the same effect as assigning a numeric expression to a long variable.

Example

```

sub main()

```

```
'Example of CLNG
dim a as long

a = clng(pi)    'convert PI to a long integer
msgbox str$(a)
end sub
```

See Also Conversions (Chapter 7)

Close Statement

Description	Closes open file(s).
Syntax	<code>Close [[#] filename% [, [#] filename%]]</code>
Comments	<p>The file number is assigned by the <code>Open</code> command.</p> <p>If no arguments are specified, this statement closes all files. Otherwise, this statement closes each specified file.</p>
Example	Input/Output Example
See Also	File Input and Output (Chapter 7)

Combobox Statement

Description	Defines a combobox that appears within a dialog box template.
Syntax	<code>ComboBox x%,y%,width%,height%,items\$(),.Field</code>
Comments	<p>The <code>items\$</code> array must be a single-dimension array of strings. The elements of this array are placed into the combobox when the dialog box is created. The <code>.Field</code> parameter defines the name used to extract which string occupies the combobox when the dialog box ends. On exit from the <code>Dialog</code> statement, the <code>.Field</code> contains an index to the item that is highlighted in the combobox.</p> <p>This statement can only appear within a dialog box template definition (<code>BEGIN DIALOG...END DIALOG</code>).</p> <p>The <code>x,y,width,height</code> parameters are specified in dialog coordinates. The <code>x,y</code> position is relative to the upper left corner of the dialog box. The <code>width</code> and <code>height</code> parameters define the dimensions of the drop-down list box.</p>
Example	Dialog Examples

See Also Dialog Creation (Chapter 7)

ComboboxEnabled Function

Description	Determines whether the specified combobox is enabled in the current window or dialog box.
Syntax 1	<code>ComboboxEnabled (name\$)</code>
Syntax 2	<code>ComboboxEnabled (id%)</code>
Comment	<p>The combobox can be specified either by <code>name\$</code> or <code>id%</code>. The <code>name\$</code> parameter specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>This function returns the integer <code>TRUE</code> if the given combobox is enabled within the current window or dialog box, <code>FALSE</code> otherwise.</p> <p>A runtime error is generated if the specified combobox does not exist.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

ComboboxExists Function

Description	Determines whether the specified combobox exists in the current window or dialog box.
Syntax 1	<code>ComboboxExists (name\$)</code>
Syntax 2	<code>ComboboxExists (id%)</code>
Comments	<p>The combobox can be specified either by <code>name\$</code> or <code>id%</code>. The <code>name\$</code> parameter specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>This function returns the integer <code>TRUE</code> if the given combobox is enabled within the current window or dialog box, <code>FALSE</code> otherwise.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

Command\$ Statement

Description	Returns a string representing the arguments from the command line used to start the application.
Syntax	<code>Command\$</code>
Comments	When running scripts from the DCL editor, you enter command-line arguments through the Arguments command on the Run menu of the DCL editor.

Const Statement

Description	Declares a constant for use within the current script.
Syntax	<code>const name = expression [,name = expression]...</code>
Comments	<p>The <code>name</code> is only valid within the current DCL script.</p> <p>The <code>expression</code> must be assembled from literals, or other constants. Calls to functions are not allowed.</p>
See Also	Variables and Constants (Chapter 7)

Cos Function

Description	Returns a double-precision number representing the cosine of a given angle.
Syntax	<code>cos (angle#)</code>
Comments	The <code>angle#</code> parameter is given in radians.
See Also	Math Statements and Functions (Chapter 7)

CSng Function

Description	Returns a single-precision number representing the result of the given numeric expression.
Syntax	<code>CSng (number#)</code>
Comments	This function has the same effect as assigning a numeric expression to a single-precision variable.
Example	<pre>sub main() 'Example of CSNG() dim adouble as double dim asingle as single adouble =pi msgbox str\$(adouble) asingle = csng(adouble) msgbox str\$(asingle) end sub</pre>
See Also	Conversions (Chapter 7)

CStr Function

Description	Returns a string representing the result of the given expression.
Syntax	<code>CStr (number#)</code>
Example	<pre>sub main() 'Example of CSTR dim adouble as double dim astring as string</pre>


```

    adouble = pi
    astring = cstr(adouble)
    msgbox astring
end sub

```

See Also Conversions (Chapter 7)

CurDir\$ Function

Description Gets the current directory.

Syntax `curdir$[(drive$)]`

Returns Returns the current directory on the specified drive. If no drive is specified, the current directory on the current drive is returned.

Comments A runtime error results if `drive$` is invalid.

Example

```

sub main()
    'Example of CurDir$()
    dim dstr as string

    dstr = CurDir$()
    msgbox dstr
    dstr = CurDir$("F")    'for a particular drive
    msgbox dstr
end sub

```

See Also File Input and Output (Chapter 7)

Date\$ Statement and Function

Description The `date$` assignment statement sets the system date. The `date$` function returns the system date.

Assignment Syntax `date$ = newdate$`

Comments Sets the system date to the specified date. The format for `newdate$` is any of the following:

```

MM-DD-YYYY
MM-DD-YY
MM/DD/YYYY

```

Example 1

```

MM/DD/YY
sub main()
    'Example of the Date$ command (not function)
    dim currentdate as string
    dim newdate as string

    currentdate = Date$
    newdate = "1-1-1980"    'This is the earliest
                           'date that can be
                           'assigned

    Date$ = newdate
    msgbox Date$
    Date$ = currentdate
    msgbox Date$
end sub

```

Example 2 date\$ = "7-28-1992"

Function Syntax Date\$ [()]

Returns The date\$ function returns the current system date as a 10 character string.

Comments The format for the returned date is MM-DD-YYYY.

Example

```

sub main()
    'Example of Date$()
    dim dstr as string

    dstr = Date$
    msgbox dstr
    dstr = Date$()
    msgbox dstr
end sub

```

See Also Date and Time Functions (Chapter 7)

DateSerial Function

Description	Returns a double-precision number representing the specified date. The number is returned in days where Dec 30, 1899 is 0.
Syntax	<code>DateSerial (year%,month%,day%)</code>
Example	<pre> sub main() 'example of DateSerial dim dserial as double dserial = DateSerial(1899,12,30) 'Earliest date should be 0 msgbox str\$(dserial) dserial = DateSerial(1999,12,30) '100 years later (leap years included) msgbox str\$(dserial) end sub </pre>
See Also	<p>Conversions (Chapter 7)</p> <p>Date and Time Functions (Chapter 7)</p>

DateValue Function

Description	Returns a double-precision number representing the date contained in the specified string argument.
Syntax	<code>DateValue (date_string\$)</code>
Comments	<p>This function interprets the passed <code>date_string\$</code> parameter looking for a valid date specification. Date specifications vary depending on the international settings contained in the INTL section of the WIN.INI file.</p> <p>The <code>date_string\$</code> parameter can contain valid date items separated by date separators such as slash (/), minus (-), or comma (.). The date items must follow the ordering determined by the current date format settings in use by Windows.</p> <p>Date strings can contain an optional time specification, but this is not used in the formation of the returned value.</p> <p>Months can appear in their abbreviated formats, such as Jan, Feb, or Mar.</p>
Example	<pre> sub main() 'Example of DateValue dim dval as double </pre>

```

        dval = DateValue(Date$) 'value of current date
        msgbox str$(dval)
    end sub

```

See Also Conversions (Chapter 7)
 Date and Time Functions (Chapter 7)

Day Function

Description Returns an integer representing the day of the month for the date encoded in the specified `serial` parameter. The value returned is between 1 and 31 inclusive.

Syntax `Day(serial#)`

Comments You can obtain the value for the `serial#` parameter by using the `DateSerial` or `DateValue` command.

Example

```

sub main()
    'Example of Day() function (day of month)
    'will show the current date's day of month.
    msgbox str$(Day(DateValue(Date$)))
end sub

```

See Also Date and Time Functions (Chapter 7)

DCLHomeDir\$ Function

Description Returns the directory containing DCL.

Syntax `DCLHomeDir$()`

Comments This function is used to locate files that are part of the DCL system itself.

Example

```

sub main()
    'DCLHomeDir$ example

    a$ = DCLHomeDir$
    msgbox a$
end sub

```

See Also DCL Environment Information (Chapter 7)

DCLOS\$ Function

Description Returns a number indicating the host operating environment.

Syntax DCLOS\$ ()

Returns 0 Windows
1 DOS

Example

```
sub main()  
    'Example of DCLOS() function  
  
    a% = DCLOS()  
    select case a%  
        case 0  
            msgbox "Windows"  
        case 1  
            msgbox "DOS"  
    end select  
end sub
```

See Also DCL Environment Information (Chapter 7)

DCLVersion\$ Function

Description Returns the version of DCL.

Syntax DCLVersion\$ ()

Comments This function returns the major and minor version numbers in the format `major.minor`, as in "1.1".

Example

```
sub main()  
    'Example of DCLVersion$()  
  
    a$ = DCLVersion$()  
    msgbox a$  
end sub
```

See Also DCL Environment Information (Chapter 7)

DDEExecute Statement

Description	Sends an execute message to another application.
Syntax	<code>DDEExecute channel%,command\$</code>
Comments	<p>The <code>channel</code> must first be initiated using <code>DDEInitiate</code>. An error will result if <code>channel</code> is invalid.</p> <p>If the receiving application does not execute the instructions, a runtime error will be generated.</p> <p>The format of <code>command\$</code> depends on the receiving application.</p>
Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDEInitiate Function

Description	Initializes a DDE link to another application.
Syntax	<code>DDEInitiate (app\$,topic\$)</code>
Returns	Returns a unique number used to subsequently refer to the open DDE channel.
Comments	<p>The function returns 0 if the link cannot be established. This will occur under the following circumstances:</p> <ul style="list-style-type: none">• The specified application is not running• The topic was invalid for that application• Insufficient memory or system resources to establish DDE link
<ul style="list-style-type: none">• Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDEPoke Statement

Description	Sets the value of a data item in the receiving application associated with an open DDE link.
Syntax	<code>DDEPoke channel%,dataItem\$,value\$</code>
Comments	<p>The <code>channel</code> must first be initiated using <code>DDEInitiate</code>. An error will result if <code>channel</code> is invalid.</p> <p>The format for <code>dataItem\$</code> and <code>value\$</code> depends on the receiving application.</p>
Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDERequest Function

Description	Gets a data item from a receiving application.
Syntax	<code>DDERequest\$(channel%,dataItem\$)</code>
Returns	Returns a string representing the value of the given data item in the receiving application associated with the open DDE channel.
Comments	<p>The <code>channel</code> must first be initiated using <code>DDEInitiate</code>. An error will result if <code>channel</code> is invalid.</p> <p>The formats for <code>dataItem\$</code> and the returned value depend on the receiving application.</p>
Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDETerminate Statement

Description	Closes the specified DDE channel.
Syntax	<code>DDETerminate channel%</code>

Comments	The <code>channel</code> must first be initiated using <code>DDEInitiate</code> . An error will result if <code>channel</code> is invalid. All open DDE channels are automatically terminated when the script ends.
Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDETerminateAll Statement

Description	Closes all currently open DDE channels.
Syntax	<code>DDETerminateAll</code>
Comments	If you do not issue this statement, all open DDE channels are automatically terminated when the script ends.
Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDETimeout Statement

Description	Sets the number of milliseconds that must elapse before a DDE command times out.
Syntax	<code>DDETimeout milliseconds&</code>
Comments	The default is 10000 (10 seconds). The timeout should be set before issuing other DDE commands.
Example	DDE Example
See Also	Dynamic Data Exchange (Chapter 7)

DDE Example

```

sub main()
    ' This script illustrates the use of the DDE commands.
    ' The current shell is used to demonstrate this.
    ' NetTools Applications Manager is assumed to be the
    ' shell.
    '
    ' Demonstrated in this script are:
    ' DDEInitiate, DDETimeout, DDERequest, DDETerminate,
    ' DDEExecute, and DDETerminateAll

    dim channel as integer
    dim winshell as string
    dim topic as string
    dim reqstr as string
    dim grouplist() as string
    dim i as integer
    dim nl as integer
    dim selitem as integer
    dim groupname as string

    winshell = "AppMan"
    topic = "AppMan"

    ' initiate a DDE conversation with the shell
    channel = DDEInitiate(winshell,topic)
    DDETimeout 5000'set time out to 5 seconds
                                '(default is 10000 milliseconds)
    if channel = 0 then
        msgbox "Unable to initiate DDE link with "+winshell+". Script will end now."
        end
    end if

    ' Get a list of the groups in the shell
    reqstr = DDERequest$(channel,"Groups")
    ' Now the group names have to be placed into an array
    ' for use in a Selection Box
    nl = LineCount(reqstr) - 1
    redim grouplist(1 to nl)
    for i = 1 to nl
        grouplist(i) = Line$(reqstr,i)
    next i

    ' Now that we have the groups in a usable list,
    ' allow the user to make selections to

```

```

'    view the items in the group
viewgroups:
    selitem = SelectBox("Groups",winshell+" Groups:",grouplist)
    if selitem = 0 then goto shutdown

'    Get the group name, and enumerate the item names
    groupname = grouplist(selitem)
    reqstr = DDERequest$(channel,groupname)
'    Display the items in a message box
    msgbox reqstr
    goto viewgroups

shutdown:
    DDETerminate channel
    msgbox "Now we'll create a group, and then delete it."
    channel = DDEInitiate(winshell,topic)
    DDEExecute channel,"[CreateGroup(TestGrp)]"
    DDEExecute channel,"[ShowGroup(TestGrp,1)]"
    msgbox "Look at your shell now and see the new group before we delete it."
    DDEExecute channel,"[DeleteGroup(TestGrp)]"
    DDETerminateAll
end
end sub

```

Declare Statement

Description	Declares a subroutine or function within another DLL.
Syntax 1	<pre>Declare sub <i>name</i> [lib <i>libname\$</i> [alias <i>aliasname\$</i>]] ← [[<i>argumentlist</i>]]</pre>
Syntax 2	<pre>Declare function <i>name</i> [lib <i>libname</i> [alias <i>aliasname\$</i>]] ← [[<i>argumentlist</i>]] [as <i>type</i>]</pre>
Comments	<p>This statement must precede any call to an external DLL routine.</p> <p>The <i>name</i> parameter is any DCL valid global name. When declaring functions, a type-declaration character can be included to indicate the return type.</p> <p>The <i>libname\$</i> needs to be specified along with an optional <i>aliasname\$</i>. The <i>libname\$</i> parameter specifies the DLL that contains the external routine. All of the Windows API routines are contained in DLLs, such as "user", "kernel", "gdi", and so on. The file extension ".EXE" is implied if another extension is not given. If the <i>libname\$</i> parameter is not the real name of the external routine as it appears within the external DLL, then an alias name must be given providing this name.</p>

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 an internal routine or if the external routine name contains invalid characters.

The optional `argumentlist` specifies the arguments received by the routine. The argument list must match exactly with that of the referenced routine. Otherwise unpredictable results may occur. By default, DCL passes arguments by reference. Many DLL routines require a value rather than a reference. The `byval` keyword does this. For example, the following C routine:

```
int MessageBeep(int);
```

would be declared as follows:

```
declare sub MessageBeep lib "user" (byval n as integer)
```

The following shows how a C routine that requires a pointer to an integer would be declared (notice the missing `byval` keyword on the third parameter):

```
int SystemParametersInfo(int,int,int far *,int);
declare function SystemParametersInfo lib "user" (byval ←
    action as integer, byval uParam as integer, pi ←
    as integer, byval updateINI as integer)
```

Strings are always passed to DLL routines by reference - the `byval` keyword in this case is unnecessary. If a DLL routine modifies a passed string variable, there must be sufficient space within the string to hold the returned characters. This can be accomplished using the `space$()` function:

```
declare sub GetWindowsDirectory lib "user" (dir$,len%)
:
:
    dim s as string
    s = space$(128)
    GetWindowsDirectory s,128
```

For function declarations, the return type can be specified using a type declaration character (i.e., \$, %, or &), or by specifying the `[as type]` clause. The valid types are integer, long, and string.

The libraries containing the routines are loaded when the routine is called for the first time (i.e., not when the script is loaded). This allows a script to reference external DLLs that potentially do not exist.

The `declare` statement must appear before the function is used.

The `declare` statement is only valid during the life of that script.

Example

```
declare sub MessageBeep lib "user" (byval n as integer)
declare function GetDebugState lib "user" alias "GetSystemDebugState" () ←
    as long

sub main()
    'Example of Declare statement
    'All declared external functions and procedures
    'exist outside of any DCL function or
    'subroutines.
    dim debugstate as long

    MessageBeep(-1)      'standard system beep
    debugstate = GetDebugState()
    msgbox str$(debugstate)
end sub
```

See Also

Procedure Statements (Chapter 7)

DEftype Statement

Description This statement controls automatic type declaration of variables.

Syntax

```
DEFInt letterrange
DEFLng letterrange
DEFStr letterrange
DEFsng letterrange
DEFdbl letterrange
```

Comments Normally, if a variable is encountered that hasn't yet been declared with the `dim` statement, or does not appear with an explicit type declaration character, the variable is declared implicitly as an integer (`DEFInt 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 declaration—using a type declaration character or using the `dim` statement.

This statement only affects compiling of scripts.

Example

```

DEFInt i - k, x - z
DEFStr s - v
DEFDbl a - c

sub main()
    'example of DEFtype statement
    'This statement is used to define certain
    'letters of the alphabetic character set to
    'be used as a certain type of variable.

    i = 1
    msgbox str$(i)
    j = 2.5
    msgbox str$(j)
    s = "test"
    msgbox s
    a = 3
    msgbox str$(a)
    b = 5.6
    msgbox str$(b)
end sub

```

See Also Variables and Constants (Chapter 7)

DesktopCascade Statement

Description Cascades all non-minimized top-level windows.

Syntax DesktopCascade

Example

```

sub main()
    'Example of DesktopCascade

    DesktopCascade
end sub

```

See Also Desktop Modifications (Chapter 7)

DesktopSetColors Statement

Description Changes the system colors to one of the predefined color sets in the CONTROL.INI file.

Syntax DesktopSetColor ControlPanelItemName\$

Example

```
sub main()
    'example of DesktopSetColor

    msgbox "Click to change to WingTips"
    DesktopSetColor "WingTips"
    msgbox "Click to change to Arizona"
    DesktopSetColor "Arizona"
end sub
```

See Also Desktop Modifications (Chapter 7)

DesktopSetWallpaper Statement

Description Changes the Windows wallpaper.

Syntax DesktopSetWallPaper filename\$,tile%

Comments This statement changes the Windows wallpaper to the bitmap specified by filename\$. The wallpaper will be tiled if tile is TRUE, otherwise the bitmap will be centered on the desktop.

To remove the wallpaper, set the filename\$ parameter to "":

```
DesktopSetWallPaper "",true
```

This statement makes permanent changes to the wallpaper by writing the new wallpaper information to the WIN.INI file.

Example

```
sub main()
    'Example of DesktopSetWallpaper

    msgbox "Click to set wallpaper to CASTLE tiled."
    DesktopSetWallpaper "castle.bmp",true
    msgbox "Click to set wallpaper to CASTLE centered."
    DesktopSetWallpaper "castle.bmp",false
    msgbox "Click to clear wallpaper."
    DesktopSetWallpaper "",true
end sub
```

See Also Desktop Modifications (Chapter 7)

DesktopTile Statement

Description Tiles all non-minimized top-level windows.

Syntax DesktopTile

Example

```
sub main()
    'Example of DesktopTile

    DesktopTile
end sub
```

See Also Desktop Modifications (Chapter 7)

Dialog Examples

OK, Cancel, and Push Buttons

```
sub main()
    'Example of a simple Dialog box
    Begin Dialog UserDialog 16,32,233,105, "Title"
        OKButton 176,7,43,14
        CancelButton 176,30,44,14
        Text 6,34,158,8, "This is a sample text field."
        PushButton 176,52,44,14, "Other #1"
        PushButton 177,76,43,14, "Other #2"
    End Dialog

    dim adialog as UserDialog

    returncode = Dialog(adialog)
    ' returncode contains the value of the button selected
    ' OK = -1
    ' Cancel = 0
    ' Other buttons are numbered 1 to N where N is the number
    ' of other buttons on the dialog
    select case returncode
        case -1
            msgbox "OK pressed."
        case 0
            msgbox "Cancel pressed."
        case else
            msgbox "Other button #" + str$(returncode) + " pressed."
    end select
```

```
end sub
```

A Comprehensive Dialog Example

```
sub main()
    'Example showing the entire process of defining, displaying,
    'and interpreting the input from a dialog box.

    'Initialization
    '-----
    'ListBox$ and ComboBox are single-dimensional arrays to hold the
    'contents of the list and combo box in the dialog box.
    Dim ListBox1$() as string
    Dim ComboBox1$() as string

    Dim stf$ as string          ' static text variable
    stf$ = "123456789012345678901234567890"

    'In this example, the Text field uses the str$
    'variable to get its field name. This can
    'be done for any of the dialog control types except:
    'OKButton, CancelButton, TextBox
    '
    'Note that the str$ variable must have a
    'value assigned BEFORE you can define the dialog box.

    'Define dialog box
    '-----
    Begin Dialog UserDialog 16,32,304,168, "Sample Dialog Box"
        OKButton 251,9,44,14
        CancelButton 252,30,44,14
        PushButton 252,51,44,14, "Pushbutton1"
        PushButton 252,73,44,14, "PushButton2"
        GroupBox 13,9,84,59, "Sample Group Box"
        OptionGroup .OptionGroup1
            OptionButton 21,24,65,14, "Option 1"
            OptionButton 21,44,66,14, "Option 2"
        CheckBox 15,78,79,14, "Sample Checkbox", .CheckBox1
        Text 14,105,79,8, stf$
        TextBox 16,120,81,12, .TextBox1
        ListBox 114,14,120,48, ListBox1$, .ListBox1
        ComboBox 113,68,120,84, ComboBox1$, .ComboBox1
    End Dialog

    'Prepare to display dialog box.
    '-----
    'Declare the dialog type using Dim ... as UserDialog command
    Dim aSampleDialog as UserDialog
```



```

'Load the list box and combo box arrays with app window names
AppList ComboBox1$
AppList ListBox1$

'Load the text box with an initial value
aSampleDialog.TextBox1 = "123456789012345678901234567890"
'Note that you reference a dialog box field as follows:
'    <DialogBoxName>.<FieldName>

'Display the Dialog
'-----
a% = Dialog(aSampleDialog)    'Returns integer indicating button chosen

'Interpret user input
'-----
'This examples builds a string describing the contents of the
'dialog box when the user finished making changes.
crlf$ = chr$(13)+chr$(10)
dlgstr$ = "Button pushed = "
'Determine button pushed.
select case a%
    case -1
        dlgstr$ = dlgstr$ + "OK"
    case 0
        dlgstr$ = dlgstr$ + "Cancel"
    case 1
        dlgstr$ = dlgstr$ + "PushButton1"
    case 2
        dlgstr$ = dlgstr$ + "PushButton2"
end select
dlgstr$ = dlgstr$ + crlf$
'Determine new value of each dialog box component
dlgstr$=dlgstr$ + "Option = " + str$(aSampleDialog.OptionGroup1)+crlf$
dlgstr$=dlgstr$ + "Checkbox = " + str$(aSampleDialog.CheckBox1)+crlf$
dlgstr$=dlgstr$ + "TextBox = " + aSampleDialog.TextBox1 + crlf$
dlgstr$=dlgstr$ + "ListBox = " + ListBox1$(aSampleDialog.ListBox1)+crlf$
dlgstr$=dlgstr$ + "ComboBox = " + aSampleDialog.ComboBox1
'Display the dlgstr$ string in a message box.
msgbox dlgstr$
end sub

```

Dialog Statement and Function

Description The `Dialog` statement displays a dialog box. The `Dialog` function displays a dialog box and returns an integer representing the button that was pushed.

Statement Syntax	Dialog UserDlg as dialog
Comments	<p>Displays the dialog box specified by the dialog template UserDlg.</p> <p>If the user selects Cancel, a runtime error is generated. This error can be trapped using ON ERROR.</p> <p>The Dialog statement returns only after the user presses OK, Cancel, or another push button.</p>
Function Syntax	Dialog(UserDlg as dialog)
Returns	<p>-1 OK button was pushed.</p> <p>0 Cancel button was pushed.</p> <p>>0 A push button was selected. The returned number represents which button was pushed based on its order in the dialog template. 1 represents the first button, 2 represents the second, etc.</p>
Comments	This function displays the dialog box associated with the template UserDlg, and returns which button was pushed. Unlike the statement form, this function will not generate a runtime error when Cancel is selected.
Example 1	<pre> sub main() 'Example of using the DIALOG function and command 'Here is our dialog box specification. Begin Dialog UserDialog 16,32,148,72, "Sample" OKButton 54,11,41,14 CancelButton 55,40,41,14 End Dialog 'Create a dialog box type variable dim mydialog as UserDialog 'First the function method retval% = Dialog(mydialog) select case retval% case -1 msgbox "OK pressed." case 0 msgbox "Cancel pressed." end select 'Now the command method 'First create an error handler in case "Cancel" 'is pressed. on error goto cancelpressed Dialog mydialog </pre>

```
        msgbox "OK pressed."
        goto endprog
cancelpressed:
        msgbox "Cancel pressed."
endprog:
        on error goto 0
end sub
```

Example 2 A Comprehensive Dialog Example (see Dialog Examples)

See Also Dialog Creation (Chapter 7)

Dim Statement

Description	<p>Declares a list of variables and their corresponding types and sizes.</p> <p>A special form of the DIM statement is used to store a dialog definition in memory.</p>
Syntax	<pre>dim name [(<subscripts>)] [as type] [,name [(<subscripts>)] [as type]]...</pre>
Comments	<p>If a type declaration character is used (such as %, &, or \$), the optional [as type] expression is not allowed.</p> <p>Up to 60 array dimensions are allowed.</p> <p>The total size of an array (not counting space for string) is limited to 42K.</p> <p>Dynamic arrays are declared by not specifying any bounds:</p> <pre>dim a()</pre> <p>Any DIM statements declared within a subroutine or function are local to that subroutine or function.</p> <p>If a variable is seen that has not been explicitly declared with DIM, it is implicitly declared using the type specifier character (% , \$, or &). If this character is missing, then integer is assumed.</p>
Example	<pre>sub main() 'Examples of the DIM statement 'These two statements declare integers dim a as integer dim b% a = 1 b% = 1 'These two statements declare strings</pre>

```

dim s as string
dim t$

s = "test"
t$ = "test"

'Array declarations
'When an array is declared, its elements are
'numbered 0 through N-1 where N is the size of
'the array. Using the OPTION BASE statement,
'you can begin numbering at 1.

dim j(5) as integer      'array of 5 integer
                        'values
for x% = 0 to 4
    j(x%) = x%
    msgbox str$(j(x%))
next x%

dim k(5) as string      'array of 5 strings
for x% = 0 to 4
    k(x%) = chr$(65+x%)
    msgbox k(x%)
next x%

dim l(3,3) as string    'two-dimensional array
                        'containing a total of
                        '9 strings, 3 by 3

for x% = 0 to 2
    for y% = 0 to 2
        l(x%,y%) = str$(x%)+str$(y%)
        msgbox l(x%,y%)
    next y%
next x%

'Using explicit array subscripts
dim p(6 to 10) as integer
for x% = 6 to 10
    p(x%) = x%
    msgbox str$(p(x%))
next x%
end sub

```

**Use with
Dialogs**

A special form of the DIM statement stores the preceding dialog definition (BeginDialog...EndDialog) in memory.

The syntax of this statement is:

```
dim dialog_name as UserDialog
```

Where `dialog_name` is a variable you define and `UserDialog` is a DCL reserved word.
For an example of this command, see [Dialog Examples](#).

- See Also**
- [Arrays \(Chapter 7\)](#)
 - [Dialog Creation \(Chapter 7\)](#)
 - [ReDim Statement](#)
 - [Variables and Constants \(Chapter 7\)](#)

Dir\$ Function

- Description** Searches a disk directory.
- Syntax** `Dir$([filespec$])`
- Returns** If `filespec$` is specified, then this function returns the first file matching that `filespec`. If `filespec$` is not specified, then this function returns the next file matching the initial `filespec`.
- Comments**
- The `filespec$` argument can include wildcards, such as `*` and `?`.
 - An error is generated if `dir$` is called without first calling it with a valid `filespec`.
 - If there is no matching `filespec`, then an empty string is returned.
 - If no path is specified on `filespec$`, then the current directory is used.
 - This function does not find hidden files and directories.

Example

```
sub main()  
    'Example of Dir$() function  
  
    a$ = dir$("c:\*.bat")    'initialize file search  
                            '(filespec parameter required)  
  
    do  
        msgbox a$          'display file name  
        a$ = dir$()        'next file name  
                            '(no parameter specified)  
    loop while a$ <> ""  
end sub
```

- See Also**
- [File Input and Output \(Chapter 7\)](#)

DirExists Function

Description Determines whether the specified `path$` exists and is a directory.

Syntax `DirExists (path$)`

Comments The function returns the integer TRUE or FALSE.

The path can be a partial path, such as "windows".

Example

```
sub main()  
    'Example of DirExists  
  
    a$ = "C:\DOS"  
    if DirExists(a$) then  
        msgbox a$+" exists"  
    else  
        msgbox a$+" does not exist"  
    end if  
end sub
```

See Also File Input and Output (Chapter 7)

DiskDrives Statement

Description This statement grabs all of the valid drive letters and packs them into the specified array.

Syntax `DiskDrives list$()`

Comments The array is resized to hold the exact number of valid drives.

The `list$` parameter must be a single dimension array of strings.

Use the functions `lbound()` and `ubound()` to determine the size of the resultant array.

Example

```
sub main()  
    'Example of DiskDrives command  
    dim ddrives() as string  
  
    DiskDrives ddrives  
    for i% = lbound(ddrives) to ubound(ddrives)  
        alldrives$ = alldrives$ + ddrives(i%)  
    next i%  
    msgbox alldrives$  
end sub
```

See Also File Input and Output (Chapter 7)

DiskFree Function

Description	Returns a long integer representing the free space (in bytes) available on the specified drive.
Syntax	<code>DiskFree([drive\$])</code>
Comments	<p>If <code>drive\$</code> is empty or not specified, then the current drive is assumed.</p> <p>Only the first character of the <code>drive\$</code> string is used.</p>
Example	<pre>sub main() 'Example of DiskFree function dim ddrives() as string dim freespace as long DiskDrives ddrives for i% = lbound(ddrives) to ubound(ddrives) if ddrives(i%) <> "A" and ddrives(i%) <> "B" then freespace = DiskFree(ddrives(i%)) msgbox "Free space on drive "+ddrives(i%)+ ": is "+str\$(freespace) end if next i% end sub</pre>
See Also	File Input and Output (Chapter 7)

Do...Loop Statement

Description	This statement repeats a block of DCL statements while a condition is TRUE or until a condition is TRUE.
Syntax 1	<pre>do {while until} <i>condition</i> <i>statements</i> loop</pre>
Comments	<p>Using <code>do while</code> causes the statements in the loop to be executed while the specified condition is true.</p> <p>Using <code>do until</code> causes the statements in the loop to be executed until the specified</p>

condition is true.

The `condition` parameter specifies any Boolean expression.

Syntax 2

```
do
    statements
loop {while | until} condition
```

Comments

Syntax 2 has the same effect as Syntax 1.

Syntax 3

```
do
    statements
loop
```

Comments

If the {`while` | `until`} conditional clause is not specified, then the loop repeats the `statements` forever (or until an `exit do` statement is encountered).

Example

```
sub main()
    'Examples of DO-LOOP statements
    'All of the following loops perform the same task.
    'The difference is primarily in WHEN the loop exit
    'condition is checked.

    a% = 1
    do while a% = 1
        a% = msgbox("DO-WHILE-LOOP Click cancel to go to next loop.",1)
    loop

a% = 1
do until a% <> 1
    a% = msgbox("DO-UNTIL-LOOP Click cancel to go to next loop.",1)
loop

a%=1
do
    a% = msgbox("DO-LOOP-WHILE Click cancel to go to next loop.",1)
loop while a% = 1

    a% = 1
do
    a% = msgbox("DO-LOOP-UNTIL Click cancel to go to next loop.",1)
loop until a% <> 1

a% = 1
do
    a% = msgbox("DO-LOOP Click cancel to go to next loop.",1)
    if a% <> 1 then
```



```

        exit do
    end if
loop
end sub

```

See Also Flow Control (Chapter 7)

DoEvents Statement

Description	The <code>DoEvents</code> statement and function yield control to other applications.
Statement Syntax	<code>DoEvents</code>
Function Syntax	<code>DoEvents[()]</code>
Returns	The function returns the value 0.
Comments	When running in <code>exclusive</code> mode, the only way other applications can multitask is for the script to call this statement/function.
Example	<pre> sub main() 'Examples of DoEvents and DoEvents() DoEvents x% = DoEvents() end sub </pre>
See Also	Flow Control (Chapter 7)

DoKeys Statement

Description	Uses the Windows journaling mechanism to play keystrokes into the Windows environment.
Syntax	<code>DoKeys KeyString\$</code>
Comments	<p>The format for <code>KeyString\$</code> is the same as that used for <code>QueKeys</code>.</p> <p>This statement will not affect the current event queue.</p>
Example	<pre> sub main() 'Example of DoKeys dim alttab as string </pre>

```

    alttab = "%{TAB}"
    msgbox "Press OK to do first Alt-Tab"
    DoKeys alttab
    msgbox "Press OK to Alt-Tab back to original application"
    DoKeys alttab
end sub

```

See Also Keyboard Manipulation (Chapter 7)

EditEnabled Function

Description	Determines if an edit box is enabled within the current window or dialog box.
Syntax 1	<code>EditEnabled(name\$)</code>
Syntax 2	<code>EditEnabled(ID%)</code>
Returns	Returns the integer TRUE if the given edit box is enabled within the active window or dialog box, FALSE otherwise.
Comments	<p>If enabled, the edit box can be given the focus using the <code>ActivateControl</code> statement.</p> <p>The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the edit control in the window list or dialog template. Alternatively, the <code>ID%</code> of the edit box can be specified.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

EditExists Function

Description	Determines if an edit box exists within the current window or dialog box.
Syntax 1	<code>EditExists(name\$)</code>
Syntax 2	<code>EditExists(id%)</code>
Returns	Returns the integer TRUE if the given edit box exists within the active window or dialog box, FALSE otherwise.
Comments	<p>If there is no active window, FALSE will be returned.</p> <p>The <code>name\$</code> parameter specifies the text that appears within the static control that</p>

immediately precedes the edit control in the window list (or dialog template). Alternatively, the ID of the edit box can be specified.

Example Dialog Examples

See Also Dialog Manipulation (Chapter 7)

EnableStopScript Statement

Description Controls whether a script may be halted by the user.

Syntax `EnableStopScript condition%`

Comments The `condition%` parameter can be one of the following constants:

- `ESS_ENABLE` (enable script breaking). This is the default.
- `ESS_DISABLE` (disable script breaking).
- `ESS_ENABLE_INTERACTIVE` (allows the script to be broken only when executing in interactive [debug] mode).

• **Example**

```
sub main()
    'Example of EnableStopScript

    EnableStopScript ESS_ENABLE
    'control-break enabled
    EnableStopScript ESS_DISABLE
    'control-break disabled
    EnableStopScript ESS_INTERACTIVE
    'control-break only in script editor
end sub
```

See Also Flow Control (Chapter 7)

End Statement

Description Terminates execution of the current script.

Syntax `end`

Comments This statement can appear multiple times in a script, so that you can conditionally end the script. It can also appear in functions and subroutines.

All open files are close. All open DDE channels are closed.

Example

```
sub main()  
    'Example of END  
  
    msgbox "You'll see this one."  
    end  
    msgbox "You won't see this one."  
end sub
```

See Also

Flow Control (Chapter 7)

ENV_BOTH

Description Constant meaning both DOS and Windows; used by the `SetEnv` command.

ENV_DOS

Description Constant meaning DOS environment; used by the `GetEnv`, `SetEnv`, `RestoreEnv`, and `SaveEnv` commands.

ENV_WINDOWS

Description Constant meaning Windows environment; used by the `GetEnv`, `SetEnv`, `RestoreEnv`, and `SaveEnv` commands.

Environ\$ Function

Description Supported only under DOS. Returns the value of the specified environment variable.

Note: For greater convenience, we recommend using the `GetEnv` function instead.

Syntax 1 `environ$(variable$)`

Syntax 2 `environ$(VariableNumber%)`

Comments If `variable$` is specified, then this function looks for that variable in the environment. If the variable name cannot be found, then an empty 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 variables, an empty string is returned. Otherwise, the entire entry from the environment is returned in the format:

`variable=value`

Example

```
sub main()
    'Example of environ$() function
```

```

a$ = environ$("path")
msgbox a$
end sub

```

See Also Environment Statements and Functions (Chapter 7)

EOF Function

Description	Determines whether end of file has been reached.
Syntax	<code>eof(filenum%)</code>
Returns	Returns the integer TRUE if the end of file has been reached for the given file, otherwise FALSE.
Comments	The <code>filenum</code> parameter is a number that is used by DCL to refer to the open file—the number passed to the <code>open</code> statement.
Example	Input/Output Example
See Also	File Input and Output (Chapter 7)

Erl Function

Description	Not used.
Syntax	<code>erl[()]</code>
Returns	Returns the integer 0 (DCL does not support line numbers).
See Also	Error Trapping (Chapter 7)

Err Statement and Function

Description	The <code>err</code> function returns an integer representing the runtime that caused the current error trap. The <code>err</code> statement sets the value returned by the <code>err</code> function to a specific value.
Statement Syntax	<code>err = value%</code>
Function Syntax	<code>err[()]</code>

Comments	<p>The <code>err</code> function can only be used while within an error trap.</p> <p>When a function or statement ends, the value returned by <code>err</code> is reset to 0.</p>
Example	<pre> sub main() 'Example of Err(), Error, and Error\$() dim i as integer i=1 nexterror: on error goto errortrap select case i case 1 error 100 case 2 error 101 case 3 error 102 end select end end errortrap: msgbox "Error #" + str\$(err()) + "; " + Error\$(err()) i = i + 1 goto nexterror end sub </pre>
See Also	Error Trapping (Chapter 7)

Error Statement

Description	Simulates the occurrence of the specified runtime error.
Syntax	<code>error errornumber%</code>
Comments	<p>The <code>errornumber</code> parameter can be a built-in error number, or a user defined error number. The <code>err</code> function can be used within the error trap handler to determine the value of the error.</p>
Example	<pre> sub main() 'Example of Err(), Error, and Error\$() dim i as integer i=1 nexterror: on error goto errortrap select case i </pre>

```

        case 1
            error 100
        case 2
            error 101
        case 3
            error 102
    end select
end

errortrap:
    msgbox "Error #" + str$(err()) + "; " + Error$(err())
    i = i + 1
    goto nexterror
end sub

```

See Also Error Trapping (Chapter 7)

Error\$ Function

Description	Returns the text corresponding to the given error number or the most recent error.
Syntax	Error\$ [(errornumber%)]
Comments	<p>If <code>errornumber</code> is omitted, then the function returns the text corresponding to the most recent runtime error. If no runtime error has occurred, then an empty string is returned ("").</p> <p>If the <code>error</code> statement was used to generate a user-defined runtime error, this function will return an empty string ("").</p>
Example	<pre> sub main() 'Example of Err(), Error, and Error\$() dim i as integer i=1 nexterror: on error goto errortrap select case i case 1 error 100 case 2 error 101 case 3 error 102 end select end </pre>


```

errortrap:
    msgbox "Error #" + str$(err()) + "; " + Error$(err())
    i = i + 1
    goto nexterror
end sub

```

See Also Error Trapping (Chapter 7)

Exclusive Statement

Description Sets or unsets exclusive mode.

Syntax `exclusive NewState%`

Comments When set (`NewState%` is `TRUE`), then the script will not yield control to other applications - no other applications will execute concurrently. When a script is not running in exclusive mode (`NewState%` is `FALSE`), then other applications can multitask while the script is running.

By default, scripts do not run in exclusive mode.

If a script is running in exclusive mode and there is an infinite loop, you will be unable to abort the script (the computer will hang). Thus, caution must be taken to make sure that a script has no errors or infinite loops.

Scripts running exclusively run faster than scripts not in exclusive mode.

If a dialog box is encountered (`MsgBox`, `AskBox$`, `AnswerBox`, ...) while a script is running in exclusive mode, then that dialog box is system modal, meaning that a user can only interact with that dialog and not any other applications that may be running concurrently. All other dialog boxes, applications, and buttons will be "locked out".

Example

```

sub main()
    'Examples of the Exclusive statement

    Exclusive TRUE      'no multitasking for now
    Exclusive FALSE     'allow other programs to run
                        'again
end sub

```

See Also Flow Control (Chapter 7)

Exit Do Statement

Description	Exits a <code>do . . . loop</code> .
Syntax	<code>exit do</code>
Comments	This statement can only appear within a <code>do . . . loop</code> statement. It causes execution to continue with the next statement after the <code>loop</code> clause.
Example	Do...Loop Statement Example
See Also	Flow Control (Chapter 7)

Exit For Statement

Description	Exits a <code>for . . . next</code> loop.
Syntax	<code>exit for</code>
Comments	<p>This statement ends the <code>for . . . next</code> block in which it appears. Execution will continue on the line immediately after the <code>next</code> statement.</p> <p>This statement can only appear within a <code>for . . . next</code> block.</p>
Example	Exit Statement Examples
See Also	Flow Control (Chapter 7)

Exit Function Statement

Description	Exits the current function.
Syntax	<code>Exit Function</code>
Comments	<p>This statement ends execution of the function in which it appears. Execution will continue on the statement or function following the call to this function.</p> <p>This statement can only appear within a function.</p>
Example	Exit Statement Examples
See Also	Procedure Statements (Chapter 7)

Exit Sub Statement

Description	Exits the current subroutine.
Syntax	Exit Sub
Comments	<p>This statement ends the current subroutine. Execution is transferred to the statement following the call to the current subroutine.</p> <p>This statement can appear anywhere within a subroutine. It cannot appear within a function.</p>
Example	Exit Statement Examples
See Also	Procedure Statements (Chapter 7)

Exit Statement Examples

```
function testfunc (a as integer) as integer
    testfunc = 0
    if a = 2 then
        Exit Function
    end if
    msgbox "no premature exit from testfunc"
    testfunc = 100
end function
```

```
sub testsub (a as integer)
    if a = 4 then
        Exit Sub
    end if
    msgbox "no premature exit from testsub"
end sub
```

```
sub main()
    'examples of EXIT statements
    for i% = 1 to 10
        if i% >= 1 and i% <= 2 then
            b% = testfunc(i%)
            msgbox str$(b%)
        end if
        if i% >= 3 and i% <= 4 then
            testsub(i%)
        end if
        if i% = 7 then
            Exit For
        else
            msgbox "no exit yet "+str$(i%)
        end if
    next i%
end sub
```

```

        end if
    next i%
    msgbox "we just exited from the for loop"
end sub

```

Exp Function

Description Returns a double-precision number representing the value of e raised to the power of x .

Syntax `Exp (x#)`

Comments Range of x : $0 \leq x \leq 709.782712893$

A runtime error is generated if x is out of the above specified range.

Example

```

sub main()
    'Example of EXP function

    dim result as double

    result = exp(1)
    msgbox str$(result)
end sub

```

See Also Math Statements and Functions (Chapter 7)

FALSE

Description Constant.

Returns 0

Comments Used in conditionals and Boolean expressions

FileAttr Function

Description Returns an integer representing the file mode or file handle.

Syntax `FileAttr (filenumber%, attribute%)`

Returns Returns the file mode (if `attribute` is 1), or the operating system file handle (if `attribute` is 2).

Comments	<p>If <code>attribute</code> is 1, then one of the following values is returned:</p> <ul style="list-style-type: none"> 1 input 2 output 8 append <p>The <code>filenumber</code> parameter is a number that is used by DCL to refer to the open file—the number passed to the <code>open</code> statement.</p>
Example	Input/Output Example
See Also	File Input and Output (Chapter 7)

FileCopy Function

Description	Copies the specified file(s) to the given destination.
Syntax	<code>FileCopy(src\$, dest\$)</code>
Comments	<p>The function returns the integer TRUE if successful; otherwise it returns FALSE.</p> <p>Wildcards are permitted in the <code>src\$</code> string. They are the same as the DOS wildcards.</p> <p>If the <code>src\$</code> parameter specifies wildcards, all files matching the <code>src\$</code> parameter are copied to <code>dest\$</code> with the proper modification made to the destination file name so that it matches the source file according to the wildcards.</p>
Example	<pre>sub main() 'Example of FileCopy a% = FileCopy("c:*.bat", "c:\batch") if a% then msgbox "File copy successful." else msgbox "File copy failed." end if end sub</pre>
See Also	File Input and Output (Chapter 7)

FileDateTime Function

Description	Returns a double-precision number representing the date and time of the given file. The
--------------------	---

number is returned in days where Dec 20, 1899 is 0.

Syntax `FileDateTime (filename$)`

Comments This function retrieves the date and time of the file specified by `filename$`. 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

```
sub main()
    'Example of FileDateTime
    dim fdt as double
    dim sfdt as string

    fdt = FileDateTime("C:\AUTOEXEC.BAT")
    sfdt = str$(month(fdt))+"/"+str$(day(fdt))+ " ←
           "/" +str$(year(fdt))+" " +str$(hour(fdt))+ " ←
           ":"+str$(minute(fdt))+":"+str$(second(fdt))
    msgbox sfdt
end sub
```

See Also File Input and Output (Chapter 7)

FileDirs Statement

Description Fills an array with directory names from disk.

Syntax `FileDirs array$() [,dirspec$]`

Comments The `array$()` is any previously declared string array. The `FileDirs` function reallocates this array to exactly hold all of the directory names matching a given specification.

The `dirspec$` parameter specifies the file search mask, such as:

`T*.* C:*.*`

If the `dirspec$` parameter is not specified, then `*.*` is used, which files the array with all the sub-directory names within the current directory.

Example

```
sub main()
    'Example of FileDirs statement
    dim fdirs() as string

    'list all directories on drive C:
    FileDirs fdirs,"C:\*.*"

    for i% = lbound(fdirs) to ubound(fdirs)
        fdirlist$ = fdirlist + fdirs(i%) + "; "
    next i%
```

```

        msgbox fdirlist$
    end sub

```

See Also Arrays (Chapter 7)
 File Input and Output (Chapter 7)

FileExists Function

Description Determines if a given filename is valid.

Syntax FileExists(filename\$)

Returns Returns the integer TRUE if the filename\$ is a valid file, FALSE otherwise.

Example

```

sub main()
    'Example of FileExists() function

    if FileExists("C:\AUTOEXEC.BAT") then
        msgbox "Autoexec Exists."
    else
        msgbox "Autoexec Does Not Exists."
    end if
    if FileExists("D:\NADA.FIL") then
        msgbox "NADA.FIL Exists."
    else
        msgbox "NADA.FIL Does Not Exists."
    end if
end sub

```

See Also File Input and Output (Chapter 7)

FileLen Function

Description Returns a long integer representing the length of the specified file in bytes.

Syntax FileLen(filename\$)

Comments This function is used to retrieve the length of a file without first opening the file. A runtime error results if the file does not exist.

Example

```

sub main()
    'Example of the FileLen function
    dim flen as long

```

```
    flen = FileLen("C:\AUTOEXEC.BAT")
    msgbox "Autoexec.Bat is"&str$(flen)&" bytes long."
end sub
```

See Also File Input and Output (Chapter 7)
 LOF Function

FileList Statement

Description Fills an array with filenames from disk.

Syntax `FileList array$() [,filespec$ [,fileattr%]]`

Comments The `array$()` is any previously declared string array. The `files` function reallocates this array to exactly hold all of the files matching a given `filespec`.

 The `filespec$` parameter specifies the file search mask, such as:

*.EXE *.DOC t*.DO?

If the `filespec$` parameter is not specified, `*.*` is used.

The `fileattr` parameter is a number indicating what types of files you want included in the list. It can be any combination of the following:

Constant	Value	Description
ATTR_NORMAL	0	Read-only, archive, subdirectory, and files with no attributes
ATTR_READONLY	1	Read-only files
ATTR_HIDDEN	2	Hidden files
ATTR_SYSTEM	4	System files
ATTR_VOLUME	8	Volume label
ATTR_DIRECTORY	16	MS-DOS directories
ATTR_ARCHIVE	32	Files changed since last backup
ATTR_NONE	64	Files with no attributes

If the `fileattr` parameter is not specified, then the value 97 is used (`ATTR_READONLY` or `ATTR_ARCHIVE` or `ATTR_NONE` or `ATTR_DIRECTORY`). This value retrieves the same set of files normally returned by the DOS `dir` command.

Example

```
sub main()
    'Example of FileList statement
    dim ffiles() as string
    dim atrib as integer
```



```
atrib = ATTR_NORMAL
filelist ffiles,"C:\*.*",atrib
for i% = lbound(ffiles) to ubound(ffiles)
    flist$ = flist$ + ffiles(i%) + "; "
next i%
msgbox flist$
end sub
```

See Also Arrays (Chapter 7)
 File Input and Output (Chapter 7)

FileParse Statement

Description Takes a given filename and extracts a given portion of the filename from it.

Syntax FileParse\$(filename\$[,operation])

Comments The filename\$ parameter can specify an valid DOS filename (does not have to exist). For example:

```
..\TEST.DAT
C:\SHEETS\TEST.DAT
TEST.DAT
```

The optional operation parameter specifies which portion of the filename\$ to extract. It can be any of the following values.

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.

Example

```
sub main()
    'Examples of FileParse$()
    dim filespec as string

    filespec = "C:\DOS\COMMAND.COM"

    'full file specification
    a$ = FileParse$(filespec,0)
    msgbox "Full Filespec = " + a$
```

```

'drive
a$ = FileParse$(filespec,1)
msgbox "Drive = " + a$

'path
a$ = FileParse$(filespec,2)
msgbox "Path = " + a$

'name
a$ = FileParse$(filespec,3)
msgbox "Filename = " + a$

'root
a$ = FileParse$(filespec,4)
msgbox "File Rootname = " + a$

'extension
a$ = FileParse$(filespec,5)
msgbox "File Extension = " + a$
end sub

```

See Also File Input and Output (Chapter 7)

FileType Function

Description	Returns an integer representing the file type.
Syntax	FileType (filename\$)
Returns	Returns one of the following file type constants: <div> <div>TYPE_DOS</div> <div>TYPE_WINDOWS</div> <div>a DOS executable file</div> <div>a Windows executable file</div> </div>
Comments	This function is used to determine whether a file is a Windows executable or DOS executable. If one of the above values is not returned, then the file type is unknown.
Example	<pre> sub DisplayFileType (ftype as integer) select case ftype case TYPE_DOS msgbox "DOS" case TYPE_WINDOWS msgbox "WINDOWS" case else msgbox "Unknown Filetype" end select end sub </pre>

```

sub main()
    'example of FileType function

    ftype% = FileType("C:\DOS\COMMAND.COM")
    DisplayFileType(ftype)
    ftype% = FileType("C:\DOS\DOSSHELL.EXE")
    DisplayFileType(ftype)
    ftype% = FileType("D:\WINDOWS\notepad.exe")
    DisplayFileType(ftype)
end sub

```

See Also File Input and Output (Chapter 7)

FindFile\$ Function

Description Searches for file\$ and, if found, returns a full path to it. If the file is not found, the return value is a null-string.

Syntax fullpath\$=FindFile\$(file\$)

Comments The search follows normal Windows search order: current directory, Windows directory, system directory, and then the PATH environment variable.

Example

```

sub main()
    'Example of FindFile$

    fp$ = FindFile$("notepad.exe")
    if fp$ <> "" then
        msgbox "File found as "+fp$
    else
        msgbox "File not found."
    end if
end sub

```

See Also File Input and Output (Chapter 7)

Fix Function

Description Returns the integer part of number.

Syntax fix(number#)

Comments This function returns the integer part of the given value by removing the fractional part. The sign is preserved. No rounding occurs. For example:

```
fix(4.5)           'returns 4
fix(-4.5)          'returns -4
```

Example

```
sub main()
    'Example of the Fix() function
    dim adouble as double
    dim aint as integer

    adouble = pi
    msgbox str$(adouble)
    aint = fix(adouble)
    msgbox str$(aint)
end sub
```

See Also CInt Function
Int Function
Math Statements and Functions (Chapter 7)

For...Next Statement

Description Repeats a block of statement a specified number of times, incrementing a loop counter by a given increment each time through the loop.

Syntax

```
for counter = start to end [step increment]
    ...
    ...
next [counter]
```

Comments If increment is not specified, then 1 is assumed.

The first time through the loop, counter is equal to start. Each time through the loop, increment is added to counter by the amount specified in increment.

The for...next statement continues executing until:

- An exit for statement is encountered

OR

- When counter is greater than end.

If end > start then increment must be positive. If end < start, then increment

must be negative.

The `for...next` statements can be nested. In such a case, the `next [counter]` statement applies to the innermost `for...next`.

The `next [counter]` can be optimized for 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):

```
next i,j
```

Example

```
sub main()
'Examples of FOR-NEXT loops
dim x as integer      'used as iteration variable
                      'for FOR-NEXT

dim xstart as integer
dim xend as integer

'simple form
for x = 1 to 5
    msgbox str$(x)
next x

'step form
for x = 1 to 10 step 2
    msgbox str$(x)
next x

'backward form
for x = 10 to 1 step -2
    msgbox str$(x)
next x

'change of indexes
for x = 69 to 74
    msgbox chr$(x)
next x

'variable range
xstart = 69
xend = 74
for x = xstart to xend
    msgbox "VAR - "+chr$(x)
next x

'premature exit
for x = 1 to 10
    if x = 6 then
        exit for
    end if
```

174 Desktop Control Language

```
        msgbox "Exit - "+str$(x)
    next x
end sub
```

See Also Flow Control (Chapter 7)

FreeFile Function

Description	Returns the next available file number.
Syntax	<code>FreeFile[()]</code>
Comment	The returned number is suitable for use in the <code>open</code> statement.
Example	<pre> sub main() 'Example of FreeFile%() dim nextfile as integer nextfile = FreeFile() msgbox str\$(nextfile) end sub </pre>
See Also	File Input and Output (Chapter 7)

Function...End Function Statement

Description	Creates a user-defined function.
Syntax	<pre> Function name[(parameter [as <type>]...)] [as <type>] name = <expression> End Function </pre>
Comments	<p>The return value is determined by the statement:</p> <pre>name = <expression></pre> <p>The name of the function following DCL naming conventions. It can include type declaration characters: %, &, and \$.</p> <p>If no assignment is encountered before the function exits, then 0 will be returned for numeric functions, and an empty string will be returned for string functions.</p> <p>The type of the return value is determined by the <code>as <type></code> clause on the function statement itself. As an alternative, a type declaration character can be added to the function name:</p> <pre>Function Test() as string</pre>

```

    Test = "Hello World"
End Function

Function Test$()
    Test = "Hello World"
End Function

```

Parameters are passed to a function by reference, meaning that any modifications to a passed parameter changes that variable in the caller. To avoid this, simply enclose variable names in parenthesis, as in the following example function calls:

```
i = UserFunction(10,12,(j))
```

If a function is not to receive a parameter by reference, then the optional `byval` keyword can be used:

```

Function Test(byval FileName as string)
    as string
End Function

```

A function returns to the caller when either of the following statements is encountered:

```

End Function
Exit Function

```

The function definition must precede the statements that call the function.

The function cannot be inside a subroutine (including `sub (main)`) or inside another function.

Functions can be recursive.

See Also Procedure Statements (Chapter 7)

GetAttr Function

Description Returns an integer containing the attributes of the specified file.

Syntax `GetAttr(filename$)`

Returns The attribute value returned contains the sum of the following attributes.

Constant	Value	Description
ATTR_NORMAL	0	Read-only, archive, subdirectory, and files with no attributes
ATTR_READONLY	1	Read-only files
ATTR_HIDDEN	2	Hidden files
ATTR_SYSTEM	4	System files

ATTR_VOLUME	8	Volume label
ATTR_DIRECTORY	16	MS-DOS directories
ATTR_ARCHIVE	32	Files changed since last backup
ATTR_NONE	64	Files with no attributes

These attributes are the same as those used by DOS.

Example

```
sub main()
    'Example of GetAttr

    i% = GetAttr("C:\IO.SYS")
    msgbox str$(i%)
end sub
```

See Also

File Input and Output (Chapter 7)

GetCheckbox Function

Description

Returns an integer representing the state of the specified check box.

Syntax 1

GetCheckbox (name\$)

Syntax 2

GetCheckbox (id%)

Comments

This function is used to determine the state of a check box, given its name\$ (the text of its label) or id%. The return value will be one of the following:

- 0 check box has no check
- 1 check box contains a check
- 2 check box is grayed

Example

Dialog Examples

See Also

Dialog Manipulation (Chapter 7)

GetComboboxItem\$ Function

Description

Returns the text corresponding to an item number in a combobox.

Syntax 1

GetComboboxItem\$ (name\$, ItemNumber%)

Syntax 2

GetComboboxItem\$ (id%, ItemNumber\$)

Comments

The combobox must exist within the current window or dialog box, otherwise a runtime

error is generated.

You can use the `name$` or `id%` parameter to specify the combobox. The `name$` parameter specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).

The `ItemNumber%` parameter is the line number of the desired combobox item.

An empty string will be returned if the combobox does not contain textual items.

Example

Dialog Examples

See Also

Dialog Manipulation (Chapter 7)

GetComboBoxItemCount Function

Description

Returns an integer representing the number of items in the specified combobox.

Syntax 1

`GetComboBoxItemCount (name$)`

Syntax 2

`GetComboBoxItemCount (id%)`

Comments

You can use the `name$` or `id%` parameter to specify the combobox. The `name$` parameter specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).

A runtime error is generated if the specified combobox does not exist within the current window or dialog box.

Example

Dialog Examples

See Also

Dialog Manipulation (Chapter 7)

GetEditText\$ Function

Description

Returns the textual content of the specified edit box.

Syntax 1

`GetEditText$ (name$)`

Syntax 2

`GetEditText$ (id%)`

Comments

The name of an edit control is determined by scanning the window list (or dialog template) for a static control labeled `name$` that is immediately followed by an edit control. A runtime error is generated if an edit control with that name cannot be found within the active window.

For edit controls that do not have a preceding static control, the `id$` can be used to absolutely reference the control. The `id%` is assigned to the control by the program. It can be obtained through Windows diagnostic utilities, such as SPY.

Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

GetEnv Function

Description	Returns the value of the given environment variable for DOS or Windows.
Syntax	<code>GetEnv\$(var\$[, mode\$])</code>
Comments	<p>The <code>mode\$</code> parameter can be <code>ENV_DOS</code> or <code>ENV_WINDOWS</code>.</p> <p>If <code>mode\$</code> is <code>ENV_DOS</code>, the variable is returned from the DOS environment, otherwise it is returned from the Windows environment.</p> <p>If <code>mode</code> is unspecified, the default is <code>ENV_WINDOWS</code>.</p>
Example	<pre>sub main() 'Example of SetEnv and GetEnv\$() tmp\$ = GetEnv\$("TMP",ENV_WINDOWS) tv\$ = AskBox\$("New Value For TMP Environment Variable:") a% = SetEnv("TMP",tv\$,ENV_WINDOWS) msgbox "New value for TMP is "+ GetEnv\$("TMP",ENV_WINDOWS) 'restore old value a% = SetEnv("TMP",tmp\$,ENV_WINDOWS) end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

GetListboxItem\$ Function

Description	Returns the string corresponding to a list box item.
Syntax 1	<code>GetListboxItem\$(name\$,item%)</code>
Syntax 2	<code>GetListboxItem\$(id%,item%)</code>
Comments	This function retrieves the text of a given item in a listbox. The <code>item%</code> parameter is the item's position in the list, where 1 is the first item. The <code>item%</code> parameter must be between 1

and the number of items in the listbox.

The listbox can be specified using either its `id%` or the `name$` (label) of the static control that immediately precedes the listbox control in the window list (or dialog template).

A runtime error is generated if the specified listbox cannot be found within the active window.

Example Dialog Examples

See Also Dialog Manipulation (Chapter 7)

GetListboxItemCount Function

Description Returns an integer representing the number of items in the specified listbox.

Syntax 1 `GetListboxItemCount (name$)`

Syntax 2 `GetListboxItemCount (id%)`

Comments The listbox can be specified using either its `id%` or the `name$` (label) of the static control that immediately precedes the listbox control in the window list (or dialog template).

A runtime error is generated if the specified listbox cannot be found within the active window.

Example Dialog Examples

See Also Dialog Manipulation (Chapter 7)

GetOption Function

Description Determines whether a given option button is checked.

Syntax 1 `GetOption (name$)`

Syntax 2 `GetOption (id%)`

Returns Returns the integer TRUE if the option is set, FALSE otherwise.

Comments The option button must exist within the current window or dialog box.

The option button can be referenced given its `name$` (label) or its `id%`. A runtime error will be generated if the given option button does not exist.

Example Dialog Examples

See Also Dialog Manipulation (Chapter 7)

GetUserName Function

Description Use the `NetUserName` function instead.

GoSub Statement

Description Executes the specified subroutine.

Syntax `Gosub label`

Comments This statement causes execution to continue at the specified label. 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. The `gosub` statement outside the context of the current function or subroutine is not allowed.

Note: It is a sounder programming practice to write a named subroutine using a `Sub . . . End Sub` block.

Example

```
sub main()
    'Examples of GOSUB and RETURN

    for x% = 1 to 5
        Gosub mylabel
    next x%
end

mylabel:
    msgbox "Here we are!"
    return
end sub
```

See Also Flow Control (Chapter 7)

Goto Statement

Description	Transfers execution to the line containing the specified label.
Syntax	<code>Goto <label></code>
Comments	<p>The compiler will produce an error if <code>label</code> does not exist.</p> <p>The <code>label</code> must appear within the same subroutine or function as the <code>goto</code>.</p> <p>Labels must begin with a letter and end with a colon. Keywords cannot be used as labels. Labels are not case sensitive.</p>
Example	<pre>sub main() 'Example of GOTO for x% = 1 to 5 if x% = 3 then goto enditall next x% msgbox "Error" end enditall: msgbox "Ended properly" end end sub</pre>
See Also	Flow Control (Chapter 7)

GroupBox Statement

Description	Defines a groupbox within a dialog box template.
Syntax	<code>GroupBox x%,y%,width%,height%,title\$</code>
Comments	<p>A groupbox is simply a visual element used to enclose other controls within a dialog box.</p> <p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x%,y%,width%,height%</code> parameters are specified in dialog coordinates. The <code>x, y</code> position is relative to the upper left corner of the dialog box.</p>
Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

Hex\$ Function

Description	Returns a string containing the hexadecimal equivalent of the specified number.
Syntax	<code>hex\$(number&)</code>
Comments	<p>The returned string contains only the number of hexadecimal digits necessary to represent the number, up to a maximum of 8.</p> <p>The <code>number</code> parameter can be any type, but is rounded to the nearest whole number before converting to hexadecimal. If the passed number is an integer, then a maximum of 4 digits are returned; otherwise, up to 8 digits can be returned.</p>
Example	<pre>sub main() 'Example of Hex\$() function i% = 31 h\$ = hex\$(i%) msgbox h\$ end sub</pre>
See Also	Conversions (Chapter 7)

HLine Statement

Description	Scrolls the window with the focus left or right by the specified number of lines. This feature is useful when the contents are wider than the window.
Syntax	<code>HLine [lines%]</code>
Comments	If the <code>lines</code> parameter is omitted, then the window is scrolled right by 1 line.
Example	<pre>sub main() 'Examples of HLINE ViewPortOpen ViewPortClear Print "Here is some test data." HLine 50 sleep 2000 HLine -50 ViewPortClose end sub</pre>
See Also	Window Manipulation (Chapter 7)

Hour Function

Description	Returns an integer representing the hour of the day encoded in the specified <code>serial</code> parameter. The value returned is between 0 and 23 inclusive.
Syntax	<code>hour(serial#)</code>
Comments	You can obtain the value for the <code>serial#</code> parameter by using the <code>TimeSerial</code> or <code>TimeValue</code> command.
Example	<pre>sub main() 'Example of hour() function dim dt as double dt = Now msgbox str\$(hour(dt)) 'current hour end sub</pre>
See Also	Date and Time Functions (Chapter 7)

HPage Statement

Description	Scrolls the window with the focus left or right by the specified number of pages. This feature is useful when the contents are wider than the window.
Syntax	<code>HPage [pages%]</code>
Comments	If the <code>pages%</code> parameter is omitted, then the window is scrolled right by 1 page.
Example	<pre>sub main() 'Examples of HPage ViewPortOpen ViewPortClear Print "This is some test data" HPage 1 sleep 2000 HPage -1 ViewPortClose end sub</pre>
See Also	Window Manipulation (Chapter 7)

HScroll Statement

Description	Sets the thumb mark on the horizontal scroll bar attached to the current window.
Syntax	<code>HScroll percentage%</code>
Comments	The position is given as a percentage of the total range associated with that scroll bar. For example, if the <code>percentage%</code> parameter is 50, then the thumb is positioned in the middle of the scroll bar.
Example	<pre> sub main() 'Example of HSCROLL ViewPortOpen ViewPortClear Print "This is some test data for the viewport scroll test." sleep 2000 HScroll 50 '50 percent scroll sleep 2000 HScroll 0 'no scroll sleep 2000 ViewPortClose end sub </pre>
See Also	Window Manipulation (Chapter 7)

If...Then...Else Statement

Description	Conditionally executes a statement or group of statements.
Syntax 1	<pre> if <condition> then <statement> [else <statement>] </pre>
Syntax 2	<pre> if <condition> then [<statement>] [elseif <condition> then [<statement>]] [else [<statement>]] </pre>

```
end if
```

Comments In the single line version, the <statement> must be a single statement. Optionally, many statements can be separated using the colon (:).

Example

```
sub main()
    'Example IF-THEN-ELSE-END IF statement
    dim a%

    if a% = 1 then
        'do this stuff if a is 1
    elseif a% = 2 then
        'otherwise if a is 2 then do this stuff
    else
        'if a is neither 1 nor 2 then do this stuff
    end if
end sub
```

See Also Flow Control (Chapter 7)

Input # Statement

Description Reads comma-delimited data from a file into variables.

Syntax `Input [#]filename%,variable[,variable]...`

Comments This statement reads data from the file referenced by `filename%` into the given variables.

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.

All data items are separated by commas in the file. Leading spaces are ignored. Strings must be enclosed in quotes:

```
10,"Hello World",192,6
```

The `filename%` parameter is a number that is used by DCL to refer to the open file—the number passed to the `open` statement.

The `filename%` must reference a file opened in `input` mode.

See Also File Input and Output (Chapter 7)

Input\$ Function

Description	Returns a character string containing the first <code>numbytes%</code> characters read from the given file.
Syntax	<code>input\$(numbytes%, [#] filenumber%)</code>
Comments	<p>The <code>input\$</code> function reads all characters, including spaces and carriage returns.</p> <p>The <code>filenumber%</code> must reference a file opened in <code>input</code> mode.</p> <p>The <code>filenumber%</code> parameter is a number that is used by DCL to refer to the open file—the number passed to the <code>Open</code> statement.</p>
See Also	File Input and Output (Chapter 7)

Input/Output Example

```
sub main()
    'Example of file input/output functions
    'Open, Close, Eof, FileAttr, Line Input#, Lof

    'Because of the use of ViewPort... commands,
    'this script should not be run under DOS.

    dim fileno as integer
    dim flen as integer
    dim fileatr as integer
    dim osfile as integer
    dim inline as string
    dim linedesc as string
    dim linemult as integer

    ViewPortOpen "AUTOEXEC.BAT"
    ViewPortClear
    fileno = FreeFile()           'get next available file number
    Open "C:\AUTOEXEC.BAT" for input as fileno
    flen = Lof(fileno)           'get length of file
    Print "File Autoexec.Bat - Length"+str$(flen)
    fileatr = FileAttr(fileno,1)
    osfile = FileAttr(fileno,2)
    Print "File is opened for ";
    Select Case fileatr
        Case 1
            Print "Input";
        Case 2
```

```

        Print "Output";
    Case 8
        Print "Append";
End Select
Print " and has an Operating System Filehandle of"+str$(osfile)
Print string$(50,"-")
while not eof(fileno)
    Line Input #fileno, inline
    Print inline
wend
Close fileno
msgbox "Click OK when you're finished viewing the file."

ViewPortClear
Print "Creating a simple output file."
fileno = FreeFile()
Open "C:\JUNK.TXT" for output as fileno
for i% = 1 to 10
    Write #fileno,"Line"+str$(i%),i% * 10
next i%
Close fileno
Print "Reading created file."
Print string$(50,"-")
fileno = FreeFile()
Open "C:\JUNK.TXT" for input as fileno
while not eof(fileno)
    print "Seek Position"+str$(Seek(fileno))
    print "File Position"+str$(loc(fileno))    'print the file position
    Input #fileno,linedesc,linemult
    Print linedesc+", Value ="&str$(linemult)
wend
Close fileno
msgbox "Click OK when you're finished viewing the file."

ViewPortClear
Print "Display Autoexec.Bat again...this time in a different way."
fileno = FreeFile()
Open "C:\AUTOEXEC.BAT" for input as fileno
flen = lof(fileno)
aexec$ = Input$(flen,fileno)    'read the entire file at once
print aexec$
Close fileno
msgbox "Click OK when you're finished viewing the file."

ViewPortClear
Print "Now create Junk.Txt using Print# instead of Write#"
fileno = FreeFile()
Open "C:\JUNK.TXT" for output as fileno
for i% = 1 to 10

```

```

        Print #fileno,i%,"test","more"
    next i%
    Close fileno
    fileno = FreeFile()
    Open "C:\JUNK.TXT" for input as fileno
    stuff$ = Input$(lof(fileno),fileno)
    Close fileno
    print stuff$
    msgbox "Click OK when you're finished viewing the file."
    ViewPortClose
end sub

```

InputBox\$ Function

Description	Presents a dialog box displaying a prompt and returns the user's response.
Syntax	<code>InputBox\$(prompt\$ [,title\$ [,default\$ [,x%,y%]])</code>
Returns	Returns the text contained in the edit box when the user presses OK. If the user Cancels the dialog box, an empty string is returned.

Comments A default response can be specified in the `default$` parameter.

The `prompt$` parameter can contain multiple lines, each separated with a Carriage Return/Line Feed (`chr$(13) + chr$(10)`).

The `title$` parameter specifies the text that appears in the dialog box's caption. If the `title$` parameter is not specified, no title appears in the dialog's caption.

The `x` and `y` parameters are specified in twips (1/20th of a point or 1/1440 of an inch). This allows the dialog box to be positioned in a device independent manner. If the position is not specified, then the dialog box is positioned on or near the object containing the executing script.

Example 1

```

sub main()
    'Example of InputBox$() function

    a$ = InputBox$("Enter your description:", "Description","",100,200)
    msgbox a$
end sub

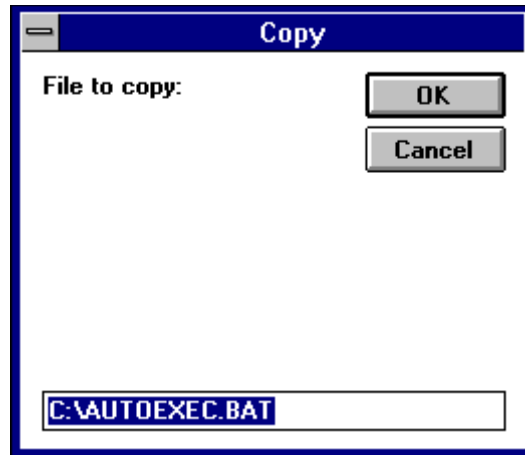
```

Example 2

```

s$ = InputBox$("File to copy:", "Copy", "C:\AUTOEXEC.BAT")

```



See Also Dialog Display (Chapter 7)

InStr Function

- Description** Searches a string for a substring.
- Syntax** `instr([start%,] search$,find$)`
- Returns** Returns an integer representing the character position of `find$` within `search$`.
- Comments** If the string is found, its character position within `search$` is returned, with 1 being the character position of the first character.
- If `start%` is specified, then the search starts at that character position within `search$`. The `start%` parameter must be between 1 and 65535. If not specified, the search starts at the beginning (`start% = 1`).
- If the string is not found, or `start%` is greater than the length of `search$`, or if `search$` is empty, then 0 is returned.

Example

```
sub main()
    'Example of InStr() function
    dim teststring as string

    teststring = "The quick brown fox jumps over the ←
        lazy dog."
    'search starting at position 1
    i% = InStr(1,teststring,"quick")
    if i% = 0 then
        msgbox "'quick' was not found"
    else
```

```

        msgbox "'quick' was found at position"+str$(i%)
    end if
    'search starting at position 10
    i% = InStr(10,teststring,"quick")
    if i% = 0 then
        msgbox "'quick' was not found"
    else
        msgbox "'quick' was found at position"+str$(i%)
    end if
end sub

```

See Also Strings (Chapter 7)

Int Function

Description Returns the integer part of a given number.

Syntax `int (number#)`

Comments This function returns the first integer less than (rounds down) `number#`. The sign is preserved.

Example `sub main()`
 `'example of INT function`
 `dim adouble as double`

 `adouble = pi`
 `msgbox str$(adouble)`
 `msgbox str$(int(adouble))`
 `end sub`

See Also `CInt Function`
 `Fix Function`
 Math Statements and Functions (Chapter 7)

Item\$ Function

Description Gets a set of contiguous, delimited items from a text string.

Syntax `item$(text$,first%,last% [,delimiters$])`

Returns Returns all of the items between `first%` and `last%` within the specified text.

Comments	<p>The <code>first%</code> parameter specifies the first item in the sequence to return. The lowest value for <code>first%</code> is 1. All items between <code>first%</code> and <code>last%</code> are returned.</p> <p>By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the <code>delimiters\$</code> parameter.</p> <p>If <code>first</code> is greater than the number of items in <code>text\$</code>, then an empty string is returned.</p> <p>If <code>last</code> is greater than the number of items in <code>text\$</code>, then all items from <code>first</code> to the end of <code>text</code> are returned.</p>
Example	<pre>sub main() 'Example of Item\$() and ItemCount dim pathstr as string pathstr = environ\$("PATH") 'get the path msgbox "There are"+str\$(itemcount(pathstr,";"))+" items in the path." msgbox pathstr msgbox item\$(pathstr,3,4,";") end sub</pre>
See Also	Strings (Chapter 7)

ItemCount Function

Description	Returns an integer representing the number of items in the specified text.
Syntax	<code>ItemCount(text\$ [, delimiters\$])</code>
Comments	<p>By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the <code>delimiters\$</code> parameter. For example, to parse items using a backslash:</p> <pre>n = itemCount(text\$, "\")</pre> <p>The first <code>text\$</code> item is 1.</p>
Example	Item\$ Example
See Also	Strings (Chapter 7)

Kill Statement

Description	Deletes one or more files.
--------------------	----------------------------

Syntax	<code>kill filespec\$</code>
Comments	<p>This command deletes all files matching <code>filespec\$</code>.</p> <p>The <code>filespec\$</code> parameter can contain wildcards, such as <code>*</code> and <code>?</code>.</p> <p>This function behaves the same as the "del" command in DOS.</p>
Example	<pre>sub main() 'Example of KILL command kill "c:\junk.txt" end sub</pre>
See Also	File Input and Output (Chapter 7)

LBound Function

Description	Determines the smallest subscript for a dimension of an array.
Syntax	<code>lbound(ArrayVariable() [,dimension%])</code>
Returns	Returns an integer representing the lower bound of the specified dimension of the specified array variable. If the array has no dimension, a runtime error is returned.
Comments	The first dimension is assumed if <code>dimension%</code> is not specified (i.e., <code>dimension% = 1</code>).
Example	<pre>sub main() 'Examples of LBOUND dim ia1(8) as integer dim ia2(65 to 70) as integer msgbox str\$(lbound(ia1)) msgbox str\$(lbound(ia2)) end sub</pre>
See Also	Arrays (Chapter 7)

LCase\$ Function

Description	Returns the lower case equivalent of the specified string.
Syntax	<code>Lcase\$(str)</code>
Example	<pre>sub main() 'Example of LCase\$</pre>

```

        dim teststr as string

        teststr = "THIS IS A TEST"
        msgbox teststr
        teststr = Lcase$(teststr)
        msgbox teststr
    end sub

```

See Also Strings (Chapter 7)

Left\$ Function

Description Returns the leftmost NumChars% characters from a given string.

Syntax Left\$(str\$, NumChars%)

Comments If NumChars% is 0, then an empty 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.

Example

```

sub main()
    'Example of left$()
    dim teststr as string

    teststr = "This is a test."
    msgbox teststr
    teststr = left$(teststr,7)
    msgbox teststr
end sub

```

See Also Strings (Chapter 7)

Len Function

Description Returns an integer representing the number of characters in a given string.

Syntax Len(str\$)

Comments If str is empty, 0 is returned.

Example

```

sub main()
    'Example of Len()
    dim teststr as string

```

```

teststr = "This is a test."
msgbox ""+teststr+" is"+str$(len(teststr))+ " characters long."
end sub

```

See Also Strings (Chapter 7)

Let Statement

Description Assigns the result of an expression to a variable.

Syntax [Let] variable = expression

Comments The `let` statement is supported for compatibility with other implementations of DCL.

Example

```

sub main()
    'Examples of Let statement

    let a% = 1
    let s$ = "test"
end sub

```

See Also Variables and Constants (Chapter 7)

Line\$ Function

Description Gets a set of lines (delimited by CR/LFs) from the specified text.

Syntax `Line$(text$, first%[, last%])`

Returns Returns a single line or group of lines between `first` and `last`.

Comments Lines are delimited by CR/LF pairs.

If `last` is not specified, then only one line is returned.

If `first` is greater than the number of lines in `text$`, an empty string is returned.

If `last` is greater than the number of lines in `text$`, all lines from `first` to the end of `text` are returned.

Example

```

sub main()
    'Example of Line$()
    dim crlf as string
    dim testlines as string

    crlf = chr$(13) + chr$(10)

```

```
        'carriage return followed by a linefeed
testlines = "line 1" + crlf
testlines = testlines + "line 2" + crlf
testlines = testlines + "line 3" + crlf
testlines = testlines + "line 4"
msgbox testlines
msgbox line$(testlines,2,3)
end sub
```

See Also Strings (Chapter 7)

LineCount Function

Description Returns an integer representing the number of lines in the specified text.

Syntax `LineCount(text$)`

Comments Lines are delimited by CR/LF pairs.

Example

```
sub main()
    'Example of LineCount
    dim crlf as string
    dim testlines as string

    crlf = chr$(13) + chr$(10)
    'carriage return followed by a linefeed
    testlines = "line 1" + crlf
    testlines = testlines + "line 2" + crlf
    testlines = testlines + "line 3" + crlf
    testlines = testlines + "line 4"
    msgbox testlines
    msgbox str$(linecount(testlines))+" lines total."
end sub
```

See Also Strings (Chapter 7)

LineInput # Statement

Description Reads a line into a string variable.

Syntax `LineInput [#]filenumber%,text$`

Comments This statement reads an entire line into the given string variable `text$`. The file is read up to the next carriage return. The file pointer is positioned after the terminating CR/LF.

The `filenumber%` parameter is a number that is used by DCL to refer to the open file—the number passed to the `open` statement.

The `filenumber%` must reference a file opened in `input` mode.

The `text$` parameter is any string variable reference.

Example Input/Output Example

See Also File Input and Output (Chapter 7)

Listbox Statement

Description	Defines a listbox that appears within a dialog box template.
Syntax	<code>Listbox x%,y%,width%,height%,items\$(),.Field</code>
Comments	<p>The <code>items\$</code> array must be a single-dimension array of strings. The elements of this array are placed into the listbox when the dialog box is created. The <code>.Field</code> parameter defines the name used to extract which string occupies the listbox when the dialog box ends. On exit from the <code>Dialog</code> statement, the <code>.Field</code> will contains an index to the item that is highlighted in the listbox.</p> <p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x%,y%,width%,height%</code> parameters are specified in dialog coordinates. The <code>x, y</code> position is relative to the upper left corner of the dialog box.</p>
Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

ListboxEnabled Function

Description	Determines whether a listbox is enabled within the current window or dialog box.
Syntax	<code>ListboxEnabled(name\$ id%)</code>
Returns	Returns the integer TRUE if the given listbox is enabled within the active window or dialog box, FALSE otherwise.
Comments	<p>If there is no active window, FALSE is returned.</p> <p>The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the listbox control in the window list (or dialog template). Alternatively, the <code>id%</code> of the listbox can be specified.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

ListboxExists Function

Description	Determines whether a listbox exists within the current window or dialog box.
Syntax	<code>ListboxExists (name\$ id%)</code>
Returns	Returns the integer TRUE if the given listbox exists within the active window or dialog box, FALSE otherwise.
Comments	<p>If there is no active window, FALSE is returned.</p> <p>The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the listbox control in the window list (or dialog template). Alternatively, the <code>id%</code> of the listbox can be specified.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

Loc Function

Description	<p>Returns an integer representing the position of the file pointer in the given file.</p> <p>Note: We recommend using the <code>Seek</code> function instead.</p>
Syntax	<code>Loc (filenumber%)</code>
Comments	The <code>filenumber%</code> parameter is a number that is used by DCL to refer to the open file—the number passed to the <code>open</code> statement.
See Also	File Input and Output (Chapter 7)

LOF Function

Description	Returns an integer representing the number of bytes in the given file.
Syntax	<code>LOF (filenumber%)</code>
Comments	The <code>filenumber</code> parameter is a number that is used by DCL to refer to the open file—the number passed to the <code>open</code> statement.
Example	Input/Output Example
See Also	File Input and Output (Chapter 7)

Log Function

Description	Returns a double-precision number representing the natural logarithm of a given number.
Syntax	<code>Log (number#)</code>
Comments	The value of <code>number</code> must be greater than 0.
Example	<pre>sub main() 'Example of log() function msgbox "Log of 1 is "+str\$(Log(1)) msgbox "Log of 2 is "+str\$(Log(2)) end sub</pre>
See Also	Math Statements and Functions (Chapter 7)

LTrim\$ Function

Description	Returns the specified string with the leading spaces removed.
Syntax	<code>Ltrim\$(str\$)</code>
Example	<pre>sub main() 'Example of LTrim\$() dim teststring as string teststring = " testing " msgbox "*" + teststring + "*" msgbox "*" + Ltrim\$(teststring) + "*" end sub</pre>
See Also	Strings (Chapter 7)

Main Statement

Description	Defines the Main subroutine for the script.
Syntax	<pre>sub main() end sub</pre>
Comments	This defines the subroutine that receives execution control from the host application.

See Also Procedure Statements (Chapter 7)

MCI Function

Description Executes MCI (multimedia) command.

Syntax `MCI(command$,result$ [,error$])`

Returns Returns the integer 0 if the function was successful, otherwise it returns an error number.

Comments If an error occurs, then the optional `error$` parameter is set to the text corresponding to the error.

If the `command$` returns a value, then that value is contained in `result$`.

The `mci` function accepts any MCI command as defined in the *Multimedia Programmers Reference* in the Windows 3.1 SDK.

Example

```
sub main()
    'Example MCI command
    'NOTE: This program may not run on your machine.

    dim resultstr as string
    dim errorstr as string

    MCI("play cdrom from 1 to 20",resultstr,errorstr)
end sub
```

See Also Environment Statements and Functions (Chapter 7)

Menu Statement

Description Issues the specified menu command from the active window.

Syntax `Menu MenuItem$`

Comments The `MenuItem$` parameter specifies the complete menu item name, each menu level separated with 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 popup 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 4th item from the third menu, use "#3,#4". Separators

counts as items.

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 popup, rather than a menu item. The menu item "File.Blank Blank" is not a valid menu item.

When comparing menu item names, this statement removes periods (.), spaces, and &. Further, 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.

Example

```
sub main()
    'Example of Menu command

    aname$ = AppFind$("Notepad")
    AppActivate aname$
    Menu "File.Page Setup"
end sub
```

See Also

Menus (Chapter 7)

MenuItemChecked Function

Description	Determines whether a menu item in the active window is checked.
Syntax	<code>MenuItemChecked (MenuItemName\$)</code>
Returns	Returns the integer TRUE if the given menu item exists and is checked, FALSE otherwise.
Comments	The <code>MenuItemName\$</code> parameter specifies a complete menu item or menu item popup following the same format as that used by the <code>Menu</code> statement.
See Also	Menus (Chapter 7)

MenuItemEnabled Function

Description	Determines whether a menu item in the active window is enabled.
--------------------	---

Syntax	<code>MenuItemEnabled (MenuItemName\$)</code>
Returns	Returns the integer TRUE if the given menu item exists and is enabled, FALSE otherwise.
Comments	The <code>MenuItemName\$</code> parameter specifies a complete menu item or menu item popup following the same format as that used by the <code>Menu</code> statement.
See Also	Menus (Chapter 7)

MenuItemExists Function

Description	Determines whether a menu item in the active window exists.
Syntax	<code>MenuItemExists (MenuItemName\$)</code>
Returns	Returns the integer TRUE if the given menu item exists, FALSE otherwise.
Comments	The <code>MenuItemName\$</code> parameter specifies a complete menu item or menu item popup following the same format as that used by the <code>Menu</code> statement.
Examples	<pre>sub main() if MenuItemExists("File.Open") then beep if MenuItemExists("File") then MsgBox "There is a File menu." end sub</pre>
See Also	Menus (Chapter 7)

Mid\$ Function

Description	The <code>mid\$</code> function returns a substring. The <code>mid\$</code> function can also replace a substring with new text.
Syntax 1	<code>Mid\$(str\$, start% [, length%])</code>
Returns	Returns a substring from the specified string. The substring starts at character position <code>start%</code> for <code>length%</code> number of characters.
Comments	<p>If <code>length%</code> is not specified, then the entire string starting at <code>start%</code> is returned.</p> <p>If <code>start%</code> is greater than the length of <code>str\$</code>, then an empty string is returned.</p>
Syntax 2	<code>Mid\$(str\$, start%[, length%]) = newvalue\$</code>
Comments	This statement replaces one part of a string with another. The <code>str\$</code> parameter specifies the string variable containing the substring to replace. The substring within <code>str\$</code> which is

replaced starts at the character position specified by `start%` for `length%` number of characters. If `length%` is not specified, then the rest of the string is assumed.

The `newvalue$` parameter is any string or string expression. The resultant string is never longer than the original length of `str$`. Any extra characters at the end of `newvalue$` are ignored.

Example

```
sub main()
    'Example of mid$() function and command
    dim teststr as string

    teststr = "This is a test."
    msgbox teststr
    msgbox mid$(teststr,3,5)      'starting at 3 and
                                'retrieving 5 characters

    mid$(teststr,3,5) = "      "
    msgbox teststr
    msgbox mid$(teststr,3,5)      'starting at 3 and
                                'retrieving 5 characters
end sub
```

See Also

Strings (Chapter 7)

Minute Function

Description

Returns an integer representing the minute of the day encoded in the specified `serial#` parameter. The value returned is between 0 and 59 inclusive.

Syntax

`Minute(serial#)`

Comment

You can obtain the value for the `serial#` parameter by using the `TimeSerial` or `TimeValue` command.

Example

```
sub main()
    'Example of minute() function
    dim dt as double

    dt = Now
    msgbox str$(minute(dt))      'current minute
end sub
```

See Also

Date and Time Functions (Chapter 7)

MkDir Statement

Description	Creates a new directory.
Syntax	MkDir dir\$
Comments	This command behaves just like the DOS "md" command.
Example	<pre>sub main() 'Example of mkdir MkDir "C:\testdir" end sub</pre>
See Also	File Input and Output (Chapter 7)

Mod

Description	Modulo operator.
Syntax	expression1 mod expression2
Returns	Returns the remainder of expression1 / expression2.
Notes	The two operands are converted to whole numbers before performing the modulo operation.
Example	<pre>sub main() 'Example of MOD msgbox str\$(5 mod 3) 'should display 2--the 'remainder after division end sub</pre>
See Also	Operators (Chapter 7)

Month Function

Description	Returns an integer representing the month of the date encoded in the specified serial# parameter. The value returned is between 1 and 12 inclusive.
Syntax	month(serial#)
Comment	You can obtain the value for the serial# parameter by using the DateSerial and DateValue command.

Example

```

sub main()
    'Example of Month

    msgbox str$(month(Now))    'display current month
end sub

```

See Also

Date and Time Functions (Chapter 7)

Message Example

```

sub main()
    'Example of MsgOpen, MsgSetText, MsgSetThermometer, and MsgClose

    MsgOpen "Message Box - 0% complete.",0,TRUE,TRUE
    on error goto cancelpressed
    for i% = 1 to 100
        sleep 100
        msg$ = "Message Box -"+str$(i%)+"% complete."
        MsgSetText msg$
        MsgSetThermometer i%
    next i%
    MsgClose
    msgbox "Finished Normally"
end

cancelpressed:
    MsgClose
    on error goto 0
    msgbox "Cancel was selected."
end
end sub

```

MsgBox Statement and Function

Description	The <code>MsgBox</code> statement displays a messages box; the <code>MsgBox</code> function displays a message box and returns an integer representing the button that was pressed.
Statement Syntax	<code>MsgBox msg\$ [,type% [,title\$]]</code>
Function Syntax	<code>MsgBox(msg\$ [,type% [,title\$]])</code>
Returns	The <code>MsgBox</code> function returns one of the following numbers:


- 1 OK was pressed
- 2 Cancel was pressed
- 3 Abort was pressed
- 4 Retry was pressed
- 5 Ignore was pressed
- 6 Yes was pressed
- 7 No was pressed


Comments

The dialog box is sized to hold the entire contents of `msg$`. The `msg$` string can contain CR/LF to separate lines. If a given line is too long, it will be word wrapped.

The `type` parameter is the sub of the any of the following values:

- 0 display OK button only
- 1 display OK, Cancel buttons
- 2 display Abort, Retry, Ignore buttons
- 3 display Yes, No, Cancel buttons
- 4 display Yes, No buttons
- 5 display Retry, Cancel buttons

- 16 display "stop" icon 

- 32 display "question mark" icon 

- 48 display "exclamation point" icon 

- 64 display "information" icon 

- 0 first button is the default button
- 256 second button is the default button
- 512 third button is the default button

- 0 Application modal - the current application is suspended until dialog box is closed

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).

The default value for `title$` is "DCL".

Example

```
sub main()
    'Examples of MsgBox function and statement

    for i% = 0 to 5
        MsgBox "message", i%, "title"
    next i%
    for i% = 0 to 5
        result = MsgBox("message", i%, "title")
        select case result
            case 1
                msgbox "OK was pressed"
            case 2
                msgbox "Cancel was pressed"
            case 3
                msgbox "Abort was pressed"
            case 4
                msgbox "Retry was pressed"
            case 5
                msgbox "Ignore was pressed"
            case 6
                msgbox "Yes was pressed"
            case 7
                msgbox "No was pressed"
        end select
    next i%

    'In this dialog box the second button is the default.
    result=msgbox("Hello World", 3+256, "title")

    'In this dialog box the Stop symbol is displayed.
    result=msgbox("Hello World", 3+256+16, "title")

end sub
```

See Also Dialog Display (Chapter 7)

MsgClose Statement

Description Closes a message window that was opened with the `MsgOpen` statement.

Syntax	<code>MsgClose</code>
Comments	Nothing will happen if there is no open message window.
Example	Message Example
See Also	Dialog Display (Chapter 7)

MsgOpen Statement

Description	Displays a window with a message.
Syntax	<code>MsgOpen msg\$, timeout%, isCancel%, isThermometer%[, x%, y%]</code>
Comments	<p>The message can be displayed permanently, or for a specified number of seconds, or until an optional Cancel button is pressed.</p> <p>The displayed message can be changed by calling the <code>MsgSetText</code> statement.</p> <p>The <code>timeout%</code> parameter causes the window to automatically be removed after that number of seconds. The <code>timeout%</code> parameter has no effect if its value is zero.</p> <p>The <code>isCancel</code> parameter controls whether or not a Cancel button appears within the window beneath the displayed message. If <code>TRUE</code>, then a Cancel button appears. If not specified, or <code>FALSE</code>, then no Cancel button is created. If a user presses the Cancel button at runtime, a trappable runtime error is generated. In this manner, a message box can be displayed and processing can continue as normal, aborting only when the process is canceled by pressing the Cancel button.</p> <p>The <code>isThermometer</code> parameter controls whether there is a thermometer. If <code>TRUE</code>, then a thermometer is created between the text and the optional Cancel button. The thermometer initially indicated 0% complete, and can be changed using the <code>MsgSetThermometer</code> statement.</p> <p>The optional <code>x</code>, <code>y</code> parameters specify the location of the upper left corner of the message box, in twips (1/20th of a point or 1/1440 of an inch). If this point is not specified, then the window is centered on top of the parent.</p> <p>Only one message window can be opened at any one time. The message window is removed automatically when a script terminates.</p>

Example 1 Message Example

Example 2 `MsgOpen "Printing. Please wait...", 0, FALSE, FALSE`

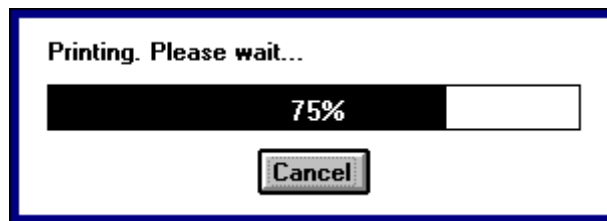


Example 3

```
MsgOpen "Printing. Please wait... ",0,TRUE,FALSE
```

**Example 4**

```
MsgOpen "Printing. Please wait... ",0,TRUE,TRUE  
MsgSetThermometer 75
```

**See Also**

Dialog Display (Chapter 7)

MsgSetText Statement

Description	Changes the text within an open message box (one that was previously opened with <code>MsgOpen</code>).
Syntax	<code>MsgSetText newtext\$</code>
Comments	The message box is resized to accommodate the new text. A runtime error will result if a message box is not currently open (using <code>MsgOpen</code>).
Example	Message Example
See Also	Dialog Display (Chapter 7)

MsgSetThermometer Statement

Description	Changes the percentage filled in the thermometer of an open message box (one that was previously opened with <code>MsgOpen</code>).
--------------------	--

Syntax	<code>MsgSetThermometer percentage%</code>
Comments	A runtime error will result if a message box is not currently open (using <code>MsgOpen</code>), or if the value of <code>percentage%</code> is not between 0 and 100 inclusive.
Example	Message Example
See Also	Dialog Display (Chapter 7)

Name Statement

Description	Renames a file.
Syntax	<code>name oldfile\$ as newfile\$</code>
Example	<pre>sub main() 'Example of Name statement Name "C:\JUNK.TXT" as "C:\NEWJUNK.TXT" 'renames JUNK.TXT to NEWJUNK.TXT end sub</pre>
See Also	File Input and Output (Chapter 7)

NetAttach Function

Description	Attaches the current user to the specified server using the given user ID and password.
Syntax	<code>NetAttach(server\$, user\$, password\$)</code>
Comments	This function returns the integer TRUE if it is successful in attaching to the specified server or FALSE if the user is already attached, the password verification fails, or the server is not available.
Example	<pre>sub main() 'Example of NetAttach s\$ = AskBox\$("Server to attach to:") u\$ = AskBox\$("Username:") p\$ = AskPassword\$("Password:") if NetAttach(s\$,u\$,p\$) then msgbox("Attach Successful.") else msgbox("Attach Failed.") end if end sub</pre>

```

        end if
    end sub

```

See Also Network Functions (Chapter 7)

NetConnectDrive Function

Description Connects a local drive letter with a network path.

Syntax `NetConnectDrive(localpath$, networkpath$ [, [root%], passwd$])`

Comments Returns the integer TRUE if the drive is successfully mapped, otherwise returns FALSE.

If `root%` is specified, then the drive is created with a *false root*. If `passwd$` is specified it is used in attempting to connect the network drive. Note that `root%` and `passwd$` may not be supported in all networks.

The default for `root%` is FALSE, and `passwd$` defaults to none.

The `localpath$` parameter should be specified in the form "E:".

The `networkpath$` parameter is a valid network path, meaning *server/volume:...* or a UNC pathname.

Example

```

sub main()
    'Example of NetConnectDrive

    lp$ = AskBox$("Local drive to connect to:")
    np$ = AskBox$("Network path to connect:")
    if NetConnectDrive(lp$,np$) then
        msgbox "Drive connected."
    else
        msgbox "Drive connection failed."
    end if
end sub

```

See Also Network Functions (Chapter 7)

NetDetach Function

Description Detaches the current user from the specified file server.

Syntax	<code>NetDetach(server\$)</code>
Comment	Returns the integer TRUE if the drive is successfully detached, otherwise returns FALSE.
Caution	No safety checks are performed during the detach operation. Detaching from a server is a dangerous thing to do, and this function in no way guarantees that it will be done without causing some sort of system failure.
Example	<pre> sub main() 'Example of NetDetach s\$ = AskBox\$("Server To Detach:") if NetDetach(s\$) then msgbox "Server Detached." else msgbox "Error Detaching Server." end if end sub </pre>
See Also	Network Functions (Chapter 7)

NetDirectoryRights Function

Description	Returns the integer TRUE if the current user has the given rights for the specified network directory path.
Syntax	<code>NetDirectoryRights(path\$, rights\$)</code>
Comments	<p>The <code>path\$</code> parameter may be any legal reference to a directory. This includes full paths with drive letter, server/volume, or UNC; or partial paths that are relative to the current directory.</p> <p>The <code>rights\$</code> parameter is a series of characters that have network specific meanings. For example, "ROS" on NetWare means Read, Open & Search permissions. If the user has all of these permissions for the specified path, the return value is TRUE.</p>
Example	<pre> sub main() 'Example of NetDirectoryRights p\$ = AskBox\$("Path to get rights for:") r\$ = AskBox\$("Rights to search for:") if NetDirectoryRights(p\$,r\$) then msgbox "You have "+r\$+" rights for "+p\$ else msgbox "You do not have "+r\$+" rights for "+p\$ end if end sub </pre>

See Also Network Functions (Chapter 7)

NetDisconnectDrive Function

Description Disconnects a local drive letter from a network path.

Syntax NetDisconnectDrive (networkdrive\$)

Comments The function takes a single string parameter which represents the drive letter that is to be disconnected. It returns an integer value 0 or -1. A 0 (or FALSE) indicates function failure, a -1 (or TRUE) indicates success.

Example

```
sub main()
    'Example of NetDisconnectDrive() function
    dim drive as string
    dim netpath as string

    drive = "k"
    netpath = "sweng_1/sys2:"

    if NetConnectDrive(drive,netpath) then
        msgbox "Drive "+drive+" has been connected."
        if NetDisconnectDrive(drive) then
            msgbox "Drive "+drive+" has been disconnected."
        else
            msgbox "Disconnect Failed."
        end if
    else
        msgbox "Connection Failed."
    end if
end sub
```

See Also Network Functions (Chapter 7)

NetGetDirectoryRights\$ Function

Description Returns a string representing the effective rights for the current user on the specified path.

Syntax NetGetDirectoryRights\$ (path\$)

Comments The rights are identified as a series of characters compatible with those used by the network

operating system that is in use.

Example

```
sub main()
    'Example of NetGetDirectoryRights$()

    p$ = AskBox$("Path to get rights list for:")
    r$ = NetGetDirectoryRights$(p$)
    msgbox "Your rights for "+p$+" are "+r$
end sub
```

See Also

Network Functions (Chapter 7)

NetMemberOf Function

Description

Determines whether the current user is a member of the specified group.

Syntax

NetMemberOf (group\$, server\$)

Comments

This function returns the integer TRUE or FALSE.

If the `server$` parameter is specified then the groups on that server are searched, otherwise the primary server is used.

If the user `USERA` is a member of the group `SMALLGRP`, and `SMALLGRP` is a member of `BIGGROUP`, `USERA` is a member of `BIGGROUP`.

Example

```
sub main()
    'Example of NetMemberOf

    g$ = AskBox$("Group To Search For:")
    s$ = AskBox$("Server To Search:")
    if NetMemberOf(g$, s$) then
        msgbox "You are a member of that group."
    else
        msgbox "You are not a member of that group."
    end if
end sub
```

See Also

Network Functions (Chapter 7)

NetStationID Function

Description	Returns a network-dependent station id as a string.
Syntax	<code>NetStationID\$()</code>
Comments	On Novell networks, the station's Ethernet address is returned.
Example	<pre>sub main() 'Example of NetStationID\$() nsi\$ = NetStationID\$() msgbox "Your ethernet address is: "+nsi\$ end sub</pre>
See Also	Network Functions (Chapter 7)

NetUserName Function

Description	Returns the user name associated with the given server.
Syntax	<code>NetUserName\$([server\$])</code>
Comments	If server isn't specified then the primary server is used.
Example	<pre>sub main() 'Example of NetUserName s\$ = AskBox\$("Server:") nu1\$ = NetUserName\$() nu2\$ = NetUserName\$(s\$) msgbox "You are "+nu1\$+" on your primary server, and "+nu2\$+" on "+s\$+"." end sub</pre>
See Also	Network Functions (Chapter 7)

NetworkStatus Function

Description	Returns an integer representing the network status as a 16 bit (WORD) set of flags.
Syntax	<code>NetworkStatus()</code>
Comments	Each bit of the returned status has different significance. Currently, there are two bit flags. NS_ACTIVE (0x0001) The network redirector is loaded

NS_LOGGEDON (0x0002) A user is actively logged onto a server/the network.

Normally, this function is compared to the value 3 to determine if additional network calls can or should be made.

Example

```
sub main()  
    'Example of NetworkStatus function  
  
    ns% = NetworkStatus()  
    msgbox str$(ns%)  
    if ns% AND NS_ACTIVE then  
        msgbox "Network is active."  
    else  
        msgbox "Network is not active."  
    end if  
    if ns% AND NS_LOGGEDON then  
        msgbox "Logged On."  
    else  
        msgbox "Not Logged On."  
    end if  
end sub
```

See Also

Network Functions (Chapter 7)

Not

Description

Not operator.

Syntax

NOT expression1

Returns

TRUE if expression1 is FALSE, otherwise returns TRUE.

If the operand is numeric, then the result is the bitwise NOT of the argument.

Notes

If the operand is a floating point value (either single or double), then it is first converted to a long, then a bitwise NOT is performed.

Example

```
sub main()  
    'Example of NOT operator  
    dim testvar as integer  
  
    testvar = TRUE  
    if testvar then  
        msgbox "TestVar is TRUE"  
    end if  
  
    testvar = FALSE  
    if NOT testvar then
```

```

        msgbox "TestVar is NOT TRUE"
    end if
end sub

```

See Also Operators (Chapter 7)

Now Function

Description Returns a double-precision number representing the current date and time. The number is returned in days where Dec 20, 1899 is 0.

Syntax `now()`

Example

```

sub main()
    'Example of the Now function
    dim cdt as double

    cdt = Now
    msgbox "Current date is "+str$(month(cdt))+"/"+str$(day(cdt))+"/"+ ←
        str$(year(cdt))
end sub

```

See Also Date and Time Functions (Chapter 7)

NS_ACTIVE

Description Constant used by the `NetworkStatus` command.

NS_LOGGEDON

Description Constant used by the `NetworkStatus` command.

Null Function

Description Returns a null string (a string that contains no characters and requires no storage).

Syntax	<code>null[()]</code>
Comments	An empty string ("") can also be used to remove all characters from a string. However, empty string still requires some memory for storage. Null strings require no memory.
Example	<pre> sub main() 'Example of the Null function dim teststr as string teststr = "testing" msgbox ""+teststr+"" teststr = null msgbox ""+teststr+"" end sub </pre>
See Also	Strings (Chapter 7)

Oct\$ Function

Description	Returns a string containing the octal equivalent of the specified number.
Syntax	<code>Oct\$(number%)</code>
Comments	<p>The returned string contains only the number of octal digits necessary to represent the number.</p> <p>The <code>number</code> parameter can be any type, but is rounded to the nearest whole number before converting to octal.</p>
Example	<pre> sub main() 'Example of Oct\$() function msgbox Oct\$(9) end sub </pre>
See Also	Conversions (Chapter 7)

OKButton Statement

Description	Defines an OK button that appears within a dialog box template.
Syntax	<code>OKButton x%,y%,width%,height%</code>
Comments	This statement can only appear within a dialog box template definition (BEGIN

DIALOG...END DIALOG).

The `x`, `y`, `width`, `height` parameters are specified in dialog coordinates. The `x`, `y` position is relative to the upper left corner of the dialog box.

Example Dialog Examples

See Also Dialog Creation (Chapter 7)

On Error Statement

Description Defines the action taken when a trappable runtime error occurs.

Syntax `on error {goto <label> | resume next | goto 0}`

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 to the next line after 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.

If an error occurs within the error handler, the current routines error trap is disabled and a runtime error results.

Example

```
sub main()
    'Example of ON-ERROR statement

    on error goto elabel      'enable error trap
    error 101                 'simulate an error
end

elabel:                     'error trap label
    on error goto 0          'disable error trapping
end
end sub
```

See Also Error Trapping (Chapter 7)

Flow Control (Chapter 7)

Open Statement

Description	Opens a file.
Syntax	<code>open filename\$ [for {input output append}] as [#] filenumber%</code>
Comments	<p>This statement opens a file for a given mode, assigning the open file to the supplied <code>filenumber</code>.</p> <p>The <code>filename</code> parameter is a string expression that contains a valid DOS filename.</p> <p>The <code>filenumber</code> parameter is a number between 1 and 255. The <code>FreeFile()</code> function can be used to determine an available file number.</p> <p>The different modes are defined as follows:</p> <p>Input Opens an existing file for input.</p> <p>Output Opens an existing file, truncating its length to zero, or creates a new file.</p> <p>Append Opens an existing file, positioning the file pointer at the end of the file, or creates a new file.</p> <p>If the <code>[for mode]</code> is missing, then <code>append</code> is used.</p>
Example	Input/Output Example
See Also	File Input and Output (Chapter 7)

OpenFileName\$ Function

Description	Displays the common file open dialog box (from <code>COMMDLG.DLL</code>), allowing the user to select a file.
Syntax	<code>OpenFileName\$(title\$,extensions\$)</code>
Returns	Returns the full DOS pathname of the file the user selected, or an empty string if the user canceled the dialog box.
Comments	<p>The <code>title\$</code> parameter specifies the title that appears on the dialog box's caption.</p> <p>The <code>extensions\$</code> parameter specifies the available file types. This string should be in the following format:</p> <pre>"type:ext[,ext] [;type:ext[,ext]]..."</pre> <p>where <code>ext</code> is a valid file extension, like <code>*.BAT</code> or <code>*.??</code>, and <code>type</code> is a string that identifies this type to the user.</p>

Example

```
sub main()
    'Example of OpenFileName$

    selffile$ = OpenFileName$("Open File","All Files:*.bmp, *
    *.wmf;Bitmaps:*.bmp; Metafiles:*.wmf")
    msgbox "Selected File = "+selfile$
end sub
```

See Also Dialog Display (Chapter 7)

Option Base Statement

Description Sets the lower bound for array declarations. By default, the lower bound used for all array declarations is 0.

Syntax Option Base {0 | 1}

Comments This statement must appear outside of any functions or subroutines.

Example

```
Option Base 1

sub main()
    'Example of Option Base statement
    dim a(5) as integer

    msgbox str$(lbound(a))
end sub
```

See Also Arrays (Chapter 7)

OptionButton Statement

Description Defines a push button with the specified text that appears within a dialog box template.

Syntax OptionButton x%,y%,width%,height%,title\$

Comments The title\$ parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for Font.

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The x%,y%,width%,height% parameters are specified in dialog coordinates. The x, y position is relative to the upper left corner of the dialog box.

Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

OptionEnabled Function

Description	Determines whether an option button is enabled within the current window or dialog box.
Syntax 1	<code>OptionEnabled (name\$)</code>
Syntax 2	<code>OptionEnabled (id%)</code>
Returns	Returns the integer TRUE if the specified option button is enabled within the current window or dialog box, otherwise this function returns FALSE.
Comments	<p>If an option button is enabled, its value can be set using the <code>SetOption</code> statement.</p> <p>The option button can be referenced given either its <code>name\$</code> (caption) or its <code>id%</code>.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

OptionExists Function

Description	Determines whether an option button exists within the current window or dialog box.
Syntax 1	<code>OptionExists (name\$)</code>
Syntax 2	<code>OptionExists (id%)</code>
Returns	Returns the integer TRUE if the specified option button exists within the current window or dialog box, otherwise this function returns FALSE.
Comments	The option button can be referenced given either its <code>name\$</code> (caption) or its <code>id%</code> .
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

OptionGroup Statement

Description	Starts a group of option buttons within a dialog box template and defines the name used to determine which option button (from the group of option buttons) is selected when the dialog box ends.
Syntax	<code>OptionGroup .Field</code>
Comments	<p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>On exit from the <code>Dialog</code> statement, the <code>.Field</code> will contain the index of the selected option button, with 0 being the first option button.</p>
Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

Or

Description	Or operator.
Syntax	<code>expression1 OR expression2</code>
Returns	TRUE if either <code>expression1</code> is TRUE or <code>expression2</code> is TRUE, otherwise FALSE.

	If the two operands are numeric, the result is the bitwise OR of the two arguments.
Notes	If either of the two operands is a floating point number, the two operands are first converted to longs, then a bitwise OR is performed.
Example	<pre> sub main() 'OR statement dim a as integer dim b as integer a = 5 b = 9 if (a < 6) OR (b > 8) then msgbox "One of the conditions was true." else msgbox "Neither condition was true." end if if (a < 6) OR (b > 9) then msgbox "One of the conditions was true." else msgbox "Neither condition was true." end if end sub </pre>
See Also	Operators (Chapter 7)

PI

Description	Pi constant.
Syntax	PI
Returns	3.141592653589793238462643383279
Notes	PI can also be determined using the following formula: $4 * \text{atn}(1)$
Example	<pre> sub main() 'Example of the PI function dim adouble as double adouble = pi msgbox str\$(adouble) end sub </pre>

See Also Math Statements and Functions (Chapter 7)

PO_LANDSCAPE

Description Constant used with the `PrinterSetOrientation` statement to align the paper horizontally.

Returns 2

See Also `PrinterGetOrientation`
`PrinterSetOrientation`

PO_PORTRAIT

Description Constant used with the `PrinterSetOrientation` statement to align the paper vertically.

Returns 1

See Also `PrinterGetOrientation`
`PrinterSetOrientation`

PopupMenu Function

Description Creates a popup menu using the string elements in the given array and returns an integer representing the user's response.

Syntax `PopupMenu (MenuItems$ ())`

Returns Returns the index of the selected item. If no item is selected (the popup menu is canceled), then a value of 1 less than the lower bound is returned.

Comments Each array element is used as a menu item. An empty string results in a separator bar in the menu.

The popup 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 popup menu can be displayed at a time. An error will result if another script executes this function while a popup menu is visible.

Example

```
sub main()
    'Example of PopupMenu
    dim apps$() as string
    dim result as integer

    AppList apps$
    result = PopupMenu(apps$)
    if result >= lbound(apps$) then
        msgbox "You chose "+apps$(result)
    else
        msgbox "You chose nothing -"+str$(result)
    end if
end sub
```

See Also

Dialog Display (Chapter 7)

Print Statement

Description

Writes data to a viewport window.

Syntax

```
print expression [{, | ;} expression]...
```

Comments

Strings are written in their literal form, with no enclosing quotes.

Integers and longs are written with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.

Each *expression* is separated either with 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 is not followed by a comma or semicolon, then a carriage return is printed to the file. If the last expression in the list ends with a semicolon, no carriage return is printed—the 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.

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.

Example

```
sub main()
    'Example of Print

    ViewPortOpen
```

```

    ViewPortClear
    print "this is some data"
    sleep 5000
    ViewPortClose
end sub

```

See Also Viewport Window Manipulation (Chapter 7)

Print # Statement

- Description** Writes data to a disk file.
- Syntax** `print #filenumber%, expression [{, | ;} expression]...`
- Comments** The `filenumber%` parameter is a number that is used by DCL to refer to the open file—the number passed to the `open` statement.
- Strings are written in their literal form, with no enclosing quotes.
- Integers and longs are written with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
- Each `expression` is separated either with 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 is not followed by a comma or semicolon, then a carriage return is printed to the file. If the last expression in the list ends with a semicolon, no carriage return is printed - the 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 a carriage-return. 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` modes.

Example

```

sub main()
    open "test.dat" for output as #1
    print #1,10,34,"Hello World";
    a = 10
    s$ = "this is a test"
    print #1,a;s$,
    print #1,67
    close #1
end sub

```

See Also File Input and Output (Chapter 7)

PrinterGetOrientation Function

Description Retrieves the orientation of the default printer—the printer specified in the `device=` line in the `[windows]` section of the WIN.INI file.

Syntax `PrinterGetOrientation()`

Returns Returns the integer `PO_PORTRAIT` if the printer orientation is set to portrait, otherwise if returns `PO_LANDSCAPE`.

Comments This function loads the printer driver and therefore may be slow.

Example

```
sub main()
    'Example of PrinterGetOrientation and
    'PrinterSetOrientation
    dim oldorient as integer

    oldorient = PrinterGetOrientation()
    select case oldorient
        case PO_PORTRAIT
            msgbox "Printer is set up for portrait print."
        case PO_LANDSCAPE
            msgbox "Printer is set up for landscape print."
    end select
    PrinterSetOrientation oldorient
end sub
```

See Also Printer Manipulation (Chapter 7)

PrinterSetOrientation Statement

Description Sets the orientation of the default printer—the printer specified in the `device=` line in the `[windows]` section of the WIN.INI file.

Syntax `PrinterSetOrientation NewSetting%`

Comments `NewSetting%` is `PO_PORTRAIT` or `PO_LANDSCAPE`.

This command loads the printer driver for the default printer, and therefore may be slow.

Example

```
sub main()  
    'Example of PrinterGetOrientation and  
    'PrinterSetOrientation  
    dim oldorient as integer  
  
    oldorient = PrinterGetOrientation()  
    select case oldorient  
        case PO_PORTRAIT  
            msgbox "Printer is set up for portrait print."  
        case PO_LANDSCAPE  
            msgbox "Printer is set up for landscape print."  
    end select  
    PrinterSetOrientation oldorient  
end sub
```

See Also

Printer Manipulation (Chapter 7)

PrintFile Function

Description	Invokes the Windows 3.1 shell functions that cause an application to execute and print a file.
Syntax	<code>PrintFile(filename\$)</code>
Returns	Returns an integer representing the ID of the executing task.
Comments	<p>This function is only available under Windows 3.1.</p> <p>The application to be executed must be associated with the file extension of the file specified by this command. This association is established in the [Extensions] section of the WIN.INI file. For example, if the Notepad application is associated with the .TXT extension, the Notepad application is started when DCL executes a PrintFile command for a .TXT file.</p> <p>This command does not support .EXE, .COM, .BAT, or .PIF files.</p>
Example	<pre>sub main() 'Example of PrintFile taskid = printfile("c:\testfile.txt") msgbox "Your file is printing as task #" + str\$(taskid) end sub</pre>
See Also	Printer Manipulation (Chapter 7)

PushButton Statement

Description	Defines a push button within a dialog box template.
Syntax	<code>PushButton x%,y%,width%,height%,title\$</code>
Comments	<p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>When a push button is selected, the <code>Dialog</code> statement ends.</p> <p>The <code>x%,y%,width%,height%</code> parameters are specified in dialog coordinates. The <code>x, y</code> position is relative to the upper left corner of the dialog box.</p> <p>The <code>title\$</code> parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for <u>F</u>ont.</p>
Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

QueEmpty Statement

Description	Empties the current event queue.
Syntax	<code>QueEmpty</code>
Comments	After this statement, <code>QueFlush</code> will do nothing.
Example	Queue Example
See Also	Keyboard Manipulation (Chapter 7) Mouse Events (Chapter 7)

QueFlush Statement

Description	Plays back events that are stored in the current event queue.
Syntax	<code>QueFlush isSaveState%</code>
Comments	<p>After <code>QueFlush</code> is finished, the queue is empty.</p> <p>The <code>QueFlush</code> 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.</p> <p>If <code>isSaveState</code> is <code>TRUE</code>, then <code>QueFlush</code> saves the state of the <code>CAPSLOCK</code>, <code>NUMLOCK</code>, <code>SCROLL LOCK</code>, and <code>INSERT</code>, and restores the state after the <code>QueFlush</code> is complete. If <code>FALSE</code>, these states are not restored.</p> <p>The function does not return until the entire queue has been played.</p>
Example	Queue Example
See Also	Keyboard Manipulation (Chapter 7) Mouse Events (Chapter 7)

QueKeyDn Statement

Description	Appends key down events for the specified keys to the end of the current event queue.
--------------------	---

Syntax	<code>QueKeyDn Keys\$</code>
Comments	<p>The format for <code>Keys\$</code> is the same as for the <code>QueKeys KeyString\$</code> parameter, with the exception that parentheses are illegal.</p> <p>The <code>QueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	<p>Keyboard Manipulation (Chapter 7)</p> <p><code>QueKeys</code> for list of special keys.</p>

QueKeys Statement

Description	Appends keystroke information to the current event queue.
Syntax	<code>QueKeys KeyString\$</code>
Comments	<p>To specify any key on the keyboard, simply use that key, such as "a" for lower case a, or "A" for upper case a.</p> <p>Sequences of keys are specified by appending them together: "abc" or "dir /w".</p> <p>The keys +, ^, ~, and % are special and must be specified within brackets. For example, to specify the percent, use "{%}".</p> <p>The keys ()[]{} also have special meaning. To specify one of these keys, enclose it within brackets, such as "{()}".</p> <p>Keys that are not displayed when you press that key are described within brackets, such as {ENTER} or {UP}.</p> <p>A list of these keys follows:</p>

{BACKSPACE}	{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}	{SCROLLLOCK}
{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:

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:

```
Shift+A, Shift+B, Shift+C      "+(abc)"
Ctrl+F1, Ctrl+F2               "^{F1}{F2}"
```

Use "~" as a shortcut for embedding ENTER within a key sequence:

```
"ab~de"
```

To embed quotes, use two quotes in a row:

```
"This is a ""test"" of the system"
```

Key sequences can be repeated using a repeat count within brackets:

```
"{a 10}"      produces 10 "a" keys
"{ENTER 2}"   produces 2 Enter keys
```

Example Queue Example

See Also Keyboard Manipulation (Chapter 7)

QueKeyUp Statement

Description Appends key up events for the specified keys to the end of the current event queue.

Syntax `QueKeyUp Keys$`

Comments The format for `Keys$` is the same as for the `QueKeys KeyString$` parameter, with the exception that parentheses are illegal.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example Queue Example

See Also Keyboard Manipulation (Chapter 7)

`QueKeys` for list of special keys.

QueueMouseClick Statement

Description	Adds a mouse click to the current event queue.
Syntax	<code>QueueMouseClick button%, x%, y%</code>
Comments	<p>A mouse click consists of a mouse button down at position <i>x</i>, <i>y</i>, immediately followed by a mouse button up.</p> <p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	Mouse Events (Chapter 7)

QueueMouseDblClk Statement

Description	Adds a mouse double click to the current event queue.
Syntax	<code>QueueMouseDblClk button%, x%, y%</code>
Comments	<p>A mouse double click consists of a mouse DN/UP/DN/UP at position <i>x</i>, <i>y</i>. The events are queued in such a way that a double click is registered during queue playback.</p> <p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	Mouse Events (Chapter 7)

QueueMouseDblDn Statement

Description	Adds a mouse double down to the current event queue.
Syntax	<code>QueueMouseDblDn button%, x%, y%</code>

Comments	<p>A double down consists of a mouse DN/UP/DN at position <code>x%, y%</code>.</p> <p>The <code>button%</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	Mouse Events (Chapter 7)

QueueMouseDown Statement

Description	Adds a mouse down to the current event queue.
Syntax	<code>QueueMouseDown button%, x%, y%</code>
Comments	<p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	Mouse Events (Chapter 7)

QueueMouseMove Statement

Description	Adds a mouse move to the current event queue.
Syntax	<code>QueueMouseMove button%, x%, y%</code>
Comments	<p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	Mouse Events (Chapter 7)

QueMouseUp Statement

Description	Adds a mouse up to the current event queue.
Syntax	<code>QueMouseUp button%, x%, y%</code>
Comments	<p>The <code>button</code> parameter specifies which button to queue: either <code>VK_LBUTTON</code> or <code>VK_RBUTTON</code>.</p> <p>The <code>QueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example	Queue Example
See Also	Mouse Events (Chapter 7)

QueSetRelativeWindow Statement

Description	Adjusts all mouse positions relative to the specified window.
Syntax	<code>QueSetRelativeWindow hWnd%</code>
Comments	<p>This statement affects all subsequent <code>Que...</code> commands.</p> <p>The <code>hWnd%</code> parameter is a handle to a window in the Windows environment. If <code>hWnd%</code> is 0, then the window with the focus is used (i.e., the active window).</p> <p>The <code>QueFlush</code> command is used to play back the events stored in the current event queue.</p>
Example 1	Queue Example
Example 2	<pre>sub main() 'Adjust mouse coordinates relative to Notepad hWnd = WinFind("Notepad") QueSetRelativeWindow hWnd end sub</pre>
See Also	Mouse Events (Chapter 7)

Queue Example

```
sub main()
'Example of QUE commands
dim npWnd as integer
dim npname as string
```

```

npname = AppFind$("Notepad")
AppActivate npname
npWnd = WinFind(npname)
QueueEmpty      'empty the queue
QueueKeyDn "D"
QueueKeyUp "a"
QueueKeys "vid"
QueueMouseClicked VK_RBUTTON,1,1
QueueMouseDblClk VK_RBUTTON,1,1
QueueMouseDblDn VK_RBUTTON,1,1
QueueMouseDn VK_RBUTTON,1,1
QueueMouseMove 100,100
QueueMouseUp VK_RBUTTON,100,100
QueueSetRelativeWindow npWnd
QueueFlush TRUE
QueueEmpty
end sub

```

Random Function

Description Generates a random number.

Syntax Random (min&,max&)

Returns Returns a long integer greater than or equal to min and less than or equal to max.

Example

```

sub main()
    'Example of Random function

    ViewPortOpen
    ViewPortClear
    for j% = 1 to 10
        i% = Random(1,100)
        print i%
    next j%
    sleep 5000
    ViewPortClose
end sub

```

See Also Math Statements and Functions (Chapter 7)

Randomize Function

Description Initializes the random number generator with a new seed.

Syntax	<code>Randomize [seed&]</code>
Comments	If seed is not specified, then the current value of the system clock is used.
Example	<pre>sub main() 'Example of Randomize statement Randomize 65 Randomize end sub</pre>
See Also	Math Statements and Functions (Chapter 7)

ReadINI\$ Function

Description	Returns the value of specified item from the specified INI file.
Syntax	<code>ReadINI\$(section\$,item\$[,filename\$])</code>
Comments	<p>The <code>filename\$</code> parameter, if specified, contains the name of the INI file to read. Otherwise, the WIN.INI file is used.</p> <p>The <code>section\$</code> parameter specifies the section that contains the desired variable, such as "windows". Section names are specified without the enclosing brackets.</p> <p>The <code>item\$</code> parameter specifies which item to retrieve the value of.</p>
Example	<pre>sub main() 'Example of ReadIni\$ winshell\$ = ReadIni\$("boot","shell","system.ini") msgbox winshell\$ end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

ReadINISection Statement

Description	Reads all of the item names from a given section of the specified INI file.
Syntax	<code>ReadINISection section\$,items\$() [,filename\$]</code>
Comments	<p>The <code>filename\$</code> parameter, if specified, contains the name of an INI file. Otherwise, the WIN.INI file is used.</p> <p>The <code>section\$</code> parameter specifies the section that contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.</p> <p>The <code>items\$</code> parameter refers to the item name on the left side of the = sign. This parameter must be a single dimension array of strings (see Dim statement). On return, this will contain one array element for each variable in the specified INI section. Use the <code>lbound</code> and <code>ubound</code> functions to determine its size on return.</p>
Example	<pre>sub main() 'Example of ReadIniSection dim iniitems() as string ReadIniSection "boot",iniitems,"system.ini"</pre>


```

ViewPortOpen
print "[Boot] section items from SYSTEM.INI"
print " "
for i% = lbound(iniitems) to ubound(iniitems)
    print iniitems(i%)
next i%
sleep 5000
ViewPortClose
end sub

```

See Also Arrays (Chapter 7)
 Environment Statements and Functions (Chapter 7)

ReDim Statement

Description Redimensions an array, specifying a new upper and lower bound for a given arrays dimensions.

Syntax `ReDim variablename (subscriptRange) [as type],...`

Comments The *variablename* parameter specifies the name of an existing array (previously declared using the `dim` statement).

Caution: The `ReDim` statement deletes any data already in the array.

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).

Arrays can be dynamically dimensioned by first declaring them with no initial size using the `dim` statement:

```
dim a$()
```

Then, using the `redim` statement, the actual size can be specified:

```
redim a$(0 to 8,6 to 10)
```

The number of dimensions of an array cannot be changed once the array has been given dimensions—either by declaring it with initial dimensions using `dim` or by a previous use of `redim`.

The *type* parameter can be used to specify the array element type. The following can be used: `integer`, `long`, `string`.

Example

```
sub main()
    'Example of ReDim
    dim stuff(5) as integer

    msgbox str$(ubound(stuff))
    redim stuff(10)
    msgbox str$(ubound(stuff))
end sub
```

See Also Arrays (Chapter 7)
 Variables and Constants (Chapter 7)

REM Statement

Description Causes the compiler to skip all characters on that line.

Syntax REM text

Example

```
sub main()
    'Example of REM

    REM this is also a comment
end sub
```

See Also Comments (Chapter 6)

RefreshIni Statement

Description Reloads the WIN.INI file from disk, thus refreshing all WIN.INI settings.

Syntax RefreshIni

Comments This forces all hand-edited changes to the WIN.INI file to be activated.

Example

```
sub main()
    'Example of RefreshIni

    RefreshIni
end sub
```

See Also Environment Statements and Functions (Chapter 7)

Reset Statement

Description	Closes all open files, writing out all I/O buffers.
Syntax	<code>Reset</code>
Example	<pre>sub main() 'Example of RESET statement Reset 'close all open files (writes out i/o buffers) end sub</pre>
See Also	File Input and Output (Chapter 7)

RestoreEnv Function

Description	Restores the set of environment variables from the virtual stack for DOS or for Windows saved by the <code>SaveEnv</code> function.
Syntax	<code>RestoreEnv ([mode\$])</code>
Comments	<p>Returns the integer TRUE if the function is successful, otherwise FALSE.</p> <p>Separate stacks are kept for the Windows and DOS environments.</p> <p>The <code>mode\$</code> parameter can be <code>ENV_DOS</code> or <code>ENV_WINDOWS</code>.</p> <p>If <code>mode</code> is unspecified, the default is <code>ENV_WINDOWS</code>.</p>
Example	<pre>sub main() 'Example of SaveEnv and RestoreEnv a% = SaveEnv(ENV_WINDOWS) a% = RestoreEnv(ENV_WINDOWS) end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

Resume Statement

Description	Ends an error handler and continues execution.
--------------------	--

Syntax	<code>Resume {[0] next label}</code>
Comments	<p>The form <code>resume [0]</code> causes execution to continue with the statement that caused the error.</p> <p>The form <code>resume next</code> causes execution to continue with the statement following the statement that caused the error.</p> <p>The form <code>resume label</code> causes execution to continue at the specified label.</p>
Example	<pre> sub main() 'Example of Resume statement on error goto testerror error 101 msgbox "resumed as anticipated" on error goto 0 end testerror: resume next end sub </pre>
See Also	Error Trapping (Chapter 7)

Return Statement

Description	Transfers execution control to the statement following the most recent <code>gosub</code> .
Syntax	<code>Return</code>
Comments	A runtime error results if a <code>return</code> statement is encountered without a corresponding <code>gosub</code> statement.
Example	<pre> sub main() 'Examples of GOSUB and RETURN for x% = 1 to 5 gosub mylabel next x% end mylabel: msgbox "Here we are!" return end sub </pre>
See Also	Flow Control (Chapter 7)

Right\$ Function

Description	Returns the rightmost NumChars characters from a specified string.
Syntax	Right\$(str\$, NumChars\$)
Comments	<p>If NumChars is greater than or equal to the length of the string, then the entire string is returned.</p> <p>If NumChars is 0, then an empty string is returned.</p>
Example	<pre>sub main() 'Example of right\$() dim teststr as string teststr = "This is a test." msgbox teststr teststr = right\$(teststr,7) msgbox teststr end sub</pre>
See Also	Strings (Chapter 7)

Rmdir Statement

Description	Removes the specified directory.
Syntax	Rmdir dir\$
Comments	This command behaves just like the DOS "rd" command.
Example	<pre>sub main() 'Example of Rmdir Rmdir "C:\testdir" end sub</pre>
See Also	File Input and Output (Chapter 7)

Rnd Function

Description	Returns a single-precision random number between 0 and 1.
Syntax	Rnd[(number#)]

Comments If `number#` is omitted, the next random number is returned. Otherwise, the `number#` parameter has the following meaning:

`number# < 0` Always returns the same number
`number# = 0` Returns the last number generated
`number# > 0` Returns the next random number

Example

```
sub main()
    'Example of Rnd function

    ViewPortOpen
    Randomize
    for i% = 1 to 10
        print Rnd(1)
    next i%
    sleep 5000
    ViewPortClose
end sub
```

See Also Math Statements and Functions (Chapter 7)

RTrim\$ Function

Description Returns the string with the trailing spaces removed.

Syntax `RTrim$(str$)`

Example

```
sub main()
    'Example of RTrim$()
    dim teststring as string

    teststring = "    testing    "
    msgbox "*" + teststring + "*"
    msgbox "*" + rtrim$(teststring) + "*"
end sub
```

See Also Strings (Chapter 7)

SaveEnv Function

Description Pushes the current environment variable set onto a virtual stack so that they can later be restored by the `RestoreEnv` function.

Syntax `SaveEnv ([mode$])`

Comments Returns the integer `TRUE` if the function is successful, otherwise `false`.

Separate stacks are kept for the Windows and DOS environments.

The `mode$` parameter can be `ENV_DOS` or `ENV_WINDOWS`.

If `mode` is unspecified, the default is `ENV_WINDOWS`.

Example

```
sub main()
    'Example of SaveEnv and RestoreEnv

    a% = SaveEnv(ENV_WINDOWS)
    a% = RestoreEnv(ENV_WINDOWS)
end sub
```

See Also Environment Statements and Functions (Chapter 7)

SaveFileName\$ Function

Description Displays the common file save dialog box (from `COMMDLG.DLL`), allowing the user to select a file. If the file already exists, the user is prompted to overwrite it.

Syntax `SaveFileName$(title$ [,extensions$])`

Returns Returns a full DOS pathname of the file that the user selected.

Comments The `title$` parameter specifies the title that appears on the dialog box's caption.

The `extensions$` parameter specifies the available file types. This string should be in the following format:

`"type:ext[,ext][;type:ext[,ext]]..."`

where `ext` is a valid file extension, like `*.BAT` or `*.??`, and `type` is a string that identifies this type to the user.

By default, the first extension in appearing within `extensions` is used.

Example

```
sub main()
    'Example of SaveFileName$

    selffile$ = SaveFileName$("Open File", "All Files:*.bmp, *.wmf; ←
        Bitmaps:*.bmp;Metafiles:*.wmf")
    msgbox "Selected File = "+selffile$
end sub
```

See Also

Dialog Display (Chapter 7)

Second Function

Description

Returns an integer representing the second of the day encoded in the specified `serial#` parameter. The value returned is between 0 and 59 inclusive.

Syntax

`second(serial#)`

Comments

You can obtain the value for the `serial#` parameter by using the `TimeSerial` or `TimeValue` command.

Example

```
sub main()
    'Example of second() function
    dim dt as double

    dt = Now
    msgbox str$(second(dt))....'current second
end sub
```

See Also

Date and Time Functions (Chapter 7)

Seek Statement and Function

Description

The `seek` function returns the file pointer for a given file. The `seek` statement sets the file pointer.

Function Syntax

`seek(filenumber%)`

Statement Syntax

`seek [#] filenumber%,position&`

Comments

The `filenumber` parameter is a number that is used by DCL to refer to the open file—the number passed to the `open` statement.

See Also File Input and Output (Chapter 7)

Select...Case Statement

Description Executes a block of DCL statements depending on the value of a given expression.

Syntax

```
select case testexpression
[case expressionlist
    [statement_block]]
[case expressionlist
    [statement_block]]
:
[case else
    [statement_block]]
end select
```

Comments The `select case` statement uses the following arguments:

<i>testexpression</i>	Any numeric or string expression
<i>statement_block</i>	Any group of DCL statements
<i>expressionlist</i>	Any of the following:
	<i>expression</i> [, <i>expression</i>]...
	<i>expression</i> to <i>expression</i>
	is <i>relational_operator</i> <i>expression</i>

If the *testexpression* matches any of the expressions contained in *expressionlist*, the accompanying block of DCL statements is executed.

The resultant type of *expression* 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 executed.

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

```
sub main()
    'Example of SELECT-CASE statement
```

```

i% = 1
select case i%
  case 1
    msgbox "i% is 1"
  case 2
    msgbox "i% is 2"
end select
end sub

```

See Also Flow Control (Chapter 7)

SelectBox Function

Description	Displays a dialog box containing a listbox of strings. If the user selects an item, the index of that item is returned.
Syntax	<code>SelectBox(title\$,prompt\$,items\$())</code>
Returns	Returns an integer representing the index of the item that the user selected. The first item is 0. The value -1 is returned if the user selects Cancel.
Comments	<p>The <code>items\$</code> parameter must be a single-dimension array of strings, otherwise a runtime error is generated.</p> <p>The <code>title\$</code> parameter specifies the name that appears in the dialog box's caption. The <code>prompt\$</code> parameter specifies the name that appears immediately above the listbox containing the array items.</p> <p>The dialog box uses the 8 point Helvetica font.</p>
Example	<pre> sub main() 'example of SelectBox function dim alist() as string AppList alist result% = SelectBox("Application List","Pick One",alist) msgbox "You selected "+alist(result%) end sub </pre>
See Also	Dialog Display (Chapter 7)

SelectButton Statement

Description	Simulates a mouse click on a button, given the button's name or ID.
Syntax 1	<code>SelectButton ButtonName\$</code>
Syntax 2	<code>SelectButton ButtonID%</code>
Comments	<p>You can reference the button by <code>name\$</code> (caption) or <code>id%</code>.</p> <p>A runtime error is generated if a button with the given <code>name\$</code> or <code>id%</code> cannot be found in the active window.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

SelectComboboxItem Statement

Description	Selects an item from a combobox, given the name or ID of the combobox and the name or line number of the item.
Syntax	<code>SelectComboboxItem name\$ id%,ItemName\$ ItemNumber% [,isDoubleClick%]</code>
Comments	<p>The combobox can be referenced either by <code>name\$</code> or by <code>id%</code>. The <code>name\$</code> parameter specifies the text that appears in the static control that immediately precedes the combobox control in the window list (or dialog template).</p> <p>A runtime error is generated if a combobox with the given <code>name\$</code> or <code>id%</code> cannot be found within the active window, or if an item with the given name or line number cannot be found within that combobox.</p> <p>If the second parameter is either 0 or an empty string (""), all selections will be removed from the combobox.</p> <p>The optional <code>isDoubleClick%</code> parameter specifies if this item is to be selected via a double click or single click. If not specified, then the item is selected using a single click.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

SelectListBoxItem Statement

Description	This statement selects an item from a list box, given the name or ID of the listbox and the name or line number of the item.
Syntax	<pre>SelectListBoxItem name\$ id%,ItemName\$ ItemNumber%← [,isDoubleClick%]</pre>
Comments	<p>The listbox must exist within the current window or dialog, otherwise a runtime error will be generated.</p> <p>The listbox can be referenced either by name\$ or by id%. The name\$ parameter specifies the text that appears in the static control that immediately precedes the listbox control in the window list (or dialog template). A runtime error is generated if a listbox with that name cannot be found within the active window.</p> <p>If the second parameter is 0 or an empty string (""), then all selections are removed from the listbox. For multi-select listboxes, <code>SelectListBoxItem</code> will select additional items (i.e., it will not remove the selection from the currently selected items).</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

SendKeys Statement

Description	Sends the specified keys to the active application, optionally waiting for the keys to be processed before continuing.
Syntax	<pre>SendKeys KeyString\$ [,wait%]</pre>
Comments	<p>The format for <code>KeyString\$</code> is the same as that used for <code>DoKeys</code>.</p> <p>If <code>wait</code> is TRUE (or not specified), the statement waits for the keys to be completely processed before continuing. Otherwise, execution continues immediately.</p>
Example	<pre>sub main() 'Example of SendKeys dim alttab as string alttab = "%{TAB}" msgbox "Press OK to do first Alt-Tab" SendKeys alttab,TRUE msgbox "Press OK to Alt-Tab back to original application" SendKeys alttab,FALSE end sub</pre>

See Also Keyboard Manipulation (Chapter 7)

SetAttr Statement

Description Changes the attributes of the specified file to the given attribute.

Syntax `SetAttr filename$, attribute%`

Comments A runtime error results if the file cannot be found.

The `attribute%` parameter contains the sum of the following attributes.

Constant	Value	Description
ATTR_NORMAL	0	Read-only, archive, subdirectory, and files with no attributes
ATTR_READONLY	1	Read-only files
ATTR_HIDDEN	2	Hidden files
ATTR_SYSTEM	4	System files
ATTR_VOLUME	8	Volume label
ATTR_DIRECTORY	16	MS-DOS directories
ATTR_ARCHIVE	32	Files changed since last backup
ATTR_NONE	64	Files with no attributes

These attributes are the same as those used by DOS.

Example

```
sub main()
    'Example of SetAttr

    SetAttr "C:\AUTOEXEC.BAT", 0
    'set file attribute to NORMAL
end sub
```

See Also File Input and Output (Chapter 7)

SetCheckbox Statement

Description Sets the state of the checkbox with the given name or ID.

Syntax 1 `SetCheckbox name$, state%`

Syntax 2 `SetCheckbox id%, state%`

Comments	<p>The checkbox can be specified either by its <code>name\$</code> or using its <code>id%</code>. The <code>name\$</code> parameter is the text of the checkbox label.</p> <p>If <code>state</code> is 1, the box is checked. If <code>state</code> is 0, the check is removed. If <code>state</code> is 2, then the box is grayed (only applicable for 3-state check boxes).</p> <p>A runtime error is generated if a check box with the specified name cannot be found in the active window.</p> <p>This statement has the side-effect of setting the focus to the given checkbox.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

SetEditText Statement

Description	Sets the content of an edit control, given its name or ID.
Syntax 1	<code>SetEditText name\$,content\$</code>
Syntax 2	<code>SetEditText id%,content\$</code>
Comments	<p>The <code>name\$</code> parameter specifies the text that appears within the static control that immediately precedes the edit control in the window list (or dialog template). Alternatively, the <code>id%</code> of the edit box can be specified.</p> <p>This statement has the side-effect of setting the focus to the given edit control.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

SetEnv Function

Description	Sets the specified environment variable to the given value for DOS, Windows, or Both.
Syntax	<code>SetEnv(var\$, value\$[, mode\$])</code>
Comments	<p>When you are running Windows, there are two environments. One environment is used for Windows applications and one for DOS applications.</p> <p>To control which environment you set variables for, use the <code>mode\$</code> parameter. The <code>mode\$</code> parameter can be:</p> <ul style="list-style-type: none"> -- <code>ENV_DOS</code>, which sets environment variables for DOS applications -- <code>ENV_WINDOWS</code>, which sets environment variables for Windows. -- <code>ENV_BOTH</code>, which sets environment variables for both DOS and Windows. <p>If it is an invalid value, or not set, the default <code>ENV_WINDOWS</code> is used.</p> <p>Changes made to Windows environment variables are available to all Windows applications—including those that have already been launched.</p> <p>The function returns <code>TRUE</code> if successful, or <code>FALSE</code> otherwise. If the function fails, the environment may be full.</p>
Example	<pre>sub main() 'Example of SetEnv and GetEnv\$() tmp\$ = GetEnv\$("TMP",ENV_WINDOWS) tv\$ = AskBox\$("New Value For TMP Environment Variable:") a% = SetEnv("TMP",tv\$,ENV_WINDOWS) msgbox "New value for TMP is "+GetEnv\$("TMP",ENV_WINDOWS) 'restore old value a% = SetEnv("TMP",tmp\$,ENV_WINDOWS) end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

SetIcon Statement

Description	Specifies which icon to show when <code>ShowIcon</code> is called.
Syntax	<code>SetIcon iconfile\$ [,icon%]</code>

Comments	<p>The <code>iconfile\$</code> specifies an .EXE, .DLL, or .ICO file. <code>icon%</code> is the zero-based index into the list of icons the file contains. If unspecified, <code>icon%</code> is assumed to be 0.</p> <p>If the icon cannot be found or is not a valid format, or if the value of <code>icon%</code> exceeds the number of icons in the file, a runtime error is generated.</p> <p>This function has no affect on scripts run from the editor or debugger.</p>
See Also	Icons (Chapter 7)

SetIconTitle Statement

Description	Sets the title to be displayed under the icon (if shown) for a compiled script.
Syntax	<code>SetIconTitle title\$</code>
Comments	This function has no affect on scripts run from the editor or debugger.
See Also	Icons (Chapter 7)

SetOption Statement

Description	Clicks on the specified option button.
Syntax 1	<code>SetOption name\$</code>
Syntax 2	<code>SetOption id%</code>
Comments	<p>The option button can be referenced either by its <code>name\$</code> (caption) or its <code>id%</code>.</p> <p>A runtime error is generated if the option button cannot be found within the active window.</p>
Example	Dialog Examples
See Also	Dialog Manipulation (Chapter 7)

Sgn Function

Description Returns an integer representing a number is less than, greater than, or equal to zero.

Syntax `sgn (number)`

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.

Example

```
sub main()
    'Example of SGN() function

    i% = -2
    msgbox str$(sgn(i%))
    i% = 0
    msgbox str$(sgn(i%))
    i% = 2
    msgbox str$(sgn(i%))
end sub
```

See Also Math Statements and Functions (Chapter 7)

Shell Function

Description Runs the program (with any specified parameters) contained in `command$`.

Syntax `shell (command$ [,WindowStyle%])`

Returns If successful, the function returns an integer representing the task ID. For other return codes, see "Errors" below.

Comments The optional `WindowStyle%` parameter specifies 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
- 3 Maximized with focus
- 4 Normal window without focus
- 7 Minimized without focus

An error is generated if unsuccessful. A return value less than or equal to 32 specifies an error.

Errors

This function returns the value 31 if there is no association for the specified file type or if there is no association for the specified action within the file type. The other possible error values are as follows:

- 0 System was out of memory, executable file was corrupt, or relocations were invalid.
- 2 File was not found.
- 3 Path was not found.
- 5 Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
- 6 Library required separate data segments for each task.
- 8 There was insufficient memory to start the application.
- 10 Windows version was incorrect.
- 11 Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
- 12 Application was designed for a different operating system.
- 13 Application was designed for MS-DOS 4.0.
- 14 Type of executable file was unknown.
- 15 Attempt was made to load a real-mode application (developed for an earlier version of Windows).
- 16 Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
- 19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
- 20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
- 21 Application requires Microsoft Windows 32-bit extensions.

Example

```
sub main()
    'Example of Shell function

    taskid% = Shell("notepad.exe",1)
end sub
```

See Also

Flow Control (Chapter 7)

ShowIcon Statement

Description

Displays an icon on the desktop while the script is running.

Syntax

```
ShowIcon [show%]
```

Comments	<p>Normally, an icon does not show up on the desktop while a compiled script is running.</p> <p>If <code>show%</code> is set to <code>TRUE</code> (the default value), an icon will appear on the desktop when the script is run. The purpose of the icon is to allow the user to stop the script if the user has been permitted this authority (through the <code>EnableStopScript</code> command).</p> <p>This function has no affect on scripts run from the editor or debugger.</p>
See Also	Icons (Chapter 7)

Sin Function

Description	Returns a double-precision number representing the sine of a given angle.
Syntax	<code>sin (angle#)</code>
Comments	The <code>angle</code> parameter is given in radians.
Example	<pre>sub main() 'Example of Sin() function result# = sin(0) msgbox str\$(result#) result# = sin(1) msgbox str\$(result#) end sub</pre>
See Also	Math Statements and Functions (Chapter 7)

Sleep Statement

Description	Causes the script to pause for a specified number of milliseconds.
Syntax	<code>sleep milliseconds&</code>
Comments	While paused, other applications can execute.
Example	<pre>sub main() 'Example of Sleep command msgbox "Press OK to sleep 5 seconds" sleep 5000 end sub</pre>

```

Example      sub main()
                  'Example of SleepUntil

                  t$ = time$()                'get current time
                  t$ = left$(t$,5)             'get hours and
                                                  'minutes only
                  m$ = right$(t$,2)            'get minutes value
                  h$ = left$(t$,2)             'get hour value

```

```

x = val(m$)      'get the value of the minutes
y = val(h$)      'get value of hour string
x = x + 2        'increment minutes
                 'to wait 2 minutes

if x > 59 then
    y = y + 1
    x = x - 59
end if
h$ = str$(y)      'convert hours
                 'back to string

h$ = right$(h$,len(h$)-1)
    'get rid of leading blank from conversion
if len(h$) = 1 then h$ = "0" + h$
m$ = str$(x)      'convert minutes
                 'back to string likewise
m$ = right$(m$,len(m$)-1)
if len(m$) = 1 then m$ = "0" + m$
t$ = h$+":"+m$    'reconstruct time
                 'value

retval% = SleepUntil(t$,TRUE, ←
    "Press CANCEL to quit.", ←
    "Sleeping Until "+t$,TRUE,TRUE)
end sub

```

See Also Flow Control (Chapter 7)

Snapshot Statement

Description Takes a snapshot of a particular section of the screen and saves it to the clipboard.

Syntax snapshot [spec%]

Comments The *spec* parameter specifies the screen area as follows:

- 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 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

```
sub main()
    'Example of SnapShot statement

    SnapShot 4 'active window
    taskid% = Shell("pbrush.exe")
    DoKeys "+{INSERT}"           'paste clipboard
                                'into paintbrush
end sub
```

See Also Clipboard Manipulation (Chapter 7)

Space\$ Function

Description Returns a string containing the specified number of spaces.

Syntax space\$ (NumSpaces%)

Comments NumSpaces must be between 0 and 32767.

Example

```
sub main()
    'Example of Space$() function

    stuff$ = null
    msgbox "*" + stuff$ + "*"
    stuff$ = space$(10)
    msgbox "*" + stuff$ + "*"
end sub
```

See Also Strings (Chapter 7)

Sqr Function

Description Returns a double-precision number representing the square root of a given value.

Syntax sqr (number#)

Comments The number parameter must be greater than or equal to 0.

Example

```
sub main()
    'Example of SQR function

    msgbox str$(sqr(100))
end sub
```

See Also Math Statements and Functions (Chapter 7)

Stop Statement

Description	Stops execution of a script.
Syntax	<code>stop</code>
Comments	This command terminates execution of the current script and displays the message: "Stopped at line X", where X is the line number containing the stop statement. All open files are closed. All open DDE channels are closed.
Example	<pre>sub main() 'Example of Stop statement Stop end sub</pre>
See Also	Flow Control (Chapter 7)

Str\$ Function

Description	Returns a string representation of the given number.
Syntax 1	<code>str\$(number%)</code>
Syntax 2	<code>str\$(number&)</code>
Syntax 3	<code>str\$(number!)</code>
Syntax 4	<code>str\$(number#)</code>
Comments	<p>If <code>number</code> is negative, then the returned string will contain a leading minus sign.</p> <p>If <code>number</code> is positive, then the returned string will contain a leading space.</p> <p>Singles are printed using only 7 significant digits. Doubles are printed using 15-16 significant digits.</p>
Example	<pre>sub main() 'Example of str\$() function i% = 3 s\$ = str\$(i%) msgbox s\$ end sub</pre>

See Also Conversions (Chapter 7)
 Strings (Chapter 7)

StrComp Function

Description	Compares two strings.
Syntax	<code>StrComp(string1\$,string2\$ [,compare%])</code>
Returns	Returns an integer value indicating the result of comparing the two string arguments: 0 <code>string1\$ = string2</code> 1 <code>string1\$ > string2\$</code> -1 <code>string1\$ < string2\$</code>
Comments	<p>The <code>StrComp</code> function compares two strings and returns an integer indicating the result of the comparison. The comparison can be either case-sensitive or case-insensitive, depending on the value of the optional <code>compare%</code> parameter:</p> <p>0 Case-sensitive comparison 1 Case-insensitive comparison</p> <p>If <code>compare%</code> is not specified, then the comparison is case-sensitive (0).</p>
Example	<pre>sub main() 'Example of StrComp function 'Case-sensitive comparison teststr\$ = "This is a test string" result% = StrComp(teststr\$,"THIS IS A TEST STRING",0) select case result% case -1 msgbox "teststr\$ is less than the compare string" case 0 msgbox "teststr\$ is equal to the compare string" case 1 msgbox "teststr\$ is greater than compare string" end select 'Case-insensitive comparison result% = StrComp(teststr\$,"THIS IS A TEST STRING",1) select case result% case -1 msgbox "teststr\$ is less than the compare string"</pre>


```

        case 0
            msgbox "teststr$ is equal to the compare string"
        case 1
            msgbox "teststr$ is greater than compare string"
    end select
end sub

```

See Also Strings (Chapter 7)

String\$ Function

Description Returns a string of `number%` length consisting of a repetition of the specified filler character.

Syntax 1 `string$(number%,CharCode%)`

Syntax 2 `string$(number%,str$)`

Comments If `CharCode` is specified, the character with this ASCII value is used as the filler character.
If `str` is specified, then the first character of this string is used as the filler character.

Example

```

sub main()
    'Example of String$() function

    s$ = String$(10,65)
    msgbox s$
    s$ = String$(10,"B")
    msgbox s$
end sub

```

See Also Strings (Chapter 7)

StringSort Function

Description Sorts a one-dimensional array of strings in ascending order.

Syntax `StringSort list$()`

Comments If an array of more than one dimension is specified, a runtime error is generated.

See Also Strings (Chapter 7)

Sub...End Sub Statement

Description	Declares a subroutine.
Syntax	<pre>sub name[(parameter [as type]...)] end sub</pre>
Comments	<p>Parameters are passed to a subroutine by reference, meaning that any modification to a passed parameter changes that variable in the caller. To avoid this, simply enclose variable names in parentheses, as in the following example function calls:</p> <pre>UserSub 10,12,(j)</pre> <p>If a subroutine is not to receive a parameter by reference, the optional <code>byval</code> keyword can be used:</p> <pre>sub Test byval FileName as string end sub</pre> <p>A subroutine terminates when one of the following statements is encountered:</p> <pre>end sub exit sub</pre> <p>The name of the subroutine must follow DCL naming conventions. It cannot include type declaration characters.</p> <p>Subroutines can be recursive.</p>
See Also	Procedure Statements (Chapter 7)

SystemFreeMemory Function

Description	Returns a long integer representing the number of bytes of free memory.
Syntax	<code>SystemFreeMemory()</code>
Example	<pre>sub main() 'Example of SystemFreeMemory msgbox "Free Memory ="&str\$(SystemFreeMemory) end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

SystemFreeResources Function

Description	Returns an integer representing the percentage of free system resources.
Syntax	<code>SystemFreeResources()</code>
Comments	The returned value is between 0 and 100.
Example	<pre>sub main() 'Example of SystemFreeResources msgbox "Free Resources ="&str\$(SystemFreeResources)&"%" end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

SystemMouseTrails Statement

Description	Turns on or off mouse trails.
Syntax	<code>SystemMouseTrails state%</code>
Comments	The setting is saved in the INI file permanently. This option is only available under Windows 3.1.
Example	<pre>sub main() 'Example of SystemMouseTrails SystemMouseTrails TRUE msgbox "Move the Mouse Around Now" SystemMouseTrails FALSE end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

SystemRestart Statement

Description	Restarts Windows, much like the Window Setup program.
Syntax	<code>SystemRestart</code>

Example

```
sub main()
    'Example of SystemRestart
    'WARNING:  this will restart Windows if you run it

    SystemRestart
end sub
```

See Also Environment Statements and Functions (Chapter 7)

SystemTotalMemory Function

Description Returns a long integer representing the total available free memory in Windows in bytes.

Syntax SystemTotalMemory()

Example

```
sub main()
    'Example of SystemTotalMemory

    msgbox "Total System Memory =" + &
        str$(SystemTotalMemory)
end sub
```

See Also Environment Statements and Functions (Chapter 7)

SystemWindowsDirectory\$ Function

Description Returns the directory where Windows is stored, such as "C:\WINDOWS".

Syntax SystemWindowsDirectory\$()

Example

```
sub main()
    'Example of SystemWindowsDirectory$

    msgbox SystemWindowsDirectory$
end sub
```

See Also Environment Statements and Functions (Chapter 7)

SystemWindowsVersion\$ Function

Description	Returns the Windows version, such as "3.0" or "3.1".
Syntax	<code>SystemWindowsVersion\$()</code>
Example	<pre>sub main() 'Example of SystemWindowsVersion\$ msgbox SystemWindowsVersion\$ end sub</pre>
See Also	Environment Statements and Functions (Chapter 7)

Tan Function

Description Returns a double-precision number representing the tangent of the specified angle.

Syntax `tan(angle#)`

Comments The `angle` parameter is given in radians.

Example

```
sub main()
    'Example of Tan() function

    i# = Tan(1)
    msgbox str$(i#)
end sub
```

See Also Math Statements and Functions (Chapter 7)

Text Statement

Description Defines a text control in a dialog template.

Syntax `Text x%,y%,width%,height%,title$`

Comments The purpose of `Text` controls is simply to display text.

The `title$` parameter will be truncated if the width of the control is insufficient to hold the entire content. The `title$` parameter may contain an ampersand character to denote an underlined accelerator, such as "&Font" for Font.

This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).

The `x`, `y`, `width`, `height` parameters are specified in dialog coordinates. The `x`, `y` position is relative to the upper left corner of the dialog box.

Example Dialog Examples

See Also Dialog Creation (Chapter 7)

TextBox Statement

Description	Defines a text-entry field that appears within a dialog box template.
Syntax	<code>TextBox x%,y%,width%,height%,.Field</code>
Comments	<p>This statement can only appear within a dialog box template definition (BEGIN DIALOG...END DIALOG).</p> <p>The <code>x</code>, <code>y</code>, <code>width</code>, <code>height</code> parameters are specified in dialog coordinates. The <code>x</code>, <code>y</code> position is relative to the upper left corner of the dialog box.</p> <p>When the dialog box is created, the content of the <code>.Field</code> is used to set the initial content of the textbox. When the <code>Dialog</code> statement returns, the <code>.Field</code> is used to determine the final content of the textbox. The <code>.Field</code> always contains a string.</p>
Example	Dialog Examples
See Also	Dialog Creation (Chapter 7)

Time\$ Statement and Function

Description	The <code>time\$</code> assignment statement sets the system time; the <code>time\$</code> function returns the system time.
Assignment Syntax	<code>time\$ = newtime\$</code>
Comments	<p>This statement sets the system time to the time contained in the specified string.</p> <p>The format for <code>newtime\$</code> must be one of the following:</p> <p>HH</p> <p>HH:MM</p> <p>HH:MM:SS</p> <p>A 24 hour clock is used.</p>
Example	<pre>sub main() 'Example of Time command time\$ = "21:30:40" end sub</pre>

Function Syntax	<code>time\$()</code>
Returns	Returns the system time as an 8 character string.
Comments	The format of the returned string is HH:MM:SS.

Example

```
sub main()  
    'Example of Time$() function  
  
    msgbox Time$()  
end sub
```

See Also Date and Time Functions (Chapter 7)

Timer Function

Description Returns a long integer representing the number of seconds that have occurred since midnight.

Syntax timer

Example

```
sub main()  
    'Example of Timer function  
    dim tval as long  
  
    tval = Timer  
    msgbox str$(tval)  
end sub
```

See Also Date and Time Functions (Chapter 7)

TimeSerial Function

Description Returns a double-precision number representing the given time with today's date. The number is returned in days where Dec 30, 1899 is 0.

Syntax TimeSerial(hour%,minute%,second%)

Example

```
sub main()  
    'Example of TimeSerial function  
    dim tser as double  
  
    tser = TimeSerial(21,40,44)  
end sub
```

See Also Date and Time Functions (Chapter 7)

TimeValue Function

Description	Returns a double-precision number representing the time contained in the specified string argument.
Syntax	<code>TimeValue(time_string\$)</code>
Comments	<p>This function interprets the passed <code>time_string\$</code> parameter looking for a valid time specification. Time specifications vary depending on the international settings contained in the INTL section of the WIN.INI file.</p> <p>The <code>time_string\$</code> parameter can contain valid time items separated by time separators such as colon (:) or period (.). The time items must follow the ordering determined by the current time format settings in use by Windows.</p> <p>Time strings can contain an optional date specification, but this is not used in the formation of the returned value.</p> <p>If a particular time item is missing, then the missing time items are set to zero. For example, the string "10 pm" would be interpreted as "22:00:00".</p>
Example	<pre>sub main() 'Example of TimeValue function dim tval as double tval = TimeValue("21:40:44") end sub</pre>
See Also	Date and Time Functions (Chapter 7)

Trim\$ Function

Description	Removes leading and trailing spaces.
Syntax	<code>trim\$(str\$)</code>
Returns	Returns a copy of the passed string expression (<code>str\$</code>) with leading and trailing spaces removed.
Example	<pre>sub main() 'Example of Trim\$() dim teststring as string teststring = " testing " msgbox "*" + teststring + "*" msgbox "*" + trim\$(teststring) + "*" end sub</pre>

See Also Strings (Chapter 7)

TRUE

Description	Constant.
Returns	-1
Comments	Used in conditionals and boolean expressions.

TYPE_DOS

Description	Constant.
Returns	1
Comment	Used with the AppType function to indicate a DOS application.

TYPE_WINDOWS

Description	Constant.
Syntax	2
Comment	Used with the AppType function to indicate a Windows application.

UBound Function

Description	Returns an integer representing the upper bound of the specified dimension of the specified array variable.
Syntax	<code>ubound(ArrayVariable() [, dimension%])</code>
Comments	The first dimension (1) is assumed if dimension is not specified.
Example	<pre>sub main() 'Examples of UBOUND dim ial(8) as integer</pre>

```

        dim ia2(65 to 70) as integer

        msgbox str$(ubound(ia1))
        msgbox str$(ubound(ia2))
    end sub

```

See Also Arrays (Chapter 7)

UCase\$ Function

Description Returns the uppercase equivalent of the specified string.

Syntax `ucase$(str$)`

Example

```

sub main()
    'Example of UCase$
    dim teststr as string

    teststr = "this is a test"
    msgbox teststr
    teststr = ucase$(teststr)
    msgbox teststr
end sub

```

See Also Strings (Chapter 7)

Val Function

Description Converts a given string expression to a double-precision number.

Syntax `val(number$)`

Comments The `number$` parameter can contain any of the following:

- Leading minus sign (for non hex or octal numbers only)
- Hexadecimal number in the format: `&H<hex digits>`
- Octal number in the format: `&O<octal digits>`
- Floating point number, which can contain a decimal point and optional exponent

Spaces, tabs, and linefeeds are ignored.

If `number$` does not contain a number, 0 is returned.

The `val()` function continues to read characters from the string up to the first non-numeric 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 1

```
sub main()
    'Example of Val function

    teststr$ = "123.3"
    tval# = val(teststr$)
    msgbox str$(tval)
end sub
```

Example 2

The following table shows valid strings and their numeric equivalent:

"1 2 3"	123
"12.3"	12.3
"&HFFFF"	-1
"&O77"	64
" 12.345E-02"	.12345

See Also

Conversions (Chapter 7)
Strings (Chapter 7)

ViewportClear Statement

Description Clears the open viewport window.

Syntax `ViewportClear`

Comments The statement has no effect if no viewport is opened.

Example

```
sub main()
    'Example of ViewPort commands

    'Create a veiwport window
    ViewPortOpen "My ViewPort"
    For i% = 1 to 20
        Print i%
    Next i%
    msgbox "Press OK to clear the viewport."
    ViewPortClear
    msgbox "Press OK to close the viewport."
```

```

        ViewPortClose
    end sub

```

See Also Viewport Window Manipulation (Chapter 7)

ViewportClose Statement

Description Closes an open viewport window.

Syntax ViewportClose

Example

```

sub main()
    'Example of ViewPort commands

    'Create a veiwport window
    ViewPortOpen "My ViewPort"
    For i% = 1 to 20
        Print i%
    Next i%
    msgbox "Press OK to clear the viewport."
    ViewPortClear
    msgbox "Press OK to close the viewport."
    ViewPortClose
end sub

```

See Also Viewport Window Manipulation (Chapter 7)

ViewportOpen Statement

Description Opens a viewport window.

Syntax ViewportOpen [title\$ [,x%,y% [,width%,height%]]]

Comments The optional `title$` parameter specifies the text to appear in the viewport's caption. The `x` and `y` parameters specify an optional initial position in twips, and the optional `width` and `height` parameters specify an optional initial width and height for the viewport window.

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 DCL 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	Scroll up by one line
Down	Scroll down by one line
Home	Scroll to the first line in the viewport window
End	Scroll to the last line in the viewport window
PgUp	Scroll the viewport window down by one page
PgUp	Scroll the viewport window up by one page
Ctrl+PgUp	Scroll the viewport window left by one page
Ctrl+PgDn	Scroll the viewport window right by one page

Only 1 viewport window can be open at any one time. Any scripts with `print` statements will output information into the same viewport window.

Example

```
sub main()
  'Example of ViewPort commands

  'Create a viewport window
  ViewPortOpen "My ViewPort"
  For i% = 1 to 20
    Print i%
  Next i%
  msgbox "Press OK to clear the viewport."
  ViewPortClear
  msgbox "Press OK to close the viewport."
  ViewPortClose
end sub
```

See Also Viewport Window Manipulation (Chapter 7)

VK_LBUTTON

Description Constant used with the `QueMouse . . .` commands to represent the left button.

Value 1

See Also Mouse Events

VK_RBUTTON

Description	Constant used with the <code>QueMouse . . .</code> commands to represent the right button.
Value	2
See Also	Mouse Events

VLine Statement

Description	Scrolls the window with the focus up or down by the specified number of lines.
Syntax	<code>VLine [lines%]</code>
Comments	If the <code>lines</code> parameter is omitted, then the window is scrolled down by 1 line.
Example	<pre>sub main() 'Examples of VLINE ViewPortOpen ViewPortClear for i% = 1 to 50 Print "Here is some test data." next i% VLine 50 sleep 2000 VLine -50 ViewPortClose end sub</pre>
See Also	Window Manipulation (Chapter 7)

VPage Statement

Description	Scrolls the window with the focus up or down by the specified number of pages.
Syntax	<code>VPage [pages%]</code>
Comments	If the <code>pages</code> parameter is omitted, then the window is scrolled down by 1 page.
Example	<pre>sub main() 'Examples of VPage</pre>

```

        ViewPortOpen
        ViewPortClear
        for i% = 1 to 50
            Print i%
        next i%
        VPage 1
        sleep 2000
        VPage -1
        ViewPortClose
    end sub

```

See Also Window Manipulation (Chapter 7)

VScroll Statement

Description Sets the thumb mark on the vertical scroll bar attached to the current window.

Syntax VScroll percentage%

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 is positioned in the middle of the scroll bar.

Example

```

sub main()
    'Example of VSCROLL

    ViewPortOpen
    ViewPortClear
    for i% = 1 to 50
        Print "Test data for viewport scroll test."
    next i%
    sleep 2000
    VScroll 50 '50 percent scroll
    sleep 2000
    VScroll 1   'no scroll
    sleep 2000
    ViewPortClose
end sub

```

See Also Window Manipulation (Chapter 7)

WaitForTaskCompletion Function

Description	Waits until the task specified by <code>taskid%</code> is exited.
Syntax	<code>WaitForTaskCompletion taskid%</code>
Comments	The <code>taskid%</code> is the return value from the <code>Shell</code> statement. If <code>taskid%</code> is not currently running, control returns immediately.
Example	<pre>sub main() 'Example of WaitForTaskCompletion taskid% = Shell("notepad",1) 'The next statement pauses until Notepad 'is shut down WaitForTaskCompletion taskid% msgbox "All done." end sub</pre>
See Also	Flow Control (Chapter 7)

Weekday Function

Description	Returns an integer representing the day of the week of the date encoded in the specified <code>serial</code> parameter. The value returned is between 1 and 7 inclusive where 1 is Sunday.
Syntax	<code>weekday(serial#)</code>
Comments	You can obtain the value for the <code>serial#</code> parameter by using the <code>DateSerial</code> or <code>DateValue</code> command.
Example	<pre>sub main() 'Example of Weekday() wday% = weekday(Now) select case wday% case 1 msgbox "Today Is Sunday" case 2 msgbox "Today Is Monday" case 3 msgbox "Today Is Tuesday" case 4 msgbox "Today Is Wednesday" case 5</pre>

```

        msgbox "Today Is Thursday"
    case 6
        msgbox "Today Is Friday"
    case 7
        msgbox "Today Is Saturday"
    end select
end sub

```

See Also Date and Time Functions (Chapter 7)

While...Wend Statement

Description Repeats a statement or group of statements while a condition is TRUE.

Syntax while <condition>
 [<statement>]
 wend

Example sub main()
 'Example of WHILE-WEND

 i% = 4
 while i% > 1
 msgbox "not yet"
 i% = i% -1
 wend
 msgbox str\$(i%)
 end sub

See Also Flow Control (Chapter 7)

WinActivate Statement

Description Activates the window with the given name or window handle.

Syntax 1 WinActivate window_name\$

Syntax 2 WinActivate hWnd%

Comments The window_name\$ parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. 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 the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

If the `hWnd%` parameter is specified rather than the `window_name$` parameter, then focus is set immediately to the window with that handle.

Windows without captions cannot be activated using this command.

Example

```
sub main()
    'Example of WinActivate

    appn$ = AppFind$("Notepad")
    WinActivate appn$
end sub
```

See Also

Window Manipulation (Chapter 7)

WinClose Statement

Description

Closes the given window.

Syntax

```
WinClose [window_name$]
```

Comments

If no `window_name$` parameter is specified, the window with the focus is closed.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. 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 the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the `AppClose` command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example

```
sub main()
    'Example of WinClose

    appn$ = AppFind$("Notepad")
    WinClose appn$
end sub
```

See Also Window Manipulation (Chapter 7)

WinFind Function

Description Returns an integer representing a handle to the specified window.

Syntax WinFind(name\$)

Comments The name\$ is specified using the same format as that used by the WinActivate statement.

Example

```
sub main()
    'Example of WinFind

    appn$ = AppFind$("Notepad")
    hWnd% = WinFind(appn$)
    msgbox str$(hWnd%)
end sub
```

See Also Window Manipulation (Chapter 7)

WinList Function

Description Fills the passed array with the handles to all the top level windows.

Syntax WinList hWnd\$()

Comments After calling this function, use the lbound() and ubound() functions to determine the new size of the array.

Example

```
sub main()
    'Example of WinList
    dim hWindows() as integer

    WinList hWindows
    for i% = lbound(hWindows) to ubound(hWindows)
```

```

        msgbox str$(hWindows(i%))
    next i%
end sub

```

See Also Window Manipulation (Chapter 7)

WinMaximize Statement

Description This command maximizes the specified window.

Syntax WinMaximize [window_name\$]

Comments If no window_name\$ parameter is specified, the window with the focus is maximized.

The window_name\$ parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. 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 the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the AppMaximize command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example

```

sub main()
    'Example of WinMaximize

    appn$ = AppFind$("Notepad")
    WinMaximize appn$
end sub

```

See Also Window Manipulation (Chapter 7)

WinMinimize Statement

Description	Minimizes the specified window.
Syntax	<code>WinMinimize [window_name\$]</code>
Comments	<p>If no <code>window_name\$</code> parameter is specified, the window with the focus is minimized.</p> <p>The <code>window_name\$</code> parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.</p> <p>This command differs from the <code>AppMinimize</code> command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.</p>
Example	<pre>sub main() 'Example of WinMinimize appn\$ = AppFind\$("Notepad") WinMinimize appn\$ end sub</pre>
See Also	Window Manipulation (Chapter 7)

WinMove Statement

Description	Moves the specified window.
Syntax	<code>WinMove x%,y% [window_name\$]</code>
Comments	<p>This command moves the given window to the given x,y position. If no <code>window_name\$</code> parameter is specified, then the window with the focus is moved.</p> <p>The <code>window_name\$</code> parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. Optionally, a partial name can be used, such as "Word" for "Microsoft Word".</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p>

```
WinActivate "Notepad|Find"
```

In the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the AppMove command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window. When moving child windows, remember that the x% and y% parameters are relative to the client area of the parent window.

Example

```
sub main()
    'Example of WinMove

    appn$ = AppFind$("Notepad")
    for i% = 0 to 100
        WinMove i%, i%, appn$
    next i%
end sub
```

See Also

Window Manipulation (Chapter 7)

WinRestore Statement

Description

Restores the specified window.

Syntax

```
WinRestore [window_name$]
```

Comments

If no window_name\$ parameter is specified, the window with the focus is restored.

The window_name\$ parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. 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 the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the AppRestore command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example

```
sub main()
```

```

'Example of WinRestore

appn$ = AppFind$("Notepad")
WinRestore appn$
end sub

```

See Also Window Manipulation (Chapter 7)

WinSize Statement

Description Resizes the specified window.

Syntax `WinSize width%,height% [window_name$]`

Comments This command resizes the given window to the specified width and height. If no `window_name$` parameter is specified, the window with the focus is resized.

The `window_name$` parameter specifies the name that appears on the desired application's title bar. The parameter is not case-sensitive. 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 the above example, the top level windows are first searched for a name that includes the string "Notepad". When found, the windows owned by the found window are searched for one that contains the string "Find" in its window title.

This command differs from the `AppSize` command in that this command operates on the current window rather than the current top-level window. The current window can be an MDI child window, a popup window, or a top-level window.

Example

```

sub main()
'Example of WinSize

appn$ = AppFind$("Notepad")
for i% = 1 to 200
    WinSize i%, i%, appn$
next i%
end sub

```

See Also Window Manipulation (Chapter 7)

Word\$ Function

Description	Extracts words from a text source.
Syntax	<code>word\$(text\$,first%[,last%])</code>
Returns	Returns a single word or sequence of words between <code>first</code> and <code>last</code> .
Comments	<p>The <code>first</code> parameter specifies the first word in the sequence to return. If <code>last</code> is not specified, then only that word is returned. If <code>last</code> is specified, then all words between <code>first</code> and <code>last</code> will be returned, including all spaces, tabs, and end-of-lines that occur between those words.</p> <p>Word are separated by spaces, tabs, and end-of-lines.</p> <p>If <code>first</code> is greater than the number of words in <code>text\$</code>, then an empty string is returned.</p> <p>If <code>last</code> is greater than the number of words in <code>text\$</code>, then all words from <code>first</code> to the end of <code>text</code> are returned.</p>
Example	<pre>sub main() 'Example of WORD\$() and WORDCOUNT functions dim teststr as string teststr = "The quick brown fox jumps over the lazy dog." for i% = 1 to wordcount(teststr) msgbox word\$(teststr,i%,i%) next i% end sub</pre>
See Also	Strings (Chapter 7)

WordCount Function

Description	Returns an integer representing the number of words in the specified text.
Syntax	<code>WordCount(text\$)</code>
Comments	Word are separated by spaces, tabs, and end-of-lines.
Example	<pre>sub main() 'Example of WORD\$() and WORDCOUNT functions dim teststr as string teststr = "The quick brown fox jumps over the lazy dog." for i% = 1 to wordcount(teststr) msgbox word\$(teststr,i%,i%) next i%</pre>

```
end sub
```

See Also Strings (Chapter 7)

Write # Statement

Description Writes a list of expressions to the specified file.

Syntax `write [#]filenumber% [,expressionlist]`

Comments The file referenced by `filenumber` must be opened in either output or append mode.

 The `filenumber` parameter is a number that is used by DCL to refer to the open file—the number passed to the `open` statement.

See Also File Input and Output (Chapter 7)

WriteINI Statement

Description Writes a new value into an INI file.

Syntax `WriteINI section$,ItemName$,value$[,filename$]`

Comments The `filename$` parameter, if specified, contains the name of an INI file. If `filename$` is not specified, the WIN.INI file is used.

 The `section$` parameter specifies the section which contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.

 The `ItemName$` parameter specifies which item from within the given section you want to change. If `ItemName$` is an empty string (""), then the entire section specified by `section$` is deleted.

 The `value$` parameter specifies the new value for the given item. If `value$` is an empty string (""), then the item specified by `ItemName$` is deleted from the INI file.

Example

```
sub main()
    'Example of WriteIni statement

    WriteIni "anewsection","anewitem","value","win.ini"
end sub
```

See Also Environment Statements and Functions (Chapter 7)

WS_MAXIMIZED

Description	Constant used with the AppSetState and AppGetState statements to indicate a maximized window state.
Value	1

WS_MINIMIZED

Description	Constant used with the AppSetState and AppGetState statements to indicate a minimized window state.
Value	2

WS_RESTORED

Description	Constant used with the AppSetState and AppGetState statements to indicate a normal window state.
Value	3

Xor

Description	Xor operator.
Syntax	<code>expression1 XOR expression2</code>
Returns	If both operands are relational, then XOR returns the logical exclusive OR of <code>expression1</code> and <code>expression2</code> . In other words, XOR returns TRUE only if both operands are not equal. If both operands are numeric, the result is the bitwise XOR of the arguments.
Notes	If either of the two operands is a floating point number, the two operands are first converted to longs, then a bitwise XOR is performed.
Example	<pre>sub main() 'Example of XOR dim a as integer</pre>

```

dim b as integer

a = 5
b = 9
if (a < 6) XOR (b > 8) then
    msgbox "Both conditions were not the same."
else
    msgbox "Both conditions were the same--either ←
        TRUE or FALSE."
end if

if (a < 6) XOR (b > 9) then
    msgbox "Both conditions were not the same."
else
    msgbox "Both conditions were the same--either ←
        TRUE or FALSE."
end if
end sub

```

See Also Operators (Chapter 7)

Year Function

Description Returns an integer representing the year of the date encoded in the specified `serial` parameter. The value returned is between 100 and 9999 inclusive.

Syntax `year(serial#)`

Comments You can obtain the value for the `serial#` parameter by using the `DateSerial` or `DateValue` command.

Example

```

sub main()
    'Example of Year

    msgbox str$(year(Now))    'display current year
end sub

```

See Also Date and Time Functions (Chapter 7)

