CALL command

Format
CALL "filename"

Purpose
Calls another script file as a subroutine script

Parameters
- String name of the script file 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

CentreDialog

Format
CentreDialog()

Purpose
Centres a User Defined Dialog on the screen when the dialog is first displayed

Parameters
None

Return Value
None

Comments
The CentreDialog function causes a User Defined Dialog to be positioned centrally on screen, both vertically and horizontally, when the dialog is first displayed.

Example
Please see the example in the CreateDialog function documentation.

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")

Close

Format
Close()
Close(stream)

Purpose
Closes a file stream, freeing it for later re-use

Parameters
- Optional stream number of file to close (1 - 10)

Return Value
None

Comments
The function closes the file in the specified stream. If there is no file open in the specified stream, this function is ignored.
The stream number parameter is optional. If omitted, all file streams will be closed

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

```
Open("C:\CONFIG.SYS", 1, READ)
IF %ERROR% == TRUE GOTO :OPENERROR

Open("C:\TEMP.DAT", 2, WRITE)
:NEXTLINE
ReadLine(1, %Buffer%)
IF %ERROR% == EOF GOTO :EOF

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

Set %Buffer% = "FILES=100"

:NOTFOUND
WriteLine(2, %Buffer%)
GOTO :NEXTLINE

:EOF
Close(1)
Close(2)

.
.

:OPENERROR
```

See Also
Open

Command Directory

CALL	CALL another script
EXIT	EXIT to higher level script
GOTO	Control branching command
IF	Comparison command
SET	Variable assignment command
STOP	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.

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"
```

GP SOFTWARE

GRAHAM PLOWMAN SOFTWARE

This Windows Help file was written by Graham Plowman
using Help Builder Version 1.09 and refers to:

Setup Version 4.04.001 / 03/04/96

CreateControl

Format
CreateControl("object type", "caption text", nId, nX, nY, nWidth, nHeight, nStyle, [%variable%[, nEditLength]])

Purpose
Creates a control on a user defined dialog which has been previously created by the CreateDialog function

Parameters

- String name of the control type to create. This may be:
  - icon
  - text
  - edit
  - groupbox
  - button
  - radiobutton
  - checkbox
- String text/caption of the control. For an icon, this is the name of the icon.
- Numeric Id of the control (relevant to edit, button, radiobutton, checkbox)
- Numeric X coordinate of control in dialog units
- Numeric Y coordinate of control in dialog units
- Numeric width of control in dialog units
- Numeric height of control in dialog units
- Numeric style of the control
- Variable to retrieve default value from and place resulting input (edit, radiobutton, checkbox only)
- Numeric number of characters to limit an edit control to

Return Value
None

Comments
Icon names
Setup has all the icons from the \ICONS\COMPUTER directory supplied with Visual Basic 3 encoded into it. All of the icons have the same names as the corresponding Visual Basic icon files, less the file extension. To use any of these icons, simply place the name in the caption/text parameter of the CreateControl function eg "Disk01".
There is also an additional "Setup" icon.

Styles
The Setup interpreter supplies the basic default styles necessary to display controls, however, nearly all of the Windows controls have specific attributes which may be selected using the style parameter of the CreateControl function. Listed below are the styles for each type of control, together with a number. To use any one or more of the styles for a control, simply add the numbers together and insert the result as the style parameter to the CreateControl function.

text

0	Allign left text (default)
1	Center text
2	Right allign text
8388608	Border

edit

	0	Allign left text (default)
	1	Center text
	2	Right allign text
	4	Multiline edit
	8	Convert to upper case
	16	Convert to lower case
	32	Display as password (*) characters
	64	Auto vertical scroll
	128	Auto horizontal scroll
	2048	Read-only edit field
	4096	Want return (multiline edit only)
	8388608	Border (default - required for CTL3D)
button		
	1	Make button the 'default' 'Ok' button
groupbox		None
icon		None
radiobutton		
		Allign text right (default)
	16	Allign text left (warning: CTL3D doesn't display correctly)
checkbox		
		Allign text right (default)
	16	Allign text left (warning: CTL3D doesn't display correctly)

Numeric Ids
Numeric identifiers are used by controls which can receive input from the user and provide an internal link for the Setup program to set and retrieve values from controls. Within a dialog, they must be unique. All controls which don't accept user input such as icon, text, groupbox should be given an Id of -1 which denotes 'not used'. It is recommend that to avoid conflict, you start numbering Ids from 20 because the lower numbers are used by the values IDOK, IDCANCEL, IDYES, IDNO etc
Note that push buttons will only respond to the standard message box key Predefined Constants.

Variables
edit, radiobutton an checkbox controls all accept input from the user. Setup provides a simple way of retrieving values from these controls by simply populating a variable with the value entered in the control. The variable is also used to 'pre-populate' a control as well. A variable should always be supplied for a control which accepts user input.

Example
Please see the example in the CreateDialog function documentation.

CreateDialog

Format
CreateDialog("template name", "caption", nX, nY, nWidth, nHeight)

Purpose
Creates a user defined dialog template in memory

Parameters
- String name of the template to create
- String caption to be displayed on the dialog
- Numeric X coordinate of dialog in dialog units
- Numeric Y coordinate of dialog in dialog units
- Numeric width of dialog in dialog units
- Numeric height of dialog in dialog units

Return Value
None

Comments
The Setup program has an internal table reserved for creating up to 10 user-defined dialogs.
A dialog is created in this table by using the CreateDialog function, although creating a dialog does not display it. To display a dialog a call must be made to the DialogBox function, passing the template name of the user defined dialog.
It is possible to create dialogs with the same name as any of the in-built dialog templates and by doing so, you effectively overide the in-built dialog template since user defined template names take precedence over in-built template names.
The lines in the script file following the CreateDialog function call must contain calls to the CreateControl function to define the controls on the dialog. A blank line should follow the last dialog definition statement.
It is possible to create a user defined dialog with the same name as an existing user defined dialog. In this situation, the interpreter automatically deletes the old template and then creates a new template. This is exactly the same as manually calling DestroyDialog before creating a dialog.

Example
The following example creates a typical setup dialog which requests a user to enter the path names for the various components of an application being installed. You can paste this example directly into a script, but make sure that the SetFocus statement at the end is followed by a blank line.

```
CreateDialog("Directories","Directories", 40, 40, 260, 145)
CreateControl("groupbox","",-1,6,4,247,110,0)
CreateControl("icon","setup",-1,15,20,0,0,0)
CreateControl("text","Please enter the directories in which to place the %Application% software files:",-
1,50,20,170,24,0)
CreateControl("text","Application Executeables",-1,30,55,145,8,0)
CreateControl("edit","",10,130,53,100,12,0,%Exes%,40)
CreateControl("text","Sample script files",-1,30,71,145,8,0)
CreateControl("edit","",10,130,69,100,12,0,%Script%,40)
CreateControl("text","Help files",-1,30,87,145,8,0)
CreateControl("edit","",10,130,85,100,12,0,%Help%,40)
CreateControl("button","&Ok",IDOK,6,126,40,14,1)
CreateControl("button","&Cancel",IDCANCEL,52,126,40,14,0)
CentreDialog()
SetFocus(10)
```

See Also
CreateControl, CentreDialog, SetFocus, DialogBox

Creating a setup procedure

To create a setup procedure the following components are required, although SETUP.EXE is only required in certain situations:

- The SETUP.EXE installer program
- The INST.EXE interpreter program
- The DIBAPI.DLL bitmap handling DLL
- The CTL3DV2.DLL 3d control handling DLL
  This is already installed on most PC's today
- The CLEANUP.EXE tidy-up utility
- 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 of the above files are required on the first diskette of your installation suite.
The SETUP.EXE program copies the INST.EXE, DIBAPI.DLL and SETUP.INF files to the WINDOWS\TEMP directory (which it also creates) of the machine on which the program is running. It also copies CTL3DV2.DLL to your WINDOWS\SYSTEM directory if the file does not already exist or it is an older version than that supplied with Setup Builder. 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 which would otherwise cause INST.EXE to fail if it was running from a diskette. You cannot run an executeable from a diskette under Windows and then remove the diskette!
INST.EXE does not remove itself, DIBAPI.DLL or SETUP.INF from the hard disk when it has completed. To do this, the script must run the CLEANUP.EXE program from your last installation diskette. This is done automatically by installation procedures created using Setup Builder.

WARNING: If SHARE.EXE is running and your script file attempts to delete INST.EXE, DIBAPI.DLL 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, DIBAPI.DLL 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 Windows Program manager as an icon with the script file name as a parameter. You can also set up a file association with 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 Builder software on your machine.

Procedure for creating install suites
In order to create a successful windows hosted software installation procedure it is adviseable 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.
Note that you cannot delete a file if it is currently opened via the Open function. A file cannot be deleted if it is currently open in another application or it in use by Windows

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.
If the specified group does not exist, this function has no effect.

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.
If the icon does not exist, this function has no effect.

Example
DeleteIcon("Editor")

See Also
MakeIcon

DestroyDialog

Format
DestroyDialog("template name")

Purpose
Deletes a dialog from the internal template table, freeing memory for creation of new dialogs

Parameters
- String name of the template to destroy

Return Value
None

Comments
Destroying a dialog simply removes the dialog template definition from the internal table memory, freeing space for another dialog to be created. Dialogs may be freely created and destroyed at any time. Attempting to destroy a dialog which does not exist, has no affect.
Note that you cannot destroy the in-built dialog templates, however, if you create a user defined dialog to overide an in-built dialog, deleting the user-defined template restores the use of the in-built template.

Example
DestroyDialog("MyDialog")

See Also
CreateDialog

DialogBox

Format
DialogBox("dialog name")

Purpose
Activates one of the predefined dialog boxes or a user-defined dialog previously created by the CreateDialog function.

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

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
•	"DEINSTALL"	Displays the de-install option dialog

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

•	%CAPTION%	Caption of the dialog
•	%MESSAGE1% to %MESSAGE6%	First static text field within the dialog 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% to %BUTTON6%	Preset status of check boxes in CHECKBOX dialogs. A '1' signifies a check ie 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% to %PUSHB_5%	Holds the text to be displayed on the push 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, CreateDialog, Predefined Constants, User Defined Dialogs

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 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 must not be 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

29 Unable to open script file
This error occurs when an attempt is made to CALL a nested script file which cannot be found.
See CALL

30 Attempt to open too many nested script files
This error occurs when an attempt is made to CALL a nested script file when the maximum number of nested script files allowed has been reached.
See CALL

31 Invalid dialog command
A command has been found within some script which is creating a user defined dialog and the command is not recognised as a dialog related command.
When creating user-defined dialogs, you can only use the dialog commands within your script until the terminating blank line at the end of the dialog creation script.
See User Defined Dialogs

32 Attempt to create too many dialogs
An attempt has been made to create more than the maximum allowed number of user defined dialogs.
See CreateDialog

33 Attempt to create too many controls
An attempt has been made to create more than the maximum allowed number of controls in a user defined dialog.
See User Defined Dialogs

34 Invalid control type
This error occurs when an invalid control name to be created is specified for the CreateControl function.

35 Failed to allocate memory for internal use
This error occurs usually only in low-memory situations and normally only when handling user-defined dialogs.

36 Failed to initialise user defined dialog
This error occurs when the DialogBox function is used to activate a user-defined dialog and the Windows API InitModalIndirect function (which is used to initialise the dialog resource template for the user-defined dialog) fails.
This normally only occurs in low memory situations.

37 Invalid file stream specified
A file stream has been specified for one of the ASCII file handling functions and the stream is not in the valid range.
See Open, Close, ReadLine, WriteLine

38 Specified file stream is already in use
An attempt has been made to open a file in a stream which is already open. A stream can only be open for

reading or writing but not both at the same time.
See Open

39 Specified file stream is not open
An attempt has been made to read data from or write data to a file stream which is not currently open.

40 Specified file stream is not open for reading
An attempt has been made to read from a file stream which has been opened for writing.
See Open

41 Specified file stream is not open for writing
An attempt has been made to write to a file stream which has been opened for reading.
See Open

42 Invalid target file name
An attempt has been made to use the CopyFile function to copy a file to an invalid target file name.

43 Unsupported language specified
An attempt has been made to use the SetLanguage function to select a language which is not supported by the script interpreter.

44 Failure while reading file
An error has occured while reading a file during file copying. This is normally due to a corrupted file or disk.

45 Failure while writing file
An error has occured while writing a file during file copying. This is normally due to a corrupted file or disk.

46 Failed to create target file
The specified target file name could not be created, either because it is an invalid name, the file/disk is write protected or a disk failure occured.

47 Failed to initialise DDEML
Setup has failed to initialise the DDEML Dynamic Link Library which is used to manage DDE conversations between Setup and Program Manager.
This could be because DDEML.DLL is missing or it is an old version.

48 Failed to start DDE coversation with Program Manager
Setup has failed to start a DDE conversation with Program Manager.
This could be because Program Manager has failed to respond or because it is not running. It can also occur when a replacement Program Manager product does not support the 'Shell Dynamic Data Exchange' protocol supported by Program Manager for maintaining groups and icons.

49 A DDE error has occured while communicating with Program Manager
This error occurs if a Program Manager function fails due to a communications failure in the underlying DDE transaction.

50 Invalid file mode
This error occurs when an attempt has been made to open a file using an invalid file mode. The file mode must be READ, WRITE or APPEND
See Open

51 Label stack full
This error occurs when an attempt is made to execute a script which has too many labels.

The interpreter opens a script file and reads all the labels within the script, appending them to a label stack. This is done so that the GOTO command can be executed directly to the appropriate location in the script. The stack which stores all the labels has a limited size. This error occurs when there are too many labels to add to the stack - around 400 are allowed.
The error normally only occurs when a script is first executed or the CALL statement is used to execute a nested script file.
See CALL

52 Unable to find or open script file
This error occurs when an attempt is made to run the script interpreter and pass it an invalid or non-existant script file name to run.

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

GetDate	Get the system date
GetTime	Get the system time
GetFileDate	Get file date
GetFileTime	Get file time
SetFileDate	Set file date
SetFileTime	Set file time

Disk & Directory Functions

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

File Related Functions

CheckExists	Check if file exists
Close	Close a file
ConfirmOverwrite	Allows user to confirm a file overwrite
CopyFile	Copy file(s) from one location to another
Delete	Delete a file
GetBackupName	Get a unique backup file name
GetFileAttr	Get file attributes
GetFileLength	Get file length
Open	Open a file for reading or writing
ReadLine	Read a line from the input file
Rename	Rename a file to another name
SetFileAttr	Set file attributes
UnCompress	Un-compress a file, renaming
WriteLine	Write a line to the output file

Language Functions

SetLanguage	Set the system language

Program Manager Functions

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

Registry Functions

Register	Register a file in the registry
UnRegister	De-Register a file from the registry

String Manipulation Functions

Instr	Find one string in another
LCase	Convert string to lower case

Left	Get left n characters of a string
Len	Get length of string
Mid	Get a sub-string from a string
Right	Get the right n characters of a string
UCase	Convert a string to upper case

User Defined Dialog Functions

CentreDialog	Centres a dialog on screen
CreateControl	Create a control in a user dialog
CreateDialog	Create a user-defined dialog
DestroyDialog	Destroy a user-defined dialog
SetFocus	Set control focus in a user dialog

Windows Interface/API Functions

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

GetBackupName

Format
GetBackupName("filename.ext", %variable%)

Purpose
Given a file name, creates a unique backup file name from it

Parameters
- String name of the file
- Variable to store the result in

Return Value
None

Comments
This function is useful when creating backup copies of files. Given a file name, it evaluates a unique file name which is guaranteed not to already exist.
The parameter file name may optionally contain a drive letter, path specification and file extension, although the later will always be replaced upon returning.
The returned file name will always contain any drive/path specifications which were specified in the parameter file name.
The example below best demonstrates how the function operates.

Example
GetBackupName("C:\AUTOEXEC.BAT", %File%)

The above example would place the file name 'C:\AUTOEXEC.001' in the %File% variable. If this file already exists, then 'C:\AUTOEXEC.002' would be returned and so on.

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 chacked.
  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 occured

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%)
```

Notes

Please note that some of the Windows .INI component files have duplicate 'entry' names. The SYSTEM.INI [386Enh] section is a good example of this where there are multiple Device= entries.
The Setup Script language does not support reading and writing of such entries: it only supports unique entry names. Indeed, the Windows API functions which the script language functions map onto do not support duplicate entry names either. To handle such entries some script code could be written which enters a loop to read every line of the .INI file, writing them to a temporary file and making adjustments at the same time. The resulting temporary file would then be renamed or copied over the original .INI file.

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 active group in Program Manager. 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
- String name of new directory to create

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

Open

Format
Open("filename", stream, mode)

Purpose
Opens a file on a disk for reading or writing

Parameters
- String name of the file to open
- Stream number to open the file in (1 - 10)
- File mode which may be:

READ	to open for reading
WRITE	to open for writing
APPEND	to open for append (add to the end of the file)

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 but not wildcard characters.
A file may be opened for reading or writing, but not both.
When a file is opened, it is opened in a stream. A stream number is any value from 1 to 10. This means that up to 10 files may be opened at a time. The file stream number is then used to identify the file for reading, writing and closing.
An error will occur if you attempt to open more than one file in a stream.
Note that if you use Setup Builder to automatically build an installation suite, stream 10 is used by the 'de-install' facility (only if you selected to use the de-install facility), so you should not use it in any 'user-script' that you may add to the setup procedure

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

```
Open("C:\CONFIG.SYS", 1, READ)
IF %ERROR% == TRUE GOTO :OPENERROR

Open("C:\TEMP.DAT", 2, WRITE)
:NEXTLINE
ReadLine(1, %Buffer%)
IF %ERROR% == EOF GOTO :EOF

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

Set %Buffer% = "FILES=100"

:NOTFOUND
WriteLine(2, %Buffer%)
```

```
GOTO :NEXTLINE

:EOF
Close(1)
Close(2)

.
.

:OPENERROR
```

See Also
Close

Overriding In-Built Dialogs

The INST.EXE Setup script interpreter supports the ability to create user-defined dialogs such that scripts can be written with user-tailored dialogs. Up to 10 dialogs may be created and they may contain up to 50 controls each.
User-defined dialogs are created as 'templates' which have a unique name determined by the developer of the install procedure. The script interpreter also has a large number of predefined dialog templates which are listed in the documentation of the DialogBox function.

Often, the developer may want to replace one or more of the standard dialogs in the standard install procedure by overriding with some user-defined dialogs. This topic discusses how it can be done.

Please refer to the DialogBox function for details on the dialog you wish to override.
All of the standard dialogs have controls which have numeric IDs as listed in the DialogBox function help.
When creating a user defined dialog you must place the same controls on your dialog with the same numeric IDs as those in the standard dialog your are overriding. It is up to you how the controls are positioned and what size they are.
The following show some example script of how the PUSHB2 dialog would be overridden. The example was actually used in older versions of the Setup Builder product install procedure.

```
CreateDialog("PUSHB2","%Caption%", 40, 40, 260, 145)
CreateControl("groupbox","",-1,6,4,247,110,0)
CreateControl("icon","setup",-1,15,20,0,0,0)
CreateControl("text","Please make a selection from the following options:",-1,60,30,170,24,0)
CreateControl("button","&Install,IDOK,38,56,40,14,1)
CreateControl("text","Install the %Application% software",-1,100,58,145,8,0)
CreateControl("button","&De-Insall",IDBUTTON2,38,80,40,14,1)
CreateControl("text","Un-Install the %Application% software",-1,100,82,145,8,0)
CreateControl("button","Cancel",IDCANCEL,6,126,40,14,0)
CentreDialog()
SetFocus(IDOK)
```

All three buttons have the same IDs as those in the PUSHB2 dialog. Although not used in the example, the text controls may also contain text strings which have variable names embedded which are the same as those used by the standard dialog. In this way, the user defined dialog becomes an exact replacement for the standard dialog.

Note that the ASKPATH dialog performs some checking to see if the directory name entered was valid and creates it if it is. Similarly, the LICENSING dialog writes the two fields to the application .INI file.
It is therefore not possible to create a direct replacment for the ASKPATH dialog without some extra supporting code. Likewise, a replacement for the LICENSING dialog would also require supporting code to handle the string values entered in the edit fields.

See Also
User Defined Dialogs

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\

WINDOWSDIRECTORY
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(stream, %Variable%)

Purpose
Reads a line from the specified file stream

Parameters
- The numeric stream of the file (1-10)
- The variable into which to read the line

Return Value
The %ERROR% variable holds the error status:

•	0	Success
•	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
To be able to read from a file, the file must have previously been opened in READ mode using the Open function. The file must have been opened with the same stream number as that passed to this function.
An error will occur if you try to read from a stream which has not been opened in READ mode

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

```
Open("C:\CONFIG.SYS", 1, READ)
IF %ERROR% == TRUE GOTO :OPENERROR

Open("C:\TEMP.DAT", 2, WRITE)
:NEXTLINE
ReadLine(1, %Buffer%)
IF %ERROR% == EOF GOTO :EOF

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

Set %Buffer% = "FILES=100"

:NOTFOUND
WriteLine(2, %Buffer%)
GOTO :NEXTLINE

:EOF
Close(1)
Close(2)

.
.
```

```
:OPENERROR
```

See Also
Close, Open, WriteLine

Register

Format
Register("filename.ext")

Purpose
Adds a file to the shared file registry

Parameters
- String fully qualified file name of the file being added

Return Value
None

Comments
Often during an installation of some software, it is common to place files in the \WINDOWS or \WINDOWS\SYSTEM directory. Sometimes, the files placed in these directories are used by more than one application, for example, .DLL files. At some point, it is likely that one or more of the applications sharing these common files will be de-installed, causing a potential problem with files that have been installed for shared use, especially if the shared files are deleted.
To resolve the problem of shared files being de-installed, Setup Builder supports a 'shared file registry' system. In simple terms, the registry is just a list of files with a count of the number of applications using them. As applications are installed, the counts are increased and when applications are de-installed, the counts are decreased. When counts reach zero, files are deleted.
The registry itself is simply the REGISTRY.INI file in your \WINDOWS directory. The file contains a list of all the files which have been registered as shared, together with a count of the number of applications which are currently installed which use the files.
When a file is registered, it may already exist in the registry (another application is already using it), in which case its count is simply increased by one. If it isn't already registered, the file is added to the registry with a count of 1. Before a file is registered, the Register function checks to see if the file exists. In this way, files which have already been installed by other applications which are not in the registry can be accounted for in the registry, so it is possible that if you Register a file, its count may start at 2 if the file already existed.
To integrate with this facility, Setup Builder creates script code which registers a file before it is actually installed/copied.
The Setup Builder de-install facility integrates with the shared file registry.
When a piece of software is de-installed, files are usually simply deleted, however, files which have been registered in the registry must only be deleted if their share count reaches zero. Therefore, shared files must be removed with the UnRegister function and not the Delete function. UnRegister retrieves the share count for a file and decreases it, writing it back to the registry. The file is not deleted until the share count reaches zero (ie all applications using the file have been de-installed) and then the registry entry for the file is removed - the file is no longer registered as being shared.
The registry functions do not support wild card file names

Example
```
Register("C:\WINDOWS\SYSTEM\CTL3DV2.DLL")
' Install the file in/overwrite check etc
STOP

' De-Install
UnRegister("C:\WINDOWS\SYSTEM\CTL3DV2.DLL")
```

See Also
UnRegister

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.

Comments
An empty or invalid/non-existant group name parameter will cause this function to have no effect.
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 of 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", rdonly, 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")

SetFocus

Format
SetFocus(nId)

Purpose
Sets the control within a User Defined dialog that will receive the focus when the dialog is first displayed

Parameters
- Numeric Id of the control to have focus

Return Value
None

Comments
The SetFocus function sets the control which will receive focus when a User Defined dialog is first displayed. The numeric Id parameter is the same as that supplied as a parameter to one of the CreateControl calls when the dialog was created. This is why all Ids should be unique within a dialog: if they're not, SetFocus() will give focus to the first control it finds with the given Id.

Example
Please see the example in the CreateDialog function documentation.

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")

SetLanguage

Format
SetLanguage("language")

Purpose
Specifies to the interpreter the language to display all system generated text on buttons and dialogs

Parameters
- A string specifying the language which may be:
  - UkEnglish
  - USEnglish
  - French
  - German
  - Italian
  - Spanish

Return Value
None. An unsupported language name causes a run-time error

Comments
The default language used by the interpreter if the SetLanguage function is not called is that specified by the [intl] iCountry= setting in the WIN.INI file. If this is not available, or it specifies an unsupported language, the default is UkEnglish.
The interpreter only supports the languages listed above.
The language name is not case sensitive

Example
```
SetLanguage("German")
```

This would cause all system displayed text on buttons and dialogs to appear in the German language

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.
If the specified group does not exist, this function has no effect.

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.
Whereever 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 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 nesting variables in a string, string concatenation is achieved:

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

It is also possible to insert special characters in 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
```

Up to three numeric digits following the ^ character are taken as the ASCII code, so for example:

```
"Here is ^0656B"
```

would display as:

```
Here is A6B
```

If less than three numeric digits are given before a non-numeric, then all the numeric digits supplied up to the non-numeric are taken as the ASCII code as in the first example above with ^13 and ^34
Note that an ASCII code zero will cause a string to be terminated as in the 'C' language.
To insert a ^ character in a string, simply place it in the string twice:

```
"Here is a ^^ character"
```

would display as:

```
Here is a ^ character
```

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 at 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)

UnRegister

Format
UnRegister("filename.ext")

Purpose
Removes a file to the shared file registry

Parameters
- String fully qualified file name of the file being removed

Return Value
None

Comments
Please see the Register function for a full description of the shared file registry

Example
UnRegister("C:\WINDOWS\SYSTEM\CTL3DV2.DLL")

See Also
Register

User Defined Dialogs

The INST.EXE Setup script interpreter supports the ability to create user-defined dialogs such that scripts can be written with user-specifically tailored dialogs. Up to 10 dialogs may be created and they may contain up to 50 controls each. It is possible to create dialogs containing the following controls:

- Icons
- Static text fields
- Edit fields
- Group boxes (border lines)
- Push Buttons
- Radio Buttons
- Check boxes

User defined dialogs are maintained using the following script language functions:

CreateDialog
CreateControl
CentreDialog
SetFocus
DestroyDialog

User defined dialogs are handled in terms of 'templates' and each template has a unique identifying name. This name may be the same as an in-built dialog, providing a way to override in-built dialogs.
A dialog template must always be created first using the CreateDialog function before any user defined dialog functionality can be performed.
CreateDialog allocates space in an internal table reserved for holding user-dialog templates, ready for controls to be added.
CreateControl adds controls to a template and SetFocus marks the template with the Id of the control which will have focus when the dialog is first displayed.
When a dialog is finished with it may be deleted using the DestroyDialog function.
It is possible to create a template at the begining of a script and use it several times via the DialogBox function. In this way you do not have to keep creating and destroying dialogs.

User defined dialogs are activated using the standard DialogBox function, passing the name of a template instead of one of the in-built standard template names.

In order to provide a convenient way to set field values and retrieve values from fields, when controls are created they may have a variable name attached. This variable will be read to pre-populate dialog fields and when the user presses Ok only (none of the other buttons), the variable will be populated with the value in the field. Therefore, simply refering to variables provides an easy method to pass data to and from a dialog.

The coordination system used by dialogs is that of 'dialog units' discussed in the Windows SDK help. These ensure that whatever screen resolution or font size your Windows system uses, dialogs will always be a consistent size with the system font: no edit fields too high or low to show all characters.

User defined dialogs can be used to define any screen the user wishes, subject to the use of the standard Windows controls supported.

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 funtion should be used to synchronise events within the Windows environment. If another application is started using the WinExec() funtion for example, script processing will continue dur to the multitasking nature of Windows. The Wait() funtion 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")
```

Notes
Please note that some of the Windows .INI component files have duplicate 'entry' names. The SYSTEM.INI [386Enh] section is a good example of this where there are multiple Device= entries.
The Setup Script language does not support reading and writing of such entries: it only supports unique entry names. Indeed, the Windows API functions which the script language functions map onto do not support duplicate entry names either. To handle such entries some script code could be written which enters a loop to read every line of the .INI file, writing them to a temporary file and making adjustments at the same time. The resulting temporary file would then be renamed or copied over the original .INI file.

See Also
GetProfileString

WriteLine

Format
WriteLine(stream, "string")

Purpose
Writes a line to the specified file stream

Parameters
- The numeric stream of the file (1-10)
- The text to write to the file

Return Value
The %ERROR% variable holds the error status:

0	Success
1	Error

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.
To be able to write to a file, the file must have previously been opened in WRITE mode using the Open function. The file must have been opened with the same stream number as that passed to this function.
An error will occur if you try to read from a stream which has not been opened in WRITE mode

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

```
Open("C:\CONFIG.SYS", 1, READ)
IF %ERROR% == TRUE GOTO :OPENERROR

Open("C:\TEMP.DAT", 2, WRITE)
:NEXTLINE
ReadLine(1, %Buffer%)
IF %ERROR% == EOF GOTO :EOF

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

Set %Buffer% = "FILES=100"

:NOTFOUND
WriteLine(2, %Buffer%)
GOTO :NEXTLINE

:EOF
Close(1)
Close(2)

.
.
```

:OPENERROR

See Also
Close, Open, ReadLine