Click Here

Click Here

Click Here

Click Here

Command List

IsLicensed
IsMenuChecked
IsMenuEnabled
IsNumber
IsRunning
ItemCount
ItemExtract
ItemSelect
LastError
LogDisk
Max
Message
Min
MouseInfo
NetAddCon
NetBrowse
NetCancelCon
NetDialog
NetGetCaps
NetGetCon
NetGetUser
Num2Char
OtherDir
OtherUpdate
ParseData
Pause
PlayMedia
PlayMidi
PlayWaveForm
Random
Return
RunHide
RunIcon
Run
RunZoom
SendKey
SetDisplay
SKDebug
SnapShot
Sounds
StrCat
StrCmp
StrFill
StrFix
StriCmp
StrIndex
StrLen
StrLower
StrReplace
StrScan
StrSub
StrTrim
StrUpper
Terminate
TextBox

TextSelect
Version
WaitForKey
WallPaper
WinActivate
WinArrange
WinCloseNot
WinClose
WinConfig
WinExeName
WinExist
WinGetActive
WinHide
WinIconize
WinItemize
WinMetrics
WinName
WinParmGet
WinParmSet
WinPlaceGet
WinPlaceSet
WinPlace
WinPosition
WinResources
WinShow
WinState
WinTitle
WinVersion
WinWaitClose
WinZoom
Yield

Click Here

Click Here

Click Here

Click Here

Task Outline

Clipboard and Data Exchange
Dialogs, Messages, Data Entry
Directory Maintenance
File Maintenance
INI File Maintenance
Language Specifics, Error Codes
Launching Applications
Multimedia Controls
Network Administration
PC System Information
Windows Session Control
WIL Programming Language

Clipboard and Data Exchange

ClipAppend
ClipGet
ClipPut
DDEExecute
DDEInitiate
DDEPoke
DDERequest
DDETerminate
DDETimeout
IsKeyDown
SendKey
SnapShot
WaitForKey

Dialogs, Messages,Data Entry

AskLine
AskYesNo
Beep
DialogBox
Display
Execute
ItemCount
ItemExtract
ItemSelect
Message
Pause
Terminate
TextBox

Directory Maintenance

DirChange
DirGet
DirHome
DirItemize
DirMake
DirRemove
DirRename
DirWindows

File Maintenance

CurrentFile
FileAppend
FileAttrGet
FileAttrSet
FileClose
FileCopy
FileDelete
FileExist
FileExtension
FileHilite
FileItemize
FileLocate
FileMove
FileOpen
FilePath
FileRead
FileRename
FileRoot
FileSize
FileTimeGet
FileTimeTouch
FileWrite

INI File Maintenance

IniDelete
IniDeletePvt
IniItemize
IniItemizePvt
IniRead
IniReadPvt
IniWrite
IniWritePvt

Language Specifics, Error Codes

Arithmetic Operators
Abs
Average
Max
Min
Random

Command Post Specific Commands
IsRunning
OtherDir
OtherUpdate
IsLicensed
IsMenuChecked
IsMenuEnabled
IsRunning
MenuChange
#Nextfile

Error Commands
LastError
ErrorMode
Error Messages

Variables
Drop
IsDefined
IsNumber
Variables Explained

Flow Control–Decisions
Delay
Exit
Goto
If...Then

Sub Programs (Procedures)
Call
CallExt
Return
Debugging Procedures
Debug
SKDebug

String (Text) Manipulation
Char2Num
Num2Char
ParseData
StrCat
StrCmp
StrFill
StrFix
StriCmp

StrIndex
StrLen
StrLower
StrReplace
StrScan
StrSub
StrTrim
StrUpper
TextSelect

Language Explained (WIL)

Launching Applications

Run
RunHide
RunIcon
RunZoom

Multimedia Controls

PlayMedia
PlayMidi
PlayWaveForm
Sounds

Network Administration

AskPassword
DiskFree
DiskHide
DiskReset
DiskScan
DiskUpdate
LogDisk
NetAddCon
NetBrowse
NetCancelCon
NetDialog
NetGetCaps
NetGetCon
NetGetUser

PC System Information

DateTime
DOSVersion
Version
Environment
MouseInfo

Windows Session Control

EndSession
Exclusive
IconArrange
IgnoreInput
IntControl
SetDisplay
WallPaper
WinActivate
WinArrange
WinClose
WinCloseNot
WinConfig
WinExeName
WinExist
WinGetActive
WinHide
WinIconize
WinItemize
WinMetrics
WinName
WinParmGet
WinParmSet
WinPlace
WinPlaceGet
WinPlaceSet
WinPosition
WinResources
WinShow
WinState
WinTitle
WinVersion
WinWaitClose
WinZoom
Yield

#Nextfile (Command Post only)

#Nextfile is a language directive used in Command Post only.

A "language directive" is a command to the CPML interpreter, as opposed to a menu statement. These begin with a pound-sign ("**#**") in column 1.
Currently there is only one directive recognized by Command Post: **#NextFile**. This directive tells the CPML interpreter to append another .CPM file to the current one before building the menus. You can append only one extra menu file in this way.

Abs

Returns the absolute value of a number.

Syntax:
Abs (integer**)**

Parameters:
integer = integer whose absolute value is desired.

Returns:
(integer) absolute value of integer.

This function returns the absolute (positive) value of the integer which is passed to it, regardless of whether that integer is positive or negative.


```
dy = Abs(y1 - y2)
Message("Years", "There are %dy% years 'twixt %y1% and    %y2%")
```

See Also:
Average, **Max**, **Min**

AskLine

Prompts the user for one line of input.

Syntax:
AskLine (title**,** prompt**,** default**)**

Parameters:

"title" =	title of the dialog box.
"prompt" =	question to be put to the user.
"default" =	default answer.

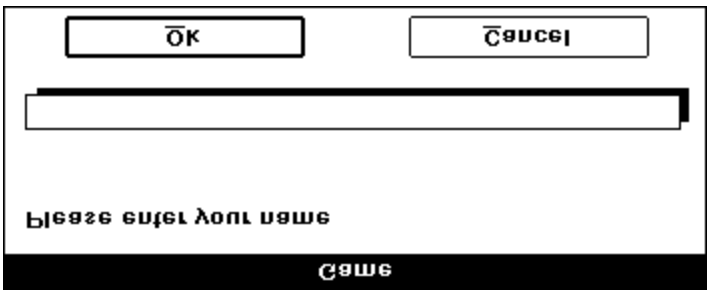
Returns:

(string)	user response.

Use this function to query the user for a line of data. The entire user response will be returned if the user presses the OK button or the Enter key. If the user presses Cancel, the batch file processing is canceled.


```
name = AskLine("Game", "Please enter your name", "")
game = AskLine("Game", "Favorite game?", "Solitaire")
message(StrCat(name,"'s favorite game is "), game)
```

produces:



And then, if Richard types "Scramble" and clicks on the **OK** button:

See Also:

AskYesNo, **Display**, **ItemSelect**, **Message**, **Pause**, **TextBox**

AskPassword

Prompts the user for a password.

Syntax:
AskPassword (title, prompt)

Parameters:

(s) title	title of the dialog box.
(s) prompt	question to be put to the user.

Returns:

(s)	user response.

Pops up a special dialog box to ask for passwords. An asterisk (*) is echoed for each character that the user types; the actual characters entered are not displayed.


```
pw = AskPassword("Security check", "Please enter your password")
If StriCmp(pw, "winguy") != 0 Then Goto nogo
Run(Environment("COMSPEC"), "")
Exit
:nogo
Pause("Security breach", "Invalid password entered")
```

See Also:
AskLine, **AskYesNo**, **DialogBox**

AskYesNo

Prompts the user for a YES or NO answer.

Syntax:
AskYesNo (title**,** question**)**

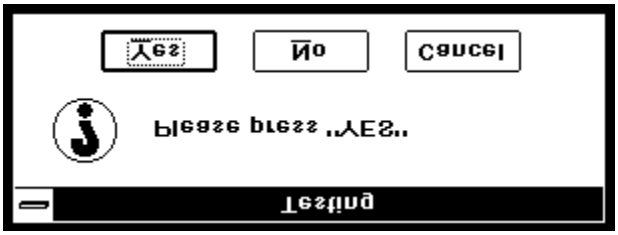
Parameters
"title" = title of the question box.
"question" = question to be put to the user.

Returns:
(integer) **@YES** or **@NO**, depending on the button pressed.

This function displays a message box with three pushbuttons - Yes, No, and Cancel. If the user presses Cancel**,** the current batch file is ended, so there is no return value.


```
q = AskYesNo('Testing', 'Please press "YES"')
If q == @YES Then Exit
Display(3, 'ERROR', 'I said press "YES"')
```

Produces:



And then, if the user presses **No**:

See Also:
AskLine, **Display**, **ItemSelect**, **Message**, **Pause**, **TextBox**

Average

Returns the average of a list of numbers.

Syntax:
Average (integer [**,** integer]...**)**

Parameters:
integer = integers to get the average of.

Returns:
(integer) average of the integers.

Use this function to compute the mean average of a series of numbers, delimited by commas. This function returns an integer value, so there can be some rounding error involved.


```
avg = Average(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
Message("The average is", avg)
```

See Also:
Abs, **Max**, **Min**

Beep

Beeps once.

Syntax:

Beep

Use this command to produce a short beep, generally to alert the user to an error situation.


```
Beep
Pause("WARNING!!!", "You are about to destroy data!")
```

Call

Calls another WBT file as a subroutine.

Syntax:
Call (filename.wbt, parameters**)**

Parameters:
"filename.wbt" = the WBT file you are calling. The WBT extension is required.
"parameters" = the parameters to pass to the file, if any, in the form "p1 p2 p3 ... pn".

Returns:
(integer) always @FALSE.

This function is used to pass control temporarily to a secondary WBT file. The main WBT file can optionally pass parameters to the secondary WBT file. All variables are common (**global**) between the calling and the called WBT files, so that the secondary WBT file may modify or create variables. The secondary WBT file should end with a **Return** statement, to pass control back to the main WBT file.
If a string of parameters is passed to the secondary WBT file, it will automatically be parsed into individual variables with the names **param1**, **param2**, etc., (maximum of nine parameters). The variable **param0** will be a count of the total number of parameters in the string.


```
; MAIN.WBT
name = AskLine("", "What is your name?", "")
age = AskLine("", "How old are you?", "")
valid = @NO
Call("chek-age.wbt", age)
If valid == @NO Then Message("", "Invalid age")

; CHEK-AGE.WBT
userage = param1
really = AskYesNo("", "%name%, are you really %userage%?")
If really == @NO Then Return
If (userage > 0) && (userage < 150) Then valid = @YES
Return
```

See Also:
CallExt, **ParseData**, **Return**

CallExt

Calls another WBT file as a separate subprogram.

Syntax:
CallExt (filename.wbt, parameters**)**

Parameters:
"filename.wbt" = the WBT file you are calling. The extension is required.
"parameters" = the parameters to pass to the file, if any, in the form "p1 p2 p3 ... pn".

Returns:
(integer) always **@FALSE**.

This function is used to pass control temporarily to a secondary WBT file. The main WBT file can optionally pass parameters to the secondary WBT file. All variables are exclusive (**local**) to their respective files, so that neither WBT file "knows about" variables being used by the other. The secondary WBT file should end with a **Return** statement, to pass control back to the main WBT file.
If a string of parameters is passed to the secondary WBT file, it will automatically be parsed into individual variables with the names **param1**, **param2**, etc. (maximum of nine parameters). The variable **param0** will be a count of the total number of parameters in the string.


```
; MAIN.WBT
old = AskLine("RENAME", "File to rename", "")
If  !FileExist(old) Then Exit
new = AskLine("RENAME", "New name for %old%", "")
If FileExist(new) Then Exit
CallExt("rename.wbt", "%old% %new%")

; RENAME.WBT
old = param1
new = param2
FileRename(old, new)
Return
```

See Also:
Call, **ParseData**, **Return**

Char2Num

Converts the first character of a string to its numeric equivalent.

Syntax:
Char2Num (string**)**

Parameters:
"string" = any text string. Only the first character will be converted.

Returns:
(integer) ANSI character code.

This function returns the 8-bit ANSI code corresponding to the first character of the string parameter.
Note: For the commonly-used characters (with codes below 128), ANSI and ASCII characters are identical.


```
; Show the hex equivalent of entered character
inpchar = AskLine("ANSI Equivalents", "Char:", "")
ansi = StrSub(inpchar, 1, 1)
ansiequiv = Char2Num(InpChar)
Message("ANSI Codes", "%ansi% => %ansiequiv%")
```

See Also:
Num2Char

ClipAppend

Appends a string to the Clipboard.

Syntax:
ClipAppend (string**)**

Parameters:
"string" = text string to add to Clipboard.

Returns:
(integer) **@TRUE** if string was appended;
@FALSE if Clipboard ran out of memory.

Use this function to append a string to the Windows Clipboard. The Clipboard must either contain text data or be empty for this function to succeed.


```
; The code below will append 2 copies of the
; Clipboard contents back to the Clipboard, resulting
; in 3 copies of the original contents with a CR/LF
; between each copy.
a = ClipGet()
crlf = StrCat(Num2Char(13), Num2Char(10))
ClipAppend(crlf)
ClipAppend(a)
ClipAppend(crlf)
ClipAppend(a)
```

See Also:
ClipGet, **ClipPut**

ClipGet

Returns the contents of the Clipboard.

Syntax:
ClipGet ( )

Parameters:
(none)

Returns:
(string) clipboard contents.

Use this function to copy text from the Windows Clipboard into a string variable.
Note: If the Clipboard contains an excessively large string a (fatal) out of memory error may occur.


```
; The code below will convert Clipboard contents to
; uppercase
ClipPut(StrUpper(ClipGet()))
a = ClipGet()
Message("UPPERCASE Clipboard Contents", a)
```

See Also:
ClipAppend, **ClipPut**

ClipPut

Copies a string to the clipboard.

Syntax:
ClipPut (string**)**

Parameters:
"string" = any text string.

Returns:
(integer) **@TRUE** if string was copied;
@FALSE if clipboard ran out of memory.

Use this function to copy a string to the Windows Clipboard. The previous Clipboard contents will be lost.


```
; The code below will convert Clipboard contents to
; lowercase
ClipPut(StrLower(ClipGet()))
a = ClipGet()
Message("lowercase Clipboard Contents", a)
```

See Also:
ClipAppend, **ClipGet**

CurrentFile (Command Post Only)

Returns the selected file name.

Syntax:
CurrentFile()

Returns:
(string) currently-selected file name.

When Command Post displays the files in the current directory, one of them is always selected. It's the one with the dotted-line box around it.

This is different than a "highlighted" file. When a file is highlighted, it shows up in inverse video (usually white-on-black). To find the file nanes that are hightlighted, see **FileItemize**.


```
;Ask which program to run (default = current file)
TheFile = AskLine ("Run It","Program:", CurrentFile())
Run (TheFile,"")
```

See Also:
FileItemize, **DirGet**, **DirItemize**

DateTime

Provides the current Date and time.

Syntax:
DateTime ( )

Parameters:
(none)

Returns:
(string) today's date and time

This function will return the current date and time in a pre-formatted string. The format it is returned in depends on how it is set up in the international section of the WIN.INI file:
ddd mm:dd:yy hh:mm:ss XX
ddd dd:mm:yy hh:mm:ss XX
ddd yy:mm:dd hh:mm:ss XX
Where:
ddd is day of the week (e.g. Mon)

mm is the month (e.g. 10)

dd is the day of the month (e.g. 23)

yy is the year (e.g. 90)

hh is the hours

mm is the minutes

ss is the seconds

XX is the Day/Night code (e.g. AM or PM)

Note: Windows provides even more formatting options than this.
The WIN.INI file will be examined to determine which format to use. You can adjust the WIN.INI file via the **International** section of **Control Panel** if the format isn't what you prefer.


```
; assuming the current standard is U.S.
; (i.e. day   dd/mm/yy hh:mm:ss AM)
Message("Current Date & Time", DateTime())
```

would produce:

Current Date & Time
Sat 5/30/95 5:23:18 PM
OK

DDEExecute

Sends commands to a DDE server application.

Syntax:
DDEExecute (channel, command string)

Parameters:
(i) channel same integer that was returned by **DDEInitiate.**
(s) command string one or more commands to be executed by the server app.

Returns:
(i) @TRUE if successful; @FALSE if unsuccessful.

Use the **DDEInitiate** function to obtain a channel number.
In order to use this function successfully, you will need appropriate documentation for the server application you wish to access, which must provide information on the DDE functions that it supports and the correct syntax to use.


```
Run("wincheck.exe", "TUT")
channel = DDEInitiate("wincheck", "TUT")
If channel == 0 Then Goto failed
result = DDEExecute(channel, '[WriteCheck:p="Shorewood
Apartments",t=580.00,l="Rent"]')
DDETerminate(channel)
WinClose("WinCheck")
If result == @FALSE Then Goto Failed
Message("DDE Execute", "Operation complete")
Exit
:failed
Message("DDE operation unsuccessful", "Check your syntax")
```

See Also:
DDEInitiate, DDEPoke, DDERequest, DDETerminate, DDETimeout

DDEInitiate

Opens a DDE channel.

Syntax:
DDEInitiate (app name, topic name)

Parameters:
(s) app name name of the application (without the **E** extension).
(s) topic name name of the topic you wish to access.

Returns:
(i) communications channel.

This function opens a DDE communications channel with a server application. The communications channel can be subsequently used by the **DDEExecute, DDEPoke, and DDERequest** functions. You should close this channel with **DDETerminate** when you are finished using it. If the communications channel cannot be opened as requested, **DDEInitiate** returns a channel number of 0.
You can call **DDEInitiate** more than once, in order to carry on multiple DDE conversations (with multiple applications) simultaneously.
In order to use this function successfully, you will need appropriate documentation for the server application you wish to access, which must provide information on the DDE functions that it supports and the correct syntax to use.


```
Run("wincheck.exe", "TUT")
channel = DDEInitiate("WinCheck", "TUT")
If channel == 0 Then Goto failed
output = DDERequest(channel, "GetChecking")
DDETerminate(channel)
WinClose("WinCheck")
If output == "" Then Goto FailedMessage("Account balance", output)
Exit
:failed
Message("DDE operation unsuccessful", "Check your syntax")
```

See Also:
DDEExecute, DDEPoke, DDERequest, DDETerminate, DDETimeout

DDEPoke

Sends data to a DDE server application.

Syntax:
DDEPoke (channel, item name, item value)

Parameters:
(i) channel same integer that was returned by **DDEInitiate.**
(s) item name identifies the type of data being sent.
(s) item value actual data to be sent to the server.

Returns:
(i) @TRUE if successful; @FALSE if unsuccessful.

Use the **DDEInitiate** function to obtain a channel number.
In order to use this function successfully, you will need appropriate documentation for the server application you wish to access, which must provide information on the DDE functions that it supports and the correct syntax to use.


```
Run("reminder.exe", "")
channel = DDEInitiate("Reminder", "items")
If channel == 0 Then Goto failed
result = DDEPoke(channel, "all", "11/3/92 Misc Remember to vote")
DDETerminate(channel)
WinClose("Reminder")
If result == @FALSE Then Goto Failed
Message("DDE Poke", "Operation complete")
Exit
:failed
Message("DDE operation unsuccessful", "Check your syntax")
```

See Also:
DDEExecute, DDEInitiate, DDERequest, DDETerminate, DDETimeout

DDERequest

Gets data from a DDE server application.

Syntax:
DDERequest (channel, item name)

Parameters:
(i) channel same integer that was returned by **DDEInitiate.**
(s) item name identifies the data to be returned by the server.

Returns:
(s) information returned from the server.

Use the **DDEInitiate** function to obtain a channel number.
In order to use this function successfully, you will need appropriate documentation for the server application you wish to access, which must provide information on the DDE functions that it supports and the correct syntax to use.


```
Run("wincheck.exe", "TUT")
channel = DDEInitiate("WinCheck", "TUT")
If channel == 0 Then Goto failed
output = DDERequest(channel, "GetChecking")
DDETerminate(channel)
WinClose("WinCheck")
If output == "" Then Goto Failed
Message("Account balance", output)
Exit
:failed
Message("DDE operation unsuccessful", "Check your syntax")
```

See Also:
DDEExecute, DDEInitiate, DDEPoke, DDETerminate, DDETimeout

DDETerminate

Closes a DDE channel.

Syntax:
DDETerminate (channel)

Parameters:
(i) channel same integer that was returned by **DDEInitiate.**

Returns:
(i) always 1.

This function closes a communications channel that was opened with **DDEInitiate**.


```
Run("wincheck.exe", "TUT")
channel = DDEInitiate("WinCheck", "TUT")
If channel == 0 Then Goto failed
output = DDERequest(channel, "GetChecking")
DDETerminate(channel)
WinClose("WinCheck")
If output == "" Then Goto Failed
Message("Account balance", output)
Exit
:failed
Message("DDE operation unsuccessful", "Check your syntax")
```

See Also:
DDEExecute, **DDEInitiate**, **DDEPoke**, **DDERequest**, **DDETimeout**

DDETimeout

Sets the DDE timeout value.

Syntax:
DDETimeout (value)

Parameters:
(i) value DDE timeout time.

Returns:
(i) previous timeout value.

Sets the timeout time for subsequent DDE functions to specified value in milliseconds (1/1000 second). Default is 3000 milliseconds (3 seconds). If the time elapses with no response, the WIL Interpreter will return an error. The value set with **DDETimeout** stays in effect until changed by another **DDETimeout** statement or until the WIL program ends, whichever comes first.


```
DDETimeout(5000)
Run("wincheck.exe", "TUT")
channel = DDEInitiate("WinCheck", "TUT")
If channel == 0 Then Goto failed
output = DDERequest(channel, "GetChecking")
DDETerminate(channel)
WinClose("WinCheck")
If output == "" Then Goto Failed
Message("Account balance", output)
Exit
:failed
Message("DDE operation unsuccessful", "Check your syntax")
```

See Also:
DDEExecute, **DDEInitiate**, **DDEPoke**, **DDERequest**, **DDETerminate**

Debug

Controls the debug mode.

Syntax:
Debug (mode**)**

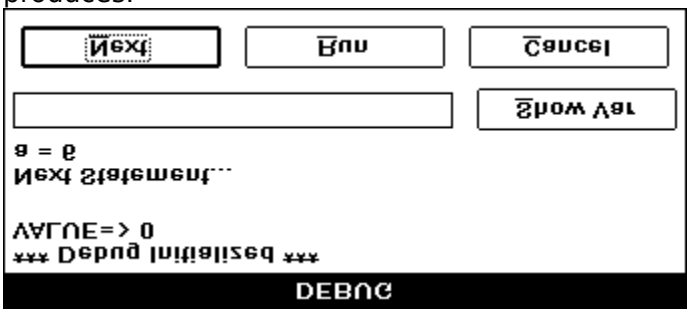
Parameters:
mode = **@ON** or **@OFF**

Returns:
(integer) previous debug mode

Use this function to turn the debug mode on or off. The default is **@OFF**.
When debug mode is on, the interpeter will display the statement just executed, its result (if any), any error conditions, and the next statement to execute.
The statements are displayed in a special dialog box. As you can see in the **Example** section following, the dialog box gives the user four options: Next, Run, Cancel and Show Var**.**
Next executes the next statement and remains in debug mode.
Run exits debug mode and runs the rest of the program normally.
Cancel terminates the current batch file.
Show Var displays the contents of a variable whose name the user entered in the edit box.


```
Debug(@ON)
a = 6
q = AskYesNo("Testing Debug Mode", "Is the Pope Catholic")
Debug(@OFF)
b = a + 4
```

produces:



... then, if the user presses **Next**:

... and presses **Next** again:

... and then presses **Yes**:

etc. (If the user had pressed **No** it would have said "VALUE=>0".)

See Also:

ErrorMode, **LastError**

Delay

Pauses execution for a specified amount of time.

Syntax:
Delay (seconds**)**

Parameters:
seconds = integer seconds to delay (**2** - **15**)

Returns:
(integer) always **@TRUE**

This function causes the currently-executing batch file to be suspended for the specified period of time. **Seconds** must be an integer between **2** and **15**. Smaller or larger numbers will be adjusted accordingly.


```
Message("Wait", "About 15 seconds")
Delay(15)
Message("Hi", "I'm Baaaaaaack")
```

See Also:
Yield

DialogBox

Pops up a Windows dialog box defined by the WBD template file.

Practical Examples: Click Here

Syntax:

DialogBox ("title", "WBD file"**)**

Parameters:

"title" = the title of the dialog box.
"WBD file" = the name of the WBD template file.

Returns:

(integer) always **0**

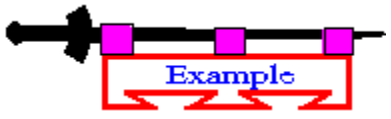
Each element in the template file is enclosed in square brackets, and consists of a variable name, followed by one of the following symbols:

Symbol	Meaning	Example
+	check box	[backup+1Save backup]
#	edit box	[newfile#]
****	file selection listbox	[editfile\ ]
^	radio button	[prog^1Note] [prog^2Write]
$	variable	[var$]

The number following the check box and radio button symbols is the value which will get assigned to the variable if its corresponding box is checked, or button is selected. Following the number is the descriptive text which will appear next to the box or button.
When used in conjunction with a file selection list box variable with the same name, two of these symbols have special meanings:

#	file mask edit box	[editfile#]
$	directory variable	[editfile$]

Anything not appearing within square brackets is displayed as text.



[editfile$]
File mask [editfile#]
[editfile\ ]
[editfile\ ]
[editfile\ ]
[editfile\ ]
[editfile\ ]
[backup+1Save backup of file]

[prog^1Notepad] [prog^2WinEdit]

DirChange

Changes the current directory. Can also log a new drive.

Syntax:
DirChange ([d:]path**)**

Parameters:
"[d:]" = an optional disk drive to log onto.
"path" = the desired path.

Returns:
(integer) **@TRUE** if directory was changed;
@FALSE if the path could not be found.

Use this function to change the current working directory to another directory, either on the same or a different disk drive.


```
DirChange("c:\")
TextBox("This is your CONFIG.SYS file", "config.sys")
```

See Also:
DirGet, **DirHome**, **LogDisk**

DirGet

Gets the current working directory.

Syntax:
DirGet ( )

Parameters:
(none)

Returns:
(string) = current working directory.

Use this function to determine which directory we are currently in. It's especially useful when changing drives or directories temporarily.


```
; Get, then restore current working directory
origdir = DirGet()
DirChange("c:\")
FileCopy("config.sys", "%origdir%xxxtemp.xyz", @FALSE)
DirChange(origdir)
```

See Also:
DirHome

DirHome

Returns directory containing the WinBatch executable files.

Syntax:
DirHome ( )

Parameters:
(none)

Returns:
(string) pathname of the home directory.

Use this function to determine the location of WINBATCH.EXE.


```
a = DirHome()
Message("WinBatch Executable is in ", a)
```

See Also:
DirGet

DirItemize

Returns a space-delimited list of directories.

Syntax:
DirItemize (dir-list**)**

Parameters:
"dir-list" = a string containing a set of subdirectory names, which may be wildcarded.

Returns:
(string) list of directories.

This function compiles a list of subdirectories and separates the names with spaces.
This is especially useful in conjunction with the **ItemSelect** function, which enables the user to choose an item from such a space-delimited list.
DirItemize("*.*") returns all dirs


```
a = DirItemize("*")
ItemSelect("Directories", a, " ")
```

See Also:
FileItemize, **WinItemize**, **ItemSelect**

DirMake

Creates a new directory.

Syntax:
DirMake ([d:]path**)**

Parameters:
"[d:]" = the desired disk drive.
"path" = the path to create.

Returns:
(integer) **@TRUE** if the directory was successfully created;
@FALSE if it wasn't.

Use this function to create a new directory.

DirMake("c:\xxxstuff")

See Also:
DirRemove, **DirRename**

DirRemove

Removes a directory.

Syntax:
DirRemove (dir-list**)**

Parameters:
"dir-list" = a space-delimited list of directory pathnames.

Returns:
(integer) **@TRUE** if the directory was successfully removed;
@FALSE if it wasn't.

Use this function to delete directories. You can delete one or more at a time by separating directory names with spaces. You cannot, however, use wildcards.

Examples:
DirRemove("c:\xxxstuff")

DirRemove("tempdir1 tempdir2 tempdir3")

See Also:
DirMake, **DirRename**

DirRename

Renames a directory.

Syntax:
DirRename ([d:]oldpath**,** [d:]newpath**)**

Parameters:
"oldpath" = existing directory name, with optional drive.
"newpath" = new name for directory.

Returns:
(integer) **@TRUE** if the directory was successfully renamed;
@FALSE if it wasn't.

DirRename("c:\temp", "c:\work")

See Also:
DirMake, **DirRemove**

DirWindows

Returns the name of the Windows or Windows System directory.

Syntax:
DirWindows (request#)

Parameters:
(i) request# see below.

Returns:
(s) directory name.

This function returns the name of either the Windows directory or the Windows System directory, depending on the request# specified.

Req#	Return value
0	Windows directory
1	Windows System directory


```
DirChange(DirWindows(0))
ini = ItemSelect("Select file to edit", FileItemize("*.ini"), " ")
Run("notepad.exe, ini)
```

See Also:
DirHome

DiskFree

Finds the total space available on a group of drives.

Syntax:
DiskFree (drive-list**)**

Parameters:
"drive-list" = at least one drive letter, separated by spaces.

Returns:
(integer) the number of bytes available on all the specified drives.

This function takes a string consisting of drive letters, separated by spaces. Only the first character of each non-blank group of characters is used to determine the drives, so you can use just the drive letters, or add a colon (**:**), or add a backslash (****), or even a whole pathname, and still get a perfectly valid result.


```
size = DiskFree("c d")
Message("Space Available on C: & D:", size)
```

See Also:
FileSize

DiskHide (CP only)

Hides disk drives from display.

Syntax:
DiskHide (drive-list)

Parameters:
(s) drive-list string of drives to hide (non-delimited).

Returns:
(i) always 1.

This function causes the drive letters specified in "drive-list" to be removed from the disk drive icon display.

DiskHide("STUVW")

See Also:
DiskReset, **DiskUpdate**

DiskReset (Command Post only)

Re-examines available disk drives.

Syntax:
DiskReset ( )

Parameters:
(none)

Returns:
(i) always 0.

Examines disk drives on system, and adds any new drives found to the display of drive icons. If an existing drive was hidden with the **DiskHide** function, it will no longer be hidden (unlike the **DiskUpdate** function).

DiskReset()

See Also:
DiskHide, **DiskUpdate**

DiskScan

Returns list of drives.

Syntax:
DiskScan (request#)

Parameters:
(i) request# see below.

Returns:
(s) drive list.

Scans disk drives on the system, and returns a space-delimited list of drives of the type specified by request#, in the form "A: B: C: D: ".
The request# is a bitmask, so adding the values together (except for 0) returns all drive types specified; eg., a request# of 3 returns floppy plus local hard drives.

Req#	Return value
0	List of unused disk IDs
1	List of floppy drives
2	List of local hard drives
4	List of remote (network) drives


```
hd = DiskScan(2)
Message("Hard drives on system", hd)
```

See Also:
DiskFree, **LogDisk**

DiskUpdate (Command Post only)

Updates drive icon display.

Syntax:
DiskUpdate ( )

Parameters:
(none)

Returns:
(i) always 0.

Examines disk drives on system, and adds any new drives found to the display of drive icons. If an existing drive was hidden with the **DiskHide** function, it will remain hidden (unlike the **DiskReset** function).

DiskUpdate()

See Also:
DiskHide, **DiskReset**

Display

Displays a message to the user for a specified period of time.

Syntax:
Display (seconds**,** title**,** text**)**

Parameters:

seconds =	integer seconds to display the message (**1**-**15**).
"title" =	title of the window to be displayed.
"text" =	text of the window to be displayed.

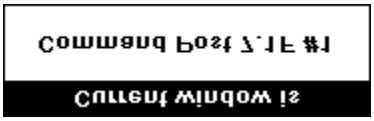
Returns:

(integer)	always **@TRUE**.

Use this function to display a message for a few seconds, and then continue processing without user input.
Seconds must be an integer between **1** and **15**. Smaller or larger numbers will be adjusted accordingly.
The display box may be prematurely canceled by the user by clicking a mouse button, or hitting any key.

Display(3, "Current window is", WinGetActive())

which produces something like this:



See Also:
Pause, **Message**

DOSVersion

Returns the version numbers of the current version of DOS.

Syntax:
DOSVersion (level)

Parameters:
level = **@MAJOR** or **@MINOR**.

Returns:
(integer) integer or decimal part of DOS version number.

@MAJOR returns the integer part (to the left of the decimal).
@MINOR returns the decimal part (to the right of the decimal).
If the version of DOS in use is **4.0**, then:
DOSVersion(@MAJOR) == **4**

DOSVersion(@MINOR) == **0**


```
i = DOSVersion(@MAJOR)
d = DOSVersion(@MINOR)
If StrLen(d) == 1 Then d = StrCat("0", d)
Message("DOS Version", "%i%.%d%")
```

See Also:
Environment, **Version**, **WinVersion**

Drop

Removes variables from memory.

Syntax:
Drop (var**,** [var]...**)**

Parameters:
var = variable names to remove.

Returns:
(integer) always **@TRUE**.

This function removes variables from the language processor's variable list, and recovers the memory associated with the variable (and possibly related string storage).


```
a = "A variable"
b = "Another one"
Drop(a, b)              ; This removes A and B from memory
```

EndSession

Ends the Windows session.

Syntax:
EndSession ( )

Parameters:
(none)

Returns:
(integer) always **0**.

Use this command to end the Windows session. This command is equivalent to closing the **Program Manager** window.


```
sure = AskYesNo ("End Session", "You SURE you want to exit          Windows?")
If sure == @No Then Goto cancel
EndSession()
:cancel
Message("", "Exit Windows canceled")
```

See Also:
Exit, **WinClose**, **WinCloseNot**

Environment

Gets a DOS environment variable.

Syntax:
Environment (env-variable**)**

Parameters:
"env-variable" = any defined environment variable.

Returns:
(string) environment variable contents.

Use this function to query the DOS environment.


```
; Display the PATH for this DOS session
currpath = Environment("PATH")
Message("Current DOS Path", currpath)
```

See Also:
IniRead, **Version**, **WinVersion**

ErrorMode

Specifies how to handle errors.

Syntax:
ErrorMode (mode**)**

Parameters:
mode = **@CANCEL** or **@NOTIFY** or **@OFF**.

Returns:
(integer) previous error setting.

Use this function to control the effects of runtime errors. The default is **@CANCEL**, meaning the execution of the batch file will be canceled for any error.
@CANCEL: All runtime errors will cause execution to be canceled. The user will be notified which error occurred.
@NOTIFY: All runtime errors will be reported to the user, and the user can choose to continue if it isn't fatal.
@OFF: Minor runtime errors will be suppressed. Moderate and fatal errors will be reported to the user. User has the option of continuing if the error is not fatal.
In general, we suggest the normal state of the program should be **ErrorMode(@CANCEL)**, especially if you are writing a batch file for others to use. You can always suppress errors you expect will occur and then re-enable **ErrorMode (@CANCEL)**.


```
; Delete xxxtest.xyz.  If file doesn't exist,
; continue execution; don't stop
prevmode = ErrorMode(@OFF)
FileDelete("c:\xxxtest.xyz")
ErrorMode(prevmode)
```

See Also:
Debug, **LastError**

Exclusive

Controls whether or not other Windows programs will get any time to execute.

Syntax:
Exclusive (mode)

Parameters:
mode = **@ON** or **@OFF**.

Returns:
(integer) previous Exclusive mode.

Exclusive(@OFF) is the default mode. In this mode,the interpeter is well-behaved toward other Windows applications.
Exclusive(@ON) allows WIL files to run somewhat faster, but causes the interpeter to be "greedier" about sharing processing time with other active Windows applications. For the most part, this mode is useful only when you have a series of WIL statements which must be executed in quick succession.


```
Exclusive(@ON)
x = 0
start = DateTime()
:add
x = x + 1
If x < 1000 Then Goto add
stop = DateTime()
crlf = StrCat(Num2Char(13), Num2Char(10))
Message("Times", "Start: %start%%crlf%Stop:   %stop%")
Exclusive(@OFF)
```

Execute

Executes a statement in a protected environment. Any errors encountered are recoverable.

Syntax:
    Execute statement

Parameters:
    "statement" =   is (hopefully) an executable statement.

Use this command to execute computed or user-entered statements. Due to the built-in error recovery associated with **Execute**, it is ideal for interactive execution of user-entered commands.
Note that the **Execute** command doesn't operate on a string, *per se*, but rather on a direct statement. If you want to put a code segment into a string variable, you must use the substitution feature of the language, as in the example below.


```
cmd = ""
cmd = AskLine("WIL Interactive", "Command:", cmd)
Execute %cmd%
```

Exit

Terminates the batch file being interpreted.

Syntax:

Exit

Use this command to prematurely exit a batch file process. An **exit** is implied at the end of each batch file.


```
a = 100
Message("The value of a is", a)
Exit
```

See Also:

Pause

FileAppend

Appends one or more files to another file.

Syntax:
FileAppend (source-list**,** destination**)**

Parameters:
"source-list" = a string containing one or more filenames, which may be wildcarded.
"destination" = target file name.

Returns:
(integer) **@TRUE** if all files were appended successfully;
@FALSE if at least one file wasn't appended.

Use this function to append an individual file or a group of files to the end of an existing file.
If "destination" does not exist, it will be created.
The file(s) specified in "source-list" will not be modified by this function.
"Source-list" may contain ***** and **?** wildcards. "Destination" may not contain wildcards of any type; it must be a single file name.

Examples:
```
FileAppend("c:\config.sys", "c:\misc\config.sav")

DirChange("c:\batch")
FileDelete("allbats.fil")
FileAppend("*.bat", "allbats.fil")
```

See Also:
FileCopy, **FileDelete**, **FileExist**

FileAttrGet

Returns file attributes.

Syntax:
FileAttrGet (filename)

Parameters:
(s) filename file whose attributes you want to determine.

Returns:
(s) attribute settings.

Returns attributes for the specified file, in a string of the form "RASH". This string is composed of four individual attribute characters, as follows:

Char	Symbol	Meaning
1	R	Read-only ON
2	A	Archive ON
3	S	System ON
4	H	Hidden ON

A hyphen in any of these positions indicates that the specified attribute is OFF. For example, the string "-A-H" indicates a file which has the Archive and Hidden attributes set.


```
editfile = "c:\config.sys"
attr = FileAttrGet(editfile)
If StrSub(attr, 1, 1) == "R" Then Goto readonly
Run("notepad.exe", editfile)
Exit
:readonly
Message("File is read-only", "Cannot edit %editfile%")
```

See Also:
FileAttrSet, **FileTimeGet**

FileAttrSet

Sets file attributes.

Syntax:
FileAttrSet (file-list, settings)

Parameters:
(s) file-list space-delimited list of files.
(s) settings new attribute settings for those file(s).

Returns:
(i) always 0.

The attribute string consists of one or more of the following characters (an upper case letter turns the specified attribute ON, a lower case letter turns it OFF):

R read only ON

A archive ON

S system ON

H hidden ON

r read only OFF

a archive OFF

s system OFF

h hidden OFF

Examples:
FileAttrSet("win.ini system.ini", "rAsH")

FileAttrSet("c:\command.com", "R")

See Also:
FileAttrGet, **FileTimeTouch**

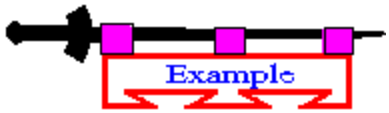
FileClose

Closes a file.

Syntax:
FileClose (filehandle)

Parameters:
filehandle = same integer that was returned by **FileOpen.**

Returns:
(integer) always 0.




```
; the hard way to copy an ASCII file
old = FileOpen("config.sys", "READ")
new = FileOpen("sample.txt", "WRITE")
:top
x = FileRead(old)
If x != "*EOF*" Then FileWrite(new, x)
If x != "*EOF*" Then Goto top
FileClose(new)
FileClose(old)
```

See Also:
FileOpen, FileRead, FileWrite

FileCopy

Copies files.

Syntax:
FileCopy (source-list**,** destination**,** warning**)**

Parameters:
"source-list" = a string containing one or more filenames, which may be wildcarded.
"destination" = target file name.

warning = **@TRUE** if you want a warning before overwriting existing files;

@FALSE if no warning desired.

Returns:
(integer) **@TRUE** if all files were copied successfully;
@FALSE if at least one file wasn't copied.

Use this function to copy an individual file, a group of files using wildcards, or several groups of files by separating the names with spaces.
You can also copy files to any **COM** or **LPT** device.
"Source-list" may contain ***** and **?** wildcards. "Destination" may contain the ***** wildcard only.

Examples:

```
FileCopy("c:\config.sys", "d:", @FALSE)

FileCopy("c:\*.sys", "d:devices\*.sys", @TRUE)

FileCopy("c:\config.sys", "LPT1", @FALSE)
```

See Also:
FileDelete, **FileExist**, **FileLocate**, **FileMove**, **FileRename**

FileDelete

Deletes files.

Syntax:
FileDelete (file-list**)**

Parameters:
"file-list" = a string containing one or more filenames, which may be wildcarded.

Returns:
(integer) **@TRUE** if all the files were deleted;
@FALSE if a file didn't exist or is marked with the READ-ONLY attribute.

Use this function to delete an individual file, a group of files using wildcards, or several groups of files by separating the names with spaces.

FileDelete("*.bak temp???.fil")

See Also:
FileExist, **FileLocate**, **FileMove**, **FileRename**

FileExist

Tests for the existence of files.

Syntax:
FileExist (filename**)**

Parameters:
"filename" = either a fully qualified filename with drive and path, or just a filename and extension.

Returns:
(integer) **@TRUE** if the file exists;
@FALSE if it doesn't or if the pathname is invalid.

This function is used to test whether or not a specified file exists.
If a fully-qualified file name is used, only the specified drive and directory will be checked for the desired file. If only the root and extension are specified, then first the current directory is checked for the file, and then, if the file is not found in the current directory, all directories in the DOS path are searched.

Examples:

```
; check for file in current directory
fex = FileExist(StrCat(DirGet(), "myfile.txt"))
tex = StrSub("NOT", 1, StrLen("NOT") * fex)
Message("MyFile.Txt"," Is %tex%in the current directory")

; check for file someplace along path
fex = FileExist("myfile.txt")
tex = StrSub("NOT", 1, StrLen("NOT") * fex)
Message("MyFile.Txt", " Is %tex% in the DOS path")
```

See Also:
FileLocate

FileExtension

Returns extension of file.

Syntax:
FileExtension (filename)

Parameters:
"filename" = [optional path]complete file name, with extension.

Returns:
(string) file extension.

FileExtension parses the passed filename and returns the extension part of the filename.


```
; prevent the user from editing a COM or E file
allfiles = FileItemize("*.*")
editfile = ItemSelect("Select file to edit", allfiles, " ")
ext = FileExtension(editfile)
If (ext == "com") || (ext == "exe") Then Goto noedit
Run("notepad.exe", editfile)
exit
:noedit
Message ("Sorry", "You may not edit a program file")
```

See Also:
FileRoot, **FilePath**

FileHilite (Command Post only)

Highlights or unhighlights files in file display.

Syntax:
FileHilite (file-masks, mode)

Parameters:

(s) file-masks	one or more file specifications, which may be wildcarded.	
(i) mode	@TRUE	Highlight matching files.
	@FALSE	Unhighlight matching files.

Returns:
(i) total number of files highlighted or unhighlighted.

This function causes one or more groups of files in the file display window to be highlighted (selected) or unhighlighted (de-selected). This is useful to select files for an operation such as **FileCopy**, or just to spotlight certain files in a directory.
Multiple file specifications must be space-delimited.

Examples:
FileHilite("*.ZIP *.LZH *.ARC", @TRUE)

FileHilite("OLD*.ZIP", @FALSE)

See Also:
CurrentFile, **FileExtension**

FileItemize

Returns a space-delimited list of files.

Syntax:
FileItemize (file-list**)**

Parameters:
"file-list" = a string containing a list of filenames, which may be wildcarded.

Returns:
(string) space-delimited list of files.

This function compiles a list of filenames and separates the names with spaces.
This is especially useful in conjunction with the **ItemSelect** function, which lets the user choose an item from such a space-delimited list.

Examples:

```
FileItemize("*.bak")                    ;all BAK files

FileItemize("*.arc *.zip *.lzh")        ;compressed files

; Get which .INI file to edit
ifiles = FileItemize("c:\windows\*.ini")
ifile = ItemSelect(".INI Files", ifiles, " ")
RunZoom("notepad", ifile)
Drop(ifiles, ifile)
```

See Also:
DirItemize, **WinItemize**, **ItemSelect**

FileLocate

Finds file in current directory or along the DOS path.

Syntax:
FileLocate (filename**)**

Parameters:
"filename" = root name, "**.**", and extension.

Returns:
(string) fully-qualified path name.

This function is used to obtain the fully qualified path name of a file. The current directory is checked first, and if the file is not found, the DOS path is searched. The first occurrence of the file is returned.


```
; Edit WIN.INI
winini = FileLocate("win.ini")
If winini == "" Then Goto notfound
Run("notepad.exe", winini)
Exit
:notfound
Message("???", "WIN.INI not found")
```

See Also:
FileExist

FileMove

Moves files.

Syntax:
FileMove (source-list**,** destination**,** warning**)**

Parameters:
"source-list" = one or more filenames separated by spaces.
"destination" = target filename.
warning = **@TRUE** if you want a warning before overwriting existing files;
@FALSE if no warning desired.

Returns:
(integer) **@TRUE** if the file was moved;
@FALSE if the source file was not found or had the READ-ONLY attribute, or the target filename is invalid.

Use this function to move an individual file, a group of files using wildcards, or several groups of files by separating the names with spaces.
You can also move files to another drive, or to any **COM** or **LPT** device.
"Source-list" may contain ***** and **?** wildcards. "Destination" may contain the ***** wildcard only.

Examples:
```
FileMove("c:\config.sys", "d:", @FALSE)

FileMove("c:\*.sys", "d:*.sys", @TRUE)
```

See Also:
FileCopy, **FileDelete**, **FileExist**, **FileLocate**, **FileRename**

FileOpen

Opens a STANDARD ASCII (only) file for reading or writing.

Syntax:
FileOpen (filename, open-type)

Parameters:
"filename" = name of the file to open.
open-type = **READ** or **WRITE**.

Returns:
(special integer) filehandle

The "**filehandle**" returned by the **FileOpen** function is subsequently used by the **FileRead, FileWrite, and FileClose** functions.

Examples:
```
; To open for reading:
FileOpen("stuff.txt", "READ")

; To open for writing:
FileOpen("stuff.txt", "WRITE")
```

See Also:
FileClose, FileRead, FileWrite

FilePath

Returns path of file.

Syntax:
FilePath (filename)

Parameters:
"filename" = fully qualified file name, including path.

Returns:
(string) fully qualified path name.

FilePath parses the passed filename and returns the drive and path of the file specification, if any.


```
coms = Environment("COMSPEC")
compath = FilePath(coms)
Message("", "Your command processor is located in the %compath% directory")
```

See Also:
FileRoot, **FileExtension**

FileRead

Reads data from a file.

Syntax:
FileRead (filehandle)

Parameters:
filehandle = same integer that was returned by **FileOpen.**

Returns:
(string) line of data read from file.

When the end of the file is reached, the string ***EOF*** will be returned.


```
handle = FileOpen("autoexec.bat", "READ")
:top
line = FileRead(handle)
Display(4, "AUTOEXEC DATA", line)
If line != "*EOF*" Then Goto top
FileClose(handle)
```

See Also:
FileOpen, FileClose, FileWrite

FileRename

Renames files.

Syntax:
FileRename (source-list**,** destination**)**

Parameters:
"source-list" = one or more filenames, separated by spaces.
"destination" = target filename.

Returns:
(integer) **@TRUE** if the file was renamed;
@FALSE if the source file was not found or had the READ-ONLY attribute, or the target filename is invalid.

Use this function to rename an individual file, a group of files using wildcards, or several groups of files by separating the names with spaces.
Note: Unlike **FileMove**, you cannot make a file change its resident disk drive with **FileRename**.
"Source-list" may contain ***** and **?** wildcards. "Destination" may contain the ***** wildcard only.

Examples:
```
FileRename("c:\config.sys", "config.old")

FileRename("c:\*.txt", "*.bak")
```

See Also:
FileCopy, **FileExist**, **FileLocate**, **FileMove**

FileRoot

Returns root of file.

Syntax:
FileRoot (filename)

Parameters:
"filename" = [optional path]complete file name, with extension.

Returns:
(string) file root.

FileRoot parses the passed filename and returns the root part of the filename.


```
allfiles = FileItemize("*.*")
editfile = ItemSelect("Select file to edit", allfiles, " ")
root = FileRoot(editfile)
ext = FileExtension(editfile)
lowerext = StrLower(ext)
nicefile = StrCat(root, ".", lowerext)
Message("", "You are about to edit %nicefile%.")
Run("notepad.exe", editfile)
```

See Also:
FileExtension, **FilePath**

FileSize

Finds the total size of a group of files.

Syntax:
FileSize (file-list**)**

Parameters:
"file-list" = zero or more filenames, separated by spaces.

Returns:
(integer) total bytes taken up by the specified files.

This function returns the total size of the specified files. Note that it doesn't handle wildcarded filenames. You can, however, use **FileItemize** on a wildcarded filename and use the resulting string as a **FileSize** parameter.


```
size = FileSize(FileItemize("*.*"))
Message("Size of All Files in Directory", size)
```

See Also:
DiskFree

FileTimeGet

Returns file date and time.

Syntax:
    FileTimeGet (filename)

Parameters:
    (s) filename   name of file for which you want the date and time.

Returns:
    (s)   file date and time.

This function will return the date and time of a file, in a pre-formatted string. The format it is returned in depends on the date format specified in the [International] section of the WIN.INI file:
ddd mm:dd:yy hh:mm:ss XX
ddd dd:mm:yy hh:mm:ss XX
ddd yy:mm:dd hh:mm:ss XX
Where:
    ddd is day of the week (e.g. Mon)

    mm is the month (e.g. 10)

    dd is the day of the month (e.g. 23)

    yy is the year (e.g. 90)

    hh is the hours

    mm is the minutes

    ss is the seconds

    XX is the Day/Night code (e.g. AM or PM)

The WIN.INI file will be examined to determine which format to use. You can adjust the WIN.INI file via the **International** section of **Control Panel** if the format isn't what you prefer.


```
oldtime = FileTimeGet("win.ini")
Run("notepad.exe", "win.ini")
WinWaitClose("Notepad - WIN.INI")
newtime = FileTimeGet("win.ini")
If StrCmp(oldtime, newtime) == 0 Then Exit
Message("", "WIN.INI has been changed")
```

See Also:

FileAttrGet, **FileTimeTouch**

FileTimeTouch

Sets file(s) to current time.

Syntax:
FileTimeTouch (file-list)

Parameters:
(s) file-list a space-delimited list of files

Returns:
(i) always 0

"File-list" is a space-delimited list of files, which may not contain wildcards. The path is searched if the file is not found in current directory and if the directory is not specified in "file-list".


```
FileTimeTouch("wac.c wac.rc")
Run("make.exe", "-fwac.mak")
```

See Also:
FileAttrSet, **FileTimeGet**

IsMenuChecked (Command Post Only)

Determines if a menuitem has a checkmark next to it.

Syntax:
IsMenuChecked (menuname**)**

Parameters:
"menuname" = name of the menu item to test.

Returns:
(integer) **@YES** if the menuitem has a checkmark;
@NO if it doesn't.

You can place a checkmark next to a menu item with the **MenuChange** command, to indicate an option has been enabled. This function lets you determine if the menu item has already been checked or not.


```
;Assume we've defined a "Misc./Prompt Often" menuitem...
Prompt = IsMenuChecked ("MiscPromptOften")
IfPrompt = strsub(";",1,(Prompt==@FALSE))
execute %IfPrompt% Confirm = AskYesNo ("???", "Do you REALLY
                want to do this?")
execute %IfPrompt% Terminate (Confirm!=@YES, "", "")
;some risky operation the user has just confirmed they want
;to carry out...
```

See Also:
IsMenuEnabled, **MenuChange**

IsMenuEnabled (Command Post Only)

Determines if a menuitem has been enabled.

Syntax:
IsMenuEnabled (menuname**)**

Parameters:
"menuname" = name of the menu item to test.

Returns:
(integer) **@YES** if the menuitem is enabled;
@NO if it is disabled & grayed.

You can disable a menu item with the **MenuChange** command if you want to prevent the user from choosing it. It shows up on the screen as a grayed item. **IsMenuEnabled** lets you determine if the menu item is currently enabled or not.


```
;Allow editing of autoexec.bat file only if choice enabled
Terminate (!IsMenuEnabled("UtilitiesEditBatFile"), "", "")
Run ("Notepad.exe", "c:\autoexec.bat")
```

See Also:
IsMenuChecked, **MenuChange**

FileWrite

Writes data to a file.

Syntax:
FileWrite(filehandle, output-data)

Parameters:
filehandle = same integer that was returned by **FileOpen**.
"output-data" = data to write to file.

Returns:
(integer) always 0.


```
handle = FileOpen("stuff.txt", "WRITE")
FileWrite(handle, "Gobbledygook")
FileClose(handle)
```

See Also:
FileOpen, **FileClose**, **FileRead**

Goto

Changes the flow of control in a batch file.

Syntax:
Goto label

Parameters:
"label" = user-defined identifier.

Goto *label* causes an unconditional branch to the batch file line marked *:label*, where the identifier is preceded by a colon (**:**).


```
If WinExist("Solitaire") == @FALSE Then Goto open
WinActivate("Solitaire")
Goto loaded
:open
Run("sol.exe", "")
:loaded
```

See Also:
If...Then

IconArrange

Rearranges icons.

Syntax:
IconArrange ( )

Parameters:
(none)

Returns:
(i) always 0.

This function rearranges the icons at the bottom of the screen, spacing them evenly. It does not change the order in which the icons appear.

IconArrange ( )

See Also:
RunIcon, **WinIconize**, **WinPlaceSet**

If...**Then**

Conditionally performs a function.

Syntax:
If condition **Then** statement

Parameters:
"condition" = an expression to be evaluated.
"statement" = any valid WIL function or command.

If the condition following the **If** keyword is true, the statement following the **Then** keyword is executed. If the condition following the **If** keyword is false, the statement following the **Then** keyword is ignored.


```
sure = AskYesNo("End Session", "Really quit Windows?")
If sure == @YES Then EndSession()
```

See Also:
Goto

IgnoreInput

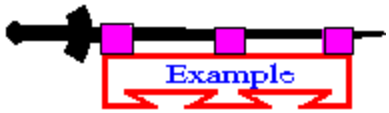
Turns off hardware input to windows.

Syntax:
IgnoreInput(mode)

Parameters:
mode = **@TRUE** or **@FALSE**.

Returns:
(integer) previous IgnoreInput mode.

IgnoreInput causes mouse movements, clicks and keyboard entry to be completely ignored. Good for self-running demos.
Warning: If you are not careful with the use of **IgnoreInput**, you can lock up your computer!




```
username = AskLine("Hello", "Please enter your name","")
IgnoreInput(@TRUE)
Call("demo.wbt", username)
IgnoreInput(@FALSE)
```

IniDelete

Removes a line or section from WIN.INI.

Syntax:
    IniDelete (section, keyname)

Parameters:
    (s) section   the major heading under which the item is located.
    (s) keyname   the name of the item to delete.

Returns:
    (i)   always 0

This function will remove the specified line from the specified section in WIN.INI. You can remove an entire section, instead of just a single line, by specifying a keyword of @WHOLESECTION. Case is not significant in section or keyname.

Examples:
    IniDelete("Desktop", "Wallpaper")

    IniDelete("Quicken",@WHOLESECTION)

See Also:
    IniDeletePvt, **IniItemize**, **IniRead**, **IniWrite**

IniDeletePvt

Removes a line or section from a private INI file.

Syntax:
IniDeletePvt (section, keyname, filename)

Parameters:

(s) section	the major heading under which the item is located.
(s) keyname	the name of the item to delete.
(s) filename	name of the INI file.

Returns:

(i)	always 0.

This function will remove the specified line from the specified section in a private INI file. You can remove an entire section, instead of just a single line, by specifying a keyword of @WHOLESECTION. Case is not significant in section or keyname.

IniDeletePvt("Current Users", "Excel", "meter.ini")

See Also:
IniDelete, **IniItemizePvt**, **IniReadPvt**, **IniWritePvt**

IniItemize

Lists keywords or sections in WIN.INI.

Syntax:
    IniItemize (section)

Parameters:
    (s) section   the major heading to itemize.

Returns:
    (s)   list of keywords or sections.

IniItemize will scan the specified section in WIN.INI, and return a space-delimited list of all keyword names contained within that section. If a null string ("") is given as the section name, **IniItemize** will return a list of all section names contained within WIN.INI. Case is not significant in section names.

Examples:
```
; Returns all keywords in the [Extensions] section
keywords = IniItemize("Extensions")

; Returns all sections in the entire WIN.INI file
sections = IniItemize("")
```

See Also:
IniDelete, **IniItemizePvt**, **IniRead**, **IniWrite**

IniItemizePvt

Lists keywords or sections in a private INI file.

Syntax:
IniItemizePvt (section, filename)

Parameters:
(s) section the major heading to itemize.
(s) filename name of the INI file.

Returns:
(s) list of keywords or sections.

IniItemizePvt will scan the specified section in a private INI file, and return a space-delimited list of all keyword names contained within that section. If a null string ("") is given as the section name, **IniItemizePvt** will return a list of all section names contained within the file. Case is not significant in section names.


```
; Returns all keywords in the [Boot] section of SYSTEM.INI
keywords = IniItemizePvt("Boot", "system.ini")
```

See Also:
IniDeletePvt, **IniItemize**, **IniReadPvt**, **IniWritePvt**

IniRead

Reads data from the WIN.INI file.

Syntax:
IniRead (section**,** keyname**,** default**)**

Parameters:
"section" = the major heading to read the data from.
"keyname = the name of the item to read.
"default" = string to return if the desired item is not found.

Returns:
(string) data from WIN.INI file.

This function allows a program to read data from the **WIN.INI** file.
The WIN.INI file has the form:

[section**]**

keyname**=**settings

Most of the entries in WIN.INI are set from the Windows **Control Panel** program, but individual applications can also use it to store option settings in their own sections.


```
; Find the default output device
a = IniRead("windows", "device", "No Default")
Message("Default Output Device", a)
```

See Also:
IniWrite, **IniReadPvt**, **IniWritePvt**, **Environment**

IniReadPvt

Reads data from a private INI file.

Syntax:
IniReadPvt (section, keyname, default, filename)

Parameters:
"section" = the major heading to read the data from.
"keyname = the name of the item to read.
"default" = string to return if the desired item is not found.
"filename" = name of the INI file.

Returns:
(string) data from the INI file.

Looks up a value in the "filename".INI file. If the value is not found, the "default" will be returned.

IniReadPvt("Main", "Lang", "English", "WB.INI")

Given the following segment from WB.INI:
[Main]
Lang=French
The batch file line above would return:
French

See Also:
IniWritePvt, **IniRead**, **IniWrite**

IniWrite

Writes data to the **WIN.INI** file.

Syntax:
IniWrite (section**,** keyname**,** data**)**

Parameters:
"section" = major heading to write the data to.
"keyname = name of the data item to write.
"data" = string to write to the WIN.INI file.

Returns:
(integer) always **@TRUE.**

This command allows a program to write data to the **WIN.INI** file. The "section" is added to the file if it doesn't already exist.


```
; Change the list of pgms to load upon Windows
; startup
loadprogs = IniRead("windows", "load", "")
newprogs = AskLine("Add Pgm To LOAD= Line", "Add:", loadprogs)
IniWrite("windows", "load", newprogs)
```

See Also:
IniRead, **IniReadPvt**, **IniWritePvt**

IniWritePvt

Writes data to a private INI file.

Syntax:
IniWritePvt (section, keyname, data, filename)

Parameters:
"section" = major heading to write the data to.
"keyname = name of the data item to write.
"data" = string to write to the INI file.
"filename" = name of the INI file.

Writes a value in the "filename".INI file.

IniWritePvt("Main", "Lang", "French, "WB.INI")

This would create the following entry in WB.INI:
[Main]
Lang=French

See Also:
IniReadPvt, **IniRead**, **IniWrite**

IntControl

Internal control functions.

Syntax:
IntControl (request#, p1, p2, p3, p4)

Parameters:
(i) request# specifies which sub-function is to be performed (see below).
(s) p1 - p4 parameters which may be required by the function (see below).

Returns:
(s) varies (see below).

Short for Internal Control, a special function that permits numerous internal operations in the CP and WB products. The first parameter of IntControl defines exactly what the function does, the other parameters are possible arguments to the function.
Warning: Many of these operations are useful only under special circumstances, and/or by technically knowledgeable users. Some could lead to adverse side effects. If it isn't clear to you what a particular function does, don't use it.

IntControl (1, p1, 0, 0, 0)
Just a test IntControl. It echoes back P1 & P2 and P3 & P4 in a pair of message boxes.

IntControl (2, 0, 0, 0, 0) (CP only)
Returns the number of Command Post windows currently open.

IntControl (3, 0, 0, 0, 0) (CP only)
Writes the positions of each open Command Post window to the WWW-PROD.INI file, using the WinPositionXY format.

IntControl (4, p1, 0, 0, 0)
Controls whether or not a dialog box with a file listbox in it has to return a file name, or may return merely a directory name or nothing.

P1	Meaning
0	May return nothing, or just a directory name
1	Must return a file name (default)

IntControl (5, p1, 0, 0, 0)
Controls whether system & hidden files are seen and processed.

P1	Meaning
0	System & Hidden files not used (default)
1	System & Hidden files seen and used

IntControl (6, 0, 0, 0, 0) (CP only)
Positions all open Command Post windows, based on the information in the WWW-PROD.INI file.

IntControl (8, 0, 0, 0, 0) (CP only)
Reloads Command Post menus, just like selecting "Reload Menu" from the system menu.

IntControl (9, p1, 0, 0, 0) (CP only)
Controls Command Post window resizing.

P1	Meaning
0	Resize automagically on open and close (default)
1	disable resize on window close
2	disable resize on window open
3	disable resize on open and close

IntControl (10, p1, 0, 0, 0)
Interrogates the Command Extender DLL status

P1	Meaning
0	Command Extender present
	0 No
	1 Yes
1	Command Extender version
	-1 No Extender present
	0 Incompatible extender present
	(other) Extender version code
2	Interpreter's Extender interface code
3	Name of Extender DLL

IntControl (11, p1, 0, 0, 0) (CP only)
Used to tell Command Post that it is (or is not) a shell, contrary to what it really is. That is, if it is really a shell, you can disable the shell-like characteristics, or if it is not a shell, enable its shell characteristics.

P1	Meaning
0	Play standard app
1	Play shell

IntControl (12, p1, 0, 0, 0) (WB only)
Used to direct WinBatch to allow itself to be terminated without warning when Windows shuts down and a batch file is still running

P1	Meaning

0 WinBatch complains on shutdown (default)

1 WinBatch will terminate quietly

IntControl (15, 0, 0, 0, 0) (WB only)
Returns currently executing WBT file name; the same as the "paramfile" variable.

IntControl (18, 0, 0, 0, 0)
Suspends the program (WB or CP) waiting for some other process to do the equivalent of IntControl(19). **This command will hang your system if used improperly**.

IntControl (19, p1, 0, 0, 0)
Un-suspends a process stopped with IntControl(18). P1 is a window handle (not a window title). Windows handles may be derived from window titles using IntControl(21).

IntControl (20, 0, 0, 0, 0)
Returns window handle of current Command Post or WinBatch window.

IntControl (21, p1, 0, 0, 0)
Returns window handle of window matching the partial window-name in p1.

IntControl (22, p1, p2, p3, p4)
Issues a Windows "SendMessage".

p1 Window handle to send to

p2 Message ID number (in decimal)

p3 wParam value

p4 assumed to be a character string. String is copied to a GMEM_LOWER buffer, and a LPSTR to the copied string is passed as lParam. The GMEM_LOWER buffer is freed immediately upon return from the SendMessage

IntControl (23, 0, 0, 0, 0)
Issues a windows PostMessage

p1 Window handle

p2 Message ID number (in decimal)

p3 wParam

p4 lParam -- assumed to be numeric

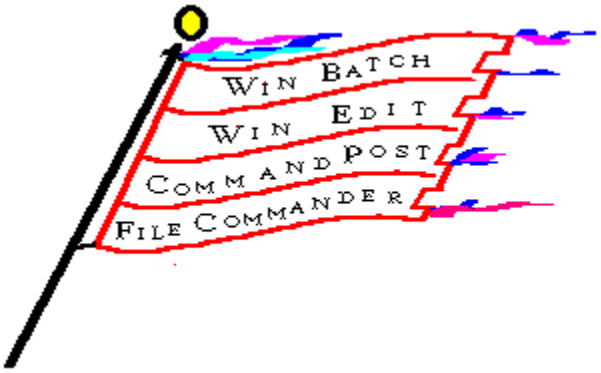
IntControl (66, 0, 0, 0, 0)
Restarts Windows, just like exiting to DOS and typing WIN again. Could be used to restart Windows after editing the SYSTEM.INI file to change video modes.

IntControl (67, 0, 0, 0, 0)
Performs a warm boot of the system, just like <Ctrl-Alt-Del>. Could be used to reboot the system after editing the AUTOEXEC.BAT or CONFIG.SYS files.
Note: IntControl(67) works only in Windows 3.1 or higher. In Windows 3.0, it behaves just like IntControl(66) and restarts Windows.

The WIL Language

Windows At Your Command

Introduction to the WIL language for Windows



WIL gives you the capability to control Windows operating on personal computers and on networks. You can do this for yourself or others with scripts that handle applications and data exchange. The WIL language is the centerpiece of the WinBatch batch language. It lets you add scripts to the menus of any Windows application.

A counterpart, Command Post. serves as a general Windows manager. It gives any user the capability of tailoring the Windows environment to their needs. launching applications, creating work sets, displaying windows, and terminating applications are just the beginning of what you can do with Command Post.

The WIL language is included in other products. PubTech Corporation's BatchWorks is an implementation of WIL. The Norton Desktop for Windows includes one as well. The WindowWare programmer's editor, WinEdit, includes a version in its macro language.

The WIL language gives you more than a hundred sixty functions and commands. Many new commands are useful in administering networks, exchanging data among Windows applications, and in customizing the Windows environment. This help application is divided into three main areas: Command ListTask list., and WIL Language. Extended help is provided in dialog box construction.

You will find Commands and help sections cross referenced by hot words. These hot words are underlined and displayed in a contrasting color. The standard Windows color, green, for these hypertext words is often considered unattractive or illegible. You can change it. To do so, you add these lines to the [Windows Help] section of your WIN.INI file (other statements may be already present in this section. Keep them there.):

```
[Windows Help]
JumpColor=0 0 255
PopupColor=255 0 0
```

The numbers specify colors in RGB format. You can see the relationship between the colors and the numbers by exploring the *Options Edit Colors* area of the Paintbrush application that comes with Windows.

You will want to use the WIL Help application while you are writing scripts. The indexes and hypertext references will be useful, but there are more features you can use to advantage. The *Edit Copy* menu item lets you copy the sample code into your scripts. The annotation feature lets you create your own comments on any page in WIL Help. Annotations are indicated on the page with a paperclip icon. They are held in a *wilhelp.ann* file in your Windows directory. Finally, you can add a menu item to WIL Help that serves as a bookmark. Use this feature to create a custom quick access menu system.

In the WIL language, we use a shorthand notation to indicate the syntax of the functions.

Function names and other actual characters you type are in **boldface**. Optional parameters are enclosed in square brackets "**[ ]**". When a function takes a variable number of parameters, the variable parts will be followed by ellipses ("**...**").

Take, for example, string concatenation:

StrCat (string[**,** string]...**)**

This says that the **StrCat** function takes at least one string parameter. Optionally, you can specify more strings to concatenate. If you do, you must separate the strings with commas.

For each function and command, we show you the **Syntax**, describe the **Parameters** (if any), the value it **Returns** (if any), a description of the function, **Example** code, and related functions. You may want to explore the "**See Also" commands**. This is particularly handy in this WIL Language Help application, since most of these are hypertext topics.

IsDefined

Determines if a variable name is currently defined.

Syntax:
IsDefined (var**)**

Parameters:
"var" = a variable name.

Returns:
(integer) **@YES** if the variable is currently defined;
@NO if it was never defined or has been dropped.

A variable is defined the first time it appears at the left of an equal sign in a statement. It stays defined until it is explicitly dropped with the **Drop** function, or until the batch file ends.


```
def = IsDefined(thisvar)
If def == @FALSE Then Message("ERROR!", "Variable not      defined")
```

See Also:
Drop

IsKeyDown

Tells about keys/mouse.

Syntax:
    IsKeyDown(keycodes)

Parameters:
    keycodes =   **@SHIFT** and/or **@CTRL**

Returns:
    (integer)   **@YES** if the key is down.
                @NO if the key is not down.

Determines if the **Shift** key or the **Ctrl** key is currently down.
Note: The right mouse button is the same as **Shift**, and the middle mouse button is the same as **Ctrl**.

Examples:
    IsKeyDown(@SHIFT)

    IsKeyDown(@CTRL)

    IsKeyDown(@CTRL | @SHIFT)

    IsKeyDown(@CTRL & @SHIFT)

IsLicensed

Tells if the WIL interpreter is licensed.

Syntax:
IsLicensed()

Parameters:
(none)

Returns:
(integer) **@YES** if current version of the WIL interpreter is licensed.
@NO if current version of the WIL interpreter is not licensed.

Returns information on whether or not the current version of the WIL interpreter is a licensed copy.

IsLicensed

IsNumber

Determines whether a variable contains a valid number.

Syntax:
IsNumber (string**)**

Parameters:
"string" = string to test to see if it represents a valid number.

Returns:
(integer) **@YES** if it contains a valid number;
@NO if it doesn't.

This function determines if a string variable contains a valid integer. Useful for checking user input prior to using it in computations.


```
a = AskLine("ISNUMBER", "Enter a number", "0")
If IsNumber(a) == @NO Then Message("", "You didn't enter a          number")
```

See Also:
Abs, **Char2Num**

IsRunning (Command Post only)

Determines if another copy of Command Post is currently running.

Syntax:
IsRunning ( )

Returns:
(integer) **@YES** if another copy of Command Post is running;
@NO if this is the only one.

There is no artificial restraint on the number of copies of Command Post you may run at once.


```
a=!(IsRunning())
Is = strsub("not ", 1, 4*a)
Message("", "Another Command Post is %Is% running.")
Drop(a, Is)
```

See Also:
OtherDir, **OtherUpdate**

ItemCount

Returns the number of items in a list.

Syntax:
ItemCount (list**,** delimiter**)**

Parameters:
"list" = a string containing a list of items to choose from.
"delimiter" = a string containing the character to act as delimiter between items in the list.

Returns:
(integer) the number of items in the list.

If you create the list with the **FileItemize** or **DirItemize** functions you will be using a space-delimited list. **WinItemize**, however, creates a tab-delimited list of window titles since titles can have embedded blanks.


```
a = FileItemize("*.*")
n = ItemCount(a, " ")
Message("Note", "There are %n% files")
```

See Also:
DirItemize, **FileItemize**, **WinItemize**, **ItemExtract**, **ItemSelect**

ItemExtract

Returns the selected item from a list.

Syntax:
ItemExtract (select**,** list**,** delimiter**)**

Parameters:

select =	the position in "list" of the item to be selected.
"list" =	a string containing a list of items to choose from.
"delimiter" =	a string containing the character to act as delimiter between items in the list.

Returns:

(string)	the selected item.

If you create the list with the **FileItemize** or **DirItemize** functions you will be using a space-delimited list. **WinItemize**, however, creates a tab-delimited list of window titles since titles can have embedded blanks.


```
bmpfiles = FileItemize("*.bmp")
bmpcount = ItemCount(bmpfiles, " ")
pos = (Random(bmpcount - 1)) + 1
paper = ItemExtract(pos, bmpfiles, " ")
Wallpaper(paper, @FALSE)
```

See Also:
DirItemize, **FileItemize**, **WinItemize**, **ItemCount**, **ItemSelect**

ItemSelect

Allows the user to choose an item from a listbox.

Syntax:
ItemSelect (title**,** list**,** delimiter**)**

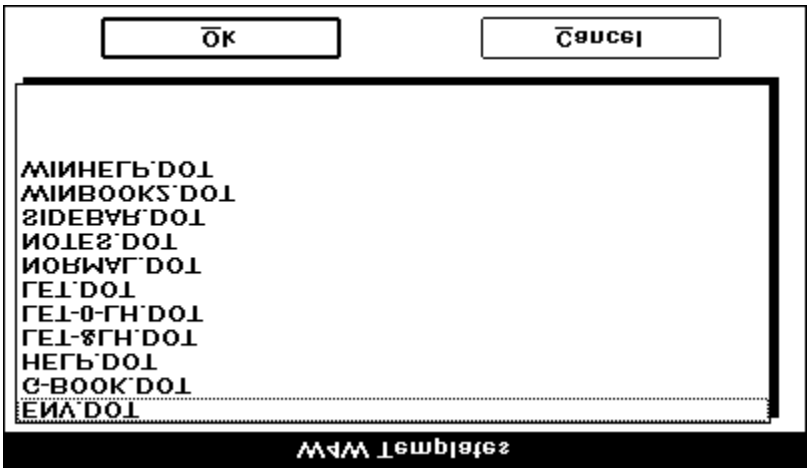
Parameters:
"title" = the title of dialog box to display.
"list" = a string containing a list of items to choose from.
"delimiter" = a string containing the character to act as delimiter between items in the list.

Returns:
(string) the selected item.

This function displays a dialog box with a listbox inside. This listbox is filled with a sorted list of items taken from a string you provide to the function.
Each item in the string must be separated ("delimited") by a character, which you also pass to the function.
The user selects one of the items by either doubleclicking on it, or single-clicking and pressing OK. The item is returned as a string.
If you create the list with the **FileItemize** or **DirItemize** functions you will be using a space-delimited list. **WinItemize**, however, creates a tab-delimited list of window titles since titles can have embedded blanks.


```
DirChange("e:\word")
alldotfiles = FileItemize("*.dot")
dotfile = ItemSelect("W4W Templates", alldotfiles, " ")
Run("winword.exe", dotfile)
```

Which would produce:



See Also:

AskYesNo, **Display**, **DirItemize**, **FileItemize**, **WinItemize**, **Message**, **Pause**, **TextBox**, **ItemCount**, **ItemExtract**

LastError

Returns the most-recent error encountered during the current batch file.

Syntax:
LastError ( )

Parameters:
(none)

Returns:
(integer) most-recent WIL error code encountered.

WIL errors are numbered according to their severity. "Minor" errors go from 1000 through 1999. Moderate errors are 2000 through 2999. Fatal errors are numbered 3000 to 3999.
Depending on which error mode is active when an error occurs, you may not get a chance to check the error code. See **ErrorMode** for a discussion of default error handling.
Don't bother checking for "fatal" error codes. When a fatal error occurs, the batch file is canceled before the next WIL statement gets to execute (regardless of which error mode is active).
Every time the **LastError** function is called, the "last error" indicator is reset to zero.
A full listing of possible errors you can encounter in processing a batch file is in **Appendix B** (pg.).


```
ErrorMode(@OFF)
FileCopy("data.dat", "c:\backups", @FALSE)
ErrorMode(@CANCEL)
If LastError() == 1006 Then Message("Error", "Please call      Tech Support at 555-9999.")
```

See Also:
Debug, **ErrorMode**

LogDisk

Logs (activates) a disk drive.

Syntax:
LogDisk (drive-letter**)**

Parameters:
"drive-letter" = the disk drive to log into.

Returns:
(integer) **@TRUE** if the current drive was changed;
@FALSE if the drive doesn't exist.

Use this function to change the logged disk drive.
This command produces the same effect as if you typed the drive name from the DOS command prompt.

LogDisk("c:")

See Also:
DirChange

Max

Returns largest number in a list of numbers.

Syntax:
Max (integer [**,** integer]...**)**

Parameters:
integer = an integer number.

Returns:
(integer) largest parameter.

Use this function to determine the largest of a set of comma-delimited integers.


```
a = Max(5, -37, 125, 34, 2345, -32767)
Message("Largest number is", a)
```

See Also:
Abs, **Average**, **Min**

MenuChange (Command Post only)

Checks, unchecks, enables, or disables a menu item.

Syntax:
MenuChange (menuname**,** flags**)**

Parameters:
"menuname" = menu item whose status you wish to change.
"flags" = **@CHECK**, **@UNCHECK**,
@ENABLE, or **@DISABLE**.

Returns:
(integer) always **@TRUE**.

There are currently two ways you can modify a menu item:
You can check and uncheck the item to imply that it corresponds to an option that can be turned on or off.
You can temporarily disable the item (it shows up as gray) and later re-enable it.
The two sets of flags (**@Check**/**@UnCheck** and **@Enable**/**@Disable**) can be combined in one function call by using the **|** (or) operator.


```
MenuChange (FilePrint, @Disable)
MenuChange (WPWrite, @Enable|@Check)
```

See Also:
IsMenuChecked, **IsMenuEnabled**

Message

Displays a message to the user.

Syntax:
Message (title**,** text**)**

Parameters:
"title" = title of the message box.
"text" = text to display in the message box.

Returns:
(integer) always **@TRUE**.

Use this function to display a message to the user. The user must respond by selecting the **OK** button before processing will continue.

Message("Current directory is", DirGet())
which produces:

See Also:
Display, **Pause**

Min

Returns lowest number in a list of numbers.

Syntax:
Min (integer [**,** integer]...**)**

Parameters:
integer = an integer number.

Returns:
(integer) lowest parameter.

Use this function to determine the lowest of a set of comma-delimited integers.


```
a = Min( 5, -37, 125, 34, 2345, -32767)
Message("Smallest number is", a)
```

See Also:
Abs, **Average**, **Max**

MouseInfo

Returns assorted mouse information.

Syntax:
MouseInfo (request#)

Parameters:
(i) request# see below.

Returns:
(s) see below.

The information returned by **MouseInfo** depends on the value of request#.

Req#	Return value
0	Window name under mouse
1	Top level parent window name under mouse
2	Mouse coordinates, assuming a 1000x1000 virtual screen
3	Mouse coordinates in absolute numbers
4	Status of mouse buttons, as a bitmask:

Binary	Decimal	Meaning
000	0	No buttons down
001	1	Right button down
010	2	Middle button down
011	3	Right and Middle buttons down
100	4	Left button down
101	5	Left and Right buttons down
110	6	Left and Middle buttons down
111	7	Left, Middle, and Right buttons down

For example, if mouse is at the center of a 640x480 screen and above the "Clock" window, and the left button is down, the following values would be returned:

Req#	Return value
1	"Clock"
2	"500 500"
3	"320 240"

4 "4"


```
Display(1, "", "Press a mouse button to continue")
:loop
buttons = MouseInfo(4)
If buttons == 0 Then Goto loop
If buttons & 4 Then Display(1, "", "Left button was pressed")
If buttons & 1 Then Display(1, "", "Right button was pressed")
```

See Also:

WinMetrics, **WinParmGet**

NetAddCon

Connects network resources to imaginary local disk drives or printer ports.

Syntax:
NetAddCon (net-path, password, local-name)

Parameters:
(s) net-path net resource or string returned by **x**.
(s) password password required to access resource, or "".
(s) local-name local drive name or printer port.

Returns:
(i) @TRUE if successful; @FALSE if unsuccessful.

You can use **NetAddCon** to connect a local drive to a network directory, in which case "local-name" will be a drive name (eg, "Z:"). You can also connect a local printer port to a network print queue, in which case "local-name" will be the name of the printer port (eg, "LPT1").
Use the **NetBrowse** function to obtain a value for "net-path".
If no password is required, use a null string ("") for the "password" parameter.


```
availdrive = DiskScan(0)
drvlen = StrLen(availdrive)
If drvlen == 0 Then Goto nomore
availdrive = StrSub(availdrive, drvlen - 2, 2)
netpath = NetBrowse(0)
pswd = AskPassword("Enter password for", netpath)
NetAddCon(netpath, pswd, availdrive)
Exit
:nomore
Message("Connect Drive to Net", "No drives avail for assignment")
```

See Also:
NetBrowse, **NetCancelCon**, **NetGetCon**

NetBrowse

Displays a dialog box allowing the user to select a network resource.

Syntax:
NetBrowse (request#)

Parameters:
(i) request# see below.

Returns:
(s) see below.

Displays a dialog box allowing the user to select a network resource. Request#=0 allows selection of a print queue and Request#=1 allows selection of a network directory. This function returns a string that can be used by **NetAddCon** to add a connection.


```
availdrive = DiskScan(0)
drvlen = StrLen(availdrive)
If drvlen == 0 Then Goto nomore
availdrive = StrSub(availdrive, drvlen - 2, 2)
netpath = NetBrowse(0)
pswd = AskPassword("Enter password for", netpath)
NetAddCon(netpath, pswd, availdrive)
Exit
:nomore
Message("Connect Drive to Net", "No drives avail for assignment")
```

See Also:
NetAddCon

NetCancelCon

Breaks a network connection.

Syntax:
NetCancelCon (name, force)

Parameters:

(s) name	network resource name or local name.
(i) force	force flag (see below).

Returns:

(i)	@TRUE if successful; @FALSE if unsuccessful.

If "force" is set to 0, **NetCancelCon** will not break the connection if any files on that connection are still open. If "force" is set to 1, the connection will be broken regardless.


```
availdrive = DiskScan(4)
n = ItemCount(availdrive, " ")
If n == 0 Then Exit
i = 1
dislist = ""
:loop
drv = ItemExtract(i, availdrive, " ")
dislist = StrCat(drv, Num2Char(9), NetGetCon(drv), "|")
i = i + 1
If i < n Then Goto loop
availdrive = ItemSelect("Disconnect", dislist, "|")
NetCancelCon(availdrive, 0)
```

See Also:
NetAddCon, NetGetCon

NetDialog

Brings up the network driver's dialog box.

Syntax:
NetDialog ( )

Parameters:
(none)

Returns:
(i) @TRUE if successful; @FALSE if unsuccessful.

A network driver's dialog box displays copyright information, and may allow access to the network, depending on the particular network driver. The WIL program will wait until the network dialog terminates before continuing.

NetDialog()
DiskUpdate()

See Also:
DiskReset, DiskUpdate

NetGetCaps

Returns information on network capabilities.

Syntax:
NetGetCaps (request#)

Parameters:
(i) request# see below.

Returns:
(i) see below.

NetGetCaps returns 0 if no network is installed (it is the only network function you can use without having a network installed and not get an error).

Req#	Return value
1	Network driver specification number
2	Type of network installed:
	0 None
	256 MS Network
	512 Lan Manager
	768 Novell NetWare
	1024 Banyan Vines
	1280 10 Net
	(other) Other network
3	Network driver version number
4	Returns 1 if any network is installed
6	Bitmask indicating whether the network driver supports the following connect functions:
	1 AddConnection
	2 CancelConnection
	4 GetConnection
	8 AutoConnect via DOS
	16 BrowseDialog
7	Bitmask indicating whether the network driver supports the following

print functions:

2	Open Print Job
4	Close Print Job
16	Hold Print Job
32	Release Print Job
64	Cancel Print Job
128	Set number of copies
256	Watch Print Queue
512	Unwatch Print Queue
1024	Lock Queue Data
2048	Unlock Queue Data
4096	Driver will send QueueChanged messages to Print Manager
8192	Abort Print Job


```
caps = NetGetCaps(6)
If caps & 16 Then Message("", "Your network supports BrowseDialog")
```

See Also:

NetGetUser, **WinConfig**, **WinMetrics**, **WinParmGet**

NetGetCon

Returns the name of a connected network resource.

Syntax:
NetGetCon (local-name)

Parameters:
(s) local-namelocal drive name or printer port.

Returns:
(s) name of network resource.

NetGetCon returns the name of the network resource currently connected to "local-name".


```
local = AskLine("NetGetCon", "Enter local drive name", "")
If local == "" Then Exit
resource = NetGetCon(local)
Message("NetGetCon", "%local% is connected to %resource%")
```

See Also:
NetAddCon, **NetCancelCon**

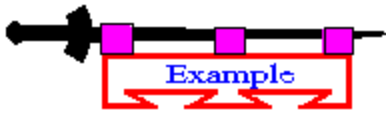
NetGetUser

Returns the name of the user currently logged into the network.

Syntax:
NetGetUser ( )

Parameters:
(none)

Returns:
(s) name of current user.



IniWritePvt("Current Users", "Excel", NetGetUser(), "usagelog.ini")
Run("excel.exe", "")

See Also:
NetGetCaps

Num2Char

Converts a number to its character equivalent.

Syntax:
Num2Char (integer**)**

Parameters:
number = any number from **0** to **255**.

Returns:
(string) one-byte string containing the character the number represents.

Use this function to convert a number to its ASCII equivalent.


```
; Build a variable containing a CRLF combo
crlf = StrCat(Num2Char(13), Num2Char(10))
Message("NUM2CHAR", StrCat("line1", crlf, "line2"))
```

See Also:
Char2Num

OtherDir (Command Post only)

Finds the directory where the other copy of Command Post is running, if any.

Syntax:
OtherDir ( )

Parameters:
"string" = pathname to "other" directory.

Returns:
(string) the directory of the second-most recently used Command Post window. The current window is considered the most recently used directory.

Use this command to determine directory of the other Command Post window. Useful in setting up copy and move operations between two Command Post copies.


```
a=DirGet()
b=OtherDir()
Message("Directory of this CmdPost window is",a)
Message("Directory of the other CmdPost window is",b)
```

See Also:
DirGet, **DirHome**, **OtherUpdate**

OtherUpdate (Command Post only)

Updates another Command Post directory display.

Syntax:
OtherUpdate ( )

Returns:
(integer) **@TRUE** if another copy of Command Post was found to update;
@FALSE if this is the only copy running.

This command updates the File Manager display of the next-most recently invoked copy of Command Post. This is useful if your menu item changes a directory; i.e. if a file or directory is created, moved, renamed, or deleted. **OtherUpdate** helps ensure the other Command Post display immediately reflects the change the user caused from *this* copy.


```
FileCopy ("MyFile.txt", OtherDir(), @FALSE)
OtherUpdate ()
```

See Also:
OtherDir, **SetDisplay**

ParseData (WB)

Parses the passed string, just like passed parameters are parsed.

Syntax:
ParseData (string**)**

Parameters:
"string" = string to be parsed.

Returns:
(integer) number of parameters in "string".

This function breaks a string constant or string variable into new sub-string variables named **param1**, **param2**, etc. (maximum of nine parameters). Blank spaces in the original string are used as delimiters to create the new variables.
Param0 is the count of how many sub-strings are found in "string".


```
username = AskLine("Hello", "Please enter your name","")
ParseData(username)
```

If the user enters:
Joe Q. User
ParseData would create the following variables:
param1 == **Joe**
param2 == **Q.**
param3 == **User**
param0 == **3**

Pause

Provides a message to user. User may cancel processing.

Syntax:
Pause (title**,** text**)**

Parameters:
"title" = title of pause box.
"text" = text of the message to be displayed.

Returns:
(integer) always **@TRUE**.

This function displays a message to the user with an exclamation point icon. The user may respond by selecting the **OK** button, or may cancel the processing by selecting **Cancel**. The **Pause** function is similar to the **Message** function, except for the addition of the **Cancel** button and icon.

Pause("Change Disks", "Insert new disk into Drive A:")

which produces:

See Also:
Display, **Message**

PlayMedia

Controls multimedia devices.

Syntax:
PlayMedia (command-string)

Parameters:
(s) command-string string to be sent to the multimedia device.

Returns:
(s) response from the device.

If the appropriate Windows multimedia extensions are present, this function can control multimedia devices. Valid command strings depend on the multimedia devices and drivers installed. The basic Windows multimedia package has a waveform device to play and record waveforms, and a sequencer device to play MID files. Refer to the appropriate documentation for information on command strings.
Many multimedia devices accept the WAIT or NOTIFY parameters as part of the command string:

WAIT Causes the system to stop processing input until the requested operation is complete. You cannot switch tasks when WAIT is specified.

NOTIFY Causes the WIL program to suspend execution until the requested operation completes. You can perform other tasks and switch between tasks when NOTIFY is specified.

WAIT NOTIFY Same as WAIT

If neither WAIT nor NOTIFY is specified, the multimedia operation is started and control returns immediately to the WIL program.
In general, if you simply want the WIL program to wait until the multimedia operation is complete, use the NOTIFY keyword. If you want the system to hang until the operation is complete, use WAIT. If you just want to start a multimedia operation and have the program continue processing, don't use either keyword.
The return value from **PlayMedia** is whatever string the driver returns. This will depend on the particular driver, as well as on the type of operation performed.


```
; Plays a music CD on a CDAudio player.  It plays whatever is in the
; drive, from start to finish
stat = PlayMedia("status cdaudio mode")
answer = 1
If stat == "playing" Then answer = AskYesNo("CD Audio", "CD is
     Playing.  Stop?")
If answer == 0 Then Exit
PlayMedia("open cdaudio shareable alias donna notify")
PlayMedia("set donna time format tmsf")
PlayMedia("play donna from 1")
PlayMedia("close donna")
```

```
Exit
:cancel
PlayMedia("set cdaudio door open")
```

See Also:
PlayMidi, **PlayWaveForm**

PlayMidi

Plays a MID or RMI sound file.

Syntax:
PlayMidi (filename, mode)

Parameters:

(s) filename	name of the MID or RMI sound file.
(i) mode	play mode (see below).

Returns:

(i)	@TRUE if successful; @FALSE if unsuccessful.

If Windows multimedia sound extensions are present, and MIDI-compatible hardware is installed, this function will play a MID or RMI sound file. If "filename" is not in the current directory and a directory is not specified, the path will be searched to find the file.
If "mode" is set to 0, the WIL program will wait for the sound file to complete before continuing. If "mode" is set to 1, it will start playing the sound file and continue immediately.


```
PlayMidi("canyon.mid", 1)
```

See Also:
PlayMedia, **PlayWaveForm**

PlayWaveForm

Plays a WAV sound file.

Syntax:
PlayWaveForm (filename, mode)

Parameters:
(s) filename
(i) mode play mode (see below).

Returns:
(i) @TRUE if successful; @FALSE if unsuccessful.

If Windows multimedia sound extensions are present, and waveform-compatible hardware is installed, this function will play a WAV sound file. If "filename" is not in the current directory and a directory is not specified, the path will be searched to find the file. If "filename is not found, the WAV file associated with the "SystemDefault" keyword is played, (unless the "NoDefault" setting is on).
Instead of specifying an actual filename, you may specify a keyword name from the [Sound] section of the WIN.INI file (eg, "SystemStart"), in which case the WAV file associated with that keyword name will be played.
"Mode" is a bitmask, composed of the following bits:

Mode	Meaning
0	Wait for the sound to end before continuing.
1	Don't wait for the sound to end. Start the sound and immediately process more statements.
2	If sound file not found, do not play a default sound
9	Continue playing the sound forever, or until a **PlayWaveForm("", 0)** statement is executed
16	If another sound is already playing, do not interrupt it. Just ignore this PlayWaveForm request.

You can combine these bits using the binary OR operator.
The command **PlayWaveForm("", 0)** can be used at any time to stop sound.

Examples:

```
PlayWaveForm("tada.wav", 0)

PlayWaveForm("SystemDefault", 1 | 16)
```

See Also:
PlayMedia, **PlayMidi**

Random

Computes a pseudo-random number.

Syntax:
Random (max**)**

Parameters:
max = largest desired integer number.

Returns:
(integer) unpredictable positive number.

This function will return a random integer between **0** and "max".


```
a = Random(79)
Message("Random number between 0 and 79", a)
```

Return

Used to return from a **Call** or a **CallExt** to the calling program.

Syntax:
Return

If the program was not called, then an **Exit** is assumed.


```
Display(2, "End of subroutine", "Returning to MAIN.WBT")
Return
```

See Also:
Call, **CallExt**, **Exit**

Run

Runs a program as a normal window.

Syntax:
Run (program-name**,** parameters**)**

Parameters:
"program-name" =the name of the desired **.EXE**, **.COM**, **.PIF, .BAT** file, or a data file.
"parameters" = optional parameters as required by the application.

Returns:
(integer) **@TRUE** if the program was found;
@FALSE if it wasn't.

Use this command to run an application.
If the drive and path are not part of the program name, the current directory will be examined first, and then the DOS path will be searched to find the desired executable file.
If the "program-name" doesn't have an extension of .EXE, .COM, .PIF, or .BAT, it will be run in accordance with whatever is in the **[extensions]** section of the WIN.INI file. When this happens, any "parameters" you specified are ignored.

Examples:

```
Run("notepad.exe", "abc.txt")

Run("clock.exe", "")

Run("paint.exe", "pict.msp")
```

See Also:
RunHide, **RunIcon**, **RunZoom**, **WinClose**, **WinWaitClose**

RunHide

Runs a program as a hidden window.

Syntax:
RunHide (program-name**,** parameters**)**

Parameters:
"program-name" =the name of the desired **.EXE**, **.COM**, **.PIF, .BAT** file, or a data file.
"parameters" = optional parameters as required by the application.

Returns:
(integer) **@TRUE** if the program was found;
@FALSE if it wasn't.

Use this command to run an application as a hidden window.
If the drive and path are not part of the program name, the current directory will be examined first, and then the DOS path will be searched to find the desired executable file.
If the "program-name" doesn't have an extension of .EXE, .COM, .PIF, or .BAT, it will be run in accordance with whatever is in the **[extensions]** section of the WIN.INI file. When this happens, any "parameters" you specified are ignored.
Note: When this command launches an application, it informs it that you want it to run as a hidden window. Whether or not the application honors your wish is beyond **RunHide**'s control.

Examples:
```
RunHide("notepad.exe", "abc.txt")

RunHide("clock.exe", "")

RunHide("paint.exe", "pict.msp")
```

See Also:
Run, **RunIcon**, **RunZoom**, **WinHide**, **WinClose**, **WinWaitClose**

RunIcon

Runs a program as an iconic (minimized) window.

Syntax:
RunIcon (program-name**,** parameters**)**

Parameters:
"program-name" =the name of the desired **.EXE**, **.COM**, **.PIF, .BAT** file, or a data file.
"parameters" = optional parameters as required by the application.

Returns:
(integer) **@TRUE** if the program was found;
@FALSE if it wasn't.

Use this command to run an application as an icon.
If the drive and path are not part of the program name, the current directory will be examined first, and then the DOS path will be searched to find the desired executable file.
If the "program-name" doesn't have an extension of .EXE, .COM, .PIF, or .BAT, it will be run in accordance with whatever is in the **[extensions]** section of the WIN.INI file. When this happens, any "parameters" you specified are ignored.
Note: When this command launches an application, it merely informs it that you want it to begin as an icon. Whether or not the application honors your wish is beyond **RunIcon**'s control.

Examples:

```
RunIcon("notepad.exe", "abc.txt")

RunIcon("clock.exe", "")

RunIcon("paint.exe", "pict.msp")
```

See Also:
Run, **RunHide**, **RunZoom**, **WinIconize**, **WinClose**, **WinWaitClose**

RunZoom

Runs a program as a full-screen (maximized) window.

Syntax:
RunZoom (program-name**,** parameters**)**

Parameters:
"program-name" =the name of the desired **.EXE**, **.COM**, **.PIF, .BAT** file, or a data file.
"parameters" = optional parameters as required by the application.

Returns:
(integer) **@TRUE** if the program was found;
@FALSE if it wasn't.

Use this command to run an application as a full-screen window.
If the drive and path are not part of the program name, the current directory will be examined first, and then the DOS path will be searched to find the desired executable file.
If the "program-name" doesn't have an extension of .EXE, .COM, .PIF, or .BAT, it will be run in accordance with whatever is in the **[extensions]** section of the WIN.INI file. When this happens, any "parameters" you specified are ignored.
Note: When this command launches an application, it merely informs it that you want it to be maximized to full-screen. Whether or not the application honors your wish is beyond **RunZoom**'s control.

Examples:
```
RunZoom("notepad.exe", "abc.txt")

RunZoom("clock.exe", "")

RunZoom("paint.exe", "pict.msp")
```

See Also:
Run, **RunHide**, **RunIcon**, **WinZoom**, **WinClose**, **WinWaitClose**

SendKey

Sends keystrokes to the active application.

Syntax:
SendKey (char-string)

Parameters:
"char-string" = string of regular and/or special characters.

Returns:
(integer) always **0**

This function is used to send keystrokes to the current window, just as if they had been entered from the keyboard. Any alphanumeric character, and most punctuation marks and other symbols which appear on the keyboard, may be sent simply by placing it in the "char-string." In addition, the following special characters, enclosed in "curly" braces, may be placed in "char-string" to send the corresponding special characters:

KeySendKey	equivalent
~	**{~}**
!	**{!}**
^	**{^}**
+	**{+}**
Backspace	**{BACKSPACE}** or **{BS}**
Break	**{BREAK}**
Clear	**{CLEAR}**
Delete	**{DELETE}** or **{DEL}**
Down Arrow	**{DOWN}**
End	**{END}**
Enter	**{ENTER}** or ~
Escape	**{ESCAPE}** or **{ESC}**
F1 through F16	**{F1}** through **{F16}**
Help	**{HELP}**
Home	**{HOME}**
Insert	**{INSERT}**
Left Arrow	**{LEFT}**
Page Down	**{PGDN}**
Page Up	**{PGUP}**
Right Arrow	**{RIGHT}**
Space	**{SPACE}** or **{SP}**
Tab	**{TAB}**
Up Arrow	**{UP}**

To enter an **Alt**, **Control**, or **Shift** key combination, precede the desired character with one or more of the following symbols:

Alt	**!**
Control	**^**
Shift	**+**

To enter **Alt**-**S**:
SendKey("!S")

To enter **Ctrl**-**Shift**-**F7**:
SendKey("^+{F7}")

You may also repeat a key by enclosing it in braces, followed by a space and the total

number of repetitions desired.
To type 20 asterisks:

```
SendKey("{* 20}")
```

To move the cursor down 8 lines:

```
SendKey("{DOWN 8}")
```

It is possible to use **SendKey** to send keystrokes to a DOS application, but only if you are running Windows in 386 Enhanced mode. You would then transfer the keystrokes to the DOS application via the Clipboard. It is only possible to send standard ASCII characters to DOS applications; you cannot send function key or Alt-key combinations.

Examples:

```
; Start Notepad, and use *.* for filenames
Run("notepad.exe", "")
SendKey("!FO*.*~")

; run DOS batch file which starts our editor
Run("edit.bat", "")
; wait 15 seconds for editor to load
Delay(15)
; send string (with carriage return) to the clipboard
crlf = StrCat(Num2Char(13), Num2Char(10))
ClipPut("Hello%crlf%")
; paste contents of clipboard to DOS window
SendKey("!{SP}EP")
```

See Also:

SKDebug

SetDisplay (Command Post Only)

Controls the display of files in the Command Post File Manager window.

Syntax:
SetDisplay (detail**,** sort-by**,** masks**)**

Parameters:
"detail" = level of detail. Use "**SHORT**" or "**LONG**".
"sort-by" = how to sort the filenames. Use "**NAME**", "**KIND**", "**SIZE**", "**DATE**" or "**UNSORTED**".
"masks" = list of masks for file display.

Returns:
(integer) **@TRUE** if valid options were specified;
@FALSE if invalid.

Use this command to change and/or update the file display.
Any of the fields may be null. If a field is null the previous setting is used. This command will alter the file display Parameters:, and then re-read all the files and update the display.
A special form of this command, **SETDISPLAY ("","","")**, will update the file display without changing any of the previously set Parameters.

Errors:
2105 "SetDisplay: Display type not SHORT or LONG"
2106 "SetDisplay: Sort Type not NAME, DATE, SIZE, KIND, or UNSORTED"


```
Windows &SDK
 &Show SDK Development Files
    SetDisplay("","","*.ICO *.CUR *.BMP *.DLG *.H")
```

SKDebug

Controls how **SendKey** works

Syntax:
SKDebug(mode)

Parameters:
mode = **@OFF** Keystrokes sent to application. No debug file written. Default mode.
@ON Keystrokes sent to application. Debug file written.
@PARSEONLY Keystrokes not sent to application. Debug file written.

Returns:
(integer) previous SKDebug mode.

This function allows you to direct the keystrokes generated by your **SendKey** statements to a disk file in addition to, or instead of, the application window. Normally, keystrokes are sent only to the application. If you specify **SKDebug (@ON)**, keystrokes are sent to a disk file as well as to the application. If you specify **SKDebug (@PARSEONLY)**, keystrokes are sent *only* to the disk file, and not to the application. **SKDebug (@OFF)** returns to the default mode.
By default, the file which will receive the parsed keystrokes is named **C:\@@SKDBUG.TXT**. You can override this by making an entry in your **WIN.INI** file, under the heading **[WinBatch]**:

[WinBatch]
SKDFile=debug.fil

where debug.fil is the filename, including complete path specification, that you want to receive the keystrokes.


```
Run("notepad.exe", "")
SKDebug(@ON)
SendKey("!FO*.*~")
SKDebug(@OFF)
```

See Also:
SendKey

SnapShot

Takes a snapshot of the screen and pastes it to the clipboard.

Syntax:
SnapShot (request#)

Parameters:
(i) request# see below.

Returns:
(i) always 0.

Req#	Meaning
0	Take snapshot of entire screen
1	Take snapshot of client area of parent window of active window
2	Take snapshot of entire area of parent window of active window
3	Take snapshot of client area of active window
4	Take snapshot of entire area of active window

SnapShot(2)

See Also:
ClipPut

Sounds

Turns sounds on or off.

Syntax:
    Sounds (request#)

Parameters:
    (i) request#   see below.

Returns:
    (i)   previous Sound setting.

If Windows multimedia sound extensions are present, this function turns sounds made by the WIL Interpreter on or off. Specify a request# of 0 to turn sounds off, and a request# of 1 to turn them on. By default, the WIL Interpreter makes noise.

Sounds

StrCat

Concatenates two or more strings.

Syntax:
StrCat (string1**,** string2[**,** stringN]...**)**

Parameters:
"string1", etc = at least two strings you want to "string" together (so to speak).

Returns:
(string) concatenation of the entire list of input strings.

Use this command to stick character strings together, or to format display messages.


```
user = AskLine("Login", "Your Name:", "")
Message("Login", StrCat("Hi, ", user))
; note that this will do the same:
Message("Login", "Hi, %user%")
```

See Also:
StrFill, **StrFix**, **StrTrim**

StrCmp

Compares two strings.

Syntax:
StrCmp (string1**,** string2**)**

Parameters:
"string1", "string2" = strings to compare.

Returns:
(integer) -1, 0, or 1; depending on whether string1 is less than, equal to, or greater than string2, respectively.

Use this command to determine whether two strings are equal, or which precedes the other in an ANSI sorting sequence.
Note: This command has been included for semantic completeness. The relational operators **>**, **>=**, **==**, **!=**, **<=**, and **<** provide the same capability.


```
a = AskLine("STRCMP", "Enter a test line", "")
b = AskLine("STRCMP", "Enter another test line", "")
c = StrCmp(a, b)
c = c + 1
d = StrSub("less than     equal to      greater     than", c * 12,  12)
; Note that above string is grouped into 12-character
; chunks.
; Desired chunk is removed with the StrSub statement.
Message("STRCMP", "%a% is %d% %b%")
```

See Also:
StriCmp, **StrIndex**, **StrLen**, **StrScan**, **StrSub**

StrFill

Creates a string filled with a series of characters.

Syntax:
StrFill (filler**,** length**)**

Parameters:
"filler" = a string to be repeated to create the return string. If the filler string is null, spaces will be used instead.
length = the length of the desired string.

Returns:
(string) character string.

Use this function to create a string consisting of multiple copies of the filler string concatenated together.

Message("My Stars", StrFill("*", 30))

which produces:

See Also:
StrCat,**StrFix**, **StrLen**, **StrTrim**

StrFix

Pads or truncates a string to a fixed length.

Syntax:
StrFix (base-string**,** pad-string**,** length**)**

Parameters:
"base-string" = string to be adjusted to a fixed length.
"pad-string" = appended to "base-string" if needed to fill out the desired length. If "pad-string" is null, spaces are used instead.
length = length of the desired string.

Returns:
(string) fixed size string.

This function "fixes" the length of a string, either by truncating it on the right, or by appending enough copies of pad-string to achieve the desired length.


```
a = StrFix("Henry", " ", 15)
b = StrFix("Betty", " ", 15)
c = StrFix("George", " ", 15)
Message("Spaced Names", StrCat(a, b, c))
```

which produces:

See Also:
StrFill, **StrLen**, **StrTrim**

StriCmp

Compares two strings without regard to case.

Syntax:
StriCmp (string1**,** string2**)**

Parameters:
"string1", "string2" = strings to compare.

Returns:
(integer) -1, 0, or 1; depending on whether string1 is less than, equal to, or greater than string2, respectively.

Use this command to determine whether two strings are equal, or which precedes the other in an ANSI sorting sequence, when case is ignored.


```
a = AskLine("STRICMP", "Enter a test line", "")
b = AskLine("STRICMP", "Enter another test line", "")
c = StriCmp(a, b)
c = c + 1
d = StrSub("less than     equal to      greater than", c * 12,     12)
; Note that above string is grouped into 12-character
; chunks.
; Desired chunk is removed with the StrSub statement.
Message("STRICMP", "%a% is %d% %b%")
```

See Also:
StrCmp, **StrIndex**, **StrLen**, **StrScan**, **StrSub**

StrIndex

Searches a string for a substring.

Syntax:
StrIndex (string**,** sub-string**,** start**,** direction**)**

Parameters:

"string" =	the string to be searched for a substring.
"substring" =	the string to look for within the main string.
start =	the position in the main string to begin search. The first character of a string is position **1**.
direction =	the search direction. **@FWDSCAN** searches forward, while **@BACKSCAN** searches backwards.

Returns:

(integer)	position of "sub-string" within "string"; **0** if not found.

This function searches for a substring within a "target" string. Starting at the "start" position, it goes forward or backward depending on the value of the "direction" parameter. It stops when it finds the "substring" within the "target" string, and returns its position.
A start position of **0** has special meaning depending on which direction you are scanning.
For **forward** searches, zero indicates the search should start at the *beginning* of the string.
For **reverse** searches, zero causes it to start at the *end* of the string.


```
instr = AskLine("STRINDEX", "Type a sentence:", "")
start = 1
end = StrIndex(instr, " ", start, @FWDSCAN)
If end == 0 Then Goto error
Message("STRINDEX", StrCat("The first word is: ", StrSub(instr, start, end - 1))
Exit
:error
Message("Sorry...", "No spaces found")
```

See Also:
StrLen, **StrScan**, **StrSub**

StrLen

Provides the length of a string.

Syntax:
StrLen (string**)**

Parameters:
"string" = any text string.

Returns:
(integer) length of string.

Use this command to determine the length of a string variable or expression.


```
myfile = AskLine("Filename", "File to process:", "")
namlen = StrLen(myfile)
If namlen > 13 Then Message("", "Filename too long!")
```

See Also:
StrFill, **StrFix**, **StrIndex**, **StrScan**, **StrTrim**

StrLower

Converts a string to lowercase.

Syntax:
StrLower (string**)**

Parameters:
"string" = any text string.

Returns:
(string) lowercase string.

Use this command to convert a text string to lower case.


```
a = AskLine("STRLOWER", "Enter text", "")
b = StrLower(a)
Message(a, b)
```

See Also:
StriCmp, **StrUpper**

StrReplace

Replaces all occurances of a substring with another.

Syntax:
StrReplace (string, old, new)

Parameters:
"string" = string in which to search.
"old" = target substring.
"new" = replacement substring.

Returns:
(string) updated "string" with "old" replaced by "new"

StrReplace scans the "string", searching for occurrences of "old" and replacing each occurrence with "new".


```
; Copy all INI files to clipboard
a = FileItemize("*.ini")
crlf = StrCat(Num2Char(13), Num2Char(10))
b = StrReplace(a, " ", crlf)
ClipPut(b)
```

StrScan

Searches string for occurrence of delimiters.

Syntax:
StrScan (string**,** delimiters**,** start**,** direction**)**

Parameters:
"string" = the string that is to be searched.
"delimiters" = a string of delimiters to search for within "string".
start = the position in the main string to begin search. The first character of a string is position **1**.
direction = the search direction. **@FWDSCAN** searches forward, while **@BACKSCAN** searches backwards.

Returns:
(integer) position of delimiter in string, or **0** if not found.

This function searches for delimiters within a target "string". Starting at the "start" position, it goes forward or backward depending on the value of the "direction" parameter. It stops when it finds any one of the characters in the "delimiters" string within the target "string".


```
thestr = "123,456.789:abc"
start = 1
end = StrScan(thestr, ",.:", start, @FWDSCAN)
If end == 0 Then Goto error
Message("The first parameter", StrSub(thestr, start, end -     start + 1))
Exit
:error
Message("Sorry...", "No delimiters found")
```

See Also:
StrLen, **StrSub**

StrSub

Extracts a substring out of an existing string.

Syntax:
StrSub (string**,** start**,** length**)**

Parameters:

"string" =	the string from which the substring is to be extracted.
start =	character position within "string" where the sub-string starts. (The first character of the string is at position **1**).
length =	length of desired substring. If you specify a length of zero it will return a null string.

Returns:

(string)	substring of parameter string.

This function extracts a substring from within a "target" string. Starting at the "start" position, it copies up to "length" characters into the substring.


```
a = "My dog has fleas"
animal = StrSub(a, 4, 3)
Message("STRSUB", "My animal is a %animal%")
```

See Also:
StrLen, **StrScan**

StrTrim

Removes leading and trailing spaces from a character string.

Syntax:
StrTrim (string**)**

Parameters:
"string" = a string with unwanted spaces at the beginning and/or the end.

Returns:
(string) string devoid of leading and trailing spaces.

Use this function to remove unwanted spaces from the beginning and end of text data.


```
myfile = AskLine("STRTRIM", "Filename ('exit' cancels)", "")
tstexit = StrTrim(StrLower(myfile))
If tstexit == "exit" Then Goto cancel
; processing of myfile continues...
: cancel
Message("Canceled", "...by user request")
```

See Also:
StrFill, **StrFix**, **StrLen**

StrUpper

Converts a string to uppercase.

Syntax:
StrUpper (string**)**

Parameters:
"string" = any text string.

Returns:
(string) uppercase string.

Use this function to convert a text string to upper case.


```
a = AskLine("STRUPPER", "Enter text","")
b = StrUpper(a)
Message(a, b)
```

See Also:
StriCmp, **StrLower**

Terminate

Conditionally ends the procedure.

Syntax:
Terminate (expression, title, message)

Parameters:
"expression" = any logical expression
"title" = the title of a message box to be displayed before termination
"message" = the message in the message box

Returns:
(integer) always @TRUE

This command ends processing for the menu item or procedure if "expression" is not zero. Note that many functions return @TRUE (1) or @FALSE (0), whick you can use to decide whether to cancel a menu item or procedure.

If either "tltle" or "message" contains a string, a messame gox with a title and a message is displayed before exiting.

Examples:

```
;Unconditional Termination w/o message
;Same as "Exit"
Terminate (@TRUE,"","")

;Basically a no-op
Terminate(@FALSE,"","This will never terminate")

;Exits with message if a is less than zero
Terminate(a<0,"Error","Cannot use negative num;bers!")

;Exits w/o message if answer isn't "YES"
Terminate(answer!="YES","","")
```

See Also:
Display, **Pause**, **Message**

TextBox

Displays a file in a listbox on the screen and returns selected line, if any.

Syntax:
TextBox (title**,** filename**)**

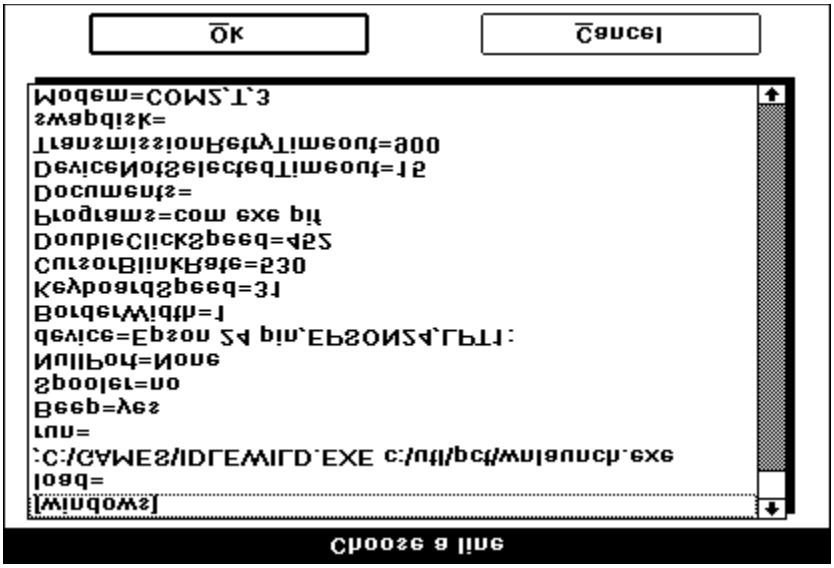
Parameters:
"title" = listbox title.
"filename" = file containing contents of listbox.

Returns:
(string) = highlighted string, if any.

This function loads a file into a Windows listbox and displays the listbox to the user. **TextBox** has two primary uses: First, it can be used to display multi-line messages to the user. In addition, because of its ability to return a selected line, it may be used as a multiple choice question box. The line highlighted by the user (if any) will be returned to the program.
If disk drive and path not are part of the filename, the current directory will be examined first, and then the DOS path will be searched to find the desired file.


```
; Display WIN.INI
a = TextBox("Choose a line", "c:\windows\win.ini")
Display(3, "Chosen line", a)
```

which produces (at least on *my* system):



and then:

See Also:

ItemSelect

TextSelect

Allows the user to choose an item from an unsorted listbox.

Syntax:
TextSelect (title, list, delimiter)

Parameters:

(s) title	the title of dialog box to display.
(s) list	a string containing a list of items to choose from.
(s) delimiter	a string containing the character to act as delimiter between items in the list.

Returns:

(s)	the selected item.

This function displays a dialog box with a listbox inside. This listbox is filled with an unsorted list of items taken from a string you provide to the function.
Each item in the string must be separated (delimited) by a character, which you also pass to the function.
The user selects one of the items by either doubleclicking on it, or single-clicking and pressing OK. The item is returned as a string.
If you create the list with the **FileItemize** or **DirItemize** functions you will be using a space-delimited list. **WinItemize**, however, creates a tab-delimited list of window titles since titles can have embedded blanks.
TextSelect is like **ItemSelect**, except that with **TextSelect** the displayed box is larger and the items in the box are not sorted alphabetically.


```
DirChange(DirWindows(0))
inifiles = FileItemize("*.ini")
ini = TextSelect("Select an INI file to edit", inifiles, " ")
If ini == "" Then Exit
RunZoom("notepad.exe", ini)
```

See Also:
AskLine, **DirItemize**, **FileItemize**, **ItemSelect**, **TextBox**, **WinItemize**

Version

Returns the version number of the currently-running WIL interpreter.

Syntax:
Version ( )

Parameters:
(none)

Returns:
(string) = WinBatch version number.

Use this function to determine the version of WinBatch that is running. It is useful to verify that a batch file generated with the latest version of the language will operate properly on what may be a different machine with a different version of WinBatch installed.


```
a = Version()
```

See Also:
Environment, **DOSVersion**, **WinVersion**

WaitForKey

Waits for a specific key to be pressed.

Syntax:
    WaitForKey (key1, key2, key3, key4, key5)

Parameters:
    (s) key1 - key5   five keystrokes to wait for.

Returns:
    (i)   position of the selected keystroke (1-5).

WaitForKey requires five parameters, each of which represents a keystroke (refer to the **SendKey** function for a list of special keycodes which can be used). The WIL program will be suspended until one of the specified keys are pressed, at which time the **WaitForKey** function will return a number from 1 to 5, indicating the position of the "key" that was selected, and the program will continue. You can specify a null string ("") for one or more of the "key" parameters if you don't need to use all five.
WaitForKey will detect its keystrokes in most, but not all, Windows applications. Any keystroke that is pressed is also passed on to the underlying application.


```
k = WaitForKey("{F11}", "{F12}", "{INSERT}", "", "")
If k == 1 Then Message("WaitForKey", "You pressed the F11 key")
If k == 2 Then Message("WaitForKey", "You pressed the F12 key")
If k == 3 Then Message("WaitForKey", "You pressed the Insert key")
```

See Also:
    IsKeyDown

WallPaper

Changes the Windows wallpaper.

Syntax:
WallPaper (bmp-name**,** tile**)**

Parameters:
"bmp-name" = Name of the BMP wallpaper file.
tile = **@TRUE** if wallpaper should be tiled.
@FALSE if wallpaper should not be tiled.

Returns:
(integer) always **0**

This function immediately changes the Windows wallpaper. It can even be used for wallpaper "slide shows."


```
DirChange("c:\windows")
a = FileItemize("*.bmp")
a = ItemSelect("Select New paper", a, " ")
tile = @FALSE
If FileSize(a) < 40000 Then tile = @TRUE
Wallpaper(a, tile)
```

WinActivate

Activates a previously running window.

Syntax:
WinActivate (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be activated.

Returns:
(integer) **@TRUE** if a window was found to activate;
@FALSE if no windows were found.

Use this function to activate windows for user input.


```
Run("notepad.exe", "")
Run("clock.exe", "")
WinActivate("Notepad")
```

See Also:
WinCloseNot, **WinGetActive**, **WinShow**

WinArrange

Arranges, tiles, and/or stacks application windows.

Syntax:
WinArrange (style**)**

Parameters:
style = one of the following: **@STACK**, **@TILE** (or **@ARRANGE**), **@ROWS**, or **@COLUMNS**.

Returns:
(integer) always **@TRUE.**

Use this function to rearrange the open windows on the screen. (Any iconized programs are unaffected.)
When you specify **@ROWS** and you have more than four open windows, or if you specify **@COLUMNS** and you have more than three open windows, WinBatch will revert to **@TILE**.


```
; Reveal all windows
WinArrange(@TILE)
```

See Also:
WinItemize, **WinHide**, **WinIconize**, **WinPlace**, **WinShow**, **WinZoom**

WinClose

Closes an open window.

Syntax:
WinClose (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be closed.

Returns:
(integer) **@TRUE** if a window was found to close;
@FALSE if no windows were found.

Use this function to close windows.
WinClose will not close the window which contains the currently-executing WIL file. You can, however, use **EndSession** to end the current Windows session.


```
Run("notepad.exe", "")
WinClose("Notepad")
```

See Also:
WinCloseNot, **WinHide**, **WinIconize**, **WinWaitClose**

WinCloseNot

Closes all windows, *except* those provided as parameters.

Syntax:
WinCloseNot (partial-windowname [**,** partial-windowname]...**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. *Any* windows whose titles match the partial names will stay open.

Returns:
(integer) always **@TRUE**.

Use this function to close all windows *except* those specifically listed in the parameter strings.
At least one partial windowname must be given. A null-string parameter would match all windows, or, in other words, close nothing.


```
; The statement below will close all windows except:
; 1) Program Manager (starts with 'Program')
; 2) Clock   (starts with 'Clo' )
WinCloseNot("Program", "Clo")
```

See Also:
WinItemize, **WinClose**, **WinHide**, **WinIconize**, **WinWaitClose**

WinConfig

Returns WIN3 mode flags.

Syntax:
WinConfig ( )

Parameters:
(none)

Returns:
(integer) sum of windows configuration bits.

Returns Windows configuration information as a number, which is the sum of the following individual bits:

1	Protected Mode
2	80286 CPU
4	80386 CPU
8	80486 CPU
16	Standard Mode
32	Enhanced Mode
64	8086 CPU
128	80186 CPU
256	Large PageFrame
512	Small PageFrame
1024	80x87 Installed

You will need to use bitwise operators to extract the individual bits.

Examples:

```
cfg = WinConfig()
If cfg & 32 Then Display(2, "Windows Mode", "Enhanced Mode")
If cfg & 16 Then Display(2, "Windows Mode", "Standard Mode")
If !(cfg & 1) Then Display(2, "Windows Mode", "Real Mode")

cfg = WinConfig()
If cfg & 1024 Then Display(2, "Math co-processor", "Yes")
If !(cfg & 1024) Then Display(2, "Math co-processor", "No")
```

WinExeName

Returns the name of the executable file which created a specified window.

Syntax:
    WinExeName (partial-windowname)

Parameters:
    (s) partial-windowname the initial part of, or an entire, window name.

Returns:
    (s) name of the E file.

Returns the name of the E file which created the first window found whose title matches "partial-windowname".
"Partial-windowname" is the initial part of a window name, and may be a complete window name. It is case-sensitive. You should specify enough characters so that "partial-windowname" matches only one existing window.


```
prog = WinExeName("WinCheck")
WinClose("WinCheck")
Delay(5)
Run(prog, "")
```

See Also:
    Run, **WinExist**, **WinGetActive**, **WinName**

WinExist

Tells if Window exists.

Syntax:
WinExist (partial-windowname)

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name.

Returns:
(integer) **@TRUE** if a matching window is found.
@FALSE if a matching window is not found.

Note: The partial window name you give must match the initial portion of the window name (as appears in the title bar) exactly, including proper case (upper or lower) and punctuation.


```
If WinExist("Clock") == @FALSE Then RunIcon("Clock", "")
```

WinGetActive

Gets the title of the active window.

Syntax:
WinGetActive ( )

Returns:
(string) title of active window.

Use this function to determine which window is currently active.


```
currentwin = WinGetActive()
```

See Also:
WinItemize, **WinActivate**

WinHide

Hides a window.

Syntax:
WinHide (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be hidden.

Returns:
(integer) **@TRUE** if a window was found to hide;
@FALSE if no windows were found.

Use this function to hide windows. The programs are still running when they are hidden. A "partial-windowname" of **""** (null string) hides the current WinBatch window.


```
Run("notepad.exe", "")
WinHide("Notepad")
Delay(3)
WinShow("Notepad")
```

See Also:
WinClose, **WinIconize**, **WinPlace**

WinIconize

Iconizes a window.

Syntax:
WinIconize (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be iconized.

Returns:
(integer) **@TRUE** if a window was found to iconize;
@FALSE if no windows were found.

Use this function to turn a window into an icon at the bottom of the screen.
A "partial-windowname" of **""** (null string) iconizes the current WinBatch window.


```
Run("clock.exe", "")
WinIconize("Clo") ; partial window name used here
```

See Also:
WinClose, **WinHide**, **WinPlace**, **WinShow**, **WinZoom**

WinItemize

Returns a tab-delimited list of all open windows.

Syntax:
WinItemize ( )

Parameters:
(none)

Returns:
(string) list of the titles of all open windows.

This function compiles a list of all the open application windows' titles and separates the titles by tabs. This is especially useful in conjunction with the **ItemSelect** function, which enables the user to choose an item from such a tab-delimited list.
Note this behaves somewhat differently than **FileItemize** and **DirItemize**, which create space-delimited lists. This is because window titles regularly contain embedded spaces.


```
; Find a window
allwins = WinItemize()
htab = Num2Char(9)
mywind = ItemSelect("Windows", allwins, htab)
WinActivate(mywind)
```

See Also:
DirItemize, **FileItemize**, **ItemSelect**

WinMetrics

Returns Windows system information.

Syntax:
    WinMetrics (request#)

Parameters:
    (i) request#   see below.

Returns:
    (i)   see below.

The request# parameter determines what piece of information will be returned.

Req#	Return value
-1	Number of colors supported by video driver
0	Width of screen, in pixels
1	Height of screen, in pixels
2	Width of arrow on vertical scrollbar
3	Height of arrow on horizontal scrollbar
4	Height of window title bar
5	Width of window border lines
6	Height of window border lines
7	Width of dialog box frame
8	Height of dialog box frame
9	Height of thumb box on scrollbar
10	Width of thumb box on scrollbar
11	Width of an icon
12	Height of an icon
13	Width of a cursor
14	Height of a cursor
15	Height of a one line menu bar
16	Width of full screen window
17	Height of a full screen window
18	Height of Kanji window (Japanese)
19	Is a mouse present (0 = No, 1 = Yes)
20	Height of arrow on vertical scrollbar
21	Width of arrow on horizontal scrollbar
22	Is debug version of Windows running (0 = No, 1 = Yes)
23	Are Left and Right mouse buttons swapped (0 = No, 1 = Yes)
24	Reserved
25	Reserved
26	Reserved
27	Reserved
28	Minimum width of a window
29	Minimum height of a window
30	Width of bitmaps in title bar
31	Height of bitmaps in title bar
32	Width of sizeable window frame
33	Height of sizeable window frame
34	Minimum tracking width of a window
35	Minimum tracking height of a window


```
mouse = "NO"
If WinMetrics(19) == 1 Then mouse = "YES"
Message("Is there a mouse installed?", mouse)
```

See Also:

MouseInfo, **NetGetCaps**, **WinConfig**, **WinParmGet**, **WinResources**

WinName

Returns the name of the current WIL Interpreter window.

Syntax:
WinName ( )

Parameters:
(none)

Returns:
(s) window name.

Returns the name of the current WIL interpreter (eg, Command Post or WinBatch) window.


```
tab = Num2Char(9)
allwins = WinItemize()
win = ItemSelect("Close window", allwins, tab)
If win == WinName() Then Goto nocando
WinClose(win)
Exit
:nocando
Message("Sorry", "I can't close myself")
```

See Also:
WinActivate, **WinExeName**, **WinGetActive**, **WinItemize**, **WinTitle**

WinParmGet

Returns system information.

Syntax:
    WinParmGet (request#)

Parameters:
    (i) request#   see below.

Returns:
    (s)   see below.

The request# parameter determines what piece of information will be returned.

Req#	Meaning	Return value
1	Beeping	0 = Off, 1 = On
2	Mouse sensitivity	"threshold1 threshold2 speed"
3	Border Width	Width in pixels
4	Keyboard Speed	Keyboard Repeat rate
5	LangDriver	name of LANGUAGE.DLL
6	Horiz. Icon Spacing	Spacing in pixels
7*	Screen Save Timeout	Timeout in seconds
8*	Is screen saver enabled	0 = No, 1 = Yes
9	Desktop Grid size	Grid Size
10	Wallpaper BMP file	BMP file name
11	Desktop Pattern	Pattern codes (string of 8 space-delimited nums.)
12*	Keyboard Delay	Delay in milliseconds
13	Vertical Icon Spacing	Spacing in pixels
14	IconTitleWrap	0 = No, 1 = Yes
15*	MenuDropAlign	0 = Right, 1 = Left
16	DoubleClickWidth	Allowable horiz. movement in pixels for DblClick
17	DoubleClickHeight	Allowable vert. movement in pixels for DblClick
18	DoubleClickSpeed	Max time in millisecs between clicks for DblClick
19	MouseButtonSwap	0 = Not swapped, 1 = swapped
20*	Fast Task Switch	0 = Off, 1 = On

Items marked with an asterisk (*) require Windows 3.1 or higher.


```
If WinParmGet(8) == 1 Then Message("", "Screen saver is active")
```

See Also:
    MouseInfo, **NetGetCaps**, **WinConfig**, **WinMetrics**, **WinParmSet**, **WinResources**

WinParmSet

Sets system information.

Syntax:
WinParmSet (request#, new-value, ini-control)

Parameters:
(i) request# see **WinParmGet**
(s) new-value see **WinParmGet**
(i) ini-control see below.

Returns:
(int) previous value of the setting.

See **WinParmSet** for a list of valid request#'s and values.
The "ini-control" parameter determines to what extent the value gets updated:
0 Set system value in memory only for future reference

1 Write new value to appropriate INI file

2 Broadcast message to all applications informing them of new value

3 Both 1 and 2


```
WinParmSet(9, "2", 3)     ; sets desktop grid size to 2
```

See Also:
WallPaper, **WinParmGet**

WinPlace

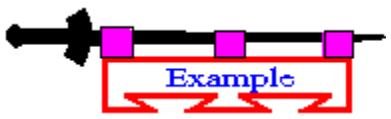
Places a window anywhere on the screen.

Syntax:
WinPlace (x-ulc**,** y-ulc**,** x-brc**,** y-brc**,** partial-windowname**)**

Parameters:
x-ulc = how far from the left of the screen to place the upper-left corner (**0**-**1000**).
y-ulc = how far from the top of the screen to place the upper-left corner (**0**-**1000**).
x-brc = how far from the left of the screen to place the bottom-right corner (**10**-**1000**) or **@NORESIZE**.
y-brc = how far from the top of the screen to place the bottom-right corner (**10**-**1000**) or **@NORESIZE** or **@ABOVEICONS**.
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be moved to the new position.

Returns:
(integer) **@TRUE** if a window was found to move;
@FALSE if no windows were found.

Use this function to move windows on the screen. (You cannot, however, move icons or windows that have been maximized to full screen.)
The "x-ulc", "y-ulc", "x-brc", and "y-brc" parameters are based on a logical screen that is 1000 points wide by 1000 points high.
You can move the window without changing the width and/or height by specifying **@NORESIZE** for the "x-brc" and/or "y-brc" parameters, respectively.
You can fix the bottom of the window to sit just above the line of icons along the bottom of the screen by specifying a "y-brc" of **@ABOVEICONS**.
Some sample parameters:
Upper left quarter of the screen: **0, 0, 500, 500**
Upper right quarter: **500, 0, 1000, 500**
Center quarter: **250, 250, 750, 750**
Lower left eighth: **0, 750, 500, 1000**
A handy utility program is included with WinBatch, called **WININFO.EXE**. This program lets you take an open window that is sized and positioned the way you like it, and automatically create the proper **WinPlace** statement for you. It puts the text into the Clipboard, from which you can paste it into your batch code:
You'll need a mouse to use **WinInfo**. While **WinInfo** is the active window, place the mouse over the window you wish to create the **WinPlace** statement for, and press the spacebar. The new statement will be placed into the Clipboard. Then press the Esc key to close **WinInfo**.




```
WinPlace(0, 0, 200, 200, "Clock")
```

See Also:
WinArrange, **WinHide**, **WinIconize**, **WinShow**, **WinZoom**

WinPlaceGet

Returns window coordinates.

Syntax:
WinPlaceGet (win-type, partial-windowname)

Parameters:

(i) win-type	@ICON, @NORMAL, or @ZOOMED
(s) partial-windowname	the initial part of, or an entire, window name.

Returns:

(s)	window coordinates (see below).

This function returns the coordinates for an iconized, normal, or zoomed window. "Partial-windowname" is the initial part of a window name, and may be a complete window name. It is case-sensitive. You should specify enough characters so that "partial-windowname" matches only one existing window. If it matches more than one window, the most recently accessed window which it matches will be used.
The returned value is a string of either 2 or 4 numbers, as follows:

Iconic windows	"x y" (upper left corner of the icon)
Normal windows	"upper-x upper-y lower-x lower-y"
Zoomed windows	"x y" (upper left corner of the window)

All coordinates are relative to a virtual 1000x1000 screen.

Examples:

```
Run("clock.exe", "")
pos = WinPlaceGet(@NORMAL, "Clock")
Delay(2)
WinPlaceSet(@NORMAL, "Clock", "250 250 750 750")
Delay(2)
WinPlaceSet(@NORMAL, "Clock", pos)
```

See Also:
WinGetActive, WinItemize, WinPlaceSet, WinPosition, WinState

WinPlaceSet

Sets window coordinates.

Syntax:
WinPlaceSet (win-type, partial-windowname, position-string)

Parameters:
(i) win-type @ICON, @NORMAL, or @ZOOMED
(s) partial-windowname the initial part of, or an entire, window name.
(s) position-string window coordinates (see below).

Returns:
(s) previous coordinates.

This function sets the coordinates for an iconized, normal, or zoomed window. The window does not have to be in the desired state to set the coordinates; for example, you can set the iconized position for a normal window so that when the window is subsequently iconized, it will go to the coordinates that you've set.
"Partial-windowname" is the initial part of a window name, and may be a complete window name. It is case-sensitive. You should specify enough characters so that "partial-windowname" matches only one existing window. If it matches more than one window, the most recently accessed window which it matches will be used.
"Position-string" is a string of either 2 or 4 numbers, as follows:

Iconic windows "x y" (upper left corner of the icon)

Normal windows "upper-x upper-y lower-x lower-y"

Zoomed windows "x y" (upper left corner of the window)

All coordinates are relative to a virtual 1000x1000 screen.

Examples:

```
WinPlaceSet(@ICON, "Clock", "10 950")

WinPlaceSet(@NORMAL, "Clock", "250 250 750 750")

WinPlaceSet(@ZOOMED, "Clock", "-5 -5")
```

See Also:
IconArrange, **WinActivate**, **WinArrange**, **WinPlace**, **WinPlaceGet**, **WinState**

WinPosition

Returns Window position.

Syntax:
WinPosition (partial-windowname)

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name.

Returns:
(string) window coordinates, delimited by commas.

Returns the current Window position information for the selected Window. It returns 4 comma-separated numbers (see **WinPlace** for details).


```
Run("clock.exe", "")                    ; start Clock
WinPlace(0,0,300,300, "Clock")          ; place Clock
pos = WinPosition("Clock")              ; save position
delay(2)
WinPlace(200,200,300,300, "Clock")      ; move Clock
delay(2)
WinPlace(%pos%, "Clock")                ; restore Clock
```

See Also:
WinPlace

WinResources

Returns information on available memory and resources.

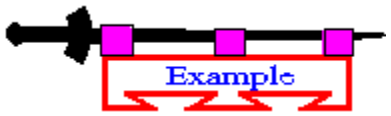
Syntax:
WinResources (request#)

Parameters:
(i) request# see below

Returns:
(i) see below.

The value of request# determined the piece of information returned.

Req#	Return value
0	Total available memory, in bytes
1	Theoretical maximum available memory, in bytes
2	Percent of free system resources (lower of GDI and USER)
3	Percent of free GDI resources
4	Percent of free USER resources




```
mem = WinResources(0)
Message("Available memory", "%mem% bytes")
```

See Also:
WinConfig, **WinMetrics**, **WinParmGet**

WinShow

Shows a window in its "normal" state.

Syntax:
WinShow (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be shown.

Returns:
(integer) **@TRUE** if a window was found to show;
@FALSE if no windows were found.

Use this function to restore a window to its "normal" size and position.
A "partial-windowname" of **""** (null string) restores the current WIL interpreter window.


```
RunZoom("notepad.exe", "")
; other processing...
WinShow("Notepad")
```

See Also:
WinArrange, **WinHide**, **WinIconize**, **WinZoom**

WinState

Returns the current state of a window.

Syntax:
WinState (partial-windowname)

Parameters:
(s) partial-windowname the initial part of, or an entire, window name.

Returns:
(i) window state (see below).

"Partial-windowname" is the initial part of a window name, and may be a complete window name. It is case-sensitive. You should specify enough characters so that "partial-windowname" matches only one existing window. If it matches more than one window, the most recently accessed window which it matches will be used.
Possible return values are as follows.

Value	Symbolic name	Meaning
-1		Specified window exists, but is hidden
0		Specified window does not exist
1	@ICON	Specified window is iconic (minimized)
2	@NORMAL	Specified window is a normal window
3	@ZOOMED	Specified window is zoomed (maximized)


```
If WinState("Notepad") == @ICON Then WinShow("Notepad")
```

See Also:
Run, **WinExist**, **WinGetActive**, **WinHide**, **WinIconize**, **WinItemize**, **WinPlace**, **WinPlaceGet**, **WinPlaceSet**, **WinPosition**, **WinShow**, **WinZoom**

WinTitle

Changes the title of a window.

Syntax:
WinTitle (partial-windowname**,** new-name**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be shown.
"new-name" = the new name of the window.

Returns:
(integer) **@TRUE** if a window was found to rename;
@FALSE if no windows were found.

Use this function to change a window's title.
A "partial-windowname" of **""** (null string) refers to the current WIL interpreter window.
Warning: Some applications may rely upon their window's title staying the same!
Therefore, the **WinTitle** function should be used with caution and adequate testing.


```
; Capitalize title of window
htab = Num2Char(9)
allwinds = WinItemize()
mywin = ItemSelect("Uppercase Windows", allwinds, htab)
WinTitle(mywin, StrUpper(mywin))
Drop(htab, allwinds, mywin)
```

See Also:
WinItemize

WinVersion

Provides the version number of the current Windows system.

Syntax:
WinVersion (level**)**

Parameters:
level = either **@MAJOR** or **@MINOR.**

Returns:
(integer) either major or minor part of the Windows version number.

Use this command to determine which version of Windows is currently running.
@MAJOR returns the integer part of the Windows version number; i.e. **1**.0, **2**.11, **3**.0, etc.
@MINOR returns the decimal part of the Windows version number; i.e. 1.**0**, 2.**11**, 3.**0**, etc.


```
minorver = WinVersion(@MINOR)
majorver = WinVersion(@MAJOR)
Message("Windows Version", StrCat(majorver, ".", minorver))
```

See Also:
Version, **DOSVersion**

WinWaitClose

Suspends the batch file execution until a specified window has been closed.

Syntax:
WinWaitClose (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. **WinWaitClose** suspends execution until all matching windows have been closed.

Returns:
(integer) **@TRUE** if at least one window was found to wait for;
@FALSE if no windows were found.

Use this function to suspend the batch file's execution until the user has finished using a given window and has manually closed it.


```
Run("clock.exe", "")
Display(4, "Note", "Close Clock to continue")
WinWaitClose("Clock")
Message("Continuing...", "Clock closed")
```

See Also:
Delay, **Yield**

WinZoom

Maximizes a window to full-screen.

Syntax:
WinZoom (partial-windowname**)**

Parameters:
"partial-windowname" =
either an initial portion of, or an entire window name. The most-recently used window whose title matches the name will be shown.

Returns:
(integer) **@TRUE** if a window was found to zoom;
@FALSE if no windows were found.

Use this function to "zoom" windows to full screen size.
A partial-windowname of **""** (null string) zooms the current WIL interpreter window.


```
Run("notepad.exe", "")
WinZoom("Notepad")
Delay(3)
WinShow("Notepad")
```

See Also:
WinHide, **WinIconize**, **WinPlace**, **WinShow**

Yield

Provides time for other windows to do processing.

Syntax:

Yield

Use this command to give other running windows time to process. This command will allow each open window to process 20 or more messages.


```
; run Excel and give it some time to start up
sheet = AskLine ("Excel", "File to run:", "")
Run("excel.exe", sheet)
Yield
Yield
Yield
```

See Also:

Delay

Click Here

Click Here

Click Here

Click Here

WIL Language Guide

WIL statements are constructed from **constants**, **variables**, **operators**, **functions**, **commands**, and **comments**.
Each line in a WIL file can be up to 255 characters long.

Wil Functions

Check Box Dialog
Command Line Parameters
CommentsConstants
Dialog Boxes
Directory List Dialog (Full)
Error Handling
File List Dialog (Plain)
Function Parameters
Identifiers
Keywords
Labels
Operators (Arithmetic, Logical)
Precedence and Evaluation Order Predefined Constants
Programming Dialog Boxes (in depth)
Radio Button Dialog
Statements
Substitution
Variables

Command-Line Parameters

WinBatch is run with the following command line:

WINBATCH filename.**WBT** p1 p2 ... p9

"filename.wbt" is any valid WIL file.

"p1 p2 ... p9" are optional parameters (maximum of nine) to be passed to the WBT file on startup, delimited by spaces.
Parameters passed to a WBT file are automatically parsed into variables named **param1**, **param2**, etc. An additional variable, **param0**, is the total number of command-line parameters.

Comments

A comment is a sequence of characters that are ignored when processing a command. A semicolon (not otherwise part of a string constant) indicates the beginning of a comment.

All characters to the right of the semicolon are considered comments, and are ignored.

Blank lines are also ignored.

Examples of comments:

```
; This is a comment
ABC = 5        ; This is also a comment
```

Constants

The programming language supports both integer and string constants.

Integer Constants

Integer constants are built from the digits **0** through **9**. They can range in magnitude from negative to positive 2_{31} -1 (approximately two billion). Constants larger than these permissible magnitudes will produce unpredictable results.

Examples of integer constants:

```
1
-45
377849
-1999999999
```

String Constants

String constants are comprised of displayable characters bounded by quote marks. You can use double quotes (**"**), single quotes (**'**), or back quotes (**`**) to enclose a string constant, as long as the same type of quote is used to both start and end it. If you need to embed the delimiting quote mark inside the string constant, use the delimiting quote mark twice.

Examples of string constants:

```
"a"
`Betty Boop`
"This constant has an embedded "" mark"
'This constant also has an embedded " mark'
```

Predefined Constants

The programming language has a number of built-in integer constants that can be used for various purposes. These start with the **@**-sign, and are **case-insensitive** (although we prefer to use ALL CAPS).

Some predefined constants:

@FALSE
@NO
@STACK
@TILE
@TRUE
@YES

Dialog Boxes–Introduction and Palette

Palette of Dialog Box Samples

 Check Boxes

 Radio Buttons

 Simple File List Box

 Complete File and Directory List Box

Checkbox Sample: An example of checkboxes in a dialog box.

Sample Checkbox

To try this dialog, follow these instructions:

1. Copy the sample dialog box code into a file called *3chkbox.wbd*. Create this file with a text editor such as the Windows Notepad, or a programming text editor like WinEdit. Use the Edit Copy and Edit Paste menu items in Windows Help and in your editor to transfer the code from this Help file to 3chkbox.wbd.

2. To simplify matters, save (*File Save AS...*) your templates in a distinctive directory. The directory you use to store your WIL language files would be a good choice.

3. Create a macro or Command Post menu item with these lines:

```
DirChange("Full name of the directory in item 2.")
DialogBox("Check Box Demo","3chkbox.wbd")
```

4. Run the macro or menu item and you will see the sample dialog box. From the code, you can see that although it is very simple, it performs a valuable function.

5. To use this dialog in your menus, include it by substituting your own labels for the *This is Box 1* and the other labels in the code. By using this dialog, the variables Box1, Box2, and Box3 will all be set to either 0 or 1 depending on whether or not the user checks them. You can then use these variables anywhere in your WIL language program.

```
;Check Box Sample Code
[Box1+1This is Box 1]
[Box2+1This is Box 2]
[Box3+1This is Box 3]
```

Labels

Labels in the WIL language **start** with a colon.

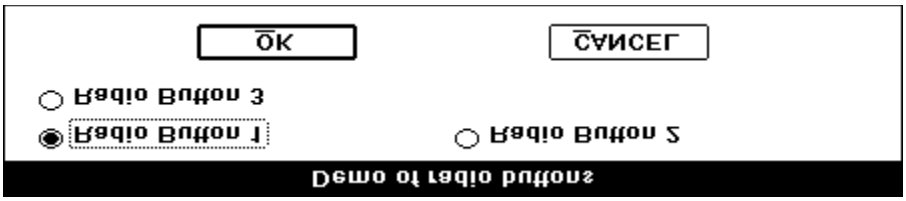
Example:

```
file="*.*"
DialogBox ("Demo of File and Directory Box","dirfiles.wbd")
If file=="" then Goto Label2
:Label1
Message ("Label Message"," You selected %file%")
Goto Goodbye
:Label2
Message("Label Message"," You entered no file at all")

:Goodbye
```

Radio Button Sample: An example of a simple file directory selection box.

Sample Radio Button Dialog



To try this dialog, follow these instructions:

1. Copy the sample dialog box code into a file called *radiobtn.wbd*. Create this file with a text editor such as the Windows Notepad, or a programming text editor like WinEdit. Use the Edit Copy and Edit Paste menu items in Windows Help and in your editor to transfer the code from this Help file to *radiobtn.wbd*.

2. To simplify matters, save (*File Save AS...*) your templates in a distinctive directory. The directory you use to store your WIL language files would be a good choice.

3. Create a macro or Command Post menu item with these lines:

```
DirChange("Full name of the directory in item 2.")
DialogBox("File List Dialog","radiobtn.wbd")
```

4. Run the macro or menu item and you will see the sample dialog box. From the code, you can see that although it is very simple, it performs a valuable function.

5. To use this dialog in your menus, remember that the user's action will store the chosen file name in the global variable *RadioBtn*. Check the code to see this. You can then use the variable *RadioBtn* anywhere in your WIL language program. If you want to erase this variable after using it, use the *Drop(RadioBtn)* statement.

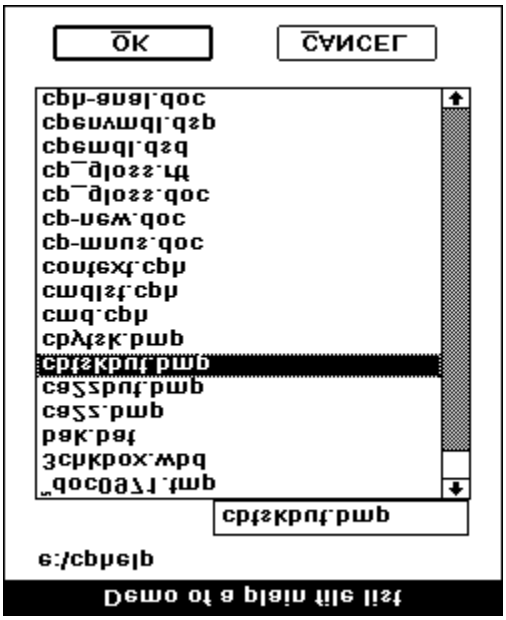
```
;Sample Code for a dialog with radio buttons.
;The variable RadioBtn will acquire the
;values 1, 2, or 3 depending on the button
;selected by the user.

;The labels Radio Button 1, and so on, can
;easily be changed to your values.

[RadioBtn^1Radio Button 1][RadioBtn^2Radio Button 2]
[RadioBtn^3Radio Button 3]
```

Filelist Sample: An example of a simple file directory selection box.

Sample Filelist Dialog



To try this dialog, follow these instructions:

1. Copy the sample dialog box code into a file called *filelist.wbd*. Create this file with a text editor such as the Windows Notepad, or a programming text editor like WinEdit. Use the Edit Copy and Edit Paste menu items in Windows Help and in your editor to transfer the code from this Help file to *filelist.wbd*.

2. To simplify matters, save (*File Save AS...*) your templates in a distinctive directory. The directory you use to store your WIL language files would be a good choice.

3. Create a macro or Command Post menu item with these lines:

```
DirChange("Full name of the directory in item 2.")
DialogBox("File List Dialog","filelist.wbd")
```

4. Run the macro or menu item and you will see the sample dialog box. From the code, you can see that although it is very simple, it performs a valuable function.

5. To use this dialog in your menus, remember that the user's action will store the chosen file name in the global variable *files*. Check the code to see this. You can then use the variable *file* anywhere in your WIL language program. If you want to erase this variable after using it, use the *Drop(file)* statement. If you want to preset the dialog for any file extension, preface

your DialogBox statement with a *Files*="*.ext" where the *ext* can be a wildcard extension, a specific file extension, or a specific file name.

```
;File List Dialog Sample.
;The file name returns in the variable file.
; You can freely substitute your own.

[file$                    ]
          [file#          ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
[file\                    ]
```

Dirfiles Sample: An example of a full directory selection box.

Sample Dirfiles

To try this dialog, follow these instructions:

1. Copy the sample dialog box code into a file called *dirfiles.wbd*. Create this file with a text editor such as the Windows Notepad, or a programming text editor like WinEdit. Use the Edit Copy and Edit Paste menu items in Windows Help and in your editor to transfer the code from this Help file to *dirfiles.wbd*.

2. To simplify matters, save (*File Save AS...*) your templates in a distinctive directory. The directory you use to store your WIL language files would be a good choice.

3. Create a macro or Command Post menu item with these lines:

```
DirChange("Full name of the directory in item 2.")
DialogBox("Full Directory Box Demo","dirfiles.wbd")
```

4. Run the macro or menu item and you will see the sample dialog box. From the code, you can see that although it is very simple, it performs a valuable function.

5. To use this dialog in your menus, remember that the user's action will store the chosen file name in the global variable *files*. Check the code to see this. You can then use the variable *file* anywhere in your WIL language program. If you want to erase this variable after using it, use the *Drop(file)* statement. If you want to preset the dialog for any file extension, preface your DialogBox statement with a *Files*="**.ext*" where the *ext* can be a wildcard extension, a specific file extension, or a specific file name.

;Full Directory Dialog Sample.
;The file name returns in the variable *file*.
; You can freely substitute your own.

Directory: [file$]

File: [file#] (Wildcards OK)

File List: [file\ ]
[file\ ]
[file\ ]
[file\ ]
[file\ ]
[file\ ]

Error Handling

There are three types of errors that can occur while processing a batch file: **Minor**, **Moderate**, and **Fatal**. What happens when an error occurs depends on the current error mode, which is set with the **ErrorMode** function. There are three possible modes you can specify:

@CANCEL
User is notified when any error occurs, and then the batch file is canceled. This is the default.

@NOTIFY
User is notified when any error occurs, and has option to continue unless the error is fatal.

@OFF
User is only notified if the error is moderate or fatal. User has option to continue unless the error is fatal.

The function **LastError** returns the code of the most-recent error encountered during the current batch file.
Minor errors are numbered from **1000** to **1999**.
Moderate errors are numbered from **2000** to **2999**.
Fatal errors are numbered from **3000** to **3999**.
Error handling is reset to **@CANCEL** at the start of each batch file.

Minor Errors

Minor errors are ignored if the current error mode has been set to **@OFF**. If the error mode is **@NOTIFY** the user has the option of continuing with the batch file or canceling it.

1006 File Copy/Move: No matching files found
1017 File Delete: No matching files found
1018 File Delete: Delete Failed
1024 File Rename: No matching files found
1025 File Rename: Rename failed
1028 LogDisk: Requested drive not online
1029 DirMake: Dir not created
1030 DirRemove: Dir not removed
1031 DirChange: Dir not found/changed
1039 WinClose: Window not found
1040 WinHide: Window not found
1041 WinIconize: Window not found
1042 WinZoom: Window not found
1043 WinShow: Window not found
1044 WinPlace: Window not found
1045 WinActivate: Window not found

1119 WinPosition: Window not found
1121 WinTitle: Window not found
1100 StrIndex/StrScan 3rd parameter out of bounds
1900 WinExec 0: Out of Memory
1902 WinExec 2: File Not Found
1903 WinExec 3: Path Not Found
1905 WinExec 5: Attempt to dynlink to a task
1906 WinExec 6: Lib requires data segs for each task
1910 WinExec 10: Incorrect Windows Version
1911 WinExec 11: Invalid E file
1912 WinExec 12: Cannot run OS/2 application
1913 WinExec 12: Cannot run DOS4.0 application
1914 WinExec 14: Unknown E type
1915 WinExec 15: Attempt to run old E in protect mode
1916 WinExec 16: Attempted 2Nd E with multiple writeable datasegs
1917 WinExec 17: Nonshareable DLLs already in use
1918 WinExec 18: App marked for protected mode only
1932 WinExec: Undefined Error

Moderate Errors
If the error mode is **@NOTIFY** or **@OFF**, the user has the option of continuing with the batch file or canceling it.
2001 SendKey: Illegal Parameters
2002 File Copy/Move: 'From' file illegal
2003 File Copy/Move: 'To' file illegal
2004 File Copy/Move: Cannot copy/move wildcards into fixed root
2005 File Copy/Move: Cannot copy/move wildcards into fixed extension
2007 File Move: Unable to rename source file
2015 File Move: Unable to remove source file
2016 File Delete: File name illegal
2019 File Rename: 'From' file illegal
2020 File Rename: 'To' file illegal
2021 File Rename: Attempt to rename across drive boundary. - Use MOVE instead.
2022 File Rename: Cannot rename wildcards into a fixed filename root
2023 File Rename: Cannot rename wildcards into a fixed filename extension
2038 WinCloseNot Function Syntax error
2045 WinActivate: Window not found
2058 StrCat function syntax error
2060 Average function syntax error
2093 Dialog Box: Bad Filespec, using *.*
2112 FileSize: File Not Found
2118 FileCopy/Move: Destination file same as source

Fatal Errors

Fatal errors cause the current batch file to be canceled with an error message, regardless of the error mode in effect. (We show the error codes here for consistency, but in practice you will never be able to call **LastError** after a fatal error.)

3008 File Copy/Move: 'From' file open error
3009 SendKey: Could not open DEBUG text file
3010 SendKey: Could not install hook - Already Active??
3011 File Copy/Move: 'From' file length error
3012 File Copy/Move: No room left on disk. Out of space??
3013 File Copy/Move: 'To' file open error
3014 File Copy/Move: I/O Error
3015 File Move: Unable to remove source file
3026 LogDisk: Illegal disk drive
3027 LogDisk: DOS reports no disks!! ???
3032 GoTo unable to lock memory for batch file
3033 GoTo label not found
3034 Clipboard owned by another app. Cannot open.
3035 Clipboard does not contain text for ClipAppend.
3036 Clipboard cannot hold that much text (>64000 bytes)
3037 Unable to allocate memory for clipboard. Close some applications
3046 Internal Error 3046. Function not defined
3047 Variable name over 30 chars. Too Long
3048 Substitution %Variable% not followed by % (Use %% for %)
3049 No variables exist??!!
3050 Undefined variable
3051 Undefined variable or function
3052 Uninitialized variable or undefined function
3053 Character string too long (>256 chars??)
3054 Unrecognizable item found on line
3055 Variable name is over 30 chars. Too Long
3056 Variable could not be converted to string
3057 Variable could not be converted to integer
3059 Illegal Bounds for StrSub function
3061 Illegal Syntax
3062 Attempt to divide by zero
3063 Internal Error 3063. Binary op not found
3064 Internal Error 3064. Unary op not found
3065 Unbalanced Parenthesis
3066 Wrong Number of Arguments in Function
3067 Function Syntax. Opening parenthesis missing.
3068 Function Syntax. Illegal delimiter found.
3069 Illegal assignment statement. (Use == for equality testing)
3070 Internal error 3070. Too many arguments defined.
3071 Missing or incomplete statement

3072 THEN not found in IF statement
3073 Goto Label not specified
3074 Expression continues past expected end.
3075 Call: Parse of file/parameter line failed
3076 FileOpen: READ or WRITE not properly specified
3077 FileOpen: Open failed
3078 FileOpen: Too many (>5) files open
3079 FileClose: Invalid file handle
3080 FileClose: File not currently open
3081 FileRead: Invalid file handle
3082 FileRead: File not currently open
3084 FileWrite: Invalid file handle
3085 FileWrite: File not currently open
3087 FileRead: File not open for reading
3088 FileRead: Attempt to read past end of file
3089 FileWrite: File not open for writing
3090 Dialog Box: File open error
3091 Dialog Box: Box too large
3092 Dialog Box: Non-text control used w/filebox.
3094 Dialog Box: Window Registration Failed
3095 Compare could not be resolved into a integer or string compare
3096 Memory allocation failure. Out of memory for string storage
3097 Memory allocation failure. Out of memory for variable storage
3098 Internal error, NULL pointer passed to xstrxxx subroutines
3099 CallExt function disabled. Not currently available.
3101 Substituted line too long. (> 256 characters)
3102 Drop: Can only drop variables
3103 IsDefined: Attempting to test non-variables item
3104 Dialog Box: Window Creation Failed
3105 Batch Compiler: CALL and CALLEXT not supported in compiled E versions
3107 Run: Filetype is not COM, EXE, PIF or BAT
3108 FileItemize: Unable to lock file info segment
3109 FileItemize: Unable to unlock file info segment
3110 FileItemize: Unable to lock file index segment
3111 FileItemize: Unable to unlock file index segment
3113 FileSize: Filelength I/O Error
3114 FileSize: Buffer Overrun Error
3115 FileDelete: Buffer Overrun Error
3116 FileRename: Buffer Overrun Error
3117 FileCopyMove: Buffer Overrun Error

Function Parameters

Most of the functions and commands in the language require parameters. These come in three types:

- **Integer**
- **String**
- **Variable name**

The interpreter performs automatic conversions between strings and integers, so in general you can use them interchangeably.
Integer parameters may be any of the following:

- **An integer (i.e. 23)**
- **A string representing an integer (i.e. "23")**
- **A variable containing an integer**
- **A variable containing a string representing an integer**

String parameters may be any of the following:

- **A string**
- **An integer**
- **A variable containing a string**
- **A variable containing an integer**

Identifiers

Identifiers are the names supplied for variables, functions, and commands in your program.

An identifier is a sequence of one or more letters or digits that begins with a letter. Identifiers may have up to 30 characters.

All identifiers are *case insensitive*. Upper- and lowercase characters may be mixed at will inside variable names, commands or functions.

For example, these statements all mean the same thing:

```
AskLine(MyTitle, Prompt, Default)
ASKLINE(MYTITLE, PROMPT, DEFAULT)
aSkLiNe(MyTiTlE, pRoMpT, dEfAuLt)
```

Keywords

"Keywords" are the predefined identifiers that have special meaning to the programming language. These cannot be used as variable names.
WIL keywords consist of the **functions**, **commands**, and **predefined constants**.
Some examples of reserved keywords:

Beep
DirChange
@Yes
FileCopy

Operators

The programming language operators take one operand ("unary operators") or two operands ("binary operators").

Unary operators (integers only):

-	Arithmetic Negation (Two's complement)
+	Identity (Unary plus)
~	Bitwise Not. Changes each **0** bit to **1**, and vice-versa.
!	Logical Not. Produces **0** (**@FALSE**) if the operand is nonzero, else **1** (**@TRUE**) if the operand is zero.

Binary arithmetic operators (integers only):

*	Multiplication
/	Division
mod	Modulo
+	Addition
-	Subtraction
<<	Left Shift
>>	Right Shift
&	Bitwise And
|	Bitwise Or
^	Bitwise Exclusive Or (XOR)
&&	Logical And
| |	Logical Or

Binary relational operators (integers and strings):

>	Greater-than
>=	Greater-than or equal
<	Less-than
<=	Less-than or equal
==	Equality
!= or **<>**	Inequality

Assignment operator (integers and strings):

=	Assigns evaluated result of an expression to a variable

Predefined Constants List

WIL provides you with a number of predefined integer constants to help make your batch files more mnemonic:

Logical Conditions

@FALSE
@NO
@OFF
@TRUE
@YES
@ON

Window Arranging

@NORESIZE
@ABOVEICONS
@STACK
@ARRANGE
@TITLE
@ROWS
@COLUMNS

String Handling
@FWDSCAN
@BACKSCAN

System Control
@MAJOR
@MINOR

Error Handling
@CANCEL
@NOTIFY
@OFF

Keyboard Status
@SHIFT

@CTRL

Debug Control
@PARSEONLY

Programming Dialog Boxes

Syntax:
DialogBox (title, WBD file)

Parameters:
("string") title the title of the dialog box.
("string") WBD file the name of the WBD template file.

Returns:
(integer) always 0.

Template File
Each element in the template file is enclosed in square brackets, and consists of a variable name, followed by one of the following symbols:

Symbol	Meaning	Example
+	check box	[backup+1Save backup]
#	edit box	[newfile#]
****	file listbox	[editfile\ ]
^	radio button	[prog^1Note] [prog^2Write]
$	variable	[var$]

The number following the check box and radio button symbols is the value which will get assigned to the variable if its corresponding box is checked, or button is selected. Following the number is the descriptive text which will appear next to the box or button.
When used in conjunction with a file selection list box variable with the same name, two of these symbols have special meanings:

#	file mask edit box	[editfile#]
$	directory variable	[editfile$]

See the EXAMPLE
Anything not appearing within square brackets is displayed as text.
See the separate section on **Dialog Boxes** later in this manual for more detailed information on using this function.

Example:

```
DialogBox("Edit a file", "edit.wbd")
If backup == 0 Then Goto nobackup
filebackupname = StrCat(FileRoot(editfile), ".", "bak")
FileCopy(editfile, filebackupname, @TRUE)
:nobackup
If prog == 1 Then Run("notepad.exe", editfile)
If prog == 2 Then Run("c:\win\apps\winedit.exe", editfile)
```

Here is the template file, EDIT.WBD:

```
[editfile$          ]
      File mask [editfile#      ]
[editfile\                        ]
[editfile\                        ]
[editfile\                        ]
[editfile\                        ]
[editfile\                        ]
[backup+1Save backup of file]
[prog^1Notepad]       [prog^2WinEdit]
```

See Also:

AskLine, AskPassword, AskYesNo, ItemSelect

Statements

Assignment Statements

Assignment statements are used to set variables to specific or computed values. Variables may be set to integers or strings.

Examples:

```
a = 5
value = Average(a, 10, 15)
location = "Northern Hemisphere"
world = StrCat(location, " ", "Southern Hemisphere")
```

Control Statements

Control statements are generally used to execute system management functions and consist of a call to a command without assigning any return values.

Examples:

```
Exit
Yield
```

Substitution

The batch language has a powerful substitution feature which inserts the contents of a string variable into a statement before the line is parsed. To substitute the contents of a variable in the statement, simply put a percent-sign (**%**) on both sides of the variable name.

Examples:

```
mycmd = "DirChange('c:\')"          ;set mycmd to a command
%mycmd%                             ;execute the command
```

Or consider this one:

```
IniWrite("PC", "User", "Richard")
...
name = IniRead("PC", "User", "somebody")
message("", "Thank you, %name%")
```

To put a single percent-sign (**%**) on a source line, specify a double percent sign(**%%**). This is required even inside quoted strings.

Note: The length of a line, after any substitution occurs, may not exceed 255 characters.

Variables

A variable may contain an integer, a string, or a string representing an integer. Automatic conversions between integers and strings are performed as a matter of course during execution.

If a function requires a string parameter and an integer parameter is supplied, the variable will be automatically modified to include the representative string.

If a function requires an integer parameter and a string parameter is supplied, an attempt will be made to convert the string to an integer. If it does not convert successfully, an error will result.

Precedence and evaluation order

The precedence of the operators affect the evaluation of operands in expressions. Operands associated with higher-precedence operators are evaluated before the lower-precedence operators.
The table below shows the precedence of the operators. Where operators have the same precedence, they are evaluated from left to right.

Operator	**Description**
( )	Parenthetical grouping
~ ! - +	Unary operators
* / **mod**	Multiplication & Division
+ -	Addition & Subtraction
<< >>	Shift operators
< <= == >= > != <>	Relational operators
& ^ |	Bit manipulation operators
&& | |	Logical operators