

This Windows Help file was written by Graham Plowman
using Help Builder Version 1.07 and refers to:

Setup Version 4.00 / 14-6-1995

Copyright 1993, 1994, 1995 G.Plowman

CALL command

Format

CALL "script"

Purpose

Calls another script file as a subroutine script

Parameters

- String name of the script to call

Return Value

None

Comments

The script interpreter supports nested calling of scripts to 6 levels.

The default file extension for script files if an extension is not supplied is .INF

Calling another script file using this command enables 'sub-routine' scripts to be used.

When a 'sub-routine' script is called, all of the commands within it are executed until the end of the script is reached, after which, control returns to the line after the CALL command in the calling script.

If a premature termination of the 'sub-routine' script is required, the EXIT command may be used to force control back to the calling script

See Also

[EXIT](#)

Chdir

Format

Chdir("Directory")

Purpose

Changes the current drive and/or directory

Parameters

- String name of directory to change to

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - directory not found
- FALSE Success

Comments

Unlike the DOS equivalent, this command changes both the drive and directory if they are specified

Example

CHDIR("C:\SETUP")

See Also

[Mkdir](#), [Rmdir](#)

CheckExists

Format

CheckExists("filename")

CheckExists("filename", "message")

Purpose

Checks for the presence of the file 'filename'.

In its second form this function checks for the existence of the parameter file and displays a message box if the file is not found. The message box continues to be displayed until a disk is inserted which contains the specified file

Parameters

- String name of file to check for
- Optional message to display if the file is not found

Return Value

The %ERROR% variable contains the return value:

Method 1:

- TRUE File(s) exist(s)
- FALSE File(s) do/does not exist

Method 2:

- IDOK File(s) exist(s)
- IDCANCEL File(s) not found and user has pressed Cancel

Comments

The file name may contain wildcards, drive and directory specifications.

This function can be used to check whether the correct disk in an installation suite has been inserted in the diskette drive

Example

```
CHECKEXISTS("C:\AUTOEXEC.BAT")
```

```
CHECKEXISTS("A:\DISK01", "Please insert DISK #1")
```

See Also

[CheckLabel](#), [GetModuleInUse](#)

CheckLabel

Format

CheckLabel("drive", "label", "message")

Purpose

Checks the disk 'drive' to see if it has the label 'label' and if not, displays a message box containing the parameter message

Parameters

- String drive letter
- String label of diskette to compare against
- String message to display if the diskette label does not match the specified label

Return Value

Sets the %ERROR% variable according to which key the user pressed to terminate the message box:

- IDOK User pressed the Ok button or the disk had the correct label
- IDCANCEL User pressed the Cancel button to quit

Example

CHECKLABEL("A:", "DISK1", "Insert disk labelled DISK1")

CloseIn

Format

CloseIn()

Purpose

Closes the file currently open for reading

Parameters

None

Return Value

None

Comments

If there is no file open, this function is ignored

Example

This example reads the CONFIG.SYS file and writes it out to TEMP.DAT having changed the FILES entry to 100

```
OpenIn("C:\CONFIG.SYS")
IF %ERROR% == TRUE GOTO :OPENERROR
```

```
OpenOut("C:\TEMP.DAT")
:NEXTLINE
ReadLine(%Buffer%)
IF %ERROR% == EOF GOTO :EOF
```

```
SET %Ptr% = Instr(1, %Buffer%, "FILES=")
IF %Ptr% == 0 GOTO :NOTFOUND
```

```
Set %Buffer% = "FILES=100"
```

```
:NOTFOUND
WriteLine(%Buffer%)
GOTO :NEXTLINE
```

```
:EOF
CloseIn()
CloseOut()
```

```
.
.
```

```
:OPENERROR
```

See Also

[CloseOut](#), [OpenIn](#), [OpenOut](#)

CloseOut

Format

CloseOut()

Purpose

Closes the file currently open for writing

Parameters

None

Return Value

None

Comments

If there is no file open, this function is ignored

Example

This example reads the CONFIG.SYS file and writes it out to TEMP.DAT having changed the FILES entry to 100

```
OpenIn("C:\CONFIG.SYS")
IF %ERROR% == TRUE GOTO :OPENERROR
```

```
OpenOut("C:\TEMP.DAT")
:NEXTLINE
ReadLine(%Buffer%)
IF %ERROR% == EOF GOTO :EOF
```

```
SET %Ptr% = Instr(1, %Buffer%, "FILES=")
IF %Ptr% == 0 GOTO :NOTFOUND
```

```
Set %Buffer% = "FILES=100"
```

```
:NOTFOUND
WriteLine(%Buffer%)
GOTO :NEXTLINE
```

```
:EOF
CloseIn()
CloseOut()
```

```
.
.
```

```
:OPENERROR
```

See Also

[CloseIn](#), [OpenIn](#), [OpenOut](#)

Command Directory

<u>CALL</u>	CALL another script
<u>EXIT</u>	EXIT to higher level script
<u>GOTO</u>	Control branching command
<u>IF</u>	Comparison command
<u>SET</u>	Variable assignment command
<u>STOP</u>	STOP all script processing

ConfirmOverwrite

Format

ConfirmOverwrite("oldfile", "Newfile", "message", inusecheck, onlyifolder)

Purpose

Checks to see if a file exists and then asks the user about overwriting it if it does exist. There are several options on how this can be done

Parameters

- Name of existing file to be overwritten
- Name of new file to overwrite with
- Message to prompt with if file found
- In use check which may be:
 - 0 No check
 - 1 Check if Windows is using the file
- Ask about overwrite only if new file is older than existing file:
 - 0 No check (always prompts with message)
 - 1 Prompt if new file is older than existing file

Return Value

The %ERROR% variable holds the error status:

- IDNO The user doesn't want the file overwritten
- IDYES Proceed to overwrite the file
- IDCANCEL File is in use by Windows

Example

```
ConfirmOverwrite("C:\WINDOWS\notepad.exe", "A:\notepad.exe", "Notepad already exists!", 0, 0)
IF %ERROR% == IDYES CopyFile("A:\notepad.exe", "C:\WINDOWS\notepad.exe")
```

See Also

[GetModuleInUse](#), [CheckExists](#)

Predefined Constants

The interpreter supports the following predefined constants whose numeric equivalents are listed. The constant or its numeric value may be used in any place where a numeric parameter is permitted.

Message Box Keys

MB_OK	0
MB_OKCANCEL	1
MB_ABORTRETRYIGNORE	2
MB_YESNOCANCEL	3
MB_YESNO	4
MB_RETRYCANCEL	5

Message Box Icons

MB_ICONSTOP	16
MB_ICONQUESTION	32
MB_ICONEXCLAMATION	48
MB_ICONINFORMATION	64

Return Keys from MessageBox / DialogBox functions

IDOK	1
IDCANCEL	2
IDABORT	3
IDRETRY	4
IDIGNORE	5
IDYES	6
IDNO	7
IDBACK	10
IDBUTTON1	11
IDBUTTON2	12
IDBUTTON3	13
IDBUTTON4	14
IDBUTTON5	15

Logical Values







TRUE	1
FALSE	0

End Of File

EOF	2
-----	---

Contents for Setup Script Help

Setup is a utility program for providing Windows-hosted procedures for installing Applications. Press the F1 key for for Help on using Windows Help.

-  [What is Setup ?](#)
-  [Creating a setup procedure](#)
-  [Standards and Notations](#)
-  [Command Directory](#)
-  [Function Directory](#)
-  [Error Message Directory](#)
-  [Predefined Variables](#)
-  [Predefined Constants](#)
-  [Suggestions for use](#)
-  [About this help file](#)

CopyFile

Format

This command has two formats:

Method 1

```
CopyFile("source file name", "target directory", uncompress, append, delete)
```

Method 2

```
CopyFile(n)
```

```
"source file name", "target directory", "message", uncompress, append, delete
```

Purpose

Copies file(s) from one location to another.

Parameters

Method 1

- String source file name which may include paths and wildcards
- Target directory/file name.
This may contain a complete filename but must not contain wildcards
- Open for uncompression
 - TRUE Open the source file and uncompress it (default)
 - FALSE Do not open for uncompression
- Copy and append
 - TRUE Copy file, appending to existing file with same name
 - FALSE Do not append to existing file - overwrite it (default)
- Delete after copy
 - TRUE Delete the source file after copying
 - FALSE Do not delete the source file after copying (default)

Method 2

The function call contains the number of files to be copied. This is so that the gauge knows how many files it is to represent and does not affect the actual number of files copied. It does not matter if you get the number wrong, but you might find the gauge visually fills up before you expected it to or not to fully fill up.

The file name parameters are listed in the lines after the command:

- String source file name which may include paths and wildcards
- Target directory/file name.
This may contain a complete filename but must not contain wildcards
- String message which can be used to tell the user in the 'copy dialog' what files are being copied
- Open for uncompression
 - TRUE Open the source file and uncompress it (default)
 - FALSE Do not open for uncompression
- Copy and append
 - TRUE Copy file, appending to existing file with same name
 - FALSE Do not append to existing file - overwrite it (default)
- Delete after copy
 - TRUE Delete the source file after copying
 - FALSE Do not delete the source file after copying (default)

Return Value

The %ERROR% variable holds the error number

Comments

Method (1) performs a straight copy whereas method (2) displays a 'gauge' to show copy progress. The number 10 in the example below would represent the number of files being copied. This is used purely for the purpose of the fuel gauge to know how many files it should represent. If it is incorrect, it doesn't matter, but you may find the gauge finishes too soon or not at all. It does not affect the number of files copied.

Note that source file names may contain drive, path and wildcard specifications.

Target file names can contain a path name and/or a file name, but they must NOT contain wildcards. Path names may end with a \ but this will be appended automatically if not supplied.

The 'Description' parameter is optional and may be left off, but if supplied will be displayed at the top of the copy files dialog to inform the user of what is being copied.

The CopyFile function will handle files compressed using the Microsoft COMPRESS.EXE utility. In this situation CopyFile will automatically read in the compressed file and write it out in its expanded form (ie uncompressed form).

Example

Method 1

```
CopyFile("A:\*.BAT", "C:\BATCH")
```

Method 2

```
CopyFile(10)
```

```
"A:\*.BAT", "C:\BATCH", "Copying: Program batch files"
```

Creating a setup procedure

To create a setup procedure three components are required, although only two are required in certain situations:

- The SETUP.EXE installer program
- The INST.EXE interpreter program
- A user written setup script (.INF) file

There are two configurations in which the Setup utility may be used:

Running from Diskette

In this case all three of the above files are required on the first diskette of your installation suite.

The SETUP.EXE program copies the INST.EXE program and the SETUP.INF file to the WINDOWS directory of the machine on which the program is running. SETUP.EXE then runs the INST.EXE interpreter program which runs the script file from the hard disk. This is necessary since install procedures often request diskette changes in which case the Setup executables would not run.

INST.EXE does not remove itself or SETUP.INF from the hard disk when it has completed however you can place commands in a script yourself to do this.

WARNING: If SHARE.EXE is running and your script file attempts to delete INST.EXE or SETUP.INF you will get a 'Share violation error' from Windows.

Running from a hard disk or network drive

In this case only INST.EXE and a .INF file are required. Note that INST.EXE takes the name of the script file as its parameter so the script file can have any name whereas SETUP.EXE above always assumes a name of SETUP.INF.

You may place INST.EXE on the Windows Program manager as an icon with the script file name as a parameter. You can also set up a file association with the File Manager such that double-clicking on a script file will cause INST.EXE to run it. This is optionally set up for you when you install the Setup utility software on your machine.

Procedure for creating install suites

In order to create a successful windows hosted software installation procedure it is advisable that you carry out the following steps:

- Plan what is to be installed
- Plan what options the user is to be given
- Plan the layout of files on the disk / diskettes
- Ensure that the installation procedure is as simple as possible - users do not expect to see technical terms
- Ensure that the user is given feedback on what is being or has been installed
- Above all, ensure that your script is bug-free by testing it in as many environments as possible

Creating a script file

A setup script file is purely an ASCII text file which may be created with any ASCII file editor. Alternatively you can use the Setup Builder application to automatically build a setup script and the appropriate diskettes for you.

Be warned that Windows Notepad has an error in it which causes a file not to have a carriage return placed on the last line of the file unless you explicitly place blank lines at the end. This can cause problems with setup and with many other ASCII file editors since the end of the file is found on reading before an end of line marker

Delete

Format

Delete("File name")

Purpose

Deletes a file

Parameters

- String name of file(s) to be deleted

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - file not found or deleted
- FALSE Success

Comments

The file name may contain both wild cards and/or drive/path specifications

Example

```
Delete("C:\AUTOEXEC.BAT")
```

```
Delete("C:\ABC.BAT")
```


DeleteGroup

Format

DeleteGroup("group name")

Purpose

Deletes a program group from Windows Program Manager

Parameters

- String name of group to be deleted

Return Value

None

Comments

The group name is the name which appears in the caption of the group when it is displayed by Program Manager

Example

```
DeleteGroup("TEST")
```

See Also

[MakeGroup](#)

DeleteIcon

Format

DeleteIcon("name")

Purpose

Deletes an icon within the currently selected program manager group

Parameters

- String name of icon to delete

Return Value

None

Comments

The icon name is the text which appears below the icon in Program Manager

Example

```
DeleteIcon("Editor")
```

See Also

[MakeIcon](#)

DialogBox

Format

DialogBox("dialog name")

Purpose

Activates one of the predefined dialog boxes.

Parameters

- String name of dialog to display

Return Value

The %ERROR% variable contains the push/command button which terminated the dialog:

- IDOK
- IDCANCEL
- IDBACK
- IDBUTTON1
- IDBUTTON2
- IDBUTTON3
- IDBUTTON4
- IDBUTTON5

Any data entered in fields or buttons is returned in the following variables:

- %BUTTON1% For CHECKBOXn dialog boxes
to
%BUTTON6%
- %RADIOBUTTON% For RADIOBn dialog boxes
- %EF_1% or For INPUTBOXn dialog boxes
%EF_2%

Comments

The dialog name may be one of the following:

- "WELCOME" Display the Welcome to setup dialog with an icon
- "ASKPATH" Display dialog to ask user install path
- "OKBOX" Display a standard Ok confirmation dialog with icon
- "CHECKBOX1" Display a dialog with 1 checkbox
- "CHECKBOX2" Display a dialog with 2 checkboxes
- "CHECKBOX3" Display a dialog with 3 checkboxes
- "CHECKBOX4" Display a dialog with 4 checkboxes
- "CHECKBOX5" Display a dialog with 5 checkboxes
- "CHECKBOX6" Display a dialog with 6 checkboxes
- "RADIOB1" Display a dialog with 1 radio button
- "RADIOB2" Display a dialog with 2 radio buttons
- "RADIOB3" Display a dialog with 3 radio buttons
- "RADIOB4" Display a dialog with 4 radio buttons
- "RADIOB5" Display a dialog with 5 radio buttons
- "RADIOB6" Display a dialog with 6 radio buttons
- "INPUTBOX1" Display a dialog with 1 input field
- "INPUTBOX2" Display a dialog with 2 input fields

- "LICENSE" Display licensing dialog
- "PUSHB2" Display a dialog with 2 push buttons
- "PUSHB3" Display a dialog with 3 push buttons
- "PUSHB4" Display a dialog with 4 push buttons
- "PUSHB5" Display a dialog with 5 push buttons

Each of the dialogs has predefined text fields which you may change by setting the following variables:

- %CAPTION% Caption of the dialog
- %MESSAGE1% First static text field within the dialog
to
%MESSAGE6% Last static text field within the dialog (depends on dialog)
- %INSTALLPATH% Sets the default prepopulated value of the edit field in the ASKPATH dialog
- %BUTTON1% Preset status of check boxes in
to
%BUTTON6% CHECKBOX dialogs. A '1' signifies a check is on.
Also used to return the states of the buttons on exit from the dialog
- %RADIOBUTTON% Holds the initially active radio button within the RADIOB dialogs.
The first button is 1, the last 6, depending on the dialog.
Also used to return the selected button.
- %INIFILE% Controls the .INI file to which the Licensing dialog writes the name/company entered by the user.
- %PUSHB_1% Holds the text to be displayed on the push
to
%PUSHB_5% buttons within the PUSHB dialogs.
The first button is 1, the last 5, depending on the dialog.

Note: The ASKPATH dialog automatically checks the path entered by the user by attempting to create the directory. Therefore the base install directory is automatically created and the programmer need not create it, only any directories required under it

Example

DialogBox("OKBOX")

See Also

[MessageBox](#), [Predefined Constants](#)

Error Message Directory

01 Invalid command

This error occurs when a command or function is encountered which the interpreter does not recognise.
See [Command Directory](#), [Function Directory](#)

02 Invalid parameters

This error occurs when the wrong 'type' of parameter is given to a command or function, for example a number given where a string is expected

03 Variable not found

This error occurs when an undefined variable is passed as a parameter. In future versions of Setup this error message will no longer occur because undefined variables will default to an empty string

04 Invalid variable name

A variable name must start with and end with a % character. If the trailing % is left off, this error will result.
See [Variables](#)

05 Label not found

A GOTO command is attempting to pass control to a label which cannot be found within the script file. Check that the label starts with a colon : both after the GOTO command and on the line to be branched to

06 Invalid string

A string must start with and end with a " character. If the trailing " is left off, this error will result.
See [Variables](#)

07 Label too long

A label may be a maximum of 20 characters. This error results if an attempt is made to use a longer name. Check that the label ends with a space character, end of line or that there is at least one space after it before a comment.
See [Labels](#)

08 String stack full

Too many strings have been defined within a command. The limit is 20. No Setup command or function should reach this limit, so if this error occurs it is likely that you have a severe syntax error!

09 Numeric stack full

The same applies to numbers as in error 08

10 Variable name too long

A variable name may be up to 20 characters long. This error occurs when an attempt is made to use a longer name or if the trailing % sign is left off of the variable name.
See [Variables](#)

11 Text too long

Text strings may be up to 254 characters long. This error occurs when an attempt is made to use a string (with no embedded variables) which is longer than 254 characters or the trailing " character has been left off.
See [Strings](#)

12 Invalid label

This error occurs when an invalid label is passed as a parameter to the GOTO command. You cannot supply strings or variables to this command

13 Invalid template name

An attempt has been made to use the DialogBox() function but an invalid dialog template name was supplied.
See [DialogBox](#)

14 String concat too long

A string may be a maximum of 254 characters. This error usually results when embedded variables in a string are used to concatenate strings and the resulting string is longer than 254 characters.
See [Strings/Variables](#)

15 No space on target drive

The CopyFile() function has been called to copy a file and there is not enough space on the target drive for the file

16 Source file not found

The CopyFile() function has been called to copy a file but the file could not be found

17 Failure while copying

The CopyFile() function failed while copying. This usually occurs if the user removes a diskette while copying from it or if a disk read failure occurs

18 Out of variable space

Setup allows up to 50 variables to be defined at a time. This error occurs when an attempt is made to create more variables. Assign variables to empty strings to clear space

19 Source and target file names the same

The CopyFile() function has been called and both the source and target file names are the same - you cannot copy a file onto itself

20 Invalid string parameter

A function has been called which expects a string parameter in the indicated position

21 Invalid numeric parameter

A function has been called which expects a numeric parameter in the indicated position

22 Missing variable name

This error occurs when the target return variable parameter is left off of the GetPrivateProfileString() function

23 Invalid comparison operator

The IF command has been supplied with an invalid comparison operator.
See [IF](#)

24 Invalid comparison value

This error occurs when the two values for comparison by an IF command are not of the same type.
See [IF](#)

25 Invalid date format specified

This error occurs when an invalid date format is specified to the date functions.
See [GetDate](#), [GetFileDate](#)

26 Invalid arithmetic operator. Operator must be + - * or /

This error occurs with the SET statement when arithmetic operations are being performed. Setup only supports addition, subtraction, multiplication and division of integer numbers.
See [Set](#)

27 String subscript out of range

This error occurs with the string handling functions when a position within a string is specified which doesn't exist.

This may be because the value specified is negative or because the value is greater than the maximum length that a string is allowed to be (ie 254 characters)

See [Left](#), [Right](#), [Mid](#), [Instr](#)

EXIT command

Format

EXIT

Purpose

Terminates the processing of the current script file, returning control to the higher level script file which called the current script

Parameters

None

Return Value

None

Comments

This command is similar to the RETURN statement found in many programming languages and is used to terminate nested scripts

See Also

STOP

ExitWindows

Format

ExitWindows(numstate)

Purpose

Restarts Windows or reboots the machine

Parameters

A numeric which is either TRUE or 1 to reboot the machine or FALSE or 0 just to restart Windows

Return Value

The %ERROR% variable holds the return value of the standard Windows ExitWindows function ie 0 if any applications fail to terminate otherwise there is no return.

Comments

WARNING: This function should be used with care since it can cause loss of data

Example

ExitWindows(TRUE)

Function Directory

Date/Time Functions

<u>GetDate</u>	Get the system date
<u>GetTime</u>	Get the system time
<u>GetFileDate</u>	Get file date
<u>GetFileTime</u>	Get file time
<u>SetFileDate</u>	Set file date
<u>SetFileTime</u>	Set file time

Disk & Directory Functions

<u>Chdir</u>	Change current drive/directory
<u>CheckLabel</u>	Check disk label
<u>GetDiskSpace</u>	Get free disk space
<u>IsWriteable</u>	Check is disk/path writeable
<u>Mkdir</u>	Make a new directory
<u>Rmdir</u>	Remove a directory

File Related Functions

<u>CheckExists</u>	Check if file exists
<u>CloseIn</u>	Close the input file
<u>CloseOut</u>	Close the output file
<u>ConfirmOverwrite</u>	Allows user to confirm a file overwrite
<u>CopyFile</u>	Copy file(s) from one location to another
<u>Delete</u>	Delete a file
<u>GetFileAttr</u>	Get file attributes
<u>GetFileLength</u>	Get file length
<u>OpenIn</u>	Open a file for reading
<u>OpenOut</u>	Open a file for reading
<u>ReadLine</u>	Read a line from the input file
<u>Rename</u>	Rename a file to another name
<u>SetFileAttr</u>	Set file attributes
<u>UnCompress</u>	Un-compress a file, renaming
<u>WriteLine</u>	Write a line to the output file

Program Manager Functions

<u>DeleteGroup</u>	Delete a Program Manager group
<u>DeleteIcon</u>	Delete a Program Manager Icon
<u>MakeGroup</u>	Make/Select a Program Manager group
<u>MakeGroupFromFile</u>	Make a Program Manager group
<u>MakeIcon</u>	Make a Program Manager Icon
<u>Reload</u>	Reload Program Manager groups
<u>ShowGroup</u>	Display a Program Manager group

String Manipulation Functions

<u>Instr</u>	Find one string in another
<u>LCase</u>	Convert string to lower case
<u>Left</u>	Get left n characters of a string
<u>Len</u>	Get length of string
<u>Mid</u>	Get a sub-string from a string
<u>Right</u>	Get the right n characters of a string
<u>UCase</u>	Convert a string to upper case

Windows Interface/API Functions

<u>DialogBox</u>	Use an inbuilt dialog
<u>ExitWindows</u>	Terminate Windows
<u>GetModuleInUse</u>	Check if Windows is using a file
<u>GetProfileString</u>	Get an .INI file string
<u>GetScreenHeight</u>	Get screen height in pixels
<u>GetScreenWidth</u>	Get screen width in pixels
<u>MessageBox</u>	Pop up a message box
<u>Release</u>	Release control to Windows
<u>SetBackDropText</u>	Write text on backdrop
<u>ShowBackDrop</u>	Display background backdrop/bitmap
<u>Wait</u>	Wait for a window to appear/disappear
<u>WinExec</u>	Execute another program
<u>WriteProfileString</u>	Write an .INI file string

GetDate

Format

GetDate(%varname%)

GetDate(%varname%, format)

Purpose

Gets the system date into a variable

Parameters

- Variable to store the result in
- Optional date format required which may be:
 - 0 For dd/mm/yy
 - 1 For yy/mm/dd

Return Value

None

Comments

The date format specifier is optional, the default being 0.

If an invalid date format is specified a run time error will occur

Example

```
GetDate(%Date%) // 21/10/93
```

```
GetDate(%Date%, 1) // 93/10/21
```

See Also

[GetTime](#)

GetDiskSpace

Format

GetDiskSpace("drive letter")

Purpose

Retrieves the amount of space available on a disk.

Parameters

- String containing the letter of the drive to be checked.
The text case is not important

Return Value

The %ERROR% variable contains the number of free bytes on the specified disk

Comments

Example

GetDiskSpace("A:")

GetFileAttr

Format

GetFileAttr("filename", %rdonly%, %hidden%, %system%, %archive%)

Purpose

Gets the attributes of a file into variables

Parameters

- String name of file to get the attributes of
- Variable to store the read only attribute in
- Variable to store the hidden attribute in
- Variable to store the system attribute in
- Variable to store the archive attribute in

Return Value

The %ERROR% variable holds the error status:

- TRUE Error - file not found
- FALSE Success

Comments

The file name must NOT contain wildcards

Example

```
GetFileAttr("C:\AUTOEXEC.BAT", %rdonly%, %hidden%, %system%, %archive%)
```

GetFileDate

Format

```
GetFileDate("filename", %varname%)  
GetFileDate("filename", %varname%, format)
```

Purpose

Gets the date of a file into a variable

Parameters

- String name of file to get the date of
- Variable to store the result in
- Optional date format required which may be:
 - 0 For dd/mm/yy
 - 1 For yy/mm/dd

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - file not found
- FALSE Success

Comments

The date format specifier is optional, the default being 0.
If an invalid date format is specified a run time error will occur

Example

```
GetFileDate("C:\AUTOEXEC.BAT", %Date%) // 29/03/92  
GetFileDate("C:\AUTOEXEC.BAT", %Date%, 1) // 92/03/29
```

See Also

[GetFileTime](#)

GetFileLength

Format

GetFileLength("filename")

Purpose

Gets the length of a file

Parameters

None

Return Value

The %ERROR% variable contains the length of the file if successful or -1 if an error occurred

Comments

Example

```
GetFileLength("C:\AUTOEXEC.BAT")
```


GetFileTime

Format

GetFileTime("filename", %varname%)

Purpose

Gets the time of a file into a variable

Parameters

- String name of file to get the time of
- Variable to store the result in

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - file not found
- FALSE Success

Comments

The file name must NOT contain wildcards

Example

```
GetFileTime("C:\AUTOEXEC.BAT", %Time%)
```

See Also

[GetFileDate](#)

GetScreenHeight

Format

GetScreenHeight(%varname%)

Purpose

Obtains the physical height of the screen in pixels. The function is intended to be used with the ShowBackDrop() function in order to determine the screen size and therefore, bitmap to display

Parameters

- Variable to store the screen height in

Return Value

None

Example

GetScreenHeight(%Height%)

See Also

[GetScreenWidth](#), [ShowBackDrop](#)

GetModuleInUse

Format

GetModuleInUse("filename")

Purpose

Determines whether a module is in use by Windows ie it is already running. This can be used to prevent installation of an executable which is being presently run by Windows

Parameters

- String name of file to be checked

Return Value

The %ERROR% variable holds the state:

- TRUE Module is in use
- FALSE Module is not in use

Comments

It is worth using this function within an installation script since the CopyFile() function will abort with an error message and terminate an installation script if an attempt is made to overwrite a file which is in use. This function enables the programmer to retain control over this situation

Example

GetModuleInUse("progman.exe")

GetPrivateProfileString

GetProfileString

Format

```
GetProfileString("section" , "entry", "default",  
                "file name", %varname%)
```

Purpose

Reads a string from a Windows .INI file into a variable

Parameters

- String [Section] of .INI file to read from
- String entry within the section to read
- String default if entry not found
- String name of .INI file to read from
- String name of variable to place the string read

Return Value

None

Comments

Along with the standard windows function, if no path is specified in the .INI file name, reading defaults to the Windows directory.

The GetPrivateProfileString function is only supplied for compatibility with earlier versions of Setup. You should use the GetProfileString function.

Example

```
GetProfileString("Windows", "Spooler", "yes", "win.ini", %spooler%)
```

See Also

[WriteProfileString](#)

GetTime

Format

GetTime(%varname%)

Purpose

Gets the system time into a variable

Parameters

- Variable to store the result in

Return Value

None

Comments

The time is in the format hh:mm:ss

Example

```
GetTime(%Time%)
```

See Also

[GetDate](#)

GetScreenWidth

Format

GetScreenWidth(%varname%)

Purpose

Obtains the physical width of the screen in pixels. The function is intended to be used with the ShowBackDrop() function in order to determine the screen size and therefore, bitmap to display

Parameters

- Variable to store the screen width in

Return Value

None

Example

GetScreenWidth(%Width%)

See Also

[GetScreenHeight](#), [ShowBackDrop](#)

GOTO command

Format

GOTO :label

Purpose

Causes a branch of execution of the script file to another line within the file. That line must start with the same label name preceeded with a colon :

Parameters

- A label

Return Value

None

Comments

A label cannot be a variable name.

A run time error will occur if the parameter label cannot be found

Example

```
GOTO :END
```

```
.
```

```
.
```

```
:END
```

See Also

[Standards and Notations](#)

IF command

Format

IF <value> <comparison operator> <value> <statement>

Purpose

Performs a comparison between two values

Parameters

There are three parameters to this command:

- A numeric/string value
- A comparison operator
- A second numeric/string value

Return Value

None

Comments

Valid comparison operators are:

- == Equals
- != Not equals
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to

Since Setup does not have a concept of variable 'types' the IF command follows certain rules depending on the comparison being performed.

If the first value is a numeric ie numeric digits or a variable name, then a numeric comparison is performed with the second value.

If the first value is a string ie text enclosed in quotes or text enclosing a variable name, then a string comparison is performed.

If the 'type' of the second parameter does not conform with the first parameter then a run time error will occur.

Any Setup command may follow the comparison and this will be executed if the result of the comparison is true

Example

Numeric comparisons

```
IF 1 < 2 GOTO :END
```

```
IF %NUMBER% == 10 GOTO :TEN
```

```
IF %ERROR% == IDBACK GOTO :BACK
```

```
IF %ERROR% == TRUE GOTO :END
```

String comparisons

```
IF "TEXT" == "TEST" GOTO :SAME
```

```
IF "%INSTALLPATH%" != "C:\" GOTO :END
```

See Also

[Standards and Notations](#)

Instr

Format

Instr(start, "SearchString", "FindString")

Purpose

Used in conjunction with the SET command, this function finds the position of one string within another string

Parameters

- Numeric position to start searching from (first character is 1)
- String to search
- String to search for

Return Value

The return value is the position within the string where the requested string was found. It is 0 if the string was not found or greater than 0 if it was found.

The return value is a numeric and is assigned to the variable in the SET statement

Example

```
Set %Var% = Instr(1, "Test String", "st") // %Var% holds 3
```

IsWriteable

Format

IsWriteable("directory name")

IsWriteable("drive spec")

Purpose

Checks to see if a drive or directory can be written to

Parameters

- Name of the directory to check
or
- Drive specification to check

Return Value

The %ERROR% variable holds the error status:

- TRUE Drive/directory is writeable
- FALSE Drive/directory is not writeable

Example

IsWriteable("C:\ABC")

IsWriteable("H:")

LCase

Format

LCase("String")

Purpose

Used in conjunction with the SET command, this function converts a string to lower case

Parameters

- String to convert

Return Value

The return value is a string and is assigned to the variable in the SET statement

Example

```
Set %Var% = LCase("Test String")      // %Var% holds 'test string'
```

See Also

[UCase](#)

Left

Format

Left("String", numchars)

Purpose

Used in conjunction with the SET command, this function extracts the number of characters specified from the start of a string

Parameters

- String to extract from
- Numeric number of characters to extract

Return Value

The return value is a string and is assigned to the variable in the SET statement

Example

```
Set %Var% = Left("Test String", 3)      // %Var% holds 'Tes'
```

See Also

[Mid](#), [Right](#)

Len

Format

Len("String")

Purpose

Used in conjunction with the SET command, this function finds the length of a string in characters

Parameters

- String to obtain the length of

Return Value

The return value is a numeric and is assigned to the variable in the SET statement

Example

```
Set %Var% = Len("Test String") // %Var% holds 11
```

MakeGroup

Format

MakeGroup("group name")

MakeGroup("group name", "group file")

Purpose

Creates a program group on Windows Program Manager

Parameters

- String name of group to be created or selected
- String name of group file to be created for the group

Return Value

None

Comments

The group name is the name which appears in the caption of the group when it is displayed by Program Manager.

The group file name may include drive/path specifiers.

Note that creating a group which already exists does not create a new group, instead it makes the existing group the current group. Therefore, this command can be used for selecting groups as well as creating them.

Example

MakeGroup("TEST")

MakeGroup("TEST", "FILE.GRP")

See Also

[DeleteGroup](#)

MakeGroupFromFile

Format

MakeGroupFromFile("group file name")

Purpose

Creates a program group on Windows Program Manager from an existing group file

Parameters

- String name of group file to be installed

Return Value

None

Comments

The group file name may include drive/path specifiers.

Note that creating a group which already exists does not create a new group.

Example

```
MakeGroupFromFile("C:\ABC\TEST.GRP")
```

See Also

[MakeGroup](#), [DeleteGroup](#)

MakeIcon

Format

MakeIcon("icon text", "file name", "icon file", iconIndex, "default directory")

Purpose

Creates an icon within the currently selected program manager group

Parameters

- String name of icon to create
- String containing command line to run the icon
- String containing file name (.exe or .dll) containing the icon to use
- Number of the icon in the .exe or .dll to use (first is 0)
- String containing default directory for program

Return Value

None

Comments

The icon text appears below the icon in Program Manager.

The icon text is used by all program manager functions to identify an icon.

Note that it is not possible to create duplicate icons with the same name - the existing icon is automatically deleted before the new one is created. The matching is performed using the icon text.

Only the first two parameters of this function are actually required, the rest defaulting if not supplied, however if you wish to set the default path for example, you can do so by supplying null parameters: see the examples below.

Icons are held in .exe and .dll files and are zero subscripted: The first icon is zero, the second is one and so on.

Example

```
// The basic default use of the function
```

```
// The icon file defaults to notepad.exe and the index to zero
```

```
MakeIcon("Editor", "notepad.exe")
```

```
// Make an icon but use the default icon (0) from cardfile.exe
```

```
MakeIcon("Editor", "notepad.exe", "cardfile.exe")
```

```
// Make an icon but use the third icon in progman.exe
```

```
MakeIcon("Editor", "notepad.exe", "progman.exe", 2)
```

```
// Make an icon, setting the icon and working directory
```

```
MakeIcon("Editor", "notepad.exe", "progman.exe", 2, "C:\MYDIR")
```

```
// Make an icon using the third icon and setting the working directory
```

```
MakeIcon("Program Manager", "progman.exe", "", 2, "C:\MYDIR")
```

```
// Make an icon, setting the working directory, but using the default icon
```

```
MakeIcon("Editor", "notepad.exe", "", -1, "C:\MYDIR")
```

See Also

[DeleteIcon](#)

Mid

Format

Mid("String", start, length)

Purpose

Used in conjunction with the SET command, this function extract a substring from another string

Parameters

- String to obtain the substring from
- Numeric position to start extracting from (first character is 1)
- Numeric number of characters to extract

Return Value

The return value is a string and is assigned to the variable in the SET statement

Example

```
Set %Var% = Mid("Test String", 2, 5)    // %Var% holds 'est S'
```

See Also

[Left](#), [Right](#)

Mkdir

Format

MkDir("directory name")

Purpose

Creates a new directory on a disk. This function now supports a multi-level directory create

Parameters

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - directory exists or could not be created
- FALSE Success

Example

Rmdir("C:\TEST")

See Also

[Rmdir](#), [Chdir](#)

MessageBox

Format

MessageBox("message", "caption", buttons, icon)

Purpose

Provides the ability to pop-up a standard Windows message box

Parameters

- String message to be displayed
- String caption of message box
- Button setting:
 - MB_OKCANCEL
 - MB_OK
 - MB_ABORTRETRYIGNORE
 - MB_YESNOCANCEL
 - MB_YESNO
 - MB_RETRYCANCEL
- Icon required:
 - 0 - no icon
 - MB_ICONQUESTION
 - MB_ICONEXCLAMATION
 - MB_ICONINFORMATION
 - MB_ICONSTOP

Return Value

The %ERROR% variable holds the button pressed:

- IDOK
- IDCANCEL
- IDABORT
- IDRETRY
- IDIGNORE
- IDYES
- IDNO
- IDBACK

Comments

By convention you should use message boxes and dialog boxes to ask the user simple questions and give them the ability to perform selective or special installations.

Always use a message box to ask the user to confirm loss of data!

Example

```
MessageBox("A test message", "Test", MB_OK, MB_ICONQUESTION)
```

See Also

[DialogBox](#), [Predefined Constants](#)

OpenIn

Format

OpenIn("filename")

Purpose

Opens a file on a disk for reading

Parameters

- String name of the file to open

Return Value

The %ERROR% variable holds the error status:

- TRUE Error - failed to open file
- FALSE Success

Comments

The file name must be a standard DOS format name which may contain drive and/or path specifications

Example

This example reads the CONFIG.SYS file and writes it out to TEMP.DAT having changed the FILES entry to 100

```
OpenIn("C:\CONFIG.SYS")
IF %ERROR% == TRUE GOTO :OPENERROR
```

```
OpenOut("C:\TEMP.DAT")
:NEXTLINE
ReadLine(%Buffer%)
IF %ERROR% == EOF GOTO :EOF
```

```
SET %Ptr% = Instr(1, %Buffer%, "FILES=")
IF %Ptr% == 0 GOTO :NOTFOUND
```

```
Set %Buffer% = "FILES=100"
```

```
:NOTFOUND
WriteLine(%Buffer%)
GOTO :NEXTLINE
```

```
:EOF
CloseIn()
CloseOut()
```

```
.
```

```
:OPENERROR
```

See Also

[CloseIn](#), [CloseOut](#), [OpenOut](#)

OpenOut

Format

OpenOut("filename")

Purpose

Opens a file on a disk for writing

Parameters

- String name of the file to open for writing

Return Value

The %ERROR% variable holds the error status:

- TRUE Error - failed to open file
- FALSE Success

Comments

The file name must be a standard DOS format name which may contain drive and/or path specifications. If the file already exists, its contents are destroyed. If the file does not exist, it is created

Example

This example reads the CONFIG.SYS file and writes it out to TEMP.DAT having changed the FILES entry to 100

```
OpenIn("C:\CONFIG.SYS")
IF %ERROR% == TRUE GOTO :OPENERROR
```

```
OpenOut("C:\TEMP.DAT")
:NEXTLINE
ReadLine(%Buffer%)
IF %ERROR% == EOF GOTO :EOF
```

```
SET %Ptr% = Instr(1, %Buffer%, "FILES=")
IF %Ptr% == 0 GOTO :NOTFOUND
```

```
Set %Buffer% = "FILES=100"
```

```
:NOTFOUND
WriteLine(%Buffer%)
GOTO :NEXTLINE
```

```
:EOF
CloseIn()
CloseOut()
```

```
.
```

```
:OPENERROR
```

See Also

[CloseIn](#), [CloseOut](#), [OpenIn](#)

Predefined Variables

CAPTION

This variable holds the text which is used as the caption for all of the predefined dialogs. It defaults to 'Setup'.

CURRENTDIRECTORY

This variable holds the full drive and path name of the current directory, complete with a trailing backslash character:

eg: C:\MAIN\TEST\

CURRENTDRIVE

This variable holds the current drive letter:

eg: A:

Note that along with CURRENTDIRECTORY, this variable is automatically updated when the ChDir() function is used.

ERROR

This variable is the 'accumulator' for return values. All functions which return a value set this variable. The contents of this variable are numeric and may be any positive number. For example a call to MessageBox() will place the value of IDOK (1) or IDCANCEL (2) in the ERROR variable whereas GetDiskSpace() will place the number of bytes free on the specified disk in the ERROR variable.

INSTALLPATH

This variable is used by the AskPath dialog and should be used as the target path for any copying

INSTALLDRIVE

This variable holds the drive letter portion of the INSTALLPATH variable.

PROGRAMFILE

This variable holds the fully qualified file name (including drive and path name) of the interpreter program.

SCRIPTFILE

This variable holds the fully qualified file name of the script file currently being executed.

SYSTEMDIRECTORY

This variable contains the drive and path of the Windows System directory which is normally C:\WINDOWS\SYSTEM\

WINDOWS DIRECTORY

This variable contains the drive and path of the Windows directory which is normally C:\WINDOWS\
This variable is normally used when a .INI file is to be copied / installed into the Windows directory during a Setup procedure.

NOTE:

The predefined variables are treated in exactly the same way as any user variable and can therefore be 'nullified' to free up the variable space, however, some will automatically recreate themselves when certain commands or functions are used!

If required, they can be assigned values although this defeats the object of some of them displaying the current system state. It is most likely that you might wish to do this with CAPTION and ERROR.

See Also [Standards and Notations](#), [DialogBox](#)

ReadLine

Format

ReadLine(%Variable%)

Purpose

Reads a line from the currently open file into a variable

Parameters

- The variable into which to read the line

Return Value

The %ERROR% variable holds the error status:

- 0 Success
- 1 File not open
- 2 End of file reached

Comments

The maximum length of line which can be read is 254 characters.

This function will only handle ASCII text files with lines ending in CR/LF

Example

This example reads the CONFIG.SYS file and writes it out to TEMP.DAT having changed the FILES entry to 100

```
OpenIn("C:\CONFIG.SYS")
IF %ERROR% == TRUE GOTO :OPENERROR
```

```
OpenOut("C:\TEMP.DAT")
:NEXTLINE
ReadLine(%Buffer%)
IF %ERROR% == EOF GOTO :EOF
```

```
SET %Ptr% = Instr(1, %Buffer%, "FILES=")
IF %Ptr% == 0 GOTO :NOTFOUND
```

```
Set %Buffer% = "FILES=100"
```

```
:NOTFOUND
WriteLine(%Buffer%)
GOTO :NEXTLINE
```

```
:EOF
CloseIn()
CloseOut()
```

```
.
```

```
:OPENERROR
```

See Also

CloseIn, CloseOut, OpenIn, OpenOut, WriteLine

Release

Format

Release()

Purpose

Relinquishes control to Windows in order to achieve multi-tasking

Parameters

None

Return Value

None

Comments

The Setup program only performs multi-tasking during file copying and while dialog or message boxes are present on the screen.

Since Windows requires applications to relinquish control in order to achieve multi-tasking, this function provides multi-tasking ability to the Setup program

Example

This example waits for a file to be created by another process. It terminates when the file becomes present

```
:WAIT  
CheckExists("TEST.TXT")  
IF %ERROR% == TRUE GOTO :FOUND
```

```
Release()  
GOTO :WAIT
```

```
:FOUND
```

Reload

Format

Reload()

Reload("group name")

Purpose

Tells Program Manager to reload all its group files from those specified in PROGMAN.INI

Tells Program Manager to reload a specific group file.

Parameters

- String name of the program group to reload.

Note that an empty string here will cause Program Manager to do nothing!

If all groups are to be reloaded, use the first format of the command with no name between the brackets

Return Value

None

Comments

This function is useful when the PROGMAN.INI file has been changed manually via the WritePrivateProfileString() function

Example

Reload()

Reload("Main")

Rename

Format

Rename("OldName", "NewName")

Purpose

Renames a file to a new name

Parameters

- String name of file to be renamed
- String new name of file

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - file not found or renamed
- FALSE Success

Comments

The file name may not contain wild cards or path/drive specifications which means that when renaming files, the file to be renamed must be in the current directory

Example

Rename("TEST.DAT", "DATA.DAT")

Right

Format

Right("String", numchars)

Purpose

Used in conjunction with the SET command, this function extracts the number of characters specified from the end of a string

Parameters

- String to extract from
- Numeric number of characters to extract

Return Value

The return value is a string and is assigned to the variable in the SET statement

Example

```
Set %Var% = Right("Test String", 3)    // %Var% holds 'ing'
```

See Also

[Left](#), [Mid](#)

Rmdir

Format

Rmdir("directory name")

Purpose

Removes a directory from a disk

Parameters

- String name of directory to remove

Return Value

The %ERROR% variable hold the error status:

- TRUE Error - directory not found or could not
 be removed possibly because it still contains files
- FALSE Success

Example

Rmdir("C:\SETUP")

See Also

[Chdir](#), [Mkdir](#)

Running a Script

There are two ways in which the Setup utility may be run in order to execute a script file:

Diskette

If Setup is to be run from a diskette then both the SETUP.EXE and INST.EXE programs will need to be copied to the diskette.

The script file must be named SETUP.INF

The SETUP.EXE program copies INST.EXE and SETUP.INF into the Windows directory and then runs the interpreter from there since the user may have requested some diskette changes within the script.

This is a typical installation diskette suite.

Fixed disk

If setup is to be run from a fixed disk then it can be set up as an icon on Windows Program Manager.

In this case, only the INST.EXE program is required and the name of the script file to be executed should be passed as a parameter to the program.

The script file may have any name.

SET command

Format

```
SET %varname% = "text"  
SET %varname% = number  
SET %varname% = %varname%  
SET %varname% = number + number
```

Purpose

Assigns a value to a variable

Parameters

- A string enclosed in double " quotes which in turn may include embedded variable names.
- An integer positive or negative number
- Another variable
- Arithmetic operators + - * and /
- A Predefined Constant

Return Value

None

Comments

The string may contain embedded variables.

Since numbers are held internally in their string form, they may be specified as string parameters.

If an attempt is made to perform an arithmetic operation and one or more of the parameters is not a valid number, a run time error will occur

Example

```
SET %varname% = "Some text"  
SET %varname% = 1024  
SET %varname% = %othervar%  
SET %varname% = IDOK  
SET %newvar% = %varname% * 3  
SET %newvar% = 3 + 4  
SET %newvar% = -2 * %varname%  
SET %newvar% = "%varname%" / "2"
```

See Also

Standards and Notations

SetBackdropText

Format

SetBackdropText(nEntity, "text", bTop, bLeft, nSize, nR, nG, nB)

Purpose

Enables an item of text to be attached to the backdrop display

Parameters

- Number of entity to set (1 - 4)
- The text to display
- A flag specifying the vertical text positioning which may be:
 - TRUE Display text at top of screen
 - FALSE Display text at bottom of screen
- A flag specifying the horizontal text positioning which may be:
 - TRUE Allign text to the left of the screen
 - FALSE Allign text to the right of the screen
- The numeric font size to use
- The numeric Red value for the text colour
- The numeric Green value for the text colour
- The numeric Blue value for the text colour

Note that the last four parameters are optional, defaulting to 50, 255, 255 and 255 respectively if not supplied

Return Value

None

Comments

This function is used to display text on a backdrop display/bitmap and sets one of four entities associated with the backdrop. You can set any number of the four entities or non at all. Entities which are set will be displayed when the backdrop is drawn.

Once set, the text will remain constantly displayed.

The text string may contain embedded carriage return characters | or \n to enable multiple line text to be displayed.

Example

```
// This example displays 'Setup Builder Installation'  
// at the top left of the screen a few lines down  
// in font size 50 (the default) and coloured white (the default)  
SetBackdropText(1, "||Setup Builder Installation", TRUE, TRUE, 50, 255, 255, 255)
```

See Also

[ShowBackdrop](#)

SetFileAttr

Format

SetFileAttr("filename", ronly, hidden, system, archive)

Purpose

Sets the attributes of a file

Parameters

- String name of file to set attributes on
- Numeric 1 or 0 to make file read only
- Numeric 1 or 0 to make file hidden
- Numeric 1 or 0 to make file a system file
- Numeric 1 or 0 to flag the file for archive

Return Value

None

Comments

The file name must NOT contain wildcards

Example

```
SetFileAttr("C:\AUTOEXEC.BAT", TRUE, FALSE, FALSE, FALSE)
```

SetFileDate

Format

SetFileDate("filename", "date")

Purpose

Sets the date of a file

Parameters

- String name of file to set the date of
- The new date in the format dd/mm/yy

Return Value

The %ERROR% variable holds the error status:

- TRUE Error - file not found
- FALSE Success

Comments

The file name must NOT contain wildcards

Example

```
SetFileDate("C:\AUTOEXEC.BAT", "01/12/92")
```

SetFileTime

Format

SetFileTime("filename", "time")

Purpose

Sets the time of a file

Parameters

- String name of file to set the time of
- The new time in the format hh/mm/ss

Return Value

The %ERROR% variable holds the error status:

- TRUE Error - file not found
- FALSE Success

Comments

The file name must NOT contain wildcards

Example

```
SetFileTime("C:\AUTOEXEC.BAT", "09:16:03")
```

ShowBackdrop

Format

ShowBackdrop("bitmap.bmp", bCentre, bTile, bSizetowindow)

Purpose

Displays a background shaded backdrop or a bitmap.

Parameters

- Name of the bitmap file to display
- Flag: Is bitmap is to be centred on screen ?
 - TRUE Centre on display
 - FALSE Display at top left of display
- Flag: Is bitmap to be 'tiled' on screen ?
 - TRUE Tile bitmap on screen
 - FALSE Do not tile the bitmap
- Flag: Is bitmap is to be stretched to fit the screen size ?
 - TRUE Stretch to fit the screen
 - FALSE Display bitmap in its normal size

Return Value

The %ERROR% variable holds the error status:

- FALSE No error
- TRUE Bitmap file not found

Comments

This function supports both 16 and 256 colour bitmaps.

If an empty string "" is supplied as the bitmap or the bitmap file cannot be found, the default shaded blue backdrop will be displayed.

If a small bitmap is centralised on screen it will be surrounded by the default shaded blue backdrop.

You cannot centre and tile at the same time. Similarly, you cannot stretch the bitmap to fit the screen and tile at the same time. The leading parameters take precedence.

If no bitmap is supplied or the bitmap file could not be found, the last three parameters are all ignored.

In order to determine a bitmap to display depending on the screen size, you should use the GetScreenHeight() and GetScreenWidth() functions.

To display application installation specific text on the backdrop, use the SetBackdropText() function

Example

```
// Display default shaded blue backdrop  
ShowBackdrop("", FALSE, FALSE, FALSE)
```

```
// Display a 256 colour tiled bitmap  
ShowBackdrop("c:\windows\256color.bmp", FALSE, TRUE, FALSE)
```

See Also

[SetBackdropText](#), [GetScreenHeight](#), [GetScreenWidth](#)

ShowGroup

Format

ShowGroup("group name", showflag)

Purpose

Displays an existing program group on Windows Program Manager

Parameters

- String name of group to be shown
- Numeric value of type of show which may be:
 - 1 Activates and displays the group window.
If the window is minimized or maximized, Windows restores it to its original size and position.
 - 2 Activates the group window and displays it as an icon.
 - 3 Activates the group window and displays it as a maximized window.
 - 4 Displays the group window in its most recent size and position. The window that is currently active remains active.
 - 5 Activates the group window and displays it in its current size and position.
 - 6 Minimizes the group window.
 - 7 Displays the group window as an icon. The window that is currently active remains active.
 - 8 Displays the group window in its current state. The window that is currently active remains active.

Return Value

None

Comments

The group name is the full name which appears in the caption of the group when it is displayed by Program Manager including spaces.

This command can be used for selecting groups.

Example

```
ShowGroup("Main", 1)
```

```
ShowGroup("Accessories", 2)
```

```
ShowGroup("Visual Basic 3.0", 1)
```

Standards and Notations

Commands

Commands/function/variable names are not case sensitive and with the exception of the open bracket on a function name, the user may use spaces as required between statements.

Wherever a string appears as a parameter to a function or command a variable may also be placed.

Variables

All variable names start with and end with a % character.

A variable name may consist up to 20 characters of the users choice.

You may create up to 50 variables at any one time.

To clear a variable from the variable space, set it to an empty "" string

See Also [Predefined Variables](#)

Strings

Strings always start with and end with a " character. Within a string you may place any combination of characters you wish, however the | (vertical bar) symbol will be translated into a carriage return.

This is useful for creating blank lines in message boxes etc.

By placing text and/or several variables into a string, string concatenation is achieved:

```
"Here is a variable:|%varname% more text"
```

It is also possible to insert special characters into a string by preceding the appropriate decimal ASCII code with a \' character:

```
"Here is a character\13return before the \34return\34 word"
```

This would display as:

```
Here is a character  
return before the "return" word
```

Because numbers can be sometimes preceded by a backslash, for example in directory names, you must use a second slash in this instance:

```
"C:\MYAPP\VERSION\204\SOURCE"
```

Would display as:

```
C:\MYAPP\VERSION\204\SOURCE
```

Numbers

Both positive and negative integer numeric values are supported with a 32 bit range.

See Also [Predefined Constants](#)

Labels

Labels start with a : character and end with a space or the end of the line. Statements on the same line after a label are not executed.

Comments

Comments may be placed in scripts using the standard 'C' language // notation.

STOP command

Format

STOP

Purpose

Terminates all processing of all scripts

Parameters

None

Return Value

None, but all script processing stops.

Comments

This command does not cause a message to appear telling the user that a STOP command has been reached, unlike other languages

See Also

[EXIT](#)

Suggestions for Use

The INST.EXE program is an interpreter program which may have many uses other than just installation scripts.

Here are some suggested uses for the Setup and Setup/Builder software products:

- Application Installation
- Application De-installation
- Windows hosted 'batch/script' programs
- Network logon scripts
- Software version upgrading (eg copying from a network)

UCase

Format

UCase("String")

Purpose

Used in conjunction with the SET command, this function converts a string to upper case

Parameters

- String to convert

Return Value

The return value is a string and is assigned to the variable in the SET statement

Example

```
Set %Var% = UCase("Test String")      // %Var% holds 'TEST STRING'
```

See Also

[LCase](#)

UnCompress

Format

UnCompress("source name", "target name", delete)

Purpose

Uncompresses a file previously compressed using the Microsoft COMPRESS.EXE program, renaming it and optionally deletes the source file afterwards

Parameters

- Name of the file to uncompress
This name may contain a path name but must not contain wildcard specifications
- Name of file to rename to
This name may contain a path name but must not contain wildcard specifications
- TRUE Delete after uncompress
FALSE Do not delete after uncompress (Default if not supplied)

Return Value

The %ERROR% variable holds the error number

Example

```
UnCompress("A:\ABC.EX_", "C:\TEST\ABC.EXE")
```

```
UnCompress("C:\ABC.EX_", "C:\ABC.EXE", TRUE)
```

Wait

Format

Wait("Window Caption", option)

Purpose

Performs a wait in a script until another window has become visible or it disappears

Parameters

- String name of window caption to check
- Option required which may be:
 - 0 Wait until window appears
 - 1 Wait until window disappears
 - 2 Wait until window appears and then disappears

Return Value

None

Comments

This function should be used to synchronise events within the Windows environment. If another application is started using the WinExec() function for example, script processing will continue due to the multitasking nature of Windows. The Wait() function can be used to stop a script from continuing until a window appears/disappears.

The 'caption' is the text which appears in the title bar of the window being checked for

Example

This example starts up Windows Notepad and waits until the user closes Notepad down. It then displays a message to show completion.

```
WinExec("notepad")
Wait("Notepad - (Untitled)", 2)
MessageBox("Notepad closed down", "Test", 0, MB_OK)
```

See Also

[Release](#)

What is Setup ?

Setup is a utility program which can be used to interpret script files written by a user to install applications in a Windows-hosted environment.

It may also be used as a tool for automating configurations of software and as a 'Windows hosted batch file interpreter'.

WinExec

Format

WinExec("Filename")

Purpose

Executes another program

Parameters

- String name of program file to be run

Return Value

The %ERROR% variable holds the return value of the standard Windows WinExec function

Example

WinExec("notepad.exe")

WritePrivateProfileString

WriteProfileString

Format

```
WriteProfileString("section" , "entry",  
                  "setting", "file name")
```

Purpose

Writes a string to a Windows .INI file.

Parameters

- String [Section] of .INI file to write to
- String entry within the section to write
- String data to be written for the entry
- String name of .INI file to write to

Return Value

None

Comments

Along with the standard windows function, if no path is specified in the .INI file name, writing defaults to the Windows directory

This function is useful for making changes to WIN.INI or SYSTEM.INI during your installation procedures.

The setup procedure which installs the Setup software for you will optionally use this function to create a File Manager file association for you.

The WritePrivateProfileString function is only supplied for compatibility with earlier versions of Setup. You should use the WriteProfileString function.

Example

```
WriteProfileString("Extensions", "inf", "inst.exe ^inf", "win.ini")
```

See Also

[GetProfileString](#)

WriteLine

Format

WriteLine("string")

Purpose

Writes a line to the currently open output file

Parameters

- The text to write to the file

Return Value

The %ERROR% variable holds the error status:

- 0 Success
- 1 File not open

Comments

The string may contain embedded variables.

This function will only write to ASCII files and automatically appends a CR/LF after any text written to a file

Example

This example reads the CONFIG.SYS file and writes it out to TEMP.DAT having changed the FILES entry to 100

```
OpenIn("C:\CONFIG.SYS")
IF %ERROR% == TRUE GOTO :OPENERROR
```

```
OpenOut("C:\TEMP.DAT")
:NEXTLINE
ReadLine(%Buffer%)
IF %ERROR% == EOF GOTO :EOF
```

```
SET %Ptr% = Instr(1, %Buffer%, "FILES=")
IF %Ptr% == 0 GOTO :NOTFOUND
```

```
Set %Buffer% = "FILES=100"
```

```
:NOTFOUND
WriteLine(%Buffer%)
GOTO :NEXTLINE
```

```
:EOF
CloseIn()
CloseOut()
```

```
.
```

```
:OPENERROR
```

See Also

[CloseIn](#), [CloseOut](#), [OpenIn](#), [OpenOut](#), [ReadLine](#)

