Many new features have been added to WinBatch 95. For a list of other changes see the Fixes and improvements text file in your WinBatch directory.

Windows 95 / Windows NT Important Issues

WinBatch FileMenu
WinBatch PopMenu

New Functions
New IntControls
File Delimiters
Partial WindowNames

In modifying WinBatch 95 to run on the 32-bit Windows platforms, the use of some functions has changed. Consider the following.

DirItemize / DiskScan / FileItemize

In the 32-bit version, the following functions, which used to return space-delimited lists of items:

DirItemize
DiskScan
FileItemize

now return tab-delimited lists. This can be changed using the new IntControl(29).

AppExist / AppWaitClose / WinExeName

In the 32-bit version, the following functions work only with 32-bit apps:

AppExist
AppWaitClose
WinExeName

In the 32-bit version, AppExist and AppWaitClose are not able to detect the existence of 16-bit DOS or Windows applications, and WinExeName will return the string "(16-bit application)".

In the 32-bit version under Windows NT, these functions accept (and return) module names instead of full program names. The module name is usually the same as the root name of the program, without the extension. For example, WinExeName("Program Manager") will return "progman".

One exception: WinExeName("") will return a full path to the program making the current call to the WIL Interpreter.

With AppExist and AppWaitClose, any file extension or path information which is part of the 'program-name' parameter is ignored; so, for example, AppExist("c:\temp\progman.exe") will return TRUE if Program Manager is running, regardless of what directory PROGMAN.EXE is actually located in.

Menu Utility for the Windows Explorer

FILEMENU is a menu utility DLL for the Windows Explorer. It allows you to add custom menu items to the context menus (that appear when you right-click on a file in the Windows Explorer). Two types of menus are supported:

1. A global menu, which is added to the context menu of every file.
2. A file-specific "local" menu, whose entries depend on the type of file that is clicked on.

FILEMENU is a menu-based WIL (Windows Interface Language) application.

Note: Please refer to the Windows Interface Language Reference Manual, Menu Files section, for information on menu file structure.

POPMENU is a WinBatch 95 desktop interface to Windows batch files written in WIL, the Windows Interface Language. POPMENU batch files are used to automate PC operations and application specific procedures. (FILEMENU, the other WinBatch 95 menu utility, is used in manipulating files in the Windows Explorer.)

Pop Menu appears as an icon on the Windows 95 Task Bar. This bar extends along one edge of the Windows 95 desktop and includes the "START" Button. A click on the POPMENU icon brings up a menu of WIL batch files. Samples are included, but you can completely modify these to meet your needs.

PopMenu is a menu-based WIL (Windows Interface Language) application.

NOTE: Please refer to the Windows Interface Language Reference Manual, Menu Files section, for information on menu file structure.

System Requirements

FILEMENU requires a version of Windows supporting the Windows Explorer, such as Windows 95..

Installation

FILEMENU is installed during the normal setup of WinBatch 95.

Operation

FILEMENU can add menu items to the following types of context menus:

1. The context menus that appear when you right-click on a file (but not a folder) in the Windows Explorer.

2. The context menus that appear when you right-click on a file (but not a folder) in a browse window (for example, if you select "Run" from the "Start" menu, and then press "Browse".

3. The Explorer "File" pull-down menu, when a file (but not a folder) is highlighted in the Explorer window.

4. Files (or Shortcuts to files) on the Windows desktop.

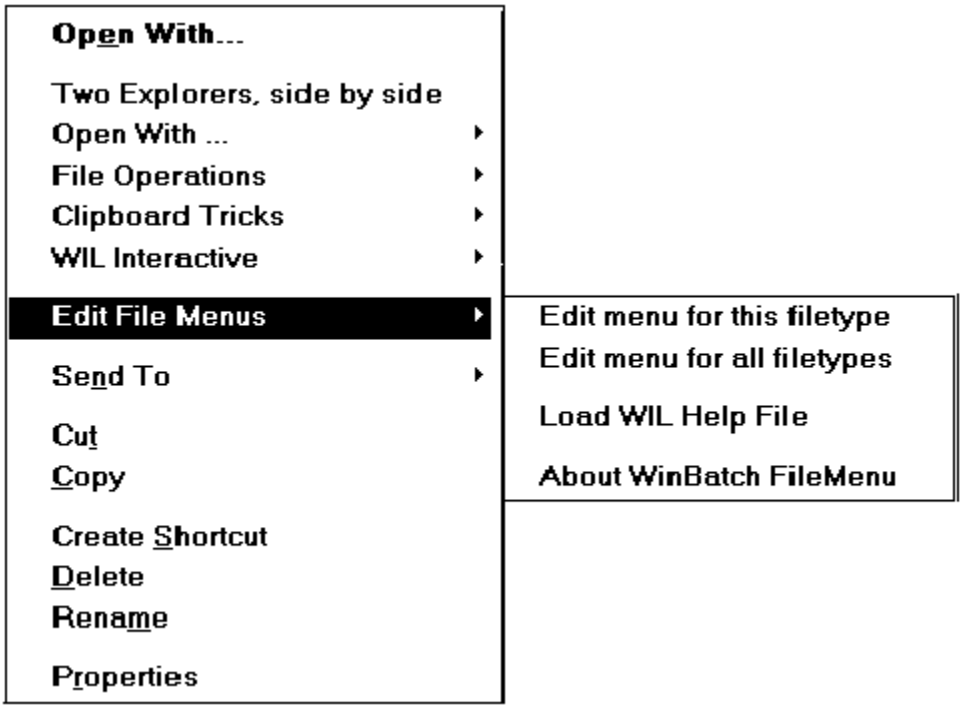
FILEMENU can add two menu files onto a file's context menu: the "all filetypes" menu, which is added to the context menu of every file, and a file-specific menu, whose entries depend on the type of file selected.

A menu file can be created or edited by selecting **Edit File Menus.** This option opens the Windows Notepad and loads either a file-specific menu or the "all filetypes" menu. Modifications to menu files are made once the file is saved.

Menu files are discussed in the Windows Interface Language manual under the topic Menu Files.

The "all filetypes" menu adds additional menu choices to the context menu which appears when you right click on any file in an Explorer window, or on the desktop.

The following is a sample context menu. The menu options displayed are samples of the file operations which can be performed.



With FILEMENU, the sample "all filetypes" menu starts with **Two Explorers, side by side** and continues down to **Edit File Menus**. When an option is highlighted, an additional explanation will be displayed on the status bar of the Windows Explorer.

The "all filetypes" menu can be modified with the context menu option **Edit File Menus / Edit menu for all filetypes**. This option opens Notepad with the "all filetypes" menu loaded. Changes are effective when the file is saved.

Note: The contents of the "all filetypes" menu file may vary from release to release as we continue to improve the sample menus.

A file-specific menu allows you to create custom menus for any file type. These menus are shown only when the file type is clicked on with the right mouse button.

File-specific menu files can be created or modified using the context menu item **Edit File Menus / Edit menu for this filetype.** When this option is selected, FILEMENU looks for an existing file type menu in the file: Filemenu.ini. If the type menu is found, it is opened in Notepad. If no file is found, FILEMENU creates a new menu file for that file type. Filemenu.ini is automatically updated and the new menu file is opened in Windows Notepad. The new file-specific menu will have a sample menu to help you get started.

The menu file names used by FILEMENU are defined in the file Filemenu.ini, which is located in your WINBATCH\SYSTEM directory. A sample Filemenu.ini is provided. The menu files can be located anywhere on your path or in your FILEMENU directory. Or, you can specify a full path in Filemenu.ini.

By default, the "all filetypes" menu is named "FileMenu for all filetypes" (the short filename will be something like; FILEME~1.MNW). This default can be changed by editing the "*CommonMenu=" line in the [FileMenu] section to point to a different menu file. If you do not wish to use the "all filetypes" menu file, specify a blank value to the right of the equals sign; i.e., "*CommonMenu= ".

To use a file-specific menu, add a line of the form "ext=menuname" to the [Menus] section, where "ext" is the extension of the file type, and "menuname" is the name of the menu file you wish to associate with that file type. For example, if you wish to add the contents of the menu file TXT.MNW to the context menus of .TXT files, add the line "txt=txt.mnw". To specify a menu file to associate with files that do not have an extension, use an extension of "."; for example ".=menufile".

Note: Extensions can be longer than three characters.

There is a limit on the number of menu items that can be added to a context menu. This limit seems to be 163 menu items, but it may vary from system to system and in different releases of Windows. FILEMENU shares these resources with other menu extender programs you may have on a first-come, first-served basis. If the maximum available menu items is 163, and you have other menu extender programs installed that use a 10 menu items, your FILEMENU menus (global + local) could contain no more than 153 menu items. Of course, FILEMENU only loads one local menu at a time. If your global menu contained 100 items, each of your local menus could contain up to 53 items.

If you exceed the limit of available menu items, a menu extender program will not be able to add additional items. If FILEMENU is unable to load one of its menus completely, it will display an error message.

Please refer to the Windows Interface Language Reference Manual, Menu Files section, for information on menu file structure.

Functions

In addition to the standard WIL functions, FILEMENU supports the following functions (which are documented in the WIL Reference Manual):

CurrentFile
CurrentPath
CurrFilePath

The following functions are NOT supported:

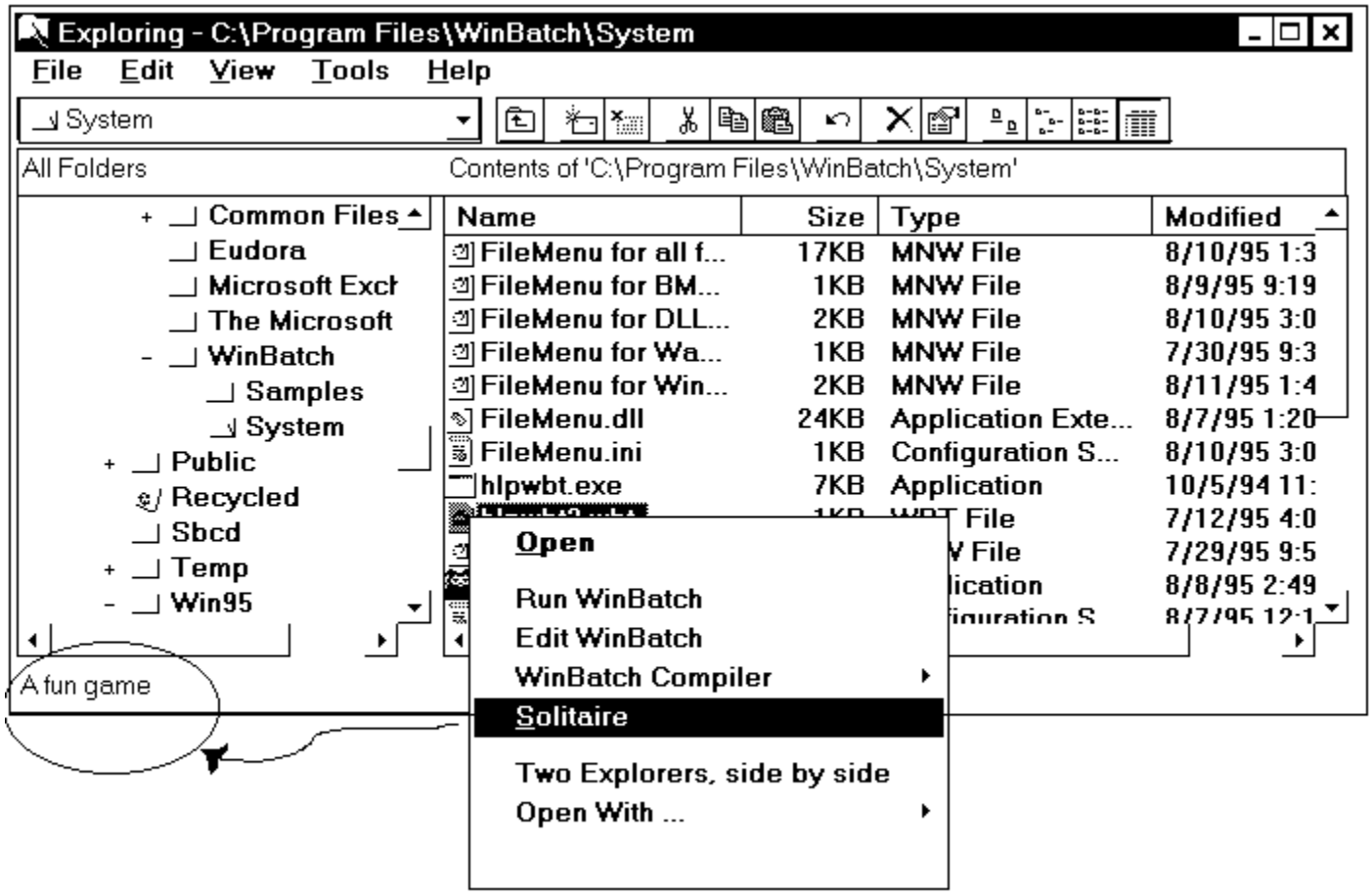
IsMenuChecked
IsMenuEnabled
MenuChange
Reload

Status Bar Comments

You can specify a comment for display in the Windows Explorer status bar. This works only for top level menu items. The comment must be on the same line as the top level item. For example, the menu item below is a main menu for running the program Solitaire.

```
&Solitaire              ; A fun game
   Run("sol.exe", "")
```

The following dialog shows how comment appears on the Explorer's status bar.



Misc....

FILEMENU processes the "Autoexec" (initialization) section of a menu file every time an item from that file is executed.

Hotkeys are not supported.

Shell extensions can be loaded and unloaded rather frequently by the operating system, so there is little benefit in using the "Drop" function.

System Requirements

POPMENU requires Windows 95 or Windows NT.

Installation

To install POPMENU:

1. Copy POPMENU.EXE to any directory on your hard drive. We will refer to the directory where POPMENU.EXE is located as your "PopMenu directory".

2. Copy the sample Popmenu.ini either to your Windows 95 directory, or to your POPMENU directory.

3. Copy WBD??32I.DLL either to your POPMENU directory, or to a directory on your path (this includes your Windows 95 and Windows 95 System directories). We recommend placing it on your path, since it can then be accessed by other WIL programs.

4. Set up your menu files (see "Menu Files", below), and place them in a directory on your path, or in your POPMENU directory. A sample menu file is included with the program. You can use it or adapt it to your requirements.

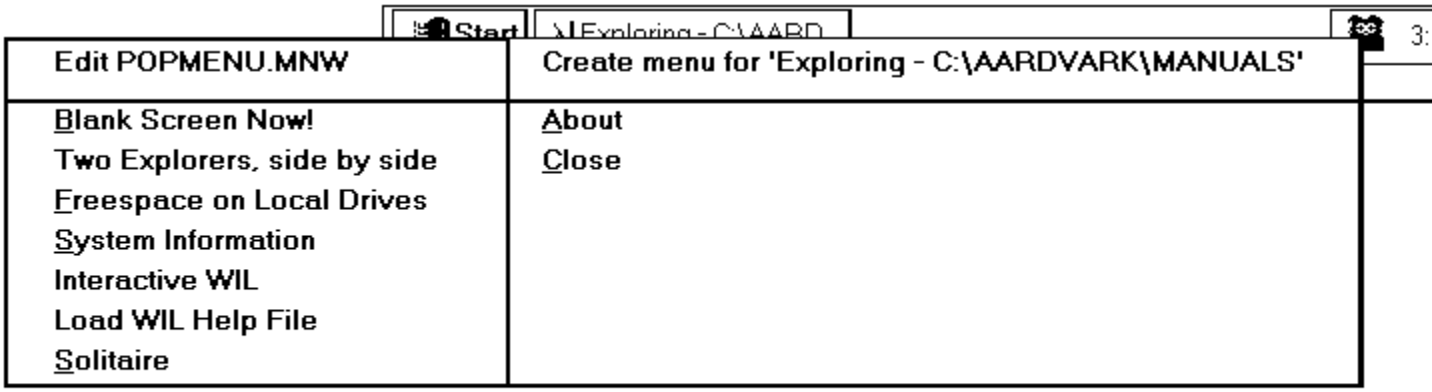
Then, run POPMENU.EXE. You should see the POPMENU icon appear in the task bar.

Operation

Start POPMENU by running POPMENU.EXE.

Activate POPMENU by clicking on its icon (you may have to click twice).

Close POPMENU by selecting "Close" from its menu.


Start
Exploring - C:\AARD
Edit POPMENU.MNW
Create menu for 'Exploring - C:\AARDVARK\MANUALS'
Blank Screen Now!
Two Explorers, side by side
Freespace on Local Drives
System Information
Interactive WIL
Load WIL Help File
Solitaire
About
Close

POPMENU allows you to specify two menu files: (1) a global menu file, and (2) a window-specific local menu file.

The default global menu file is named POPMENU.MNW. You can change this by editing the INI file (see "INI Settings", below).

The name of the window-specific local menu file is based on the class name (a specific Windows program identifier) of the most-recently-active parent window, with an extension of .MNW added. So, for example, the local menu file for Explorer (whose class name is "Progman") would be "Progman.MNW". POPMENU will add a menu item at the top of each menu, allowing you to create or edit the appropriate menu file for that window, so in general you do not need to know the actual class names.

Each menu file can contain a maximum of 1000 menu items.

POPMENU searches for menu files using the following sequence:

1. If the menu name contains a path, use it as-is and don't search
2. Menu directory ("MenuDir=" INI setting), if set
3. Home directory ("HOMEPATH" environment variable), if set
4. Windows 95 directory
5. PopMenu directory
6. Other directories on your path

By default, new menu files created by POPMENU will be placed in your PopMenu directory (the directory where POPMENU.EXE is located), unless you are running POPMENU from a network drive. On a network, menu files will be created in your home directory (the directory pointed to by the "HOMEPATH" environment variable) if it is set, or your Windows 95 directory otherwise. You can change this by editing the INI file (see "INI Settings", below).

Please refer to the Windows Interface Language Reference Manual, Menu Files section, for information on menu file structure and how to create the appropriate menu files.

The following settings can be added to the [PopMenu] section of Popmenu.ini:

MenuDir=d:\path

where "d:\path" is the directory where you want POPMENU to place menu files that it creates. This will also be the first place POPMENU looks for menus. The default is the POPMENU directory, unless you are running POPMENU from a network drive (see "Menu Files", above, for further information).

Editor=editor

where "editor" is the editor you wish to use to edit your menu files. The default is "Notepad.exe".

GlobalMenu=menufile.mnw

where "menufile.mnw" is the name of the global menu file you wish to use. The default is "POPMENU.MNW".

SkipGlobalMenu=1

Causes POPMENU not to load the global menu file. By default, the global menu file will be loaded.

SkipLocalMenu=1

Causes POPMENU not to load the window-specific local menu file. By default, the local menu file will be loaded.

SkipGlobalEdit=1

Causes POPMENU not to add a "Create/Edit menu" item at the top of the global menu. By default, the menu item will be added.

SkipLocalEdit=1

Causes POPMENU not to add a "Create/Edit menu" item at the top of the local menu. By default, the menu item will be added.

Functions

In addition to the standard WIL functions, POPMENU supports the following functions (which are documented in the WinBatch User's Guide):

BoxOpen
BoxShut
BoxText
BoxTitle

The following optional WIL menu functions are NOT supported by POPMENU:

CurrentFile
CurrentPath
CurrFilePath
IsMenuChecked
IsMenuEnabled
MenuChange
Reload

Misc...

You can only run one copy of POPMENU at a time.

You can only run one POPMENU menu item at a time (if you click on the POPMENU icon while a menu item is currently executing, it will beep).

Sometimes you may have to click on the POPMENU icon twice for the menu to pop up.

POPMENU reloads the menu files every time you bring up its menu. You can dynamically change the current global menu file while POPMENU is running by updating the "GlobalMenu=" setting in the [PopMenu] section of POPMENU.INI (you can even do this from within a menu script using the IniWritePvt function).

POPMENU processes the "Autoexec" (initialization) section of a menu file every time an item from that file is executed.

Hotkeys are not supported.

Horizontal menu separators ('_') are not added for top-level menu items.

Status bar comments are not supported.

The Windows Interface Language has new functions. Some are general WIL functions others will run only on either Windows 95 or 32 bit systems.

The following symbols will be used to indicate a function or a section of the manual which applies only to a specific implementation of the WIL Interpreter.

{*M} menu-based implementations of WIL Interpreter, plus POPMENU and FILEMENU.
{*95} Windows 95 operating system.
{*32} 32 bit Windows operating systems.

BinaryIndexNC**(handle, offset, string, direction)**
Searches a buffer for a string, ignoring case.
CurrFilePath {*M}**( )**
( )Returns the full path plus filename of the currently-selected file.
DirAttrGet**([d:]path)**
Gets directory attributes.
DirAttrSet**(dir-list, settings)**
Sets directory attributes.
FileNameLong {*32}**(filename)**
Returns the long version of a filename.
FileNameShort {*32}**(filename)**
Returns the short (ie, 8.3) version of a filename.
InstallFile **(filename, targname, default-targdir, delete-old, flags)**
Installs a file.
MouseClick**(click-type, modifiers)**
Clicks mouse button(s).
MouseClickBtn**(parent-windowname, child-windowname, button-text)**
Clicks on the specified button control.
MouseMove**(X, Y, parent-windowname, child-windowname)**
Moves the mouse to the specified X-Y coordinates.
RegApp {*32}**(program-name, path)**
Creates registry entries for a program under "App Paths".
RegDelValue {*32}**(handle, subkey-string)**
Deletes a named value data item for the specified subkey from the registry.
RegQueryBin {*32}**{*32}(handle, subkey-string)**
Returns binary value at subkey position.
RegQueryDword {*32}**{*32}(handle, subkey-string)**
Returns DWORD value at subkey position.
RegQueryItem {*32}**{*32}(handle, subkey-string)**
Returns a list of named data items for a subkey.
RegSetBin {*32}**{*32}(handle, subkey-string, value)**
Sets a binary value in the Registration Database.
RegSetDword {*32}**{*32}(handle, subkey-string, value)**
Sets a DWORD value in the Registration Database.
ShellExecute**(program-name, params, directory, display mode, operation)**
Runs a program via the Windows ShellExecute command
ShortcutEdit {*95}**(link-name, target, params, start-dir, show-mode)**
Modifies the specified shortcut file.
ShortcutExtra {*95}**(link-name, description, hotkey, icon-file, icon-index)**
Sets additional information for the specified shortcut file.

ShortcutInfo {*95}**(link-name)**
Returns information on the specified shortcut file.
ShortcutMake {*95}**(link-name, target, params, start-dir, show-mode)**
Creates a Windows 95 shortcut for the specified filename or directory.
TimeJulToYmd**(julian-date)**
Returns the Julian day given a datetime.
TimeSubtract**(datetime, datetime difference)**
Subtracts one YmdHms variable from another.
WinIdGet**(partial-winname)**
Returns a unique "Window ID" (pseudo-handle) for the specified window name.
WinItemNameId**( )**
Returns a list of all open windows and their Window ID's.
WinMetrics (new request #'s)
Returns Windows system information.
WinSysInfo( ) {*32}**()**
Returns system configuration information.

IntControl(29, p1**, 0, 0, 0)**

Changes the default file delimiter.
p1 New delimiter

We have added the ability to change the file delimiter to a character of your own choosing, using the new IntControl 29. If you are using the 32-bit version of WIL, and want to make the file delimiter a space for compatibility with existing scripts, you can place the following line at the beginning of each of your scripts:

```
IntControl(29, " ", 0, 0, 0)
```

Conversely, if you want to standardize on a TAB delimiter, you can use:

```
IntControl(29, @TAB, 0, 0, 0)
```

The first parameter for IntControl is the new file delimiter you want to use, and must be a single character. The return value of the function is the previous file delimiter character. If you specify an empty string ("") as the first parameter, the function will return the current file delimiter character but the file delimiter will not be changed.

IntControl(30, p1**,** p2**, 0, 0) {*NT}**
Performs a delayed file move.

p1 source file
p2 destination

The file is not actually moved until the operating system is restarted. This can be useful for replacing system files. "Sourcefile" must be a single file name, with no wildcards. "Destination" may be a file name (which may contain wildcards) or a directory name. The destination file MUST be on the same drive as the source file. If the destination file exists, it will be replaced without warning. "Destination" can also be a NULL string (""), in which case the source file will be deleted when the operating system is restarted.

Under Windows 95, and in the 16-bit version, this function performs a regular (non-delayed) FileMove.
This function returns "1" on success, "2" if it performed a regular FileMove instead, and "0" on failure.

IntControl(31, 0, 0, 0, 0) {*95}
Returns "Window ID's" for all Explorer windows.

This function returns a tab-delimited list of Window ID's for all open Windows 95 Explorer windows.

IntControl(32, address, "data type", 0, 0)
Returns the contents of the memory location specified by "address".
"Data type" specifies the type of data to be retrieved:

"BYTE" - returns a byte

"WORD" - returns a word
"LONG" - returns a long integer

IntControl(33, p1, 0, 0, 0)

Controls whether a listbox control in a dialog box allows multiple items to be selected.

P1	Meaning
0	Single selection
1	Multiple selection (default)

IntControl(34, p1, 0, 0, 0)

Returns the error message string which corresponds to the specified WIL error.

p1 error number.

IntControl(35, p1, 0, 0, 0)

Slows down SendKey.

p1 amount of time to delay between each keypress, in milliseconds (1000 milliseconds = 1 second);
0 = no delay (default).

Returns previous delay setting.

IntControl(36, p1, p2, 0, 0) (32-bit version only)

Waits until an application is waiting for user input.
p1 = window name associated with application
p2 = time-out, in milliseconds (-1 = no time-out)

This function waits until the process which created the specified window has finished its initialization and is waiting for user input with no input pending, or until the specified time-out interval has elapsed. It can only be used with 32-bit GUI applications. It returns @TRUE if it has successfully waited, or FALSE if a time-out has occurred (or if it was unable to initiate a wait).

In order to support long file names in Windows NT and Windows 95, which can contain embedded spaces, we have changed the default file delimiter, used to delimit lists of files and directories, to a TAB in the 32-bit version of WIL. In the 16-bit version of WIL, the default delimiter has not changed, and remains a space.

Note that this is the "default" file delimiter. We have added the ability to change the file delimiter to a character of your own choosing, using the new IntControl 29.

The most important functions affected by this change are:

DirItemize
DiskScan
FileItemize

which now return lists delimited by the current file delimiter character.

The following functions, which take file or directory lists as input parameters, now expect the lists to be delimited by the current file delimiter character. However, they now also accept lists delimited with a TAB or a vertical bar ("|", which may be easier to code in a WIL script):

DirItemize	FileItemize
DirRemove	FileMove
DiskFree	FileRename
FileAppend	FileSize
FileAttrSet	FileTimeSet
FileCopy	FileTimeTouch
FileDelete	

Note that DiskFree will continue to accept space-delimited lists as input.

Those WIL functions which take a partial windowname as a parameter can be directed to accept only an exact match by ending the window name with a tilde (~).

A tilde (~) used as the first character of the window name will match any window containing the specified string anywhere in its title. For example, WinShow("~Notepad") will match a window title of "(Untitled) - Notepad" and a window title of "My Notepad Application", as well as a window title of "Notepad - (Untitled)".

A tilde (~) used as the last character of the window name indicates that the name must match the window title through to the end of the title. For example, WinShow("Note~") would only match a window whose title was "Note"; it would not match "Notepad". Furthermore, WinShow("~Notepad~") will match a window title of "Notepad" and a window title of "(Untitled) -Notepad", but will not match a window title of "Notepad - (Untitled)".

When using partial windownames as parameters, you can specify the full name if you wish, but in most circumstances, it isn't necessary. Remember that the case (upper or lower) of the title is significant. If the case is not correct, a match will not be made.

Searches a buffer for a string, ignoring case.

Syntax:

BinaryIndexNc(handle, offset, string, direction)

Parameters:

(i) handle	handle of buffer.
(i) offset	offset in the buffer to begin search.
(s) string	the string to search for within the buffer.
(i) direction	the search direction. @FWDSCAN searches forwards, while @BACKSCAN searches backwards.

Returns:

(i)	offset of string within the buffer, or 0 if not found.

This function is like BinaryIndex, but performs a case-insensitive search for a **string** within a buffer. Starting at the **offset** position, it goes forwards or backwards depending on the value of the **direction** parameter. It stops when it finds the **string** within the buffer and returns the **string's** beginning offset.

Note 1: The **string** parameter may be composed of any characters except the null (00) character. This function cannot process a null character. If you need to search for a null character, use the **BinaryPeek** function in a **for** loop.

Note 2: The return value of this function is possibly ambiguous. A zero return value may mean the **string** was not found, or it may mean the **string** was found starting at **offset** 0. If there is a possibility that the **string** to be searched for could begin at the beginning of the buffer, you must determine some other way of resolving the ambiguity, such as using **BinaryPeekStr**.

Example:

```
; Find line number of line in config.sys where HIMEM occurs
fs1 = FileSize( "C:\CONFIG.SYS" )
binbuf1 = binaryalloc( fs1 )
a1 = BinaryRead( binbuf1, "C:\CONFIG.SYS" )
a = BinaryIndex( binbuf1, 0, "HIMEM", @FWDSCAN )  ;  find HIMEM
if a == 0
    Message("Hmmm", "HIMEM not found in CONFIG.SYS file")
else
    c = BinaryStrCnt( binbuf1, 0, a, @CRLF) + 1
    Message("Hmmm",  "HIMEM found on line %c%")
endif
```

See Also:

Binary Operations, BinaryCopy, BinaryIndex, BinaryEodGet, BinaryEodSet, BinaryStrCnt

Returns the full path plus filename of the currently-selected file.

Syntax:

CurrFilePath()

Parameters:

(none)

Returns:

(s) path and filename of currently-selected file.

Example:

```
;Get the filename before changing directories.
myfile =CurrFilePath()
DirChange("c:\word")
Run("winword.exe", myfile)
```

See Also:

CurrentFile, CurrentPath

Gets directory attributes.

Syntax:

DirAttrGet([d:]path)

Parameters:

(s) [d:]path directory pathname whose attributes you want to determine.

Returns:

(s) the attributes of the specified directory pathname.

Returns attributes for the specified directory, 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 directory which has the Archive and Hidden attributes set.

Example:

```
dir = "c:\temp"
attr = DirAttrGet(dir)
Message("Attributes of Directory, %dir", attr)
```

See Also:

DirAttrSet, FileAttrGet, FileAttrSet, FileTimeGet

Sets directory attributes.

Syntax:

DirAttrSet(dir-list, settings)

Parameters:

(s) dir-list	a list of one or more sub-directory names.
(s) settings	new attribute settings for the directories.

Returns:

(s)	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):

Symbol	Meaning
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

Example:

```
DirAttrSet("c:\temp", "rASh")
```

See Also:

DirAttrGet, FileAttrGet, FileAttrSet, FileTimeTouch

Returns the long version of a filename.

Syntax:

FileNameLong(filename)

Parameters:

(s) filename fully qualified file name, path optional.

Returns:

(s) the long version of a filename.

FileNameLong searches the path for the filename specified, returning the long filename if found.

Example:

```
DirChange("C:\win95")
a=FileNameLong("carved~1.bmp")
message("Long Filename", a)
```

See Also:

FileFullName, FileNameShort

Returns the short (ie, 8.3) version of a filename.

Syntax:

FileNameShort(filename)

Parameters:

(s) filename fully qualified file name, path optional.

Returns:

(s) the short version of a filename.

FileNameShort searches the path for the filename specified, returning the short filename if found.

Example:

```
DirChange("C:\win95")
a=FileNameShort("carved stone.bmp")
message("Short Filename", a)
```

See Also:

FileFullName, FileNameLong

Installs a file.

Syntax:

InstallFile(filename, targname, default-targdir, delete-old, flags)

Parameters:

(s) filename	source file to be installed. (path optional)
(s) targname	the name of the target file to be created. (without path)
(s) default-targdir	directory where the file is to be installed.
(i) delete-old	@TRUE - to delete existing same name files. @FALSE - to ignore existing same name files.
(i) flags	1 - shared file 2 - force install

Returns:

(s)	"result|tempname", or "result|"

InstallFile works like **FileCopy** but is far more capable. When installing image files (EXE's, DLL's, etc.), this function uses the version information embedded in the files to determine whether a file being installed is newer than an existing file with the same name. When installing any other type of file, which does not contain appropriate version information, this function uses the time stamps of the respective files instead.
The return value is in the form:

"result|tempname", or
"result|"

where "result" is the value returned by the "VerInstallFile" Windows API function; and "tempname" is the name of the temporary file that was created if the file could not be installed, or blank otherwise.
"Default-targdir" is the directory where you want the file to be installed. The file will be installed to this directory, unless it is a shared file or a file with the same name already exists elsewhere.
If "Delete-old" is **@TRUE** (or non-zero), and a file with the same name as the file being installed already exists, it will be deleted, even if it is located in a directory (on the path) other than the target directory. If "delete-old" is **@FALSE**, such a file will not be deleted.
"Flags" specifies other optional flags that affect the operation of this function, combined with the OR ('|') operator. They are:

1 - shared file (file should be installed to a shared directory)
2 - force install (install file even if older than existing file)

Note: The image version can only be interpreted by a corresponding platform version, ie. 32-bit images by a 32-bit platform.

Example:

```
InstallFile("a:\ctl3d.dl_", "ctl3d.dll", DirWindows(1), @TRUE, 1)
```

See Also:

FileCopy, RegApp

Clicks mouse button(s).

Syntax:

MouseClick(click-type, modifiers)

Parameters:

(i) click-type a mouse button press.
(i) modifiers click variations for mouse button presses.

Returns:

(i) **@TRUE** on success; **@FALSE** on error.

This function performs a mouse click at the current mouse cursor position.
"Modifiers" can be set to 0 if none are needed.
Click-types:

@LCLICK	left click
@RCLICK	right click
@MCLICK	middle click
@LDBLCLICK	left double-click
@RDBLCLICK	right double-click
@MDBLCLICK	middle double-click

Modifiers (multiple modifiers can be linked together with a logical OR, "|"):

@SHIFT	hold down shift key
@CTRL	hold down control key
@LBUTTON	hold down left mouse button
@RBUTTON	hold down right mouse button
@MBUTTON	hold down middle mouse button

Example:

```
winpos = WinPlaceGet(@NORMAL, "~Notepad")
; get coordinates for upper right corner of window
x = ItemExtract(3, winpos, " ") - 10
y = ItemExtract(2, winpos, " ") + 10
WinActivate("~Notepad")
MouseMove(x - 10, y + 10, "", "")
MouseClick(@LCLICK, 0)
```

See Also:

MouseClick, MouseClickBtn, MouseMove, SendKey

Clicks on the specified button control.

Syntax:

MouseClickBtn(parent-windowname, child-windowname, button-text)

Parameters:

(s) parent-windowname — the initial part of, or an entire parent window name.

(s) child-windowname — the initial part or, or an entire child window name.

(s) button-text — text specifying a button control.

Returns:

(i) **@TRUE** on success; **@FALSE** on error.

This function clicks on the pushbutton, radio button, or checkbox whose text is specified by "button-text".
If the button is located within a top-level window, specify the window name in "parent-windowname" and specify a blank string ("") for "child-windowname".
If the button is located within a child window, specify the top-level window name in "parent-windowname" and the child window name in "child-windowname".

Example:

```
SendMenusTo("Exploring", "Tools | Map Network Drive")
Delay(3)
MouseClickBtn("Map Network Drive", "", "Reconnect at logon")
```

See Also:

MouseClick, MouseClickBtn, MouseMove, SendKey

Moves the mouse to the specified X-Y coordinates.

Syntax:

MouseMove(X, Y, parent-windowname, child-windowname)

Parameters:

(i) X	integer specifying the coordinate X.	
(i) Y	integer specifying the coordinate Y.	
(s) parent-windowname	the initial part of, or an entire parent window name.	
(s) child-windowname	the initial part or, or an entire child window name.	

Returns:

(i) **@TRUE** on success; **@FALSE** on error.

If "parent-windowname" specifies a top-level window and "child-windowname" is a blank string, the specified X-Y coordinates are relative to "parent-windowname".
If "parent-windowname" specifies a top-level window and "child-windowname" specifies a child window of "parent-windowname", the specified X-Y coordinates are relative to "child-windowname".
If "parent-windowname" and "child-windowname" are both blank strings, the specified X-Y coordinates are relative to the Windows desktop.
All coordinates are based on a virtual 1000 x 1000 screen.

Example:

```
MouseMove(335, 110, "Control Panel", "")
```

See Also:

MouseClick, MouseClickBtn, MouseMove, SendKey

Creates registry entries for a program under "App Paths".

Syntax:

RegApp(program-name, path)

Parameters:

(s) program-name the name of an executable program (EXE), optionally containing a path.

(s) path optional desired "PATH" setting for the specified program.

Returns:

(i) **@TRUE** Entry was created;
@FALSE Operation failed.

This function creates (or updates) a sub-key in the registration database for the specified program, of the form PROGNAME.EXE, under the key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\

If "program-name" does not contain a path, the function will search for it on the path.
The function creates a "(Default)" value for the key, containing the full path to the specified program.
If the "path" parameter is not a blank string (""), the function also creates a "Path" value for the key. This should contain one or more directories (separated by semi-colons) which you want to be prepended to the existing "PATH" environment variable when the program is run.

Example:

```
RegApp("excel.exe", "c:\excel;c:\word")
```

See Also:

InstallFile, RegOpenKey, RegCreateKey, RegQueryValue, RegQueryKey, and the section on Registration Database Operations

Deletes a named value data item for the specified subkey from the registry.

Syntax:

RegDelValue(handle, subkey-string)

Parameters:

(i) handle handle to a registration database key.
(s) subkey-string a path from the key provided to the desired key.

Returns:

(i) **@TRUE** Data item was deleted;
@FALSE Data was not found.

"Subkey-string" must be enclosed in square brackets (see **RegSetValue**). "Subkey-string" of "[]" deletes the "default" value.

Example:

```
RegDelValue(@REGROOT, ".diz")
```

See Also:

RegOpenKey, RegCreateKey, RegCloseKey, RegDeleteKey, RegQueryValue, RegQueryKey, and the section on Registration Database Operations

Returns binary value at subkey position.

Syntax:

RegQueryBin(handle, subkey-string)

Parameters:

(i) handle handle to a registration database key.
(s) subkey-string a path from the key provided to the desired key.

Returns:

(s) contents of data item at key position desired.

The value is returned as a space-delimited string of hex bytes; e.g.:
"AB 45 3E 01".

Example:

```
value = RegQueryBin(@REGCURRENT, "Control Panel\Appearance\[CustomColors]")
Message("CustomColors", value)
```

See Also:

RegQueryDword, RegQueryValue

Returns DWORD value at subkey position.

Syntax:

RegQueryDword(handle, subkey-string)

Parameters:

(i) handle handle to a registration database key.
(s) subkey-string a path from the key provided to the desired key.

Returns:

(i) contents of data item at key position desired.

Example:

```
value = RegQueryDword(@REGCURRENT, "Control Panel\Desktop\[ScreenSaveUsePassword]")
Message("ScreenSaveUsePassword", value)
```

See Also:

RegQueryBin, RegQueryValue

Returns a list of named data items for a subkey.

Syntax:

RegQueryItem(handle, subkey-string)

Parameters:

(i) handle handle to a registration database key.
(s) subkey-string a path from the key provided to the desired key.

Returns:

(s) tab-delimited list of named data items for the specified subkey-string.

Example:

```
items = RegQueryItem(@REGCURRENT,     "Software\Microsoft\Windows\CurrentVersion\Extensions")
item = TextSelect("Select an item", items, @TAB)
value = RegQueryValue(@REGCURRENT,    "Software\Microsoft\Windows\CurrentVersion\
Extensions[%item%]")
Message(item, value)
```

See Also:

RegQueryValue, and the section on Registration Database Operations

Sets a binary value in the Registration Database.

Syntax:

RegSetBin(handle, subkey-string, value)

Parameters:

(i) handle handle to a registration database key.
(s) subkey-string a path from the key provided to the desired key.
(s) value data to be stored into the database at desired key.

Returns:

(i) always 1.

The value is specified as a space-delimited string of hex bytes; e.g.: "AB 45 3E 01".

Example:

```
RegSetBin(@REGCURRENT, "A Test Key\[My Binary Value]", "00 01 22 AB FF 00")
```

See Also:

RegSetDword , RegSetValue

Sets a DWORD value in the Registration Database.

Syntax:

RegSetDword(handle, subkey-string, value)

Parameters:

(i) handle handle to a registration database key.
(s) subkey-string a path from the key provided to the desired key.
(s) value data to be stored into the database at desired key.

Returns:

(i) always 1.

Example:

```
RegSetDword(@REGCURRENT, "A Test Key\[My DWORD Value]", 32)
```

See Also:

RegSetBin, RegSetValue

Runs a program via the Windows ShellExecute command

Syntax:

ShellExecute(program-name, params, directory, display mode, operation)

Parameters:

(s) program-name the name of the desired .EXE, .COM, .PIF, .BAT file or a data file.
(s) params optional parameters as required by the application.
(s) directory current working directory (if applicable).
(i) display mode @NORMAL, @ICON, @ZOOMED, @HIDDEN; or 0 for the default mode.
(i) operation operation to perform on the specified file.

Returns:

(i) @TRUE on success; @FALSE on failure.

This function uses the Windows ShellExecute API to launch the specified file. The similar RunShell function also uses the ShellExecute API in the 16-bit version, but uses the CreateProcess API in the 32-bit version. Note that RunShell has a "wait" parameter, while this function does not.

"operation" is the operation to perform on the file ("Open", "Print", etc.), which may or may not correspond to an available "verb" on the context menu for the file. This parameter may be case-sensitive. Specify a blank string "" for the file's default operation.

Note: If you use this function to launch a shortcut, and the shortcut points to an invalid path, Windows will display a "Missing Shortcut" dialog box asking if you wish to update the shortcut. This would not be suitable to use in unattended operation. Instead, you could use one of the Run.. functions to launch the shortcut, which would return an error #1932 if the shortcut could not be launched, and this error could be trapped using the ErrorMode function.

Example:

```
; launches a shortcut to a "Dial-Up Networking" item on the desktop
ShellExecute("d:\win95\desktop\netcom.lnk", "", "", @NORMAL, "")
WinWaitClose("Connect To")
```

See Also:

RunShell

Modifies the specified shortcut file.

Syntax:

ShortcutEdit(link-name, target, params, start-dir, show-mode)

Parameters:

(s) link-name the name of shortcut .LNK file to be created.
(s) target file or directory name which "link-name" will point to.
(s) params optional command-line parameters for "target"
(s) start-dir "Start in" directory for "target".
(i) show-mode "Run" mode for "target": 1 (**@ICON**), 2 (**@NORMAL**), or 3 (**@ZOOMED**).

Returns:

(i) **@TRUE** if the shortcut was successfully modified;
@FALSE if it wasn't.

See **ShortcutMake** for further information on these parameters.

Example:

```
DirChange("C:\Win95\Desktop")
ShortcutMake("system~1.LNK", "c:\Program Files\winbatch\system~1.wbt", "", "", @NORMAL)
ShortcutEdit("system~1.LNK", "", "", "c:\Win95\desktop", @NORMAL)
```

See Also:

ShortCutExtra, ShortcutInfo, ShortcutMake

Sets additional information for the specified shortcut file.

Syntax:

ShortcutExtra(link-name, description, hotkey, icon-file, icon-index)

Parameters:

(s) link-name the name of shortcut .LNK file to be modified.
(s) description the internal description for the shortcut.
(s) hotkey the "shortcut key" to be assigned to the shortcut.
(s) icon-file a file containing an icon to be used for the shortcut, with optional path.
(i) icon-index the 0-based index position of the desired icon within "icon-file".

Returns:

(i) **@TRUE** if the shortcut was successfully modified;
@FALSE if it wasn't.

The "description" parameter only sets an internal description, which is not actually displayed anywhere.
If "hotkey" is not a blank string (""), it specifies the hotkey ("shortcut key") for the shortcut. This can be an alphanumeric or special character (see **SendKey** for a list of special key characters), optionally preceded by one or more of the following modifiers:

! (Alt)
^ (Control)
+ (Shift)

Note that this function can be used to set hotkeys which would be impossible to set from within the shortcut properties dialog in Explorer.
"Icon-file" can be used to specify an .EXE (or .DLL) file or an .ICO file containing an icon which you want to be used for the shortcut. If "icon-file" specifies an .EXE (or .DLL) file (which can contain multiple icons), then "icon-index" can be used to specify the offset of a particular icon within "icon-file", where 0 indicates the first icon in the file, 1 indicates the second icon, etc. If "icon-file" specifies an .ICO file, then "icon-index" should be 0.
You can specify a blank string ("") for "icon-file", and 0 for "icon-index", to use the default icon.

Example:

```
DirChange("C:\Win95\Desktop")
ShortcutMake("system~1.LNK", "c:\Program Files\winbatch\system~1.wbt","", "c:\Program Files\
Winbatch", @NORMAL)
ShortcutExtra("system~1.LNK", "WinBatch Version Info", "^!j", "", 0)
```

See Also:

ShortcutEdit, ShortcutInfo, ShortcutMake

Returns information on the specified shortcut file.

Syntax:

ShortcutInfo(link-name)

Parameters:

(s) link-name the name of shortcut .LNK file.

Returns:

(s) a TAB delimited list of information on the shortcut file.

ShortcutInfo returns a TAB-delimited list containing the following items (some of which may be blank):

target	file or directory name which the shortcut points to.
params	command-line parameters for "target".
start-dir	"Start in" directory for "target".
show-mode	"Run" mode for "target": 1 (**@ICON**), 2 (**@NORMAL**), or 3 (**@ZOOMED**).
description	the internal description for the shortcut.
hotkey	the "shortcut key" for the shortcut.
icon-file	the name of the icon file being used by the shortcut.
icon-index	the 0-based index position within "icon-file" of the icon being used.

Example:

```
DirChange("C:\Win95\Desktop")
ShortcutMake("system~1.LNK", "c:\Program Files\winbatch\system~1.wbt", "", "", @NORMAL)
ShortcutExtra("system~1.LNK", "WinBatch Version Info", "^!j", "", 0)
info=ShortcutInfo("system~1.LNK")

target=    ItemExtract(1, info, @tab)
params=    ItemExtract(2, info, @tab)
startdir=  ItemExtract(3, info, @tab)
showmode=  ItemExtract(4, info, @tab)
desc=      ItemExtract(5, info, @tab)
hotkey=    ItemExtract(6, info, @tab)
iconfile=  ItemExtract(7, info, @tab)
iconindex= ItemExtract(8, info, @tab)

editinfo=StrCat("filename=",filename,@cr,"params=",params,@cr,"workdir=",workdir,@cr,"showmod
e=",showmode)
extrainfo=StrCat("desc=",desc,@cr,"hotkey=",hotkey,@cr,"iconpath=",iconpath,@cr,"iconindex=",
iconindex)
Message("ShortcutInfo Syntax", StrCat(editinfo,@cr, extrainfo))
```

See Also:

ShortcutEdit, ShortCutExtra, ShortcutMake

Creates a Windows 95 shortcut for the specified filename or directory.

Syntax:

ShortcutMake(link-name, target, params, start-dir, show-mode)

Parameters:

(s) link-name the name of shortcut .LNK file to be created.
(s) target file or directory name which "link-name" will point to.
(s) params optional command-line parameters for "target".
(s) start-dir "Start in" directory for "target".
(i) show-mode "Run" mode for "target": 1 (**@ICON**), 2 (**@NORMAL**), or 3 (**@ZOOMED**).

Returns:

(i) **@TRUE** if the shortcut was successfully created;
@FALSE if it wasn't.

This function can be used to create a shortcut file which points to a filename or to a directory.
"Params" and "start-dir" are optional, and can be set to blank strings (""). "Show-mode" is optional, and can be set to 0.
If "target" specifies a directory, the other parameters are meaningless.

Example:

```
DirChange("C:\Win95\Desktop")
ShortcutMake("system~1.LNK", "c:\Program Files\winbatch\system~1.wbt","", "c:\Program Files\
Winbatch", @NORMAL)
```

See Also:

ShortcutEdit, ShortCutExtra, ShortcutInfo

Returns the Julian day given a datetime.

Syntax:

TimeJulToYmd(julian-date)

Parameters:

(i) julian-date a Julian date.

Returns:

(s) the datetime corresponding to the specified Julian date.

This function converts the specified (numeric) Julian date value to a datetime in YmdHms format. The "Hms" portion of the returned YmdHms string will always be "00:00:00".

Example:

```
today = TimeYmdHms()
jul_today = TimeJulianDay(today)
jul_lastweek = jul_today - 7
lastweek = TimeJulToYmd(jul_lastweek)
FileTimeSet("myfile.log", lastweek)
```

See Also:

TimeJulianDay

Subtracts one YmdHms variable from another.

Syntax:

TimeSubtract(datetime, datetime difference)

Parameters:

(s) datetime a datetime using the format of YY:MM:DD:HH:MM:SS.
(s) datetime difference a datetime to be subtracted from the original using the same format

Returns:

(s)

Use this function to subtract a specified date/time from an original date/time. TimeSubtract uses normalized conversion so a valid date/time will be returned. "datetime difference" can not be larger than "datetime".

Example:

```
time_now = TimeYmdHms()
time_yesterday = TimeSubtract(time_now, "00:00:01:00:00:00")
FileTimeSet("myfile.log", time_yesterday)
```

See Also:

TimeAdd

Returns a unique "Window ID" (pseudo-handle) for the specified window name.

Syntax:

WinIdGet(partial-winname)

Parameters:

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

Returns:

(s) the unique "Window ID".

Use this function to obtain the unique "Window ID" (pseudo-handle) for the specified parent window name. All functions which accept a partial window name as a parameter now accept the Window ID obtained with **WinIdGet**. This can be useful to distinguish between multiple windows with the same name, or to track a window whose title changes.

Example:

```
Run("notepad.exe", "")
winid1 = WinIdGet("~Notepad")  ; gets the most-recently-accessed Notepad
Run("notepad.exe", "")
winid2 = WinIdGet("~Notepad")  ; gets the most-recently-accessed Notepad
WinPlace(0, 0, 500, @ABOVEICONS, winid1)
WinPlace(500, 0, 1000, @ABOVEICONS, winid2)
WinActivate(winid1)
```

See Also:

DllHwnd, WinExist, WinGetActive, WinTitle

Returns a list of all open windows and their Window ID's.

Syntax:

WinItemNameId()

Parameters:

(none)

Returns:

(s) list of the titles and Window ID's of all open windows.

This function returns a list of top-level window titles and their corresponding "Window ID's", in the form:

"window1-name|window1-ID|window2-name|window2-ID|..."

Example:

```
winlist = WinItemNameId()
TextSelect("Windows and ID's", winlist, "|")
```

See Also:

WinIdGet, WinItemize

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**
-4	Windows Platform 0 = Other 1 = Windows 2 = Windows for Workgroups 3 = Win32s 4 = Windows NT 5 = Windows 95
-3	WIL EXE type 0=Win16, 1=Intel32, 2=Alpha32, 3=Mips32
-2	WIL platform 1=Win16, 2=Win32
-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

Additional request #'s for WinMetrics (32-bit version only):

41	TRUE or non-zero if the Microsoft Windows for Pen computing extensions are installed; zero, or FALSE, otherwise.
42	TRUE or non-zero if the double-byte character set (DBCS) version of USER.EXE is installed; FALSE, or zero otherwise.
43	Number of buttons on mouse, or zero if no mouse is installed.
44	(Win95 only) TRUE if security is present, FALSE otherwise.
63	(Win95 only) The least significant bit is set if a network is present; otherwise, it is cleared. The other bits are reserved for future use.
67	(Win95 only) Value that specifies how the system was started: 0 - Normal boot 1 - Fail-safe boot 2 - Fail-safe with network boot Fail-safe boot (also called SafeBoot) bypasses the user's startup files.
70	TRUE or non-zero if the user requires an application to present information visually in situations where it would otherwise present the information only in audible form; FALSE, or zero, otherwise.
73	(Win95 only) TRUE if the computer has a low-end (slow) processor.
74	(Win95 only) TRUE if the system is enabled for Hebrew/Arabic languages.

There are a number of other request #'s which can be specified, but are of limited usefulness and therefore not documented here. Details on these can be obtained from Win32 programming references, available from Microsoft (and others).

Example:

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

See Also:

Environment, MouseInfo, NetInfo, WinParmGet, WinResources

Returns system configuration information.

Syntax:

WinSysInfo( )

Parameters:

(none)

Returns:

(s) a TAB delimited list of system configuration information.

WinSysInfo returns a TAB-delimited list containing the following items:

1. computer name of the current system.
2. processor architecture.
3. page size (specifies granularity of page protection and commitment).
4. mask representing the set of processors configured into the system.
5. number of processors in the system.
6. processor type.
7. granularity in which memory will be allocated.
8. system's architecture-dependent processor level.
9. architecture-dependent processor revision.

Note: This function should be used instead of WinConfig in the 32-bit version.

Example:

```
sysinfo = WinSysInfo()
computer  = ItemExtract(1, sysinfo, @TAB)
processor = ItemExtract(6, sysinfo, @TAB)
Message(computer, "is a %processor%")
```

See Also:

WinMetrics, WinParmGet, WinResources