WinBatch User’s Guide

Manual Revision Jan., 99

WinBatch
THE BATCH LANGUAGE
FOR WINDOWS

Wilson WindowWare, Inc.

5421 California Ave. SW

Seattle WA 98136 USA

Orders: 1.800.762.8383 (USA only)

International: 1.206.938.1740

Fax: 1.206.935.7129

Technical Support: Phone 1.206.937.9335

Internet: sales@windowware.com

www: http://www.windowware.com

CONTENTS

Welcome

Overview

From R. Petersen on CompuServe; "Our entire company is becoming 100% dependent on dozens of WinBatch programs that make everything hang together."

WinBatch can automate all versions of Windows. WinBatch manipulates the Windows interface, Windows applications, and network connections.

So, any operations in or from Windows 3.1/95/98, or Windows NT, can be done at the click of a mouse button with WinBatch.

Testing values, getting system information, working with directories, logging events, and manipulating files are just a few of these capabilities.

WinBatch is often used to assemble reports, install software, automate testing, control processes, acquire data, and add efficiency to the Windows workstation.

WinBatch excels in tailoring the Windows interface to fit any user. Standard operations are easy to program in WinBatch.

WinBatch functions can manipulate:

- The operating system
- The Windows interface
- Any and all Windows applications
- Most MS-DOS applications
- Most networks.

WinBatch has two components:

- A system control language called WIL (Windows Interface Language)
- An interpreter that reads a text file written in the WIL language and performs the required operations.

Note: WinBatch is a **batch file** based implementation of WIL. A WinBatch batch file is a text file containing one or more lines of WIL functions and commands.

WinBatch is continually being updated to function with future versions of Microsoft Windows.

It is easy to get started in Windows programming with WinBatch. The WinBatch Studio interface makes editing and debugging streamlined and easy. Useful system utilities are produced quickly with WinBatch. All the things you couldn't do before in Windows are suddenly just a few minutes away.

When projects demand an advanced solution, the flexiblilty of WinBatch is ready to speed development. A visual dialog editor, a window information grabber, a debugger, and the power of structured programming are included in the WinBatch package.

WinBatch has these capabilities: mathematical operators, text manipulation, binary file editing completely in memory, network connectivity, and Windows system manipulation.

Many WinBatch functions can accomplish more in a single line statement, than what could take pages of forms design, property setting, and coding in other programming languages.

WinBatch is optimized for making quick work of custom system management utilities.

Registering Your Copy of WinBatch

Registered users of WinBatch receive manuals, technical support, use of Wilson WindowWare on-line information services, and special offers on new versions of WinBatch and other Wilson WindowWare products.

You must register your copy to obtain these benefits.

You can register WinBatch by mailing your registration card, faxing your registration card, or calling Wilson WindowWare.

New in WinBatch

Registered users can share their WinBatch experience with other users on the Wilson WindowWare Web BBS.

The latest versions of WinBatch are available on-line. The addresses here may change at any time - check your installation sheet.

- Internet FTP: ftp.windowware.com

- Internet Web page: http://www.windowware.com

There is a comprehensive database of tech support questions, answers, and sample scripts available on the Wilson WindowWare Web site.

- Internet Tech Support page: http://techsupt.windowware.com

What WinBatch Can Do

With 367 general functions and commands, tons of networking functions, 86 physical constants, and 24 operators, more than 9 additional extender DLL's for various specific operations. WinBatch can:

- Solve numerous system management problems
- Run Windows and DOS programs
- Send keystrokes directly to applications
- Schedule scripts to run at a specific time
- Send menu items directly to Windows applications
- Rearrange, resize, hide, and close windows
- Run programs either concurrently or sequentially
- Display information to the user in various formats
- Prompt the user for any needed input
- Present scrollable file and directory lists
- Copy, move, delete, and rename files
- Read and write files directly
- Copy text to and from the Clipboard
- Perform string and arithmetic operations
- Make branching decisions based upon numerous factors
- Call Dynamic Link Libraries
- Act as an OLE 2.0 automation client

And much, much more.

How WinBatch is Used

Launch one WinBatch icon or file and you can run from one to thousands of operations. One WinBatch script can squeeze any number of operations into a single batch file that runs just like any other Windows program. It can run from a Windows shell or any application that can run another application.

WinBatch excels in controlling other applications both in Windows and MS-DOS. From getting system information, through controlling software, to accessing the network, WinBatch can do it all from Windows.

System Requirements

WinBatch requires an IBM PC or compatible running Microsoft Windows.

About This Manual

WinBatch uses Wilson WindowWare's Windows Interface Language (WIL). Please refer to the **WIL Reference Manual** for an introduction to WIL, as well as for complete documentation of the many functions available in WIL (and in WinBatch).

This User's Guide includes topics and functions which are exclusive to WinBatch, or which behave differently in WinBatch. Since the WIL language is distributed with software other than WinBatch, we have contained those functions and utilities that are specific to WinBatch in this manual.

Network manipulation functions are dealt with in extensions to WIL, called WIL Extender DLLs. The extenders are dynamic link libraries that can be accessed with the WIL AddExtender( ) function. Each extender has its own help file.

To obtain any additional extender DLLs, either download them from our website, or copy them from the WinBatch CD. To install extenders from the CD, locate the Extenders directory on the CD. The Extender directory will contain all the available extender subdirectories, select the appropriate extender subdirectory and run the corresponding SETUP.WBT file.

Notational Conventions

Throughout this manual, we use the following conventions to distinguish elements of text:

Boldface

Used for important points, programs, function names, and parts of syntax that must appear as shown.

System

Used for items in menus and dialogs, as they appear to the user.

`Bold fixed-width`

Used for WIL sample code.

Italics

Used for emphasis, and to liven up the documentation just a bit.

Acknowledgments

WinBatch software developed by Morrie Wilson.

Documentation written by Richard Merit, Tina Browning, Jim Stiles, Deana Dahley and Laura Plaut.

WinBatch Setup

Installing and Configuring WinBatch

WinBatch is easy to install.

If you have purchased WinBatch+Compiler, go to Appendix B, page 91, for instructions on installing WinBatch+Compiler.

The WinBatch installation program is itself a Windows application, so make sure Windows is running.

Insert your WinBatch CD into your CD-ROM disk drive. If you have your CD-ROM configured to "Auto-Start", the setup will run automatically. Otherwise, open the Explorer to your CD-ROM drive and double click on SETUP.EXE.

You will be prompted for the directory to install WinBatch into. The default will be C:\PROGRAM FILES\WINBATCH. You may change the directory if you wish, but you cannot install WinBatch on a network drive. *To install WinBatch on a server, you need to have purchased a special site license.*

Hit <u>OK</u>, and then examine the install specification screen.

Select the desired items, and let the setup process copy the files. The Network extender DLLs and HLP files are installed by default, so if you do not want these installed, de-select the checkbox for this item. All other WIL Extenders are available on the CD-ROM, in the Extenders subdirectory, along with their appropriate SETUP.WBT files.

You will be asked for license numbers. They are printed on the inside back cover of this manual.

WIL extenders must be installed separately.

WinBatch also includes WIL extender Dlls. These are special Dlls designed to extend the built-in function set of the WIL processor. These Dlls typically add functions not provided in the basic WIL set, such as network commands for particular networks (Novell, Windows for WorkGroups, LAN Manager and others), MAPI, ODBC, and other important Application Program Interface functions as may be defined by the various players in the computer industry from time to time.

To install WIL extender Dlls:

Insert the WinBatch CD in your CD-ROM drive.If you have your CD-ROM configured to “Auto-Start”, the setup will run automatically. Otherwise, open the Explorer to your CD-ROM drive and double click on SETUP.EXE.

Click on the “Explore CD” button on the install specification screen. Select “Extenders” from the directory listing. Now select the directory of the extender you wish to install. In each directory you will find a file called SETUP.WBT. Double-click on the SETUP.WBT file, and the extender will be installed automatically.

Note: To copy WinBatch files to floppy disks:

Floppy copies are available off the CD-ROM Install directory. Copy the contents of each of the following directories Disk1, Disk2, Disk3 to its own formatted floppy disk.

License Numbers: Temporary and Permanent

Purchase of this software includes technical support, a full package of software materials, notification of updates and enhancements

To Enter License Information:

You can find your licensing information on the inside the back cover of this manual. To license WinBatch do the following:

1) Start WinBatch by running WinBatch.exe, and hit the “Enter License Info” button on the registration reminder screen.

2) Enter the information from the label inside the back cover of your WinBatch User’s Guide into the licensing dialog box. Generally the top line is a serial number and the bottom line is the control number. Almost all control numbers begin with WG01 (Whiskey Golf Zero One) or YG01 (Yankee Golf Zero One) followed by four more letters and/or numbers.

3) After entering the information, select the OK button.

You may be prompted to enter your name at this point. After entering your name, select the OK button. That should do it. You will need to enter this information once each for the WinBatch.exe and the WBCompiler.exe.

You may need to re-enter this number if you re-install Windows, update to a new version of Windows, or update to a new revision of WinBatch. Please keep the number private. It is not required for Tech Support services unless asked.

WinBatch will run without license numbers, but a screen will appear to remind you to register the software.

Keep license numbers in a safe place. They will be needed whenever WinBatch is reinstalled. There will be a fee for replacing lost license numbers.

If you have been issued a temporary license number, it will expire after a period of time and the evaluation screens will re-appear. To prevent this, obtain a permanent license number in place of a temporary one.

Once you register your copy of WinBatch, you can enter your registration information. To make the registration screen appear, for a temporarly licensed copy, hold the shift key down while starting a WinBatch utility. Enter the license numbers into the screen that will pop up.

Once you have WinBatch successfully installed, you can check the version number by launching WinBatch.exe (which will launch the DEFAULT.WBT) and then selecting the “About WinBatch” item in the item select listbox. The version should be something like “98F” or “97D” depending on which version you have.

There are several other utilities that can also be run from that DEFAULT.WBT dialog box that might also be useful.

WinBatch Studio

WinBatch Studio Overview

WinBatch Studio is an ASCII text editor capable of editing numerous WinBatch files of an almost unlimited size (limited only by available Windows memory). WinBatch Studio has many features designed for creating and maintaining WinBatch program source code. Build, debug and run your WinBatch programs directly from WinBatch Studio.

As an ASCII text editor, WinBatch Studio allows you to open numerous .wbt files at once, print half sized “two-up” pages side by side in landscape orientation, print headers and footer text (document name, date and time, page number) and, merge files together.

WinBatch Studio Menus

File Menu

The File menu offers the following commands:

New	Creates a new document.
Open	Opens an existing document.
Close	Closes an opened document.
Merge	Inserts the contents of a new document into the current document.
Save	Saves an opened document using the same file name.
Save As	Saves an opened document to a specified file name.
Save All	Saves all opened documents.
Revert	Re-reads the current document from disk, returning to its original state.
Page Setup	Displays printing options.

Print Setup	Selects a printer and printer connection.
Print	Prints a document.
Print Preview	Displays the document on the screen as it would appear printed.
Send...	Sends the active document through electronic mail.
Properties	Displays information about the current document.
Exit	Exits WinBatch Studio.

Edit Menu

The Edit menu offers the following commands:

Undo	Reverse previous editing operation.
Redo	Reverse previous undo operation.
Cut	Deletes data from the document and moves it to the clipboard.
Copy	Copies data from the document to the clipboard.
Paste	Pastes data from the clipboard into the document.
Delete	Deletes data from the document.
Copy Other	Copies specific data from the document to the clipboard.
Cut Other	Cuts specific data from the document to the clipboard.
Change Case	Changes the case of selected data from the document.
Select All	Selects all the data in the document.

View Menu

The View menu offers the following commands:

Toolbars	Shows, hides, or customizes the toolbars.

Status Bar	Shows or hides the status bar.
Output	Shows or hides the output window.
Watch	Shows or hides the watch window.
Options	Editor, keyboard, and file specific settings are maintained in this dialog box.

Search Menu

The Search menu offers the following commands:

Find	Searches the current document for the specified text.
Find next	Repeats the last find operation, using the same options.
Replace	Searches the current document for the specified text, and replaces the found text with specified text.
Find In Files	Searches one or more files for the specified text.
Go To Line	Moves the caret to the specified line number.
Match Brace	If the caret is placed on a brace character "().{}, or []" the caret is moved to the matching brace character.

Window Menu

The Window menu offers the following commands, which enable you to arrange multiple views of multiple documents in the application window:

New Window	Creates a new window that views the same document.
Split	Split the active window into panes.
Cascade	Arranges windows in an overlapped fashion.
Tile Horizontally	Arranges windows in non-overlapped tiles horizontally.
Tile Vertically	Arranges windows in non-overlapped tiles vertically.
Close All	Closes all open windows.

Arrange Icons	Arranges icons of closed windows.

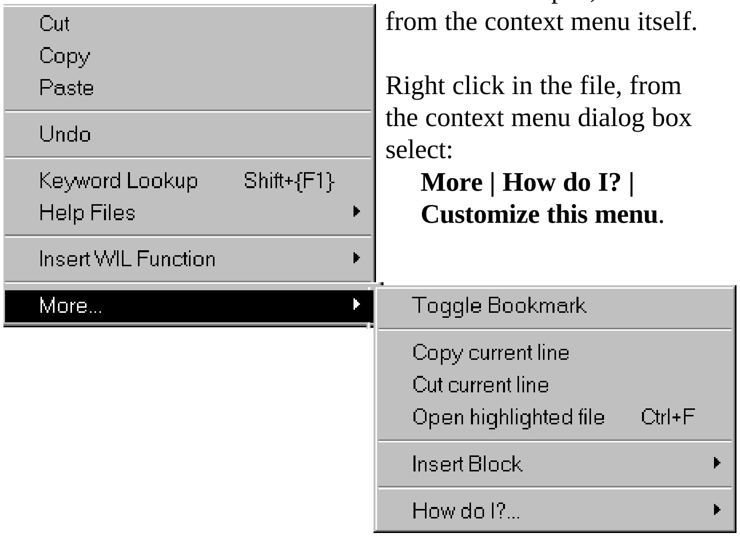
Help Menu

The Help menu offers the following commands, which provide you assistance with this application:

Help Contents	Offers you an index to topics on which you can get help.
About WinBatch Studio	Displays the version number of this application.

Context Menu

WinBatch has a completely configurable context menu accessed by clicking the right mouse button anywhere within an open file. Right clicking results in a context menu drop down list filled with many useful macros. Using the Windows Interface Language, you can write your own macros and place them on this menu for easy access.



To make changes to the context menu, open the WSPOPUP.MNU file with File/Open, or access it from the context menu itself.

Right click in the file, from the context menu dialog box select:

More | How do I? | Customize this menu.

Note: For more information on editing menu files, see the section “**Menu Files**” in the Windows Interface Language Reference manual or help file.

***Important:** If you have made any modifications to the file WSPOPUP.MNU, make sure to make backups of the file. You can save the file as WSPOPUP_BKUP.MNU. WSPOPUP.MNU is automatically overwritten upon each install.*

Using WinBatch

Creating WinBatch Script Files

WinBatch is a script file interpreter. Before you can do anything useful with the WinBatch interpreter, you must have at least one WinBatch script file to interpret.

Your WinBatch installation puts several sample scripts into your WinBatch\Samples directory.

WinBatch script files must be formatted as plain text files. You can create them with WinBatch Studio (included), WinEdit (Wilson WindowWare's optional text editor for programmers), the Windows Notepad or another text editor.

Word processors can also save scripts in plain text formatted files.

When you installed WinBatch, an association is automatically established between WinBatch and the .WBT file extension. The .WBT extension is used in this manual for batch file extensions, but you can use other file types as well. If you want to double click on a batch file and have Windows run it, be sure that you associate it in Windows with your WinBatch executable program file.

Each line in a WinBatch script file contains a statement written in WIL, Wilson WindowWare's Windows Interface Language.

A statement can be a maximum of 255 characters long (refer to the WIL Reference Manual for information on the commands you can use in WinBatch). Indentation does not matter. A statement can contain functions, commands, and comments. Functions and constants are not case-sensitive.

You can give each WinBatch script file a name which has an extension of WBT (e.g. TEST.WBT). We'll use the terms WinBatch script files and WBT files interchangeably.

WinBatch Operation:, Running WinBatch Utilities

WinBatch utilities are very versatile. They can be run from:

- icons in the Windows Explorer.
- as automatic execution macros for Windows, via the “WINDOWS\Start Menu\Programs\StartUp\” directory.
- from macros in word processors and spreadsheets.
- from a command line entry, such as “Start…Run...” Taskbar menu option in Windows.
- by double clicking or dragging and dropping file names on the WinBatch icon.
- from menu items on the Windows “System Tray” using PopMenu, an accessory program included with WinBatch.
- from other WinBatch scripts to serve as single or multiple “agents”, event handlers, or schedulers.
- from any Windows application or application macro language that can execute another Windows program. Software suite macro languages and application builders like Visual Basic and PowerBuilder are examples of these.

WinBatch system utilities run like any other Windows programs. They can run from a command line, a desktop icon, or from a file listing such as the Windows and Windows NT File Managers or Explorer.

WinBatch utilities are usually run as files with the extension .WBT. They can accept as many as 9 passed parameters when run from a command line.

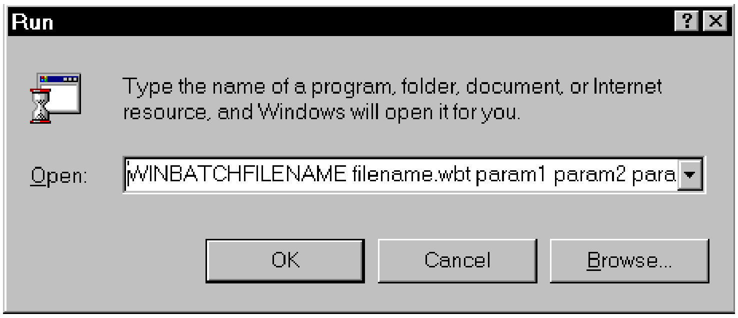
This capability can be used from the command line in the **File** **Run** menu items of both the Windows File Manager and the Program Manager or the **Start** **Run** menu items in Windows 95/98 and NT 4.0.

Parameters can be also be passed through the command line entry included in the item properties of any icon in the Explorer. Finally, an application can send parameters to a WinBatch utility it launches from a command line or from a function in a macro language.

A command like this runs a WinBatch file from a command line or an icon:

```
WINBATCHFILENAME filename.wbt param1 param2 ... param9
```

This command line can be entered into a Command Line text entry box like this one from the Windows 95/98 **Start Run** menu option.


Run
?
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
WINBATCHFILENAME filename.wbt param1 param2 para
OK
Cancel
Browse...

The command line is longer than the dialog can show, but it can be easily edited with the arrow keys.

WINBATCHFILENAME is the generic name of your WinBatch executable. The specific, or actual, name for the WinBatch application will change to reflect the operating system in use: Windows 3.1, Windows 95/98, and the different Windows NT versions.

(See Appendix A, page 85 for more information on file names).

"filename.wbt" is any valid WBT file, and is a required parameter.

"param1 param2 ... param9" are optional parameters (there are a maximum of nine of these) to be passed to the WBT file on startup. Each is delimited from the next by one space character.

These parameters will be automatically inserted into variables named **param1**, **param2**, …up to **param9**. The WinBatch utility will be able to use these. An additional variable, **param0**, gives you the total number of command-line parameters.

Example command line:

```
C:\Program files\Winbatch\System\Winbatch.exe Example.wbt John Joe
Mary
```

Example script:

```
;contents of example.wbt
names=StrCat(param1,@CRLF,param2,@CRLF,param3)
Message("Names",names)
exit
```

Using Icons to Pass Parameters to WinBatch Utilities

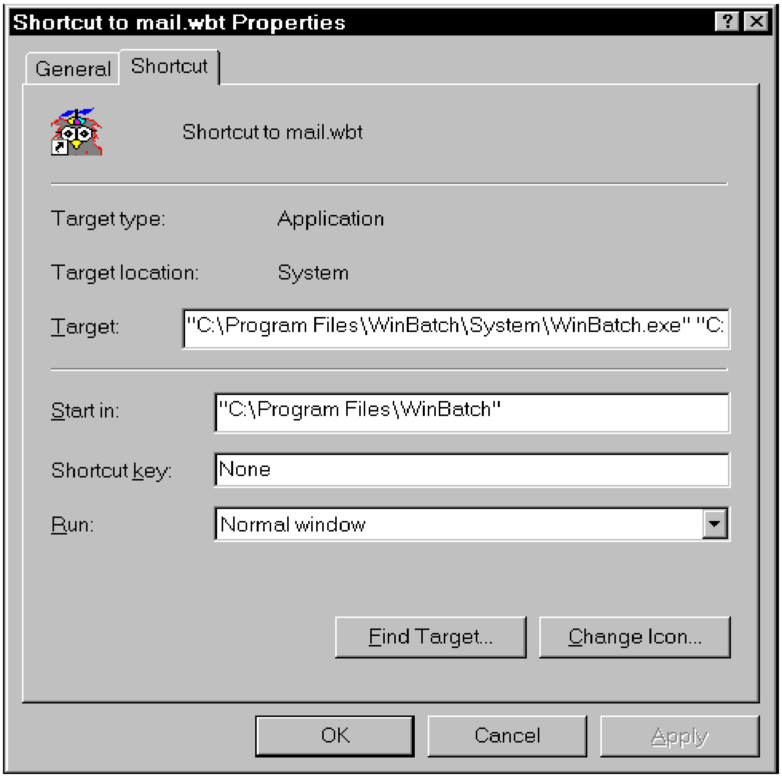
In order to pass parameters to a WinBatch .wbt file, you ***must*** *run the WinBatch executable, itself, and it must be followed by the name of the WinBatch script file and any other desired parameters.*

WBT files run from the Explorer as shortcut icons must have their complete path in the Properties dialog box in order for command line parameters to be received.

For example, the command line for “**MAIL.WBT**”, an imaginary WinBatch utility that runs mail with a password passed as a parameter might be:

```
“C:\PROGRAM FILES\WINBATCH\SYSTEM\WINBATCH.EXE
C:\PROGRAM FILES\WINBATCH\MAIL.WBT PASSWORD”.
```

To edit icon properties, highlight the icon, hold down ALT, and press ENTER. The program item properties box should look like the following:


Shortcut to mail.wbt Properties

Shortcut to mail.wbt

Target type: Application

Target location: System

Target: "C:\Program Files\WinBatch\System\WinBatch.exe" "C:

Start in: "C:\Program Files\WinBatch"

Shortcut key: None

Run: Normal window

Example: Displaying passed parameters in a message box.

To determine the total number of command line parameters, display the **param0** variable in a message box.

WinBatch works like the DOS Batch language when inserting parameters into text. Enclosing them in percent (%) signs works in WinBatch, too. This example is a simple one line WinBatch function that:

1. Displays a Message box with an OK button.
2. Specifies a title.
3. Specifies a message.
4. Puts varying information into the title or the message.

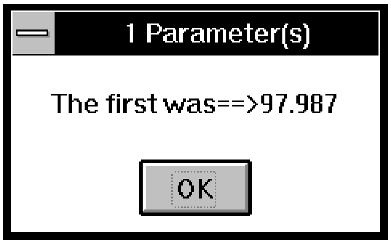
The **Message** function has this form:

```
Message(“title in quotes”,”message in quotes”)
```

The actual statement used to produce this dialog box was:

```
Message(“%param0% Parameter(s)”, “The first was==> %param1%”)
```

It produced:



The first was==>97.987

The command line that executed the utility producing the statement above was:

```
“C:\PROGRAM FILES\WINBATCH\SYSTEM\WINBATCH.EXE” “C:\TEMP\MESSAGE.WBT”
97.987
```

Note: Full path names were used for both the WinBatch executable file and for the WinBatch utility. Spaces separate the three parts of the command line. Quotes are necessary around path and filenames, if the path name contains spaces. Otherwise the quotes are ignored.

Passing Parameters Between WinBatch Script Files:

You can pass command line parameters from one WinBatch script file to another WinBatch script file. To do this, place percent characters (%) around the variables as in: %variable%.

Example:

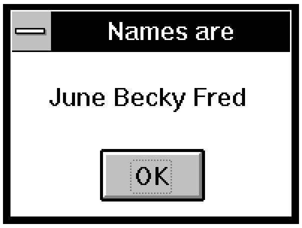
The first WBT calls a second WBT then passes three parameters.

```
Call(“test.wbt”, “Fred Becky June”)
```

TEST.WBT contains the following line:

```
Message(“Names are”, “%param3% %param2% %param1%”)
```

which produces:



June Becky Fred

WinBatch Functions

Function Reference Introduction

This section includes only those additional WinBatch functions which do not appear in the **WIL Reference Manual.** The **WIL Reference Manual** is your primary reference for the functions available in WinBatch.

> **Note**: The functions listed under the **See Also** headings may be documented either in this **User's Guide** or in the **WIL Reference Manual**.

Function List

BoxOpen(title, text)
Opens a WinBatch message box.

BoxShut( )
Closes the WinBatch message box.

BoxText(text)
Changes the text in the WinBatch message box.

BoxTitle(title)
Changes the title of the WinBatch message box.

Breakpoint
Causes a breakpoint on the next statement when used with a script debugger, like WinBatch Studio. Otherwise the command does nothing, outside of WinBatch Studio.

Graphical Box Functions

BoxButtonDraw(box ID, button ID, text, coordinates)
Creates a push-button in a WinBatch box.

BoxButtonKill(box ID, button ID)
Removes a push-button from a WinBatch box.

BoxButtonStat(box ID, button ID)
Determines whether a push-button in a WinBatch box has been pressed.

BoxButtonWait( )
Waits for any button in any box to be pressed.

BoxCaption(box ID, caption)
Changes the title of a WinBatch box.

BoxColor(box ID, color, wash color)
Sets the background color for use with a WinBatch object.

BoxDestroy(box ID)
Removes a WinBatch box.

BoxDrawCircle(box ID, coordinates, style)
Draws an ellipse in a WinBatch box.

BoxDrawLine(box ID, coordinates)
Draws a line in a WinBatch box.

BoxDrawRect(box ID, coordinates, style)
Draws a rectangle in a WinBatch box.

BoxDrawText(box ID, coordinates, text, erase flag, alignment)
Displays text in a WinBatch box.

BoxesUp(coordinates, show mode)
Displays WinBatch boxes.

BoxMapMode(box ID, map mode)
Sets the mapping mode for a WinBatch box.

BoxNew(box ID, coordinates, style)
Creates a WinBatch box.

BoxPen(box ID, color, width)
Sets the pen for a WinBatch box.

BoxTextColor(box ID, color)
Sets the text color for a WinBatch box.

BoxTextFont(box ID, name, size, style, pitch & family)
Sets the font for a WinBatch box.

BoxUpdates(box ID, update flag)
Sets the update mode for, and/or updates, a WinBatch box.

BoxDataClear(box ID, tag)
Removes commands from a WinBatch box command stack.

BoxDataTag(box ID, tag)
Creates a tag entry in a WinBatch box command stack.

Note: In our shorthand method for indicating syntax, on the following pages next to "parameters", the (s) in front of a parameter indicates that it is a string. An (i) indicates that it is an integer and a (f) indicates a floating point number parameter.

BoxOpen

Opens a WinBatch message box.

Syntax:

BoxOpen (title, text)

Parameters:

(s) title	title of the message box.
(s) text	text to display in the message box.

Returns:

(i)	always 1.

This function opens a message box with the specified title and text. The message box stays in the foreground while the WIL program continues to process.

The title of an existing message box can be changed with the **BoxTitle** function, and the text inside the box can be changed with the **BoxText** function.

Use **BoxShut** to close the message box.

Example:

```
BoxOpen(“Processing”, “Be patient”)
Delay(2)
BoxTitle(“Still processing”)
Delay(2)
BoxText (“Almost done”)
Delay(2)
BoxShut()
```

See Also:

BoxShut, BoxText , BoxTitle, Display, Message

BoxShut

Closes the WinBatch message box.

Syntax:

BoxShut ( )

Parameters:

(none)

Returns:

(i)	always 1.

This function closes the message box that was opened with BoxOpen.

Example:

```
BoxOpen(“Processing”, “Be patient”)
Delay(2)
BoxTitle(“Still processing”)
Delay(2)
BoxText (“Almost done”)
Delay(2)
BoxShut()
```

See Also:

BoxOpen, BoxText, BoxTitle

BoxText

Changes the text in the WinBatch message box.

Syntax:

BoxText (text)

Parameters:

(s) text	text to display in the message box.

Returns:

(i)	always 1.

Example:

```
BoxOpen(“Processing”, “Be patient”)
Delay(2)
BoxTitle(“Still processing”)
Delay(2)
BoxText(“Almost done”)
Delay(2)

BoxShut()
```

See Also:

BoxOpen, BoxShut, BoxTitle

BoxTitle

Changes the title of the WinBatch message box.

Syntax:

BoxTitle (title)

Parameters:

(s) title	title of the message box.

Returns:

(i)	always 1.

Example:

```
BoxOpen("Processing", "Be patient")
Delay(2)
BoxTitle("Still processing")
Delay(2)
BoxText ("Almost done")
Delay(2)
BoxShut()
```

See Also:

BoxOpen, BoxShut, BoxText, WinTitle

Breakpoint

Causes a breakpoint on the next statement when used with a script debugger, like WinBatch Studio. Otherwise the command does nothing.

Syntax:

Breakpoint

Parameters:

(none)

Returns:

(not applicable)

Use this command with WinBatch Studio to cause execution to stop in the debugger. If this command is encountered outside of the WinBatch Studio debugger it is ignored.

Debuggers usually have a method of setting a breakpoint on particular lines of a script or even stepping through the lines one at a time. Sometimes problems occur after extensive script execution where it would be tedious to step through. For example if you wish to investigate what happens on the 782'd pass of a FOR loop you could do something similar to the example code below:

Example:

```
a=1
   b=1000
   For xx = 1 to b
   If xx == 782 then BreakPoint
   c=a+xx+1
next
Message(“Loop”,”Complete”)
```

See Also:

Debug, DebugTrace…(see Windows Interface Language)

Graphical Box Functions

These WinBatch box functions generate attractive boxes with graphical interface elements. With a small number of primitive functions, very complex screens may be generated. The Box functions can draw lines, rectangles, circles, ellipses, text, and even additional windows on the screen. Plus they provide control over the size, placement, and color of the images.

[**Note:** Another way to create graphical dialogs is to use the **HTML Dialog Extender**. For further information see the **HTML Dialog Extender help file**.]

The WinBatch 95/98/NT setup program uses WinBatch box functions to display the GUI part of the user interface. Additional “box-drawing” wbt files can be found in the WinBatch\Samples subdirectory.

First, before we get into detailed descriptions of the box functions, we must define two very important data types. These are the “coordinate” and the “color” data type parameters.

Coordinate Parameters

A coordinate is a WinBatch string variable (actually a list) containing four numbers separated by commas. These four numbers define two points on the screen. The first number is the “X” coordinate of the first point, the second number is the “Y” coordinate of the first point, the third number is the “X” coordinate of the second point, and finally the fourth number is the “Y” coordinate of the second point.

The “0,0” point is in the upper left of the screen, and the “1000,1000” point is at the lower right.

With just these two points, WinBatch can size and place a number of items.

Rectangles: The first point defines the upper left corner of a rectangle, and the second point defines the lower right.

Circles and Ellipses: The first point defines the upper left corner of a bounding box for the Ellipse, and the second point defines the lower right corner of the bounding box. The ellipse will touch the bounding box at the center of each side of the bounding box.

Lines: The two points represent the beginning and end of a line.

Windows: The first point defines the upper left corner of a window, and the second point defines the lower right.

Color Parameters

A “color” data type is a WinBatch string variable (actually a list) containing three numbers separated by commas. These three numbers define the amount of red, green, and blue that the color has in it. Each number may vary from 0 (none) to 255 (max.). White has the maximum amount of all colors, while black lacks them all. A sample list of colors follow:

WHITE=”255,255,255”	RED=”255,0,0”	DKRED=”128,0,0”
BLACK=”0,0,0”	GREEN=”0,255,0”	DKGREEN=”0,128,0”
LTGRAY=”192,192,192”	BLUE=”0,0,255”	DKBLUE=”0,0,128”
GRAY=”128,128,128”	YELLOW=”255,255,0”	DKYELLOW=”128,128,0”
DKGRAY=”64,64,64”	CYAN=”0,255,255”	DKCYAN=”0,128,128”
LTPURPLE=”255,128,255”	PURPLE=”255,0,255”	DKPURPLE=”128,0,128”

BoxButtonDraw

Creates a push-button in a WinBatch box.

Syntax:

BoxButtonDraw(box ID, button ID, text, coordinates)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(i) button ID	the ID number of the desired push-button.
(s) text	text to appear in the button
(s) coordinates	dimensions of button, in virtual units (upper-x upper-y lower-x lower-y).

Returns:

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

Draws a button using standard Windows colors and fonts by specifying a unique "ID", text and coordinates. If an existing button "ID" is re-used, the text will be changed and then the button will be moved.

Note: If a button is moved, it is best to do so before the background is painted in order to color over the button's original position. Moving buttons does cause some "flashing" on the screen.

Example:

```
;;  sample code for BoxButtonDraw
bDraw1=1
bDraw2=2
bDraw3=3

BoxesUp("100,100,900,900", @normal)
BoxDrawText(1, "0,210,1000,1000","Drawing Buttons", @FALSE, 1)
TimeDelay(2)
BoxButtonDraw(1, bDraw1, "Button 1", "100,450,300,550")
TimeDelay(2)
BoxButtonDraw(1, bDraw2, "Button 2", "400,450,600,550")
TimeDelay(2)
BoxButtonDraw(1, bDraw3, "Button 3", "700,450,900,550")
bWho=0
while bWho == 0
for x =1 to 3
if BoxButtonStat(1,x) then bWho=x
next
endwhile
Message("Excuse Me", "Please, don't push my buttons")
BoxDestroy(1)
```

See Also:

BoxesUp, BoxNew, BoxButtonKill, BoxButtonStat, BoxButtonWait

BoxButtonKill

Removes a push-button from a WinBatch box.

Syntax:

BoxButtonKill(box ID, button ID)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(i) button ID	the ID number of the desired push-button.

Returns:

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

Example:

```
;;  sample code for BoxButtonKill
bDraw1=1
bDraw2=2
bDraw3=3

BoxesUp(“100,100,900,900”, @normal)
BoxDrawText(1,”0,210,1000,1000”,”Select a Button”, @FALSE, 1)
BoxButtonDraw(1, bDraw1, “Button 1”, “100,450,300,550”)
BoxButtonDraw(1, bDraw2, “Button 2”, “400,450,600,550”)
BoxButtonDraw(1, bDraw3, “Button 3”, “700,450,900,550”)
bWho=0
while bWho == 0
for x =1 to 3
if BoxButtonStat(1,x) then bWho=x
next
endwhile
Switch bWho
;Message(“Excuse Me”, “Please, don’t push my buttons”)
Case 1
BoxDrawText(1, “0,310,1000,1000”, “Killing Button %Bwho%”, @TRUE, 1)
TimeDelay(2)
BoxButtonKill(1, bDraw1)
Break
Case 2
BoxDrawText(1, “0,310,1000,1000”, “Killing Button %Bwho%”, @TRUE, 1)
BoxButtonKill(1, bDraw2)
TimeDelay(2)
Break
Case 3
BoxDrawText(1, “0,310,1000,1000”, “Killing Button %Bwho%”, @TRUE, 1)
BoxButtonKill(1, bDraw3)
TimeDelay(2)
Break
endswitch
```

See Also:

BoxesUp, BoxNew, BoxButtonDraw, BoxButtonStat, BoxButtonWait

BoxButtonStat

Determines whether a push-button in a WinBatch box has been pressed.

Syntax:

BoxButtonStat(box ID, button ID)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(i) button ID	the ID number of the desired push-button.

Returns:

(i) **@TRUE** if the button has been pressed; **@FALSE** if it hasn't.

This function will also toggle the button back to "unpressed".

Example:

```
;;  sample script for BoxButtonStat
bDraw1=1
bDraw2=2

BoxesUp("200,200,700,700", @normal)
BoxDrawText(1, "0,310,1000,1000","Pick a Button", @FALSE, 1)
BoxButtonDraw(1, bDraw1, "Button 1", "200,464,450,558")
BoxButtonDraw(1, bDraw2, "Button 2", "550,464,800,558")
bWho=0
while bWho == 0
for x =1 to 2
if BoxButtonStat(1,x) then bWho=x
next
endwhile
Switch bWho
case 1
Display(3,"Button Example", "You pushed Button 1")
break
case 2
Display(3,"Button Example", "You pushed Button 2")
Break
endswitch
```

See Also:

BoxesUp, BoxNew, BoxButtonDraw, BoxButtonKill, BoxButtonWait

BoxButtonWait

Waits for any button in any box to be pressed.

Syntax:

BoxButtonWait( )

Returns:

(i) always 1.

This function will stay in a loop while all buttons are false. If any of the buttons are true when this command is issued, the command will not wait.

Example:

```
;;  sample script for BoxButtonWait
bDraw1=1
bDraw2=2
```

```
bWho=0

BoxesUp(“200,200,700,700”, @normal)
BoxDrawText(1, “0,310,1000,1000”,”Pick a Button”, @FALSE, 1)
BoxButtonDraw(1, bDraw1, “Button 1”, “200,464,450,558”)
BoxButtonDraw(1, bDraw2, “Button 2”, “550,464,800,558”)
BoxButtonWait()
for x =1 to 2
if BoxButtonStat(1,x) then bWho=x
next
Switch bWho
case 1
Display(3,”Button Example”, “You pushed Button 1”)
break
case 2
Display(3,”Button Example”, “You pushed Button 2”)
Break
endswitch
```

See Also:

BoxesUp, BoxNew, BoxButtonDraw, BoxButtonKill, BoxButtonStat

BoxCaption

Changes the title of a WinBatch box.

Syntax:

BoxCaption(box ID, caption)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) caption	title for the box.

Returns:

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

This function sets the title of the Window. The main window always has a title (caption) bar. Windows created with the **BoxNew** function, using a “2” for the style parameter, also have a caption bar. If the box does not have a caption bar, the function is effectively ignored.

Example:

```
;;  sample script for BoxCaption
BoxesUp(“200,200,700,700”, @normal)
BoxDrawText(1,”0,310,1000,1000”,”Watch the Title Bar”, @FALSE, 1)
BoxCaption(1, “WinBatch BoxCaption Example”)
TimeDelay(5)
BoxCaption(1, “Change the title to whatever you like”)
TimeDelay(3)
BoxCaption(1, “You have the power”)
```

```
TimeDelay(3)
```

See Also:

BoxesUp, BoxNew

BoxColor

Sets the background color for use with a WinBatch object.

Syntax:

BoxColor(box ID, color, wash color)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) normal color	the background color, a string in the form: “red, green, blue”.
(i) wash color	color used to create a background gradient effect.

Returns:

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

Sets the background color for use with a WinBatch object, either a rectangle, a circle, or a line.

If a gradient effect is not desired, specify “0” for “wash color”. If “wash color” is “0”, or if a 16-color video driver is installed, then “ normal color” will be used. Default is white, no wash.

Normal Color

BLACK=”0,0,0”	DKGRAY=”192,192,192”
WHITE=”255,255,255”	GRAY=”128,128,128”
RED=”255,0,0”	DKRED=”128,0,0”
GREEN=”0,255,0”	DKGREEN=”0,128,0”
BLUE=”0,0,255”	DKBLUE=”0,0,128”
PURPLE=”255,0,255”	DKPURPLE=”128,0,128”
YELLOW=”255,255,0”	DKYELLOW=”128,128,0”
CYAN=”0,255,255”	DKCYAN=”0,128,128”

Wash color

0	No Wash
1	Red
2	Green
3	Yellow
4	Blue

5	Magenta
6	Cyan
7	White

Example:

```
; sample code for various wash colors
BoxesUp(“0,0,1000,1000”, @zoomed)
for i=1 to 7
BoxColor(1,”255,0,0”,i)   ;sets the background color
BoxDrawRect(1,”0,0,1000,1000”,2);object that will use the color
Message(“Wash Code”,i)
next
```

See Also:

BoxesUp, BoxNew, BoxPen, BoxTextColor

BoxDestroy

Removes a WinBatch box.

Syntax:

BoxDestroy(box ID)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.

Returns:

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

Removes a WinBatch box and any buttons in the box from the screen. If you specify a box ID of 1, all boxes vanish.

Example:

```
;; sample script for BoxDestroy
BoxesUp(“0,0,1000,1000”, @normal)
BoxDrawText(1, “0,700,1000,1000”, “BoxDestroy”, @FALSE, 1)
BoxCaption(1, “WinBatch BoxDestroy Example  Box 1”)
BoxNew(2,”30,41,310,365”,  1)
BoxDrawText(2, “0,500,1000,1000”, “Box 2”, @TRUE, 1)
BoxNew(3,”330,41,610,365”, 1)
BoxDrawText(3, “0,500,1000,1000”, “Box 3”, @TRUE, 1)
BoxNew(4,”639,41,919,365”,  2)
BoxDrawText(4, “0,500,1000,1000”, “Box 4”, @TRUE, 1)
for i=2 to 4
Message(“BoxDestroy”, “Destroying Box Number %i%”)
BoxDestroy(i)
next
```

See Also:

BoxesUp, BoxNew

BoxDrawCircle

Draws an ellipse in a WinBatch box.

Syntax:

BoxDrawCircle(box ID, coordinates, style)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) coordinates	dimensions of circle, in virtual units (upper-x upper-y lower-x lower-y).
(i) style	style of circle to be drawn.

Returns:

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

Draws an ellipse on the screen using the current **BoxPen** for the outline, and the current **BoxColor** for the inside of the box.

Style:

0	empty circle with border
1	filled circle with border
2	filled circle with no border

Example:

```
;; sample script for BoxDrawCircle
BoxesUp(“0,0,1000,1000”, @normal)
BoxColor(1,”0,0,255”,4)
BoxDrawText(1,”0,500,1000,1000”,”BoxDrawCircle”, @FALSE, 1)
BoxCaption(1, “WinBatch BoxDrawCircle Example”)

BoxDrawCircle(1, “30,41,310,365”, 0)
BoxDrawText(1,”30,381,310,400”,”Style 0-empty with border”,@FALSE,1)

BoxDrawCircle(1, “330,41,610,365”, 1)
BoxDrawText(1, “330,381,610,400”, “Style 1-filled with border “,@FALSE,1)

BoxColor(1,”255,0,0”,4)
BoxDrawCircle(1, “639,41,919,365”, 2)
BoxDrawText(1,”639,381,919,400”,”Style 2-filled with no border”,@FALSE,1)
Delay(5)
```

See Also:

BoxesUp, BoxNew, BoxDrawLine, BoxDrawRect, BoxDrawText

BoxDrawLine

Draws a line in a WinBatch box.

Syntax:

BoxDrawLine(box ID, coordinates)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) coordinates	starting and ending points for a line, in virtual units (start-x, start-y, end-x, end-y).

Returns:

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

Draws a line from first point to the second using the current **BoxPen**.

Example:

```
;; sample script for BoxDrawLine
BoxesUp(“100,100,800,800”, @normal)
BoxDrawText(1,”0,600,1000,1000”,”BoxDrawLine”, @FALSE, 1)
BoxCaption(1, “WinBatch BoxDrawLine Example”)
co1=200
co2=200
co3=500
co4=500

For i=1 to 5
TimeDelay(1)
BoxDrawLine(1,”%co1%,%co2%,%co3%,%co4%”)
co1=co1+10
co2=co2+-20
co3=co3+-5
co4=co4+15
next
TimeDelay(2)
```

See Also:

BoxesUp, BoxNew, BoxDrawCircle, BoxDrawRect, BoxDrawText

BoxDrawRect

Draws a rectangle in a WinBatch box.

Syntax:

BoxDrawRect(box ID, coordinates, style)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) coordinates	dimensions of rectangle, in virtual units (upper-x upper-y lower-x lower-y).
(i) style	style of rectangle to be drawn.

Returns:

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

Draws a rectangle on the screen using the current **BoxPen** for the outline, and the current **BoxColor** for the inside of the box.

Style:

0	empty rectangle with border
1	filled rectangle with border
2	filled rectangle with no border

Example:

```
;; sample script for BoxDrawRect
BoxesUp(“0,0,1000,1000”, @normal)
BoxColor(1,”255,0,0”,0)
BoxDrawText(1, “0,900,1000,1000”,”BoxDrawRect”, @FALSE, 1)
BoxCaption(1, “WinBatch BoxDrawRect Example”)
BoxDrawRect(1, “30,41,310,465”, 0)
BoxDrawText(1,”30,500,310,665”,”Style 0-empty with border”,@FALSE,1)
BoxDrawRect(1, “330,41,610,365”, 1)
BoxDrawText(1, “330,381,610,365”,”Style 1-filled with border”, @FALSE, 1)
BoxColor(1,”0,0,255”,0)
BoxDrawRect(1, “696,114,839,841”, 2)
BoxDrawText(1,”696,881,839,841”,”Style 2-filled with no border”,@FALSE,1)
Delay(5)
```

See Also:

BoxesUp, BoxNew, BoxDrawCircle, BoxDrawLine, BoxDrawText

BoxDrawText

Displays text in a WinBatch box.

Syntax:

BoxDrawText(box ID, coordinates, text, erase flag, alignment)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) coordinates	dimensions of bounding rectangle for text, in virtual units (upper-x upper-y lower-x lower-y).
(s) text	text to be displayed.
(i) erase flag	**@TRUE** if background should be cleared; **@FALSE** if it should not be cleared.
(i) alignment	alignment mode for text.

Returns:

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

Draws text on the screen using the current **BoxTextColor** and **BoxTextFont**. Text may extend beyond the box boundaries if the allotted space is exceeded or size of the text is too large.

Note: Inorder to update text, make sure the proper coordinates are specified for the given text.

Alignment is a bitmask, consisting of one or more of the following optional flags (OR'ed together):

0	left justified
1	centered horizontally
2	right-justified
4	centered vertically
8	bottom-justified (single line only)
16	wrap long lines
32	adjust font so that text fills width of bounding rectangle (single line only)
64	right-justify text by adding space between words
128	clip (truncate) text if it doesn't fit within specified rectangle

Example:

```
;; sample code for BoxDrawText
BoxesUp("200,200,800,800", @normal)
BoxDrawText(1, "200,200,750,250", "WinBatch Box Example - BoxDrawText
",@TRUE, 0)
BoxCaption(1, "WinBatch BoxDrawText Example")
BoxDrawText(1, "100,350,900,400", "Use BoxDrawText to display information
to your user's. ", @TRUE, 0)
TimeDelay(5)
```

See Also:

BoxesUp, BoxNew, BoxDrawCircle, BoxDrawLine, BoxDrawRect

BoxesUp

Displays WinBatch boxes.

Syntax:

BoxesUp(coordinates, show mode)

Parameters:

(s) coordinates	window coordinates for placement of top-level WinBatch box, in virtual units (upper-x upper-y lower-x lower-y).
(i) show mode	**@NORMAL**, **@ICON**, **@ZOOMED**, or **@HIDDEN**.

Returns:

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

Places a WinBatch box on the screen for which drawing tools can be defined. “Coordinates” specify the placement on the screen when the window is not zoomed (maximized). The “box ID” of this main box (window) is 1. Up to 7 more boxes (windows) may be defined with the **BoxNew** function.

Note: Drawing tool definitions and drawing commands refer to a particular “box ID”. Different drawing tools can be defined for separate boxes.

Example:

```
;; sample script for BoxesUp
Message(“Example”,”BoxesUp can display a box in Normal Mode. “)
BoxesUp(“200,200,800,800”, @normal)
BoxDrawText(1,”500,200,500,200”,”BoxesUp %@crlf% Normal Mode”, @FALSE, 1)
BoxCaption(1, “Normal Mode”)
Message(“Example”,”BoxesUp can display the box as an Icon.”)
BoxDestroy(1)
BoxesUp(“200,200,800,800”, @icon)
BoxDrawText(1,”500,200,500,200”,”BoxesUp %@crlf% Icon Mode”, @FALSE, 1)
BoxCaption(1, “Example - Icon Mode”)
Message(“Example”, “BoxesUp can display in a Zoomed mode.”)
BoxDestroy(1)
```

```
BoxesUp(“200,200,800,800”, @zoomed)
BoxDrawText(1,”500,200,500,200”,”BoxesUp %@crlf% Zoomed Mode”, @FALSE, 1)
BoxCaption(1, “Zoomed Mode”)
Message(“Example”,”Finally, WinBatch can set hidden mode to the box.”)
```

See Also:

BoxNew

BoxMapMode

Sets the mapping mode for a WinBatch box.

Syntax:

BoxMapMode(box ID, map mode)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(i) map mode	**@ON** to map coordinates to client scale (default). One Unit is 1/1000 (or 0.1%) of the size of the current box. **@OFF** for screen scale. One unit is 1/1000 (or 0.1%) of the size of the screen.

Returns:

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

BoxMapMode defines how a function's “coordinate” parameters will be interpreted. The default setting, @ON, allows WinBatch boxes to automatically resize themselves per the user’s monitor adjustments. In the default “mapping” mode each window is assumed to be 1000x1000. This makes it easy to write a WinBatch program that will run on anybody’s screen.

Note: The Default setting is **highly** recommended.

Example:

```
;; sample script for BoxMapMode
IntControl(12,5,0,0,0)
title=”BoxMapMode Example”
BoxesUp(“100,100,900,900”,@ZOOMED)

BoxMapMode(1,1)   ; Default map mode
BoxColor(1,”255,255,0”,0)
BoxPen(1,”0,0,255”,10)
BoxTextFont(1, “”, 30, 0, 0)
BoxTextColor(1,”0,0,0”)

BoxDrawRect(1,”50,50,150,150”,1)
BoxDrawCircle(1,”200,50,350,150”,1)
```

```
BoxDrawLine(1,"400,100,500,100")
BoxDrawLine(1,"450,50,450,150")
BoxDrawText(1, "50,160,500,190", "Map Mode = 1 Using sizes based on
window", 0, 0)
BoxMapMode(1,0)
BoxColor(1,"255,255,0",0)
BoxPen(1,"0,0,255",10)
BoxTextFont(1, "", 30, 0, 0)

BoxDrawRect(1,"50,200,150,300",1)
BoxDrawCircle(1,"200,200,350,300",1)
BoxDrawLine(1,"400,250,500,250")
BoxDrawLine(1,"450,200,450,300")
BoxDrawText(1, "50,310,500,340", "Map Mode = 0 Using sizes based on
screen", 0, 0)
Message(title,"Note that both sets of objects look pretty much the
same.")
WinPlace(0,0,750,750,"")
Message(title,"Note that when we changed the size of the window the
       MapMode=1 object were resized proportionally, whileas the
       MapMode=0 objects stayed the same.")
WinPlace(0,0,500,500,"")
Message(title,"MapMode=1 objects resized again.")
WinPlace(0,0,200,1000,"")
Message(title,"Note that while most objects scale reasonably well,
       fonts are based on Window height.")
WinPlace(0,0,1000,200,"")
Message(title,"Giving us teeny tiny fonts in this sort of Window.")
WinPlace(50,50,950,950,"")
BoxMapMode(1,1)   ; Default map mode
BoxTextFont(1, "", 30, 0, 0)
BoxTextColor(1,"255,0,0")
BoxDrawText(1,"50,500,500,700","Resize the window with the mouse and
       watch what happens.  Hit ESC when you are done. (This message
       drawn with MapMode=1)",0,16)
WaitForKey("{ESC}","","","","")
```

See Also:
BoxesUp, BoxNew

BoxNew
Creates a WinBatch box.

Syntax:
BoxNew(box ID, coordinates, style)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.

(s) coordinates	dimensions of box, in virtual units (upper-x upper-y lower-x lower-y).
(i) style	style of box to create.

Returns:

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

This function makes a new box inside the top level (box ID 1) box. If an existing box ID is used, the newly specified coordinates and style will be adopted.

Style allows a selection from three different kinds of boxes.

0	No border
1	Border
2	Border and caption

Example:

```
;; sample script for BoxNew
BoxesUp(“0,0,1000,1000”, @normal)
BoxDrawText(1, “500,500,500,500”, “BoxNew “, @FALSE, 1)
BoxCaption(1, “WinBatch BoxNew Example”)
BoxColor(1,”255,255,0”,0)
BoxDrawRect( 1, “0,0,1000,1000”, 2)

BoxNew(2, “30,41,310,465”, 0)
BoxDrawText(1, “30,681,310,665”,”Style 0-No border “,@FALSE,1)

BoxNew(3, “330,41,610,365”, 1)
BoxDrawText(1, “330,381,610,365”,”Style 1-Border “,@FALSE,1)

BoxNew(4, “696,114,839,841”, 2)
BoxDrawText(1, “696,881,839,841”,”Style 2-Border with caption “,@FALSE,1)
BoxCaption(4, “Style 2 BoxNew”)
Delay(7)
```

See Also:

BoxesUp

BoxPen

Sets the pen for a WinBatch box.

Syntax:

BoxPen(box ID, color, width)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.

(s) color	color of pen to use.
(i) width	width of pen to use, in virtual units.

Returns:

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

Defines the color and width of a “pen”. Pens are used to draw lines and borders of rectangles and ellipses. The default is black, 1 pixel wide.

Width is defined according to the current mapping mode, (see BoxMapMode). In the default mapping mode, a width of 10 is 1% of whichever is smaller, the width or the height of the box.

“Color” is a string in the form: “red, green, blue”.

BLACK=”0,0,0”	DKGRAY=”192,192,192”
WHITE=”255,255,255”	GRAY=”128,128,128”
RED=”255,0,0”	DKRED=”128,0,0”
GREEN=”0,255,0”	DKGREEN=”0,128,0”
BLUE=”0,0,255”	DKBLUE=”0,0,128”
PURPLE=”255,0,255”	DKPURPLE=”128,0,128”
YELLOW=”255,255,0”	DKYELLOW=”128,128,0”
CYAN=”0,255,255”	DKCYAN=’0,128,128”

Example:

```
;; sample script for BoxPen
BoxesUp(“100,100,900,900”, @normal)
BoxColor(1,”255,255,0”,0)
BoxDrawRect( 1, “0,0,1000,1000”, 2)
BoxDrawText(1, “0,200,1000,1000”, “BoxPen “, @FALSE, 1)
BoxCaption(1, “WinBatch BoxPen Example”)
BoxColor(1,”0,0,255”, 0)
BoxPen(1,”255,0,0”,25)
BoxDrawRect(1,”350,350,650,650”, 1)
BoxDrawLine(1, “350,700,800,700”)
TimeDelay(5)
```

See Also:

BoxesUp, BoxNew , BoxColor, BoxTextColor

BoxTextColor

Sets the text color for a WinBatch box.

Syntax:

BoxTextColor(box ID, color)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) color	text color.

Returns:

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

BoxTextColor defines the color of text for a particular box. The default is black.

“Color” is a string in the form: “red, green, blue”.

BLACK=”0,0,0”	DKGRAY=”192,192,192”
WHITE=”255,255,255”	GRAY=”128,128,128”
RED=”255,0,0”	DKRED=”128,0,0”
GREEN=”0,255,0”	DKGREEN=”0,128,0”
BLUE=”0,0,255”	DKBLUE=”0,0,128”
PURPLE=”255,0,255”	DKPURPLE=”128,0,128”
YELLOW=”255,255,0”	DKYELLOW=”128,128,0”
CYAN=”0,255,255”	DKCYAN=”0,128,128”

Example:

```
;; sample script for BoxTextColor
BoxesUp(“200,200,800,800”, @normal)
BoxCaption(1, “WinBatch BoxTextColor Example”)
x1=”0,0,0”          ;BLACK
x2=”0,0,128”        ;DKBLUE
x3=”255,0,0”        ;RED
x4=”0,255,0”        ;GREEN
x5=”255,0,255”      ;PURPLE
x6=”255,255,0”      ;YELLOW
x7=”0,255,255”      ;CYAN
for i=1 to 7
      BoxTextColor(1,x%i%)
      BoxDrawText(1, “0,350,1000,1000”, “BoxTextColor”, @True, 1)
      delay(2)
next
```

See Also:

BoxesUp, BoxNew, BoxTextFont, BoxColor, BoxPen

BoxTextFont

Sets the font for a WinBatch box.

Syntax:

BoxTextFont(box ID, font-name, size, style, pitch & family)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) font-name	name of font typeface.
(i) size	size of font, in virtual units.
(i) style	style flags for font.
(i) pitch & family	font pitch and family.

Returns:

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

When defining the font using **BoxTextFont**, size is based on mapping mode. In the default, a height of 100 is 10% of the height of the box.

Style: (the following numbers may be added together)

0	Default.
1-99	Weight (40 = Normal, 70 = Bold)
100	Italics
1000	Underlined

A style of 1170 give you a bold, underlined, italic font.

Pitch & Family parameters do not override the typeface supplied in the font-name parameter. If a match cannot be made, (font name mis-spelled, font not on system) they supply a general description for selecting a default font. To combine one pitch flag with one family flag, use the binary OR (“|”) operator.

Pitch:

0	Default
1	Fixed pitch
2	Variable pitch

Family:

0	Default
16	Roman (Times Roman, Century Schoolbook, etc.)
32	Swiss (Helvetica, Swiss, etc.)
48	Modern (Pica, Elite, Courier, etc.)
64	Script
80	Decorative (Old English, etc.)

Example:

```
;; sample script for BoxTextFont
BoxesUp(“100,100,900,900”, @normal)
BoxCaption(1, “WinBatch BoxTextFont Example”)
x1=”0,0,0”          ;BLACK
x2=”0,0,128”        ;DKBLUE
x3=”255,0,0”        ;RED
x4=”255,0,255”     ;PURPLE
x5=”0,0,255”        ;BLUE
f1=”Times Roman”
f2=”Helvetica”
f3=”Courier New”
f4=”Brush Script MT”
f5=”Book Antiqua”
fam=16
size=20

for i=1 to 5
BoxTextColor(1,x%i%)
BoxTextFont(1, f%i%, size, 0, fam)
BoxDrawText(1,”1%size%,2%size%,1000,1000”,”BoxTextFont”, @False, 0)
Fam=fam+16
size=size+16
TimeDelay(2)
next
```

See Also:
BoxesUp, BoxNew, BoxTextColor

BoxUpdates
Sets the update mode for, and/or updates, a WinBatch box.

Syntax:
BoxUpdates(box ID, update flag)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(i) update flag	see below.

Returns:

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

BoxUpdates controls how particular boxes are updated. Screen updates can be suppressed so that images seem to suddenly appear on the screen, rather than slowly form as they are drawn. This function is rarely required.

Update flag:
0 Suppress screen updates

1	Enable updates (this is the default setting)
2	Catch up on updates
3	Redraw the entire box

Example:

```
title=”BoxUpdates Example”
BoxesUp(“100,100,900,900”,@ZOOMED)
BoxColor(1,”255,255,0”,0)
BoxDrawRect(1,”0,0,1000,1000”,2)
BoxCaption(1,title)
BoxDataTag(1,”NEARTOP”)
Message(title,”First we show drawing objects with the default(code=1)
       mode of BoxUpdates”)

gosub drawalot
Message(title,”You could see the objects being drawn.  Ok, but users
       could see objects being built.  Next we are clearing the screen
       with a BoxDataClear and redrawing it with a BoxUpdates code=3”)
BoxDataClear(1,”NEARTOP”)
BoxUpdates(1,3)

BoxUpdates(1,0)
gosub drawalot
Message(title,”Next we show update off processing followed by a
       catch-up (code = 2) request.  Note that it draws faster once it
       gets started”)
BoxUpdates(1,2)
Message(title,’Faster.  It can make complicated objects just “appear”
       on the screen.’)
Message(title,”Now, we are going to redraw the screen with BoxUpdates
code=3.  Should be quick.  Don’t blink.”)
BoxUpdates(1,3)
Message(title,”That should have been pretty quick.  Next some quick,
       repetitive drawing using the code=3 technique.”)
BoxUpdates(1,1)
BoxColor(1,”255,255,255”,0)
BoxDrawRect(1,”0,0,1000,1000”,2)
BoxDataClear(1,”TOP”)
BoxUpdates(1,0)
BoxColor(1,”255,0,0”,0)
BoxDrawRect(1,”100,100,200,200”,1)
BoxDrawCircle(1,”300,100,500,200”,1)

BoxDrawRect(1,”100,300,200,400”,1)
BoxDrawCircle(1,”300,300,500,400”,1)
BoxDrawRect(1,”100,500,200,600”,1)
BoxDrawCircle(1,”300,500,500,600”,1)
BoxDrawRect(1,”100,700,200,800”,1)
BoxDrawCircle(1,”300,700,500,800”,1)
```

```
BoxColor(1,”0,0,255”,0)
BoxDrawRect(1,”100,100,200,200”,1)
BoxDrawCircle(1,”300,100,500,200”,1)
BoxDrawRect(1,”100,300,200,400”,1)
BoxDrawCircle(1,”300,300,500,400”,1)
BoxDrawRect(1,”100,500,200,600”,1)
BoxDrawCircle(1,”300,500,500,600”,1)
BoxDrawRect(1,”100,700,200,800”,1)
BoxDrawCircle(1,”300,700,500,800”,1)

BoxColor(1,”0,255,0”,0)
BoxDrawRect(1,”100,100,200,200”,1)
BoxDrawCircle(1,”300,100,500,200”,1)
BoxDrawRect(1,”100,300,200,400”,1)
BoxDrawCircle(1,”300,300,500,400”,1)
BoxDrawRect(1,”100,500,200,600”,1)
BoxDrawCircle(1,”300,500,500,600”,1)
BoxDrawRect(1,”100,700,200,800”,1)
BoxDrawCircle(1,”300,700,500,800”,1)

BoxColor(1,”255,255,0”,0)
BoxDrawRect(1,”100,100,200,200”,1)
BoxDrawCircle(1,”300,100,500,200”,1)
BoxDrawRect(1,”100,300,200,400”,1)
BoxDrawCircle(1,”300,300,500,400”,1)
BoxDrawRect(1,”100,500,200,600”,1)
BoxDrawCircle(1,”300,500,500,600”,1)
BoxDrawRect(1,”100,700,200,800”,1)
BoxDrawCircle(1,”300,700,500,800”,1)

BoxUpdates(1,2)
for x=1 to 100
BoxUpdates(1,3)
next
Message(title,”That’s all folks”)
exit

:DRAWALOT
BoxColor(1,”0,0,255”,0)
BoxPen(1,”255,0,0”,10)
for i=0 to 8
p1=50+i*100
p2=p1+75
BoxDrawRect(1,”%p1%,50,%p2%,125”,1)
BoxDrawRect(1,”%p1%,150,%p2%,225”,1)
BoxDrawRect(1,”%p1%,250,%p2%,325”,1)
BoxDrawRect(1,”%p1%,350,%p2%,425”,1)
BoxDrawRect(1,”%p1%,450,%p2%,525”,1)
BoxDrawRect(1,”%p1%,550,%p2%,625”,1)
BoxDrawRect(1,”%p1%,650,%p2%,725”,1)
BoxDrawRect(1,”%p1%,750,%p2%,825”,1)
BoxDrawRect(1,”%p1%,850,%p2%,925”,1)
next
```

```
return
```

See Also:

BoxesUp, BoxNew

Drawing Stack Management

In general, WinBatch lets you draw objects in various boxes using simple linear programming as with true message-based Windows programming. However, there is a fundamental discrepancy between the message-based Windows programming methods, and the traditional linear method used by WinBatch.

In a normal Windows application, the application must be ready to redraw all or any portion of its window at any time. This adds considerable complexity to a true Windows program. In WinBatch, the programmer is shielded from the gory details of the dynamic redrawing required by Windows, while WinBatch maintains a simple, traditional linear programming style.

In order to do this, WinBatch maintains a small database of the Box commands requested by the programmer, and refers to this database when Windows requests a redraw. In general, and for simpler applications, the existence of this database is completely transparent to the programmer. There are cases, however, in which the database must be managed by the programmer to avoid reaching the maximum limits of the database. If the maximum limits are reached, the program will die with a Box Stack exceeded error.

If there are some objects that constantly change, such that the limit of about 150 Box commands in the stack will be exceeded, then you must manage the Box Data. The idea is to draw all the fixed, non-changing objects first, and then place a “TAG” into the Data stack. Then draw the first version of the constantly changing object(s). When it comes time to update those objects, a **BoxDataClear** will erase all items below the “TAG”, and all remaining data space will again be available for reuse.

The thermometer bar and the text for the note in the setup program use this feature. All of the examples that do continuous screen draws also use these functions

BoxDataClear

Removes commands from a WinBatch box command stack.

Syntax:

BoxDataClear(box ID, tag)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) tag	tag to be removed.

Returns:

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

This function removes all commands above “tag” from the command stack. “Tag” is not removed.

All buttons and Box commands after the tag are forever erased.

Example:

```
;; sample script for BoxDataClear
BoxesUp(“100,100,900,900”, @normal)
BoxColor(1,”255,255,0”,0)
BoxDrawRect( 1, “0,0,1000,1000”, 2)
BoxDataTag(1, “tag1”);Insert the TAG before the varying text element
BoxDrawText(1,”0,200,1000,1000”,”BoxDataClear”,@FALSE, 1)
BoxCaption(1, “WinBatch BoxDataClear Example”)

BoxColor(1,”0,0,255”, 0)
BoxPen(1,”255,0,0”,25)
BoxDrawRect(1,”350,350,650,650”, 1)
BoxDrawLine(1, “350,700,800,700”)
TimeDelay(2)

BoxDataClear(1, “tag1”);set this before the next varying text.
BoxDrawText(1, “0,240,1000,1000”, “BoxDataClear-Clearing Tags to
            redraw contents”, @FALSE, 1)
TimeDelay(3)
BoxUpdates(1,3); this illustrates the effect of clearing a tag
BoxColor(1,”255,0,0”, 0)
BoxPen(1,”0,0,255”,50)
BoxDrawRect(1,”350,350,650,650”, 1)
BoxDrawLine(1, “350,700,800,700”)
TimeDelay(4)
```

See Also:

BoxesUp, BoxNew BoxDataTag

BoxDataTag

Creates a tag entry in a WinBatch box command stack.

Syntax:

BoxDataTag(box ID, tag)

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) tag	tag to be created.

Returns:

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

Places a tag into the data stack for the specified box. Usually one tag per box is all that is needed. Multiple tags are allowed, but not advised. The tag “TOP” is automatically placed at the top of the data stack .

Example:

```
;; sample script for BoxDataTag
BoxesUp(“100,100,900,900”, @normal)
BoxColor(1,”255,255,0”,0)
BoxDrawRect( 1, “0,0,1000,1000”, 2)
BoxDataTag(1, “tag1”);Insert the TAG before the varying text element
BoxDrawText(1,”0,200,1000,1000”,”BoxDataTag”, @FALSE, 1)
BoxCaption(1, “WinBatch BoxDataTag Example”)
BoxColor(1,”0,0,255”, 0)
BoxPen(1,”255,0,0”,25)
BoxDrawRect(1,”350,350,650,650”, 1)
BoxDrawLine(1, “350,700,800,700”)
TimeDelay(2)
BoxDataClear(1, “tag1”);set this before the next varying text.
BoxDrawText(1, “0,240,1000,1000”, “BoxDataTag - Clearing Tags to
            redraw contents”, @FALSE, 1)
TimeDelay(3)
BoxColor(1,”255,0,0”, 0)
BoxPen(1,”0,0,255”,50)
BoxDrawRect(1,”350,350,650,650”, 1)
BoxDrawLine(1, “350,700,800,700”)
TimeDelay(4)
```

See Also:

BoxesUp, BoxNew, BoxDataClear

Language Extenders

Network and other extenders are documented fully in the on-line help files. For more extensive information look there, for a brief overview, see below.

WIL extender DLLs are special DLLs designed to extend the built-in function set of the WIL processor. These DLLs typically add functions not provided in the basic WIL set, such as network commands for particular networks (Novell, Windows for WorkGroups, LAN Manager and others), MAPI, ODBC, and other important Application Program Interface (API) functions as may be defined by the various players in the computer industry from time to time. These DLLs may also include custom built function libraries either by the original authors, or by independent third party developers. (An Extender SDK is available) for you to write your own extender DLL's, if you are familiar with C programming.. Custom extender DLLs may add nearly any sort of function to the WIL language, from the mundane network math or database extensions, to items that can control fancy peripherals, including laboratory or manufacturing equipment.

WIL extenders must be installed separately; they are not installed by default. To obtain any additional extender DLLs, either download them from our website, or copy them from the WinBatch CD. To install extenders from the CD, locate the Extenders directory on the CD. The Extender directory will contain all the available extender subdirectories, select the appropriate extender subdirectory and run the corresponding SETUP.WBT file.

Up to 10 extender DLLs may be added to a single WIL script. The total number of added items may not exceed 100 functions and constants. The **AddExtender** function must be executed before attempting to use any functions in the extender library. The **AddExtender** function should be only executed once per extender in each WIL script that requires it.

At the top of each script in which you use a WIL extender, for example, network commands, add the appropriate extender with the AddExtender ( ) command.

```
  AddExtender("extender.dll")
```

Remember you can add up to 10 extender DLLs or a combined total of 100 functions.

Note: Extender Dlls must match the platform you’re running them under (i.e., 32-bit Dlls can only be run by a 32-bit version of WinBatch on a 32-bit platform.)

The following is an abbreviated summary of the network extenders. Refer to the extenders in the on-line help file for function names and more details.

Novell 3.x Network Extender

This extender provides standard support for Novell 3.x networks. It may be used in addition to other extenders, such as the Win32 Network extender.

In order to use any of the Novell extender DLLs, you must have the Novell NetWare Client installed.

Note: If you want to use any of the following commands you need to add the following line to the top of your script.

```
AddExtender(“wwn3x32i.dll”)
```

Other required Novell DLL’s: CALWIN32.DLL

This particular DLL, wwn3x32i.dll, is for use on 32-bit versions of Windows on Intel type processors. Your system may require the use of a different DLL.

See Filenames: Appendix A, page 85, for more information on DLL filenames.

For more information and a list of functions see the Netware 3 Extender Help file.

Novell 4.x Network Extender

This extender provides standard support for Novell 4.x networks. It may be used in addition to other extenders, such as the Win32 Network extender, and the Novell 3.x extender.

Note: There are certain differences in using Novell 4 over Novell 3. In Novell 4, you must login to the network before attaching to a File Server.

Note: If you want to use any of the following commands you need to add the following line to the top of your script.

```
AddExtender(“wwn4x32i.dll”)
```

Other required Novell DLL’s: CALWIN32.DLL, NETWIN32.DLL, LOCWIN32.DLL

This particular DLL, wwn4x32i.dll, is for use on 32-bit versions of Windows on Intel type processors. Your system may require the use of a different DLL.

See Filenames: Appendix A, page 85 for more information on DLL filenames.

For more information and a list of functions see the Netware 4 Extender Help file.

Basic Windows Network Extender (Novell, DEC Pathworks, Artisoft Lantastic)

This extender provides basic links to networks (like Novell, DEC Pathworks, Artisoft Lantastic and others that work from Windows) for the Windows 3.1 environment. It is **not** designed for Windows for Workgroups, Windows 95/98, or for other versions of Windows. There are alternate extenders available for those products. In addition, some networks, like Novell, have better, more fully featured extenders available.

Note: If you want to use any of the Basic Network commands you need to add the following line to the top of your script.

```
AddExtender("www3a16i.dll")
```

Additional DLLs required: NONE

This particular DLL, wwn3a16i.dll, is for use on 16-bit versions of Windows on Intel type processors. Your system may require the use of a different DLL.

See Filenames: Appendix A, page 85, for more information on DLL filenames.

For more information and a list of functions see the Basic Net Extender Help file.

Multinet, Windows for Workgroups Network Extender

This extender is designed for versions of Windows 3.xx containing the Microsoft Multinet network driver support. This includes Windows for Workgroups and newer versions of Windows. The commands in this package handle the Windows and Microsoft networks. It is designed to work in conjunction with other extenders for other networks, such as extenders for Novell networks.

Note: If you want to use any of the following commands you need to add the following line to the top of your script.

```
AddExtender("wwwn16i.dll")
```

Additional DLLs required: NONE

This particular DLL, wwwn16i.dll, is for use on 16-bit versions of Windows on Intel type processors. Your system may require the use of a different DLL.

See Filenames: Appendix A, page 85, for more information on DLL filenames.

For more information and a list of functions see the Multinet Extender Help file.

Windows 32 / Windows NT Network Extender

This extender provides standard support for computers running 32 bit versions of Windows, such as Windows 95/98/NT. It may be used in conjunction with other 32 bit Intel extenders.

This extender is only for 32 bit versions of Windows.

Basic DLLs:

32 Bit Intel Version
    AddExtender("wwnet32i.dll")

32 Bit DEC Alpha Version
    AddExtender("wwnet32d.dll")

32 Bit MIPS Version
    AddExtender("wwnet32m.dll")

32 Bit PowerPC Version
    AddExtender("wwnet32p.dll")

NT specific DLL:

For use on Windows NT workstations. Can control Windows NT servers.

32 Bit Intel Version
    AddExtender("wwwnt32i.dll")

95/98 specific DLLs:

For use on Windows 95/98 workstations. Can control Windows 95/98 servers.

32 Bit Intel Version
    AddExtender("www9532i.dll")

9X specific DLLs:

For use on Windows 95/98 workstations. Can control Windows NT servers.

32 Bit Intel Version
    AddExtender("www9x32i.dll")

For more information and a list of functions see the Win32 Network Extender Help file.

UTILITIES

DIALOG EDITOR

The WIL Dialog Editor (see Filenames: Appendix A, page 85 for filename) provides a convenient method of creating dialog box templates for use with the **Dialog** function.

Visual programming of dialog boxes is quick and accurate. Use generic variable names so you can reuse your favorite dialogs.

It displays a graphical representation of a dialog box, and allows you to create, modify, and move individual controls which appear in the dialog box.

After you have defined your dialog box, the Dialog Editor will generate the appropriate WIL code, which you can save to a file or copy to the Clipboard for pasting into your WIL program.

Note: The WIL Dialog Editor comes with an on-line help section in the WinBatch help file, as well as detailed instructions in the next section.

Note: Another way to create graphical dialogs, is to use the HTML Dialog Extender. For further information see the HTML Dialog Extender help file. Available on the CD-ROM, or downloadable from our website.

You can have as many as 100 controls in a WinBatch dialog. However, too many controls can be confusing. Aim for simple dialogs with a consistent appearance between different ones.

The WIL Dialog Editor offers quick production of custom dialog boxes for your WinBatch programs.

The WIL Dialog Editor allows you to create dialog box templates for WIL into a file with the .WDL file extension. The Dialog Editor will write the WIL script statements necessary to create and display the dialog.

You can visually design your dialog box on the screen and then save the template either as a .WDL file or the Windows Clipboard.

You can include the dialog template code directly in your batch code, or you can use the batch language "Call" command to execute the dialog template. For example:

```
Call("Sample.WDL", "")
```

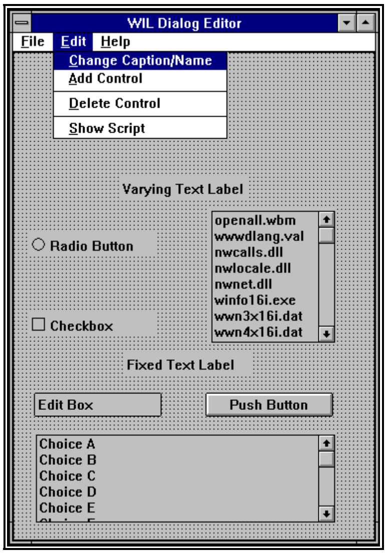
Getting Started

Using the Dialog Editor is easy. Once it is loaded, these hints offer a quick way to become comfortable with dialog box construction.

The dialog editor filename for 16 bit Windows use is: wwd1g16i.exe.

Launch the dialog editor executable, (see Filenames: Appendix A on page 85 for filename).

The editor will look like the following:



To control the size of your dialog box, resize the WIL Dialog Editor, by dragging the corners of the box with the mouse. Your dialog will be the same size as the editor's window.

Menu Commands

There are three standard menus in this program: FILE, EDIT, and HELP.

File

New

When you select New, any currently loaded template will be discarded and the slate will be clean for a new dialog. You will be prompted to enter the caption (title) for your dialog box, and a WIL variable name used to refer to the dialog box in the WIL scripts.

Load

Loads a dialog template from a file.

Save

Saves a dialog template to the current file.

Save As

Saves the dialog template to a file using a different filename.

Load from Clipboard

Loads a dialog template from the Windows Clipboard.

Save to Clipboard

Saves the dialog template to the Windows Clipboard.

Edit

Change Caption/Name

Allows you to change the Dialog caption (title) and/or the variable name used to refer to the dialog.

Note: Left Mouse double-clicking the dialog box background will also execute this menu item.

Add Control

Adds a new control to your dialog template.

Note: Right Mouse double-clicking has the same effect.

Delete Control

Surprisingly enough, Delete Control does not actually delete a control. It just reminds you how to do it. To delete a control, position the mouse cursor over the control and press the delete key.

Show Script

Displays the WIL script generated during the dialog edit session. Once you learn how the dialog scripts operate, viewing the script is a quick way to scan for errors. You will notice that some script lines cannot be viewed in their entirety, in which case simply double click it to view the entire line.

Help

Dialog Editor Help

Displays the Index of the On-line help information.

How to use Help

Activates the Microsoft Windows Index to Using Help.

About

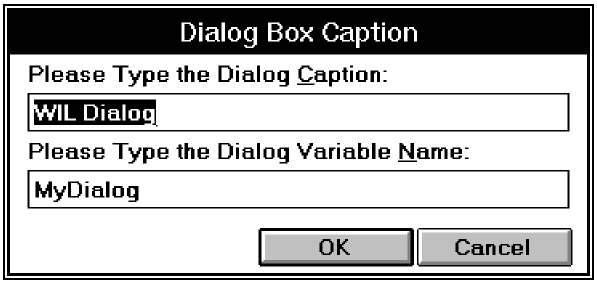
Displays the WIL Dialog Editor About dialog which includes the version number of the program.

User Interface

Here are the techniques for designing your titles and controls.

The Caption Box

Double click with the left mouse button on the workspace background to display the caption box.


Dialog Box Caption

Please Type the Dialog Caption: WIL Dialog

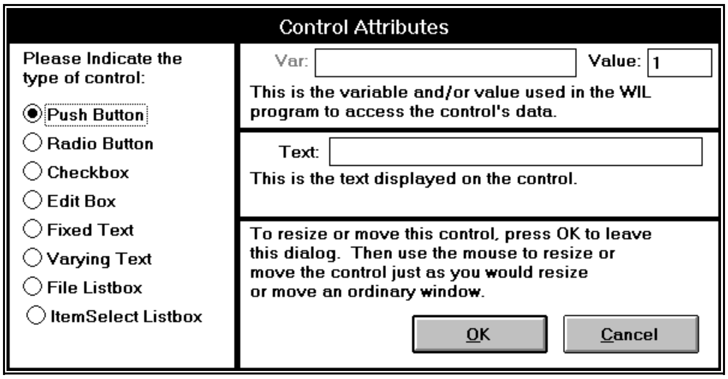
Please Type the Dialog Variable Name: MyDialog

The **Dialog Caption** is the title of the dialog box as it appears in the title bar. The **Variable Name** is the name of the dialog as seen in the script.

This information can be entered or changed at any time. However, we suggest filling it whenever you start a new dialog box. To change the caption, double click on the workspace (not on a control) with the left mouse button.

Setting Control Attributes

To add a control, double click with the right mouse button where you want the control. Fill in the information in the resulting "Control Attributes" dialog box.


Control Attributes

Please Indicate the type of control:

(●) Push Button
( ) Radio Button
( ) Checkbox
( ) Edit Box
( ) Fixed Text
( ) Varying Text
( ) File Listbox
( ) ItemSelect Listbox

Var: [ ] Value: [1]

This is the variable and/or value used in the WIL program to access the control's data.

Text: [ ]

This is the text displayed on the control.

To resize or move this control, press OK to leave this dialog. Then use the mouse to resize or move the control just as you would resize or move an ordinary window.

Choose the control on the left and fill in the appropriate attributes on the right. The control may need a **Variable** name, a **Value** or **Text**. Not all information will be needed for each control. Fill in only the items which are not grayed out. The following table is a quick reference of what attributes are required for each control.

Control	Variable	Value	Text
Push Button		✓	✓
Radio Button	✓	✓	✓
Check Box	✓	✓	✓
Edit Box	✓		✓
Fixed Text			✓
Varying Text	✓		✓
File Listbox	✓		
ItemSelect Listbox	✓		

To **MOVE** the control, click on it and drag it to a new position with the left mouse button.

To **SIZE** a control, click on the edge and drag with the left mouse button.

To **DELETE** a control, position the mouse cursor over the control and press the delete key.

Save

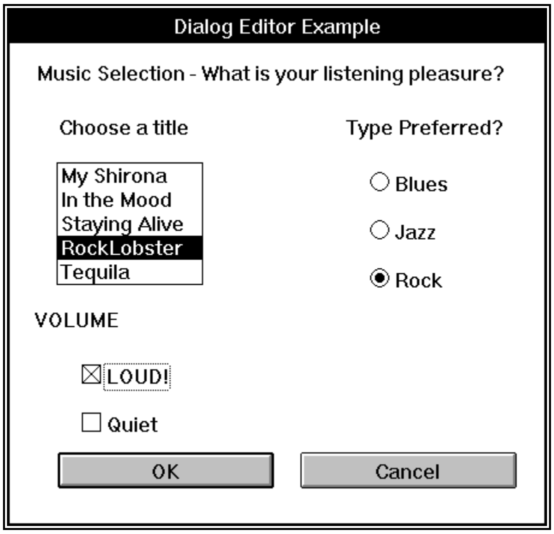
Once you are happy with your work, choose **“Save”** or **“SaveAs”** from the **File** menu to save your work to a file. Choose **“Save to Clipboard”** to put the work into the clipboard so that it can be easily pasted into one of your WIL scripts.

View the Script

Take a peek at the resulting script with the **File “Show Script**” command to see what WIL Dialog Scripts look like.

Analyze the Script

The Dialog Editor follows a specific format when creating your script. For example, here is a dialog box we created. We saved the script with the variable name “Example”.



The first line of a script sets the format and specifies the version of the Dialog Editor being used. As you can see in the example code below, the dialog variable name, in this case "Example" precedes all of the keywords.

```
ExampleFormat=`WWWDLGED,5.0`
```

The next section establishes the caption which will appear in the title bar of the dialog box along with the coordinates, size and number of controls in the dialog box.

```
ExampleCaption=`Dialog Editor Example`
ExampleX=120
ExampleY=50
ExampleWidth=179
ExampleHeight=160
ExampleNumControls=12
```

The third section contains the code for the actual controls. Each line has specific information. There will be one line for every control in the dialog box.

```
  Example01=`16,136,72,DEFAULT,PUSHBUTTON,DEFAULT,”OK”,1`
  Example02=`97,136,72,DEFAULT,PUSHBUTTON,DEFAULT,”Cancel”,0`
  Example03=`120,40,48,DEFAULT,RADIOBUTTON,music,”Blues”,1`
  Example04=`120,56,56,DEFAULT,RADIOBUTTON,music,”Jazz”,2`
  Example05=`120,72,56,DEFAULT,RADIOBUTTON,music,”Rock”,3`
```

The table below shows what the first line (Example01...) means.

Code	Definition
Example	Dialog Variable Name
01	Control Number
16,136,72,DEFAULT	Coordinates of the control
PUSHBUTTON	Control Type
DEFAULT,	Variable name
OK	Text
1	Value

Each Dialog script will end with the following line, making it easy to test the push-button return values.

```
ButtonPushed=Dialog(“Example”)
```

The variable "ButtonPushed" will be equal to the value of whichever button was pushed by the user. So in the example below, if ButtonPushed == 1, then the user pushed the OK button. If ButtonPushed == 0, then the user pushed the cancel button.

Put all the parts together and the completed script looks like the following.

```
ExampleFormat=`WWWDLGED,5.0`
ExampleCaption=`Dialog Editor Example`
ExampleX=120
ExampleY=50
ExampleWidth=179
ExampleHeight=160
ExampleNumControls=12

Example01=`16,136,72,DEFAULT,PUSHBUTTON,DEFAULT,”OK”,1`
Example02=`97,136,72,DEFAULT,PUSHBUTTON,DEFAULT,”Cancel”,0`
Example03=`120,40,48,DEFAULT,RADIOBUTTON,music,”Blues”,1`
Example04=`120,56,56,DEFAULT,RADIOBUTTON,music,”Jazz”,2`
```

```
Example05=`120,72,56,DEFAULT,RADIOBUTTON,music,"Rock",3`
Example06=`24,104,112,DEFAULT,CHECKBOX,volume,"LOUD!",1`
Example07=`24,120,104,DEFAULT,CHECKBOX,volume2,"Quiet",2`
Example08=`8,88,64,DEFAULT,STATICTEXT,DEFAULT,"VOLUME"`
Example09=`9,6,164,DEFAULT,STATICTEXT,DEFAULT,"Music Selection
-  What is your listening pleasure?"`
Example10=`16,40,48,40,ITEMBOX,tunes,DEFAULT`
Example11=`112,24,56,DEFAULT,STATICTEXT,DEFAULT,"Type
   Preferred?"`
Example12=`16,24,49,DEFAULT,VARYTEXT,song,"Choose a title"`

ButtonPushed=Dialog("Example")
```

Note: The songs that appear in the ItemSelect Listbox are listed earlier in the script on one continuous line as the variable, *tunes*. i.e.

```
tunes="My Shirona%@tab%In the Mood%@tab%Staying Alive%@tab%
RockLobster%@tab%Tequila"
```

Variables can be defined above the dialog script or in another WBT file above the statement which calls the dialog file.

Control Attribute Specifics

Some of the Controls require extra knowledge or special handling, as in the following:

Setting Variables

Any information which is needed by the Dialog Box Controls should be set up in the script prior to the dialog code. By setting the variables, you can pass lists, files, and set which options are chosen by default.

Push Button

When creating **Push Buttons**, it is a good idea to assign the value of 1 to your "OK" button equivalent and 0 to your "Cancel" button equivalent. Each button will have a separate value. The Dialog Editor adds a line to the end of your script which helps to test return values.

```
ButtonPushed=Dialog("MyDialog")
```

To test a return value of a Push Button do the following:

```
If Buttonpushed == 1 then goto label
```

"Cancel" or the value 0 will generally look for a label **:cancel**. If not found, it will exit. For more information on redefining the cancel button ("Cancel" processing), see **Things to Know** in the **WIL Reference Manual**.

Radio Button

Used in situations to choose one item over another. You can have 9 choices per variable.

In using a **Radio Button**, the variable assigned is the same for each of the choices but the value is different. For example, the script in a Dialog may look like:

```
Example03=`120,40,48,DEFAULT,RADIOBUTTON,music,”Blues”,1`
Example04=`120,56,56,DEFAULT,RADIOBUTTON,music,”Jazz”,2`
```

The variable “music” is the same on both lines. The text and the values are different on each line.

Note: **Radio Button** cannot have a value of 0.

Check Box

Offers a choice of options. Any number may be marked or left unmarked.

Each **Check Box** has its own specific information. Variable, Value and Text are different, allowing the user to choose more than one check box.

Edit Box

Use this control to create an edit box in which a choice can be entered by default and then altered by the user.

Note: Variable names that begin with “PW_”, will be treated as password fields causing asterisks to be echoed for the actual characters that the user types.

Fixed Text

Use **Fixed Text** to display labels, descriptions, explanations, or instructions. The Control Attribute box will let you type an endless amount of information into the text box. However, only about 60 characters will be displayed.

Varying Text

Use **Varying Text** to grab data which may change, like a date or a password, from somewhere else.

File Listbox

Use **File Listbox** to allow the user to choose a file from a list box. Set your variable to display a directory path and file mask or the result of **FileItemize**.

```
wbtfiles=”C:\WINBATCH\*.WBT”
wbtfiles=FileItemize(“*.bak”)
```

This box can be tied with the variable to an **Edit Box** or to **Fixed Text**. When the user chooses a file, it will be displayed in the **Edit Box** or in the place of **Fixed Text** if the variable is the same.

Note: When **File Listbox** is used, the dialog editor assumes that a file must be chosen before it proceeds. Add the following WIL command to the top of your script if you wish to allow the dialog to proceed without a file selection.

```
    IntControl(4,0,0,0,0)
```

When no file is selected, the return value of the filename variable is:

```
        ”NOFILESELECTED”
```

See the WIL manual for more information on **IntControl**.

ItemSelect Listbox

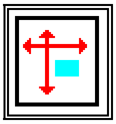
Use the **ItemSelect Listbox** to allow the user to choose an item from a list box. This option is similar to the WIL commands **AskItemList**, and **ItemSelect**. Set your variable to display a list of items delimited by a tab. Use **@tab,** a predefined constant, as the delimiter.

```
    tunes=”My Shirona%@tab%In the Mood%@tab%Staying Alive%@tab%
                  RockLobster%@tab%Tequila”
```

Window Information Utlility

The Window Information utility can grab window position settings from windows displayed on your monitor.

Using the Window Information Utility

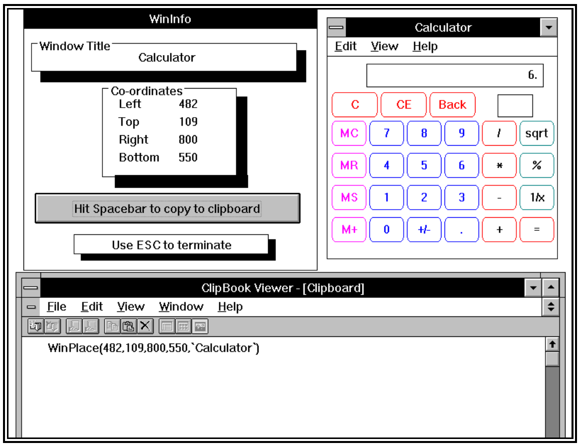


The Window Information utility is a handy window name and position grabber

The Window Information utility (see Filename Appendix A for filename) 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. (**WinPlace** is a WIL function, useful for reposistioning or resizing windows.) It puts the text into the Clipboard, from which you can paste it into your WIL program.

The Window Information utility captures coordinates in a 1000 by 1000 format that is relative to the current screen size. Since WinBatch considers every screen to have a 1000 by 1000 size, your sizing will always take up the same percentage of the user's screen. One eighth of a screen at 1024 by 768 screen resolution is actually much larger than the same eighth is at 640 by 480 pixels resolution.

Design your dialog boxes to be about 250 by 250 in size or larger. Then they will be prominent at all resolutions.



```
WinPlace(482,109,800,550,`Calculator`)
```

The **Window Information utility** captures relative screen coordinates. You'll need a mouse to use The **Window Information utility**. While the **Window Information utility** is the active window, place the mouse cursor over the window you wish to create the **WinPlace** statement for, and press the spacebar. The new statement will be copied into the Clipboard. Then press the **Esc** key to close the dialog.

WinBatch FILEMENU

Menu Utility for the Windows Explorer

Description

FILEMENU is a menu-based WIL (Windows Interface Language) application. FILEMENU is a menu utility DLL for the Windows Explorer. FileMenu 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.

System Requirements

FILEMENU requires a version of Windows supporting the Windows Explorer, such as Windows 95/98/NT.

Installation

FILEMENU is installed during the normal setup of WinBatch.

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 **Start** **Run** from the Taskbar, 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.

Menu Files

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.

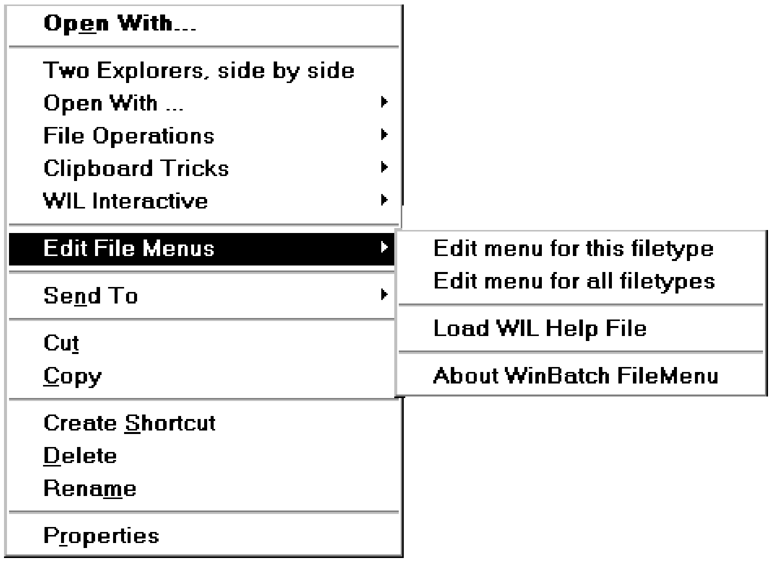
A menu file can be created or edited by selecting **Edit File Menus** from **Filemenu.** This option opens the Windows Notepad and loads either a file-specific menu or the "all filetypes" menu. Modifications to menu files are made once the file is saved.

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

Using the "all filetypes" File Menu

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 a popout option is highlighted, an additional explanation of what the option does 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.

Creating/Modifying File-Specific 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 right-clicked on the in the Windows Explorer

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.

FileMenu.ini

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, 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 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

Usage Tips, Known Problems and Limitations, etc.

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

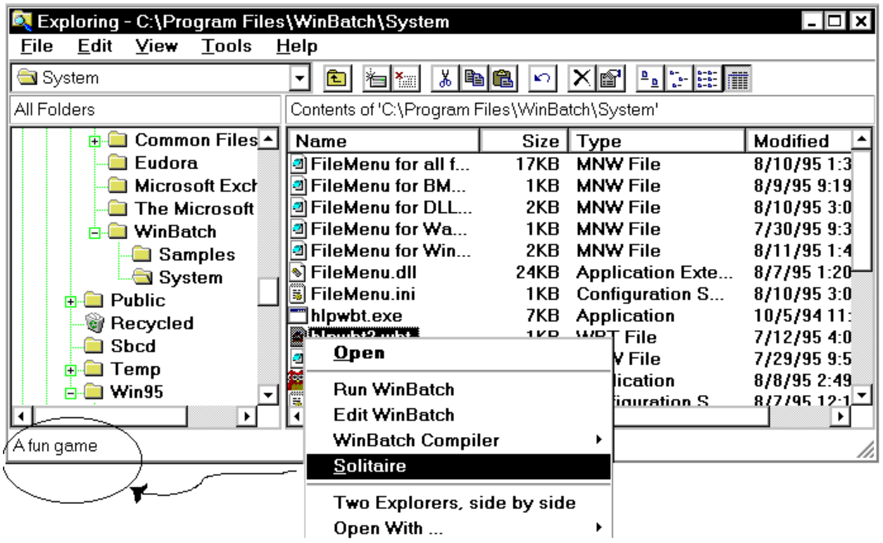
Hotkeys are not supported.

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

You can specify a comment to 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 the comment appears on the Explorer’s status bar.



Name	Size	Type	Modified
FileMenu for all f...	17KB	MNW File	8/10/95 1:3
FileMenu for BM...	1KB	MNW File	8/9/95 9:19
FileMenu for DLL...	2KB	MNW File	8/10/95 3:0
FileMenu for Wa...	1KB	MNW File	7/30/95 9:3
FileMenu for Win...	2KB	MNW File	8/11/95 1:4
FileMenu.dll	24KB	Application Exte...	8/7/95 1:20
FileMenu.ini	1KB	Configuration S...	8/10/95 3:0
hlpwbt.exe	7KB	Application	10/5/94 11:
	1KB	WPT File	7/12/95 4:0
		V File	7/29/95 9:5
		lication	8/8/95 2:49

WinBatch POPMENU

Pop-up menu for the Windows 95/98/NT Taskbar

Description

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

POPMENU appears as an icon in the systray area of the Windows 95/98/NT Taskbar. The Taskbar extends along one edge of the Windows 95/98/NT desktop and includes the "START" Button. A click on the POPMENU (owl) icon brings up a menu of WIL batch files. Samples are included, but you can completely modify these to meet your needs.

Start
Exploring - C:\AARD...
3:50 PM

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

System Requirements

POPMENU requires Windows 95/98 or Windows NT.

Installation

To install POPMENU:

POPMENU can be installed during the initial install of WinBatch. Make sure the checkbox option is checked on the setup screen.

OR

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 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 and Windows 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 Taskbar.

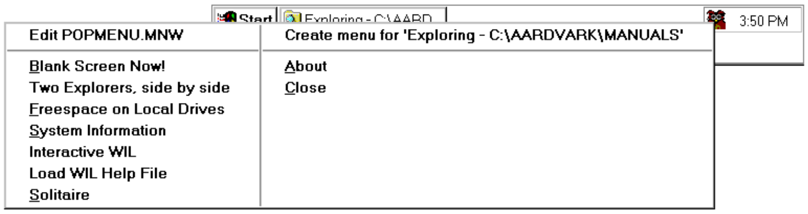
Operation

POPMENU is launched at start-up by default. If the POPMENU icon is not displayed in the Systray on the Taskbar, you can start POPMENU by running POPMENU.EXE.

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

De-activate POPMENU by clicking anywhere outside of the menu.

Close POPMENU by selecting “Close” from its menu. Selecting “Close” will actually exit POPMENU.



Menu Files

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), it uses this directory if set in Popmenu.ini.

3.) Home directory (“HOMEPATH” environment variable), if set in Windows.

4.) Windows directory.

5.) PopMenu directory.

6.) Other directories on your path.

By default, new menu files created by POPMENU will be placed in your Winbatch\System 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 directory otherwise. You can change this by editing the POPMENU.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.

INI Settings

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

Usage Tips, Known Problems and Limitations, etc.

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.

APPENDIX A: Filenames

WinBatch and Accessories

There are several different platforms on which WinBatch and its utilities may be run. When a file name is generated, it is made up of four or five characters which specify WHAT the file is, three characters which specify which platform the PC is running under and an .EXE or .DLL file extension.

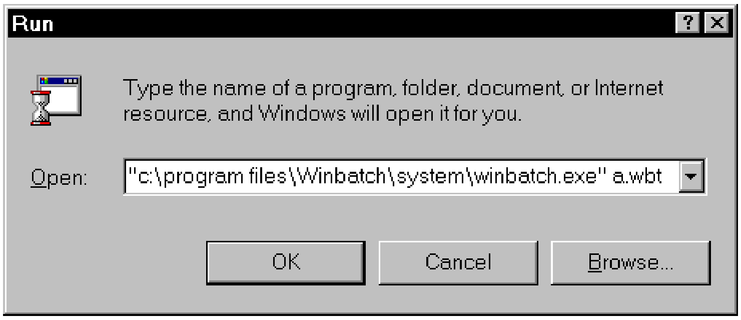
File Name Summary

File names are important in these areas:

1. Running WinBatch scripts.

WinBatch scripts are text files the WinBatch interpreter translates into action. To do this from a program launcher such as the **Start** **Run** menu item on the Taskbar, the file name of WinBatch has to be entered followed by a space and the name of a script.

Start **Run** from the Windows Taskbar produces this dialog:


Run
?
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

"c:\program files\Winbatch\system\winbatch.exe" a.wbt

2. Compiling WinBatch files with WinBatch Compiler.

If you have the WinBatch Compiler, you have the option of including in the executable batch file all, or just the minimum, number of files WinBatch needs to run a particular script. The Compiler includes selection dialogs for choosing options. The file name tables are here for general information.

3. Using Accessories.

WinBatch comes with a window position and name grabber called **Window Information.exe**. And, WinBatch comes with a Dialog Editor. The filenames for these utilities as well as filenames for WinBatch, Compiler, and WinBatch DLLs are listed below.

WinBatch and Compiler Programs: File Names

Environment	WinBatch	Compiler
Windows 3.1, 3.11	WBAT16I.EXE	WBC-16I.EXE
Windows 95/98/NT-Intel long filenames disabled.	WBAT32I.EXE	WBC-32I.EXE
Windows 95/98/NT-Intel with long filenames	WINBATCH.EXE	WBCOMPILER.EXE
Windows 95/NT-DEC Alpha	WBAT32D.EXE	WBC-32D.EXE
Windows 95/NT-PowerPC	WBAT32P.EXE	WBC-32P.EXE
Windows 95/NT-MIPS	WBAT32M.EXE	WBC-32M.EXE

WinBatch Required DLL's: File Names

(The ?? stands for a two letter code unique to the version of WinBatch your running.)

Environment	WinBatch OLE DLL	WinBatch WIL DLL
Windows 3.1, 3.11	WBO??16I.DLL	WBD??16I.DLL
Windows 95/98/NT-Intel	WBO??32I.DLL	WBD??32I.DLL
Windows 95/98/NT-DEC Alpha	WBO??32D.DLL	WBD??32D.DLL
Windows 95/98/NT-MIPS	WBO??32M.DLL	WBD??32M.DLL
Windows 95/98/NT PowerPC	WBO??32P.DLL	WBD??32P.DLL

WinBatch Accessories: File Names

Environment	Dialog Editor	WinInfo
Windows 3.1, 3.11	WWDLG16I.EXE	WINFO16I.EXE
Windows 95/98/NT Intel long filenames disabled.	WWDLG32I.EXE	WINFO32I.EXE
Windows 95/98/NT Intel with long filenames.	WIL DIALOG EDITOR.EXE	WINDOW INFORMATION.EXE
Windows 95/98/NT DEC Alpha	WWDLG32D.EXE	WINFO32D.EXE
Windows 95/98/NT MIPS	WWDLG32M.EXE	WINFO32M.EXE
Windows 95/98/NT PowerPC	WWDLG32P.EXE	WINFO32P.EXE

Network Extenders: File Names

Environment	Novell 3.x	Novell 4.x
Windows 3.1, 3.11	WWN3X16I.DLL	WWN4X16I.DLL
Windows 95/98/NT Intel	WWN3X32I.DLL	WWN4X32I.DLL
Windows 95/98/NT DEC Alpha	WWN3X32D.DLL	WWN4X32D.DLL
Windows 95/98/NT MIPS	WWN3X32M.DLL	WWN4X32M.DLL
Windows 95/98/NT PowerPC	WWN3X32P.DLL	WWN4X32P.DLL
Special: Windows 95 with 16 bit Novel NetWare Client	WWN3Z32I.DLL	N/A

Win32 Network Extenders: File Names

Environment	Extender
Basic: all Windows clients	WWWNET32I.DLL
Windows 95/98 client	WWW9532I.DLL
Windows 9x:Windows 95 client/NT server	WWW9X32I.DLL
Windows NT client	WWWNT32I.DLL

Note: Some of the above extender filenames may not exist for the specified platform.

File Naming Conventions

The following tables show how the filename, minus the extension, is broken down and defined.

WinBatch for Windows 95 running on a PC with an Intel, or compatible, microprocessor (the majority of installed Pcs) will have the file name, WINBATCH.EXE. **WinInfo** is WINDOW INFORMATION.EXE. The Dialog Editor is WIL DIALOG EDITOR.EXE. The WinBatch Compiler is WBCOMPILER.EXE.

The First 4--5 Characters in the File Name

Program/Utility	Filename
WinBatch	WINBATCH
WinInfo	WINDOW INFORMATION
Dialog Editor	WIL DIALOG EDITOR
WinBatch Compiler	WBCOMPILER

Network Extender	Filename
Novell 3.x extender	WWN3X
Novell 4.x extender	WWN4X
Basic Win 3.1 extender	WWW3A
Win32 network extenders	WWW95 WWW9X WWWNT
Multinet, WinForWrkGrp, Win4 extender	WWWN

The Second 3 Characters in the File Name

Platform	Filename
Intel 16-bit version (Windows 3.1)	16I
Intel 32-bit version (Windows 95/NT)	32I
DEC Alpha 32-bit version	32D
MIPS 32-bit version	32M
PowerPC 32-bit version	32P

If you have Windows 98/98/NT and ordered the single-user version of WinBatch, the executable files you received are WINBATCH.EXE, WIL DIALOG EDITOR.EXE, and WINDOW INFORMATION.EXE. By default, WinBatch installs the files that are suitable to the platform. If you need access to the old 16 bit legacy version of the WinBatch and it's DLLs, you can copy them from the CD-ROM or download them from our website.

WinBatch DLLs

WinBatch uses two DLLs: a WBO DLL, used for OLE functions, and a WBD DLL, used for all other WIL functions.

Inorder for WinBatch scripts or compiled EXEs to find and use them, they must be either in the directory where the WinBatch script file is, or on a system or network search path. They can be copied there manually, or wil be automatically written to disk at runtime, with the Large EXE—standalone option of the Compiler.

When a script is compiled with the Large EXE option, all the necessary DLLs will be bundled into to the executable.When the EXE starts up, it scouts around looking for its DLL. If it cannot locate a copy either in the current directory or anywhere along the path or in the Windows directory then it unstuffs the emergency copy of the DLL and places it in the same directory as the EXE. If located on a fixed hard drive, it will make the DLLs in the same directory as the EXE. If executing on removable media (floppy, zip, jaz drives), it will create the DLLs in the Windows directory.

To decrease file sizes, the Compiler also has a Small EXE option.

Small WinBatch executables need to be able to find the WinBatch DLLs. They can be located in the current directory, on the DOS path, on the search path, or in the Windows directory. The easiest way to get the initial copies of the DLLs there, is to create a simple WinBatch utility that uses all the DLLs, extenders, and so forth. Run this once in any directory on the DOS or network search path.

Once the DLLs are extracted, they can be copied anywhere they will be needed. A convenient place for them is often in the Windows directory since it is always on the search path.

Names for the WinBatch DLLs

The WinBatch DLL names are made up of 3 parts.

The first three digits identify the DLL type.

WBD - WIL Language Interpreter DLL
WBO - WIL OLE Interpreter DLL

The second two digits are used for version identification purposes. The letters are chosen at random, will match for both the WBO and the WBD DLL, and will change for each new version of the DLL.

XX = BP, or BQ, (some combination of letters)

The final three digits reference the operating environment of the DLL.

16I - 16-bit Windows (Windows 3.1/WFW 3.11)
32I - 32-bit Windows (Windows 95/NT)

Here is are examples of a pair of DLLs for use on 32-bit versions of Windows on Intel class processors.

WBDBQ32I.DLL

WBOBQ32I.DLL

Appendix B: WinBatch+ Compiler

Installing and Using WinBatch+Compiler

NOTE: This section is applicable only if you purchased WinBatch+Compiler. This is NOT a "shareware" software product. The Compiler is a separate product and is NOT included in the purchase of WinBatch, the single-user version. If you would like additional information on the Compiler and its capabilities, please call Customer Service.

Because WinBatch+Compiler includes both WinBatch and the WinBatch Compiler, registered users of WinBatch can always upgrade to WinBatch+Compiler at a special price.

The WinBatch Compiler can change a WinBatch .WBT file into any one of the following:

- A small Windows EXE file.
- A large standalone Windows EXE file.
- An encoded and encrypted WinBatch script file.
- A password protected WinBatch script file.

No royalties of any kind are required for distribution of any file created by this compiler.

Compiler Installation

WinBatch and the Compiler install from the CD_ROM in your WinBatch+Compiler package. The installation program is itself a Windows application, so make sure Windows is running, when you are ready to install.

Make sure that <u>NO</u> WinBatch WBT files are currently running. Insert WinBatch + Compiler CD-ROM on your workstation. If you have your CD-ROM configured to “Auto-Start”, the setup will run automatically. Otherwise, open the Explorer to your CD-ROM drive and double click on SETUP.EXE

You will be prompted for the directory to install WinBatch + Compiler into. The default will be C:\PROGRAM FILES\WINBATCH. You may change the directory if you wish, but you cannot install WinBatch on a network drive. *To install WinBatch on a server, you need to have purchased a site license.*

The first time you run the Compiler you will be asked to enter your license number. The license numbers can be found in the back of this WinBatch User’s guide.

Compiler Usage

The compiler may be run in interactive mode. In interactive mode, the user is prompted to provide all necessary information via a popup dialog box.

Before you can do anything useful with the Compiler, you must use the WinBatch file interpreter to create and test a WinBatch script file. Each WinBatch script file should have a file extension of .WBT, or .WIL.

Notes about the compiler:

The compiler allows you to specify version information strings to be embedded in the EXE (under “Version Info”).

The compiler also creates a configuration file for each source file you compile. It will be placed in the same directory as the source file, and will have the same base name with an extension of “.CMP”. For example, if you compile “C:\UTIL\TEST.WBT”, it will create a configuration file named “C:\UTIL\TEST.CMP”. Make sure you do not delete the .CMP files, if you plan to reuse or remember the previous compiler settings. The .CMP file is basically a record of the compiled files' last compile settings.

Running the Compiler in Interactive Mode

Start the compiler by double-clicking the compiler icon or the WBCompiler.EXE file name (or by choosing the appropriate item in any menu system you may be using).

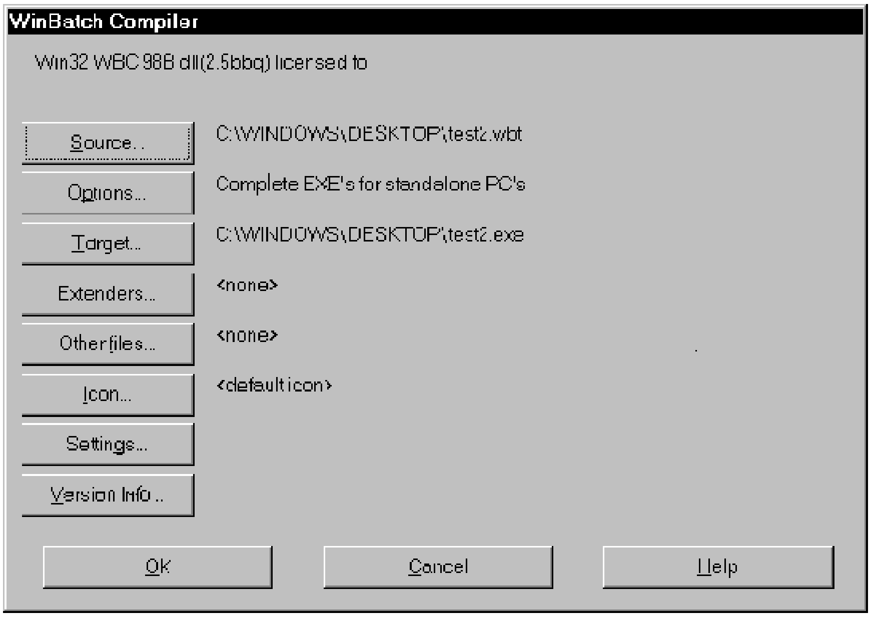
The compiler also supports “Drag and Drop” compiling. Select the Winbatch source file (WBT or WIL File) and drag it over the WBCOMPILER.EXE icon and drop the source file. A dialog box will be displayed asking for input. The source file you specified will be automatically displayed as the source file for the compile.

Select the type of compile desired (large EXE, small EXE, encoded or encrypted), choose the source .WBT file, and supply an output file name. If you wish, choose an icon along with any necessary extenders. Press the OK button.

The compiler will process for 5 to 10 seconds, and then report that the file has been compiled. The compiler does not perform error checking. It is assumed the WBT file has been properly debugged with the WinBatch interpreter prior to the compile step.

Interactive Mode

When you launch the Compiler EXE, a dialog box similar to the following will be displayed:



Win32 WBC 98B dll(2.5bbq) licensed to

C:\WINDOWS\DESKTOP\test2.wbt

Complete EXE's for standalone PC's

C:\WINDOWS\DESKTOP\test2.exe

<none>

<none>

<default icon>

Options

The OPTIONS button allows you to select which type of executable file you would like to create from your WBT file.

Large Executable Utilities for Standalone PC's (includes accessory DLLs, Extenders, OLE 2.0, etc.)

This option creates an EXE designed for Standalone PC's and does not require any extra DLLs, unless you specify additional DLLs in your script with the Addextender function. When a Standalone EXE is launched on a PC, the necessary DLLs are automatically written into the current directory. If for some reason, they cannot be written to that directory (perhaps the directory is set to be "Read Only"), the large compiled file will not run.

The DLLs can also be copied into a directory on a computer’s PATH where the compiled EXE will find them there and execute successfully. The Compiler has a small EXE option that takes advantage of this configuration.

The DLLs need to be placed on the PATH only once. Subsequent EXE files installed on this same machine can be compiled with the Small EXE option., so that multiple compiled EXEs can use a copy of the WinBatch DLL located in a single place, resulting in easier file maintenance.

If Network commands have been used, you will need to compile the Network Extender DLLs into the EXE, by clicking on the "Extenders" button. This is explained more specifically in the section, EXTENDERS, see page 96.

Small Utilities for Networked PC’s (without accessory files)

This option is suitable for network file server installation, and for distribution with separate DLL files. Makes a smaller exe that loads faster over a network. You also know what copies of the DLLs are being used as you have to manually place them.

One trick is to compile the setup program with the Large EXE option, and have it create the required DLLs. Subsequent EXEs can all be compiled with the small EXE option, because the DLLs already exist. For Network installations we recommend the small EXE with copies of the required DLLs in the same directory.

You have a couple of choices in terms of where to locate the DLLs:

1. Place a copy of the DLL in some directory that is on your path (windows directory?). The exe will use that copy instead of making a new one.

   OR

2. Make a directory for the exe and just put a shortcut to it from the desktop.

When a small WinBatch utility is run, it will look in the Windows directory and the directories in the environment PATH variable for the DLL’s. Place the WinBatch DLL’s, and network extender DLL’s on the path or search drive. If you launch this utility on a PC in which a large standalone utility has been run previously, the small utility can use the same DLLs the large one installed.

> **Hint:** You can automatically install the DLL’s on the PATH in a computer. Create a large executable containing only a single statement: Display(1,”WinBatch”,”WinBatch installed. Thank You.”). You can change this statement as you like. Then, compile this as a large EXE with all the DLL’s your scripts are ever likely to need. Copy it into the Windows System directory, and run it from there. The DLL’s will be installed once and for all. Any subsequent batch files run on that computer can be small compiled EXEs that don’t need the DLL’s already installed on that computer.

Encode for Call's from EXE files

This option makes files that can be "called" from a compiled EXEs. It creates an encoded WBT file. Encoded WBT files provide the following:

- Source code that is protected from unauthorized or accidental modification.
- Encoded WBT files may be CALL'ed from compiled files.

If your code has a Call to another WBT file, the called WBT must be compiled with this option. Otherwise, when you run your EXE, you will get an "Encrypted/Encoded Verification Failed" Error.

> **Note:** When you compile your file, a new target filename will be created with a .WBC extension. It is necessary to have a different filename from the original filename. You cannot compile a file to its own name without corrupting the file. To protect the innocent, the default Target extension is .WBC. After compiling, edit the uncompiled script file and change the Call statement to reflect the new filename extension .WBC. Then recompile the main EXE.

Encrypted with Password

This option encrypts a WBT file and uses a default Target extension of .WBE. The WinBatch interpreter (WINBATCH.EXE, or version specific WinBatch interpreter) is needed to access the encrypted file. During compilation, you must input a password in to the compiler dialog. The same password must be supplied when the WBT file is run. The purpose of an encrypted WBT file is to prevent unauthorized personnel from executing it.

Since encryption is easily added to WinBatch utilities, this option is rarely used. In fact, no one has ever been known to use it. Like the human appendix, it reminds one of evolutionary events, while avoiding the performance of any useful function.

Extenders

The EXTENDERS button displays a list of extenders which can be chosen and compiled into a Standalone EXE option. More than one extender may be chosen. If any of the Network extender functions are used, the corresponding extender must be compiled into the Standalone, or placed in the Windows directory, or on the network path for a Small EXE to access. The extenders will be displayed in the WinBatch Compiler Dialog box when you click on the EXTENDERS button. After you make a selection of all the extenders you would like to include, they will be displayed next to the extenders button on the dialog box.

Source

The SOURCE button displays a File Selection Box. Select your file or type the filename and path into the File Name box and press OK. The path and filename will be displayed in the WinBatch Compiler dialog box next to the SOURCE button.

Note 1: Source and Target names:

After you select a SOURCE file, a default TARGET name will be generated and displayed next to the TARGET button. To change the default name, click on the TARGET button.

Note 2: A WinBatch executable cannot have the same name as an executable application it runs:

Your compiled file will have an extension of EXE. If your WinBatch script has the same name as the program you want to run from the WinBatch utility. The WinBatch utility may actually run itself, rather the application EXE. This cannot be resolved by using full path names for the program you want to run.

The solution is to make certain that the WinBatch script and the other application have different names. Either choose a different name for your utility, or rename the other application and run it with that name.

Target

The TARGET button displays a File Selection Box. Select your file or type the filename and path into the File Name box and press OK. The path and filename will be displayed in the WinBatch Compiler dialog box next to the TARGET button.

Note: A default target filename and path will generally be generated from the SOURCE filename and path.

Other Files

The OTHER FILES button displays a File Selection Box of files which can be selected and compiled into a Standalone EXE. More than one file may be chosen. The selected files will be displayed in the WinBatch Compiler Dialog box next to the OTHERFILES button.

Icon

The ICON button displays a File Selection Box which allows you to choose an icon. Select your .ICO file and press OK. The path and icon filename will be displayed in the WinBatch Compiler dialog box next to the ICON button.

WinBatch+Compiler comes with icons you can use. These are in the ICONS subdirectory of your WinBatch directory.

Settings

The SETTINGS button displays a dialog for configuration settings. The “Modify Configuration Settings” dialog allows you to specify a URL (i.e., http://www.example.com) for technical support, which will be displayed in WinBatch error messages. You also have the option to select a check box to “run the script hidden”. If this option is selected the WinBatch icon will not appear on the desktop or in the Taskbar, when the EXE is executed. The selected modifications are not displayed in the WinBatch Compiler Dialog box next to the SETTINGS button.

Version Info

The VERSION INFO button displays a dialog that allows you to specify Version information strings. These are the version strings that will be displayed in the Properties dialog, that is displayed when you right-click, and select "Properties",on the compiled EXE. The selected modifications are not displayed in the WinBatch Compiler Dialog box next to the SETTINGS button

#Include

This command gives you the ability to embed external files when running or compiling a WinBatch script, by using the “**#include**” pre-processor directive.

The syntax is:

```
  #include “filename”
```

The keyword “**#include**” must begin in column 1, and must be all lower-case.

The file name must be delimited by double quotes. The file name may contain path information, and does not need to have an extension of WBT. Nothing else should appear on the line.

Each line containing a **#include** directive will be replaced by the contents of the specified file. If the file cannot be found, an error will occur.

When using the WinBatch compiler, the “included” file(s) must be present at compile time; they will be embedded in the compiled EXE, and therefore do not need to be distributed as separate files.

When using the interpreted version of WinBatch, the “included” file(s) must be present at the point in time when the script is launched (they cannot be created “on-the-fly” from within the script).

You can have as many **#include** directives as you wish, and they may be nested (ie, “included” files may themselves contain **#include** directives).

Network Considerations

If you plan to put the compiled EXEs on a network, the following information will be helpful:

1) Set the compiled EXE files to read-only so that multiple users may access the same file.

2) Copy the DLLs from the WinBatch\System subdirectory in the Explorer, to a file server directory in the search path and set the DLLs as read-only. (see Filename Appendix B)

3) When the compiler, or any compiled WBTs with the large Standalone option selected, are run, they will search the entire PATH for any required DLLs (see Filename Appendix B). If the DLLs are not found, they will be created in the user’s WINDOWS directory.

Restrictions

The **CallExt** function is not supported in compiled Exe’s.

The compiler itself is licensed for a single user. A special license is required to operate the compiler on a network drive or from a diskless workstation. If you need a capability of this sort, please call Customer Service.

Error Appendix

10102, “10102: WinBatch - Unrecognized ParentProcess request code”
10103, “10103: WinBatch Compiler - CallExt not available”
10104, “10104: WinBatch: EnvironSet Var and/or Value too long”
10105, “10105: WinBatch: EnvironSet - Failed. No space?”
10106, “10106: WinBatch: EnvironGet - Failed. Name too long?”
10107, “10107: WinBatch: EnvironGet - Failed. Value too long?”
10108, “10108: Box functions: Box command stack full”
10109, “10109: Box functions: Invalid box ID”
10110, “10110: BoxButtonDraw: Invalid button ID”
10111, “10111: BoxButtonDraw: Invalid ‘rect’ string”
10112, “10112: BoxButtonStat: Invalid button ID”
10113, “10113: BoxColor: Invalid color string”
10114, “10114: BoxColor: Invalid ‘wash’ color”
10115, “10115: BoxDrawRect: Invalid ‘rect’ string”
10116, “10116: BoxDrawLine: Invalid ‘rect’ string”
10117, “10117: BoxNew: Invalid ‘rect’ string”
10118, “10118: BoxNew: Invalid ‘style’ flag”
10119, “10119: BoxNew: Unable to create box”
10120, “10120: BoxPen: Invalid color string”
10121, “10121: BoxPen: Invalid pen width”
10122, “10122: BoxTextColor: Invalid color string”
10123, “10123: BoxTextFont: Invalid font size”
10124, “10124: BoxTextFont: Invalid font style”
10125, “10125: BoxTextFont: Invalid font family”
10126, “10126: BoxDrawText: Invalid ‘erase’ flag”
10127, “10127: BoxDrawText: Invalid ‘alignment’ flag”
10128, “10128: BoxDrawText: Invalid ‘rect’ string”
10129, “10129: BoxUpdates: Invalid ‘update’ flag”
10130, “10130: BoxesUp: Invalid ‘rect’ string”
10131, “10131: BoxesUp: Invalid ‘show’ mode”
10132, “10132: BoxMapMode: Invalid map mode”
10133, “10133: BoxDrawRect: Invalid style”
10134, “10134: BoxDrawCircle: Invalid ‘rect’ string”
10135, “10135: BoxDrawCircle: Invalid style”
10136, “10136: BoxButtonDraw: Unable to create button”
10137, “10137: BoxButtonKill: Invalid button ID”
10138, “10138: BoxDataClear: Specified tag not found”
10139, “10139: IntControl: Unrecognised Request”

Glossary of Terms

(f)

A value that must be in floating point format. This means that the number must include a number, a decimal point, and another number as in 0.123 or 456.2256. Exponentials of the form 4.9e7 are always floating point. Floating point numbers can include signs before the entire number, or before the “e” that stands for exponent.

(i)

A value that must be an integer.

(s)

A string value is used here. A string is from one to many alphabetic and/or numeric characters enclosed in quotation marks, double quotes or single quotes are equally acceptable.

applications

Software programs that use one main window along with a menu bar. An example of an application would be a spreadsheet.

batch language

A programming language that automates a process, by processing events in a step-by-step fashion. Batch languages are interpreted at run time.

constant

A value that does not change. WinBatch has many built-in constants. Useful ones include @CRLF and @TAB for inserting these into lines of text in dialogs.

Dynamic Link Libraries

Also called “DLLs”. A DLL is an accessory file used by other Windows programs. DLLs can be actual program files without the EXE extension, libraries of executable routines, or even simple data files. The information in DLLs can be utilized through the WIL **DllCall**() function.

exception handling routines

Event evaluation routines that return their results in a message box.

icon

A small picture that represents an application. Several sizes are available for use under Windows.

interpreter

A software program that reads a script file one line at a time. It examines the line, finds directives that indicate action, and then instructs the computer to carry out them out.

macro scripting language

A computer language that reads lines in a text file and turns them into action on that computer. The term “macro” comes from the capability of completing numerous operations in sequence.

menu items

Selections from a list of items found at the top border of the main windows of a Windows application.

MS-DOS

A personal computer operating system produced and marketed by Microsoft Corporation.

OLE 2.0

A specific version of a scheme of inter-program data exchange called Object Linking and Embedding. It is an extension of DDE and the Windows Clipboard. There are several OLE capabilities. WinBatch and WIL support OLE 2.0 automation. This is the capacity of one program to automate anything that an application can do.

operators

Actions that transform one numeric value into another. An example is the addition operator “+”. It takes one numeric value, adds it to a second and makes the result available for display or use by another operator.

plain text

Text containing only letters and numbers. It must not contain hidden codes for formatting or null characters. Word processors do not normally produce plain text. They can, however, be directed to do so. Windows word processors generally provide this option in the File SaveAs menu.

register

Obtain a license to use software.

script file

A computer file generated by a text editor. It contains a list of statements that can contain both directives and comments.

system management utilities

Short programs for manipulating any operations on a computer. Generally they automate activities that otherwise need to be done repetitively and manually.

utilities

Utilities manipulate applications, the operating system, and the Windows interface.

WINBATCHFILENAME

In 32 bit Windows, the WINBATCHFILENAME is WinBatch.exe. For Windows 3.1, the WinBatch executable file is named WBAT16I.EXE.

WIL

Windows Interface Language (WIL) is the actual programming language used by WinBatch. WinBatch is a processor that interprets WIL directives and directs the computer to carry out these directives.