noyesyesyesyesyesWinBatch Help FileTRUEWinBatchyesyes20/10/98

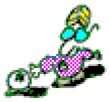
WinBatch Help File

functions winbatch

WinBatch is an application which uses Wilson WindowWare's Windows Interface Language (WIL) to automate Windows and Windows applications. WinBatch controls Windows, Windows applications, and network connections.

There are over 500 functions available to help you get the job done -- what would take pages of code in other languages is often taken care of by a single WIL function.

Your WinBatch scripts can be run in many different ways, including clicking on an icon or choosing from a **popup menu**.



There are several utilities included with WinBatch to help automate your system, including a **Dialog Editor** that lets you design custom user interfaces for your scripts without tedious coding.

The optional **"WinBatch+Compiler"** lets you compile your applications into .EXE files and distribute them to anyone you want, royalty-free -- great for network system administrators.

WinBatch is available for every version of Windows, including Windows 3.1, Windows 95/98, and Windows NT, running on Intel, DEC Alpha, PowerPC and other platforms.

We've included a Step by step guide to learning WIL so you can get started right away. See the **WIL Tutorial** in the Windows Interface Language Help file.

WinBatch Help File

Acknowledgments

This software designed by Morrie Wilson and Richard Merit.

Documentation written by Tina Browning and Deana Dahley.

Getting Started

WinBatch is the system automation utility for Windows. WinBatch lets you take nearly anything your computer can do, and assign it to a simple menu selection, command or icon.

WinBatch controls Windows, Windows applications, and network connections.

WinBatch Help File

About WinBatch

WinBatch is the system automation utility for Windows. Using the power of our Windows Interface Language (WIL), WinBatch lets you take nearly anything your computer can do, and assign it to a simple menu selection, command or icon.

WinBatch controls Windows, Windows applications, and network connections. WinBatch includes the power of over 500 functions to help you get the job done.

- What is WinBatch?
- What can WinBatch do?
- Ways to run WinBatch scripts
- Installing WinBatch
- System Requirements
- Using WinBatch

What is WinBatch?

WinBatch is the Windows desktop automation solution that lets you take nearly anything your computer can do, and assign it to a simple menu selection, command or icon. You've got the power of a complete programming language at your fingertips, so you can create simple, useful applications, like a utility that prints files overnight.

The "WinBatch+Compiler" lets you compile your applications into .EXE files and distribute them to anyone you want, royalty-free -- great for network system administrators.

WinBatch controls Windows, Windows applications, and network connections.

WinBatch is available for every version of Windows, including Windows 3.1, Windows 95/98, and Windows NT, running on Intel, DEC Alpha, PowerPC and other platforms.

WinBatch is one of several Wilson WindowWare products that use our Windows Interface Language (WIL), a full-featured programming language with over 500 functions for controlling nearly every aspect of your computer. Many WIL 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.

- What can WinBatch do?
- Ways to run WinBatch scripts

What WinBatch can do

WinBatch utilities manipulate:

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

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

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

Activate 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 a Windows program. It can run from a Windows shell or any application that can run another application.

With over 500 functions, WinBatch can:

- Solve numerous system management problems
- Run Windows and DOS programs
- Send keystrokes directly to applications
- 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

- What is WinBatch?
- Ways to run WinBatch scripts
- Using WinBatch

- Make branching decisions based upon numerous factors
- Call Dynamic Link Libraries
- Act as an OLE 2.0 automation client
- And much, much more

Ways to Run WinBatch Scripts

WinBatch scripts are simple text files, but they can be launched in many different ways:

- run from icons in the Windows Explorer, Desktop, or Task Bar.
- as automatic execution macros for Windows via the Startup directory.
- from macros in word processors and spreadsheets
- from the Task Bar "Start ..Run..." command line entry in Windows
- by double clicking or dragging and dropping file names in the Windows Explorer.
- 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 such as Visual Basic and PowerBuilder

- What is WinBatch?
- What can WinBatch do?
- Writing a Script
- Running a Script
- Menu Files

Installing WinBatch

WinBatch is easy to install. You will find the necessary diskettes in your WinBatch package. If you have purchased "WinBatch+Compiler", see the instructions for "installing WinBatch+Compiler".

Windows must be running to install WinBatch.

Insert your WinBatch disk into your disk drive. From the File Run File Manager, or from the Task Bar select "Start…Run" type [CD_Drive]:\SETUP.EXE (ie., M:\SETUP.EXE), depending on which drive contains the WinBatch disk. Follow the prompts; the program will install the necessary files in a directory of your choice. You will be asked for license numbers.

Note: WinBatch cannot be installed from a drive shared over a network.

- What is WinBatch?
- Using WinBatch
- Registering your copy
- System Requirements

System Requirements

WinBatch requires an IBM PC or compatible running Microsoft Windows version 3.1 or higher. WinBatch 32 requires a 32 bit version of Microsoft Windows or Windows NT.

WinBatch scripts use about 150 kilobytes of system memory and 2% of system resources. This memory is returned to the system when the WinBatch utility ends.

A WinBatch script file cannot exceed 64K in filesize.

- Installing WinBatch
- What is WinBatch?
- Using WinBatch

WinBatch Help File

Using WinBatch

- Writing a Script
- Running a Script
- Parameters
- Menu Files
- Installing WinBatch
- System Requirements

WinBatch is an application which uses Wilson WindowWare's Windows Interface Language (WIL). There are over 500 functions available to help you get the job done. (What would take pages of code in other languages is often taken care of by a single WIL function.)

Network manipulation functions provided by extenders to WIL. The extenders are included in dynamic link libraries accessed with the WIL AddExtender ( ) function. Each extender has its own help file.

Note: WinBatch is based on "batch files". A WinBatch batch file is a text file containing one or more lines of WIL functions and commands. **PopMenu** and **FileMenu**, two WinBatch utilities, are implementations based on menu files.

Writing a Script

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\Sample directory.

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

Word processors like WordPerfect, AmiPro, and Word can also save scripts in plain text formatted files.

The .WBT extension is used throughout these instructions for batch file extensions, but you can use others just as well. If you want to click on a batch file and have Windows run it, be sure that you associate it in Windows with your WinBatch executable program file. When you installed WinBatch, an association is automatically established between WinBatch and .WBT files.

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.

A single WinBatch script file cannot exceed 64K in file size.

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.

- Running a Script
- Parameters
- Menu Files

Running a Script

WinBatch utilities run like any other Windows programs. They can run from a command line, or from a file listing such as the Windows 95/98 and Windows NT Explorer.

WinBatch utilities are usually run as files with the extension .WBT. When some WinBatch utilities are used, they need information passed to them when they run. This is easily done by passing command line parameters to them.

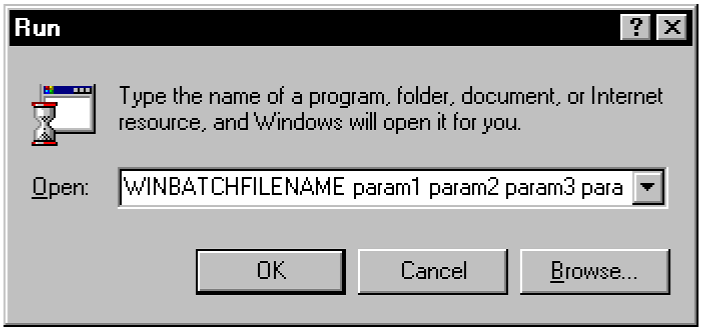
This capability can be used from the command line "Start..Run" menu items in Windows 95/98 and NT 4.0.   An example dialog is shown below.

Parameters can be also be passed through the command line entry included in the item properties of any icon in Program Manager. 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 system utility 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 and NT "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 param1 param2 param3 para

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, Windows 98 and the different Windows NT versions.

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

"p1 p2 ... p9" 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.

- Writing a Script
- Parameters
- Menu Files
- Ways to run WinBatch Scripts

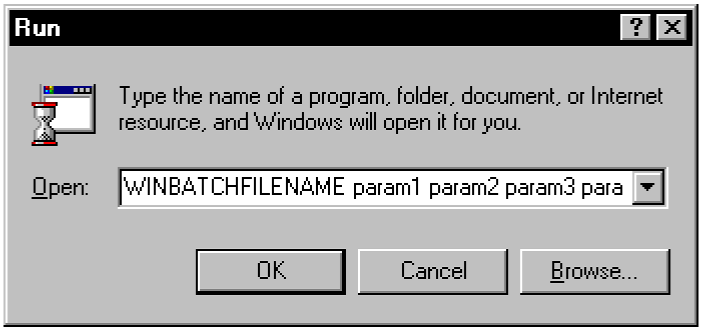
Parameters

A command like this runs a WinBatch system utility 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/NT "START…Run.." menu option.

This command line can be entered into a Command Line text entry box like this one:



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: WINBATCHFILENAME param1 param2 param3 para

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, Windows 98, and the different Windows NT versions.

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

"p1 p2 ... p9" 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.

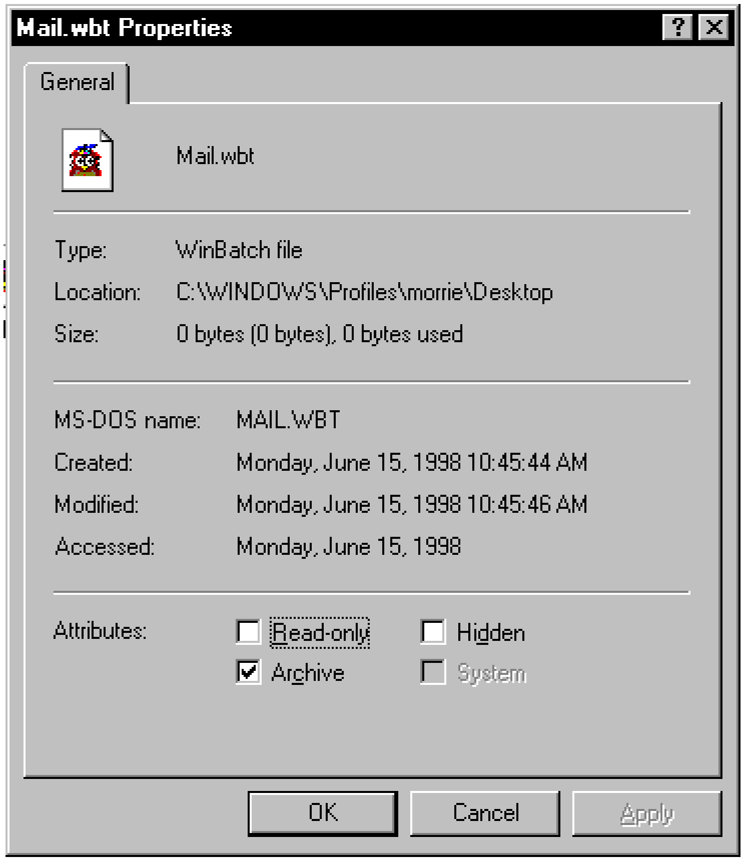
In order to pass parameters to a WinBatch script file, you must run the WinBatch executable, itself, and it must be followed by the name of the WinBatch script file and any other desired parameters. WBT files run from the Desktop as shortcuts must have their complete path in the Properties dialog box 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:\WINBATCH\SYSTEM\WINBATCH.EXE C\WINBATCH\MAIL.WBT
PASSWORD
```

- Displaying Passed Parameters in a Message Box
- Passing Parameters between Script Files
- Writing a Script
- Parameters
- Menu Files
- Ways to run WinBatch Scripts

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



Mail.wbt

Type: WinBatch file
Location: C:\WINDOWS\Profiles\morrie\Desktop
Size: 0 bytes (0 bytes), 0 bytes used

MS-DOS name: MAIL.WBT
Created: Monday, June 15, 1998 10:45:44 AM
Modified: Monday, June 15, 1998 10:45:46 AM
Accessed: Monday, June 15, 1998

Attributes: □ Read-only □ Hidden
☑ Archive □ System

Parameters passed to a WBT file will be automatically inserted into variables named param1, param2, etc. The WinBatch utility will be able to use these. An additional variable, param0, gives you the total number of command-line parameters.

Displaying Passed Parameters in a Message Box

To display the total number of command line parameters, use param0 as a variable in a message box. WinBatch works like the DOS Batch language to put parameters into text. Enclosing them in percent (%) signs works in WinBatch, too.

This example is a simple one line WinBatch function that:

1. Designs a dialog box with an OK button.

2. Specifies a title.

3. Specifies a message.

4. Puts varying information into the title or the message.

5. Formats the message in more than one line.

6. Returns a value that can indicate whether the operation has succeeded or not.

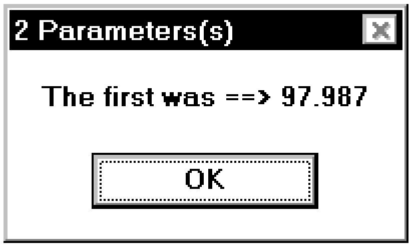
The Message function has this form:

```
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:

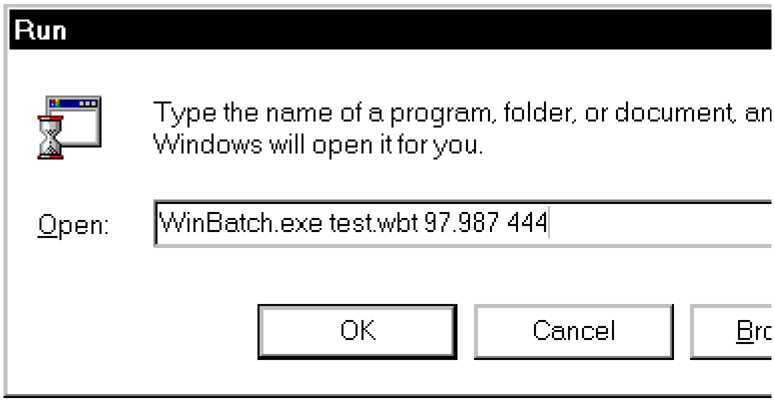


2 Parameters(s)

The first was ==> 97.987

The command line which produced the statement above was:

- Parameters
- Passing Parameters between Script Files
- Writing a Script
- Menu Files
- Ways to run WinBatch Scripts



Type the name of a program, folder, or document, an

Windows will open it for you.

Open: WinBatch.exe test.wbt 97.987 444

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.

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:

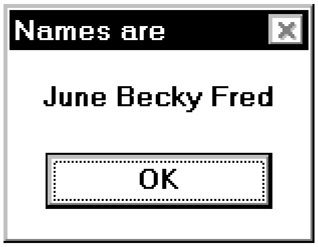
The first WBT calls a second WBT then passes three parameters.

```
  Call("test.wbt", "Fred Becky June")
```

TEST.WBT contains the following line:

```
  Message("Names are", "%param3% %param2% %param1%")
```

which produces:



Names are

June Becky Fred

- Parameters
- Displaying Passed Parameters in a Message Box
- Writing a Script
- Menu Files
- Ways to run WinBatch Scripts

Menu Files

Windows Interface Language (WIL) scripts are written in a plain text file, which can be created by Notepad or most word processors. (Of course, we recommend our own WinEdit, which has many features designed expressly for programmers, including a full-featured implementation of WIL itself.)

These text files can take one of two forms, depending on your particular implementation of WIL: batch files or menu files.

WinBatch executes batch files. A batch file is simply a list of WIL commands and function calls, executed in order (just like the old DOS batch language).

A menu file is similar to a batch file, except that multiple chunks of WIL code are organized into menu and sub-menus, and each routine is launched by pressing the appropriate keystroke or selecting an item from the menu. (The name and location of the menus vary depending on the particular implementation of WIL menu files.)

WinBatch comes with two utilities that use menu files: **FileMenu** and **PopMenu**.

The menu file format is explained in the Windows Interface Language Reference manual and in the on-line WIL help file.

WinBa

FileMenu

- PopMenu

Contacting Wilson WindowWare

Wilson WindowWare, Inc.
5421 California Ave. SW
Seattle, WA 98136 USA

Orders: (800) 762-8383
Voice: (206) 938-1740
Fax: (206) 935-7129

Email: info@windowware.com

- Registering your copy
- Ordering Information
- Order form
- Technical Support

Registered users of our software get manuals, technical support, use of Wilson WindowWare on-line information services, and special offers on new versions of Wilson WindowWare products.

How to get Technical Support

The Wilson WindowWare website is an excellent technical resource. Access to the entire Technical Support Database is at your fingertips. In the Technical Support area use the keyword search to find answers to common problems, alternate scripting methods, and sample code.   Or join the Wilson WindowWare Web BBS, a new Web forum.   The BBS provides an outlet for registered users to share their experiences with other users.

See the information on **registering your copy** if you haven't done so yet.

The latest versions of our software are available on-line. The places here may change at any time -- check your installation sheet for the most recent addresses.

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

Internet Technical Support Articles & Web BBS:
http://techsupt.windowware.com

Internet FTP: ftp.windowware.com in /wwwftp/wilson

- Registering your copy
- Ordering Information
- Order form

Registering your copy

Registered users of our software get 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 can register your software by mailing your registration card, faxing your registration card, or calling Wilson WindowWare.

- Ordering Information
- Order form

Entering your license numbers while installing WinBatch

The **WinBatch installation** program will request entry of both a control number and an ID number. Either upper or lower case will do. These numbers are located inside the back cover of your WinBatch User's guide.

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

Keep your license numbers in a safe place. They will be needed whenever WinBatch is reinstalled. If you have been issued a temporary license number, it will expire and the evaluation screens will 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.

Entering your license numbers after installation

To make the registration screen appear, hold the shift key down while starting a WinBatch utility. Select Enter License Info and enter the license numbers into the resulting dialog box.

Ordering Information

Licensing our products brings you wonderful benefits. Some of these are:

- Gets rid of that pesky reminder window that comes up when you start up the software.
- Entitles you to one hour free phone support for 90 days (Your dime).
- Ensures that you have the latest version of the product.
- Encourages the authors of these programs to continue bringing you updated/better versions and new products.
- Gets you on our mailing list so you are occasionally notified of spectacular updates and our other Windows products.
- And, of course, our 90-day money back guarantee.

International Customers

Although we do prefer payment by Credit Card we can accept non-US-bank checks under certain conditions. The check MUST be in your currency -- NOT IN US$ -- Just look in your newspaper for the current exchange rates, make out your check and send mail it to us. We will take care of the rest. No Eurocheques please.

Send to: Wilson WindowWare, Inc.
5421 California Ave. SW
Seattle, WA 98136
USA

or call: (800) 762-8383 (USA orders only)
(206) 938-1740 (customer service)
(206) 937-9335 (tech support)
(206) 935-7129 (fax)

(Please allow 2 to 3 weeks for delivery)

- Order form
- Contacting Wilson WindowWare

WILSON WINDOWWARE ORDER FORM

WILSON WINDOWWARE
5421 California Ave. SW ▪ Seattle ▪ WA ▪ 98136
Ph - (206)938-1740 Fax - (206)935-7129

- Ordering Information
- Contacting Wilson WindowWare

Name: ____

Company:____

Address:____

____

City: ________________________ St:______ Zip:___________

Phone: (______)_________________
Country:________________

Products
____ WinBatch @ $99.95 : _______.____

____ WinBatch Compiler @$495.00 : _______.____

____ WinEdit @$99.95 : _______.____

Shipping
____ US and Canada shipping @ $5.00 : _______.____

____ Foreign air shipping
(except Canada) @ $14.50 : _______.____

Total: _______.____

Please enclose a check payable to Wilson WindowWare or you may use Access, AMEX, Visa, MasterCharge, or EuroCard. For credit cards, please enter the information below:

Card #:__ __ __ __ - __ __ __ __ - __ __ __ __ - __ __ __ __ Expiration date: ____/____

Signature: ____

Where did you hear about or get a copy of our products?

____

International customers please see note on Ordering information.

winbatch functions

- Writing a Script
- Ways to run WinBatch Scripts

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 to the functions available in WinBatch.

Note: The functions listed under the **See Also** headings may be documented either in this Help file or in the **WIL Reference help file**.

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. Otherwise the command does nothing.

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 graghical dialogs, is to use the HTML Extender. For further information see the HTML extender help file.

The WinBatch setup program uses WinBatch box functions to display the GUI part of the user interface. Additional "box" wbt files can be found in the Winbatch\Samples directory.

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
Color Parameters

Additional Box functions are:

BoxButtonDraw(box ID, button ID, text, coordinates)
BoxButtonKill(box ID, button ID)
BoxButtonStat(box ID, button ID)
BoxButtonWait( )

BoxCaption(box ID, caption)

BoxColor(box ID, color, wash color)

BoxDestroy(box ID)

BoxDrawCircle(box ID, coordinates, style)

BoxDrawLine(box ID, coordinates)

BoxDrawRect(box ID, coordinates, style)

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

BoxesUp(coordinates, show mode)

BoxMapMode(box ID, map mode)

BoxNew(box ID, coordinates, style)

BoxPen(box ID, color, width)

BoxTextColor(box ID, color)

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

BoxUpdates(box ID, update flag)

Drawing Stack Management

BoxDataClear(box ID, tag)

BoxDataTag(box ID, tag)

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.

Note: In our shorthand method for indicating syntax 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.

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 *(both found in main WIL documentation)*

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 *(found in main WIL documentation)*

Breakpoint

Causes a breakpoint on the next statement when used with a script debugger.   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 a 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 help)

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 blacks lacks them all. A sample list of colors follow:

WHITE="255,255,255"
BLACK="0,0,0"
LTGRAY="192,192,192"
GRAY="128,128,128"
DKGRAY="64,64,64"
LTPURPLE="255,128,255"

RED="255,0,0"
GREEN="0,255,0"
BLUE="0,0,255"
YELLOW="255,255,0"
CYAN="0,255,255"
PURPLE="255,0,255"

DKRED="128,0,0"
DKGREEN="0,128,0"
DKBLUE="0,0,128"
DKYELLOW="128,128,0"
DKCYAN="0,128,128"
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	An ID number from 1 - 16 specifying the desired WinBatch box. Maximum boxes allowed, 16.
(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 reused, 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 buttons 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", "WinBatch Box Example - BoxButtonDraw %@CRLF%
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:

BoxButtonKill, BoxButtonStat, BoxButtonWait, BoxesUp, BoxNew

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", "WinBatch Box Example - BoxButtonKill
%@CRLF% 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:

BoxButtonDraw, BoxButtonStat, BoxButtonWait, BoxesUp, BoxNew

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", "WinBatch Box Example - BoxButtonStat
%@crlf% 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:

BoxButtonDraw, BoxButtonKill, BoxButtonWait, BoxesUp, BoxNew

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", "WinBatch Box Example - BoxButtonWait
%@crlf% 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:

BoxButtonDraw, BoxButtonKill, BoxButtonStat, BoxesUp, BoxNew

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", "WinBatch Box Example - BoxCaption
%@crlf%%@crlf% Keep your eye on 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"
WHITE="255,255,255"
RED="255,0,0"
GREEN="0,255,0"
BLUE="0,0,255"
PURPLE="255,0,255"
YELLOW="255,255,0"
CYAN="0,255,255"

DKGRAY="128,128,128"
GRAY="192,192,192"
DKRED="128,0,0"
DKGREEN="0,128,0"
DKBLUE="0,0,128"
DKPURPLE="128,0,128"
DKYELLOW="128,128,0"
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 which 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", "WinBatch Box Example - BoxDestroy %@crlf%%@crlf%
", @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", "WinBatch Box Example - 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:

BoxDrawLine, BoxDrawRect, BoxDrawText, BoxesUp, BoxNew

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", "WinBatch Box Example - 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:

BoxDrawCircle, BoxDrawRect, BoxDrawText, BoxesUp, BoxNew

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", "WinBatch Box Example - 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:

BoxDrawCircle, BoxDrawLine, BoxDrawText, BoxesUp, BoxNew

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 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 shouldn't.

(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 boxes boundaries if the allotted space is exceeded or size of the text is too large.

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, "300,300,500,500", "WinBatch Box Example - BoxDrawText ",
            @TRUE, 1)
BoxCaption(1, "WinBatch BoxDrawText Example")
BoxDrawText(1, "575,575,500,500", "Use BoxDrawText to display information to
            your user's. ", @TRUE, 1)
TimeDelay(5)
```

See Also:

BoxDrawCircle, BoxDrawLine, BoxDrawRect, BoxesUp, BoxNew

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("WinBatch BoxesUp Example",
         "BoxesUp can display a box in Normal Mode. ")
BoxesUp("200,200,800,800", @normal)
BoxDrawText(1, "500,200,500,200", "WinBatch Box Example - BoxesUp %@crlf%
Normal Mode", @FALSE, 1)
BoxCaption(1, "WinBatch BoxesUp Example - Normal Mode")

Message("WinBatch BoxesUp Example", "BoxesUp can display the box as an Icon.")
BoxDestroy(1)
BoxesUp("200,200,800,800", @icon)
BoxDrawText(1, "500,200,500,200", "WinBatch Box Example - BoxesUp %@crlf% Icon
            Mode", @FALSE, 1)
BoxCaption(1, "WinBatch BoxesUp Example - Icon Mode")

Message("WinBatch BoxesUp Example", "BoxesUp can display in a Zoomed mode.")
BoxDestroy(1)
BoxesUp("200,200,800,800", @zoomed)
BoxDrawText(1, "500,200,500,200", "WinBatch Box Example - BoxesUp %@crlf%
             Zoomed Mode", @FALSE, 1)
BoxCaption(1, "WinBatch BoxesUp Example - Zoomed Mode")

Message("WinBatch BoxesUp Example", "In addition, WinBatch can set a 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 functions "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	An ID number from 1 - 8 specifying the desired WinBatch box. Maximum boxes allowed, 8.
(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", "WinBatch Box Example - 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="128,128,128"
WHITE="255,255,255"	GRAY="192,192,192"
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", "WinBatch Box Example - 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")

delay(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"
WHITE="255,255,255"
RED="255,0,0"
GREEN="0,255,0"
BLUE="0,0,255"
PURPLE="255,0,255"
YELLOW="255,255,0"
CYAN="0,255,255"

DKGRAY="128,128,128"
GRAY="192,192,192"
DKRED="128,0,0"
DKGREEN="0,128,0"
DKBLUE="0,0,128"
DKPURPLE="128,0,128"
DKYELLOW="128,128,0"
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", "WinBatch Box Example-BoxTextColor", @True,
1)
      delay(2)
next
```

See Also:

BoxesUp, BoxNew, BoxColor, BoxTextFont, BoxPen

BoxTextFont

Sets the font for a WinBatch box.

Syntax:

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

Parameters:

(i) box ID	the ID number of the desired WinBatch box.
(s) name	name of font typeface, or "".
(i) size	size of font, in virtual units.
(i) style	style flags for font.
(i) font pitch, family and character set.	See below.

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.

If a blank string is specified for "name", the system will select the best available font based upon the other parameters supplied.

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 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.)

In addition to the existing flags for pitch and family, you can now specify one of the following flags specifying a character set (combined with the pitch and/or family flags using the binary "OR" ("|") operator):

Character Set:

ANSI_CHARSET	0 (default)
DEFAULT_CHARSET	256
SYMBOL_CHARSET	512

SHIFTJIS_CHARSET	32768 (Kanji)
HANGEUL_CHARSET	33024
GB2312_CHARSET	34304
CHINESEBIG5_CHARSET	34816
OEM_CHARSET	65280

The following flags are for Windows 95/98 and NT 4.0 only:

JOHAB_CHARSET	33280
HEBREW_CHARSET	45312
ARABIC_CHARSET	45568
GREEK_CHARSET	41216
TURKISH_CHARSET	41472
THAI_CHARSET	56832
EASTEUROPE_CHARSET	60928
RUSSIAN_CHARSET	52224
MAC_CHARSET	19712
BALTIC_CHARSET	47616

The character set flags will not override the typeface specified by the font "name" parameter. If you would rather specify a character set than a specific typeface, specify a blank string ("") for font "name".

Note: If you wish to use a Kanji (Japanese) font, you must specify the SHIFTJIS_CHARSET flag (32768).

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", "WinBatch Box Example-
                  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.  No bad, 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 a BoxUpdates
code=3.  Should be quick.  Don't blink.")
BoxUpdates(1,3)
Message(title,"That should have been pretty quick.  Next is 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, and maintains the 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 constantlt changing object(s).   When it comes time to update those objects, a **BoxDataClear** will erase all items added since the (BoxDataTag) "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 added since the (BoxDataTag) "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)
BoxDrawText(1, "0,200,1000,1000", "WinBatch Box Example - BoxDataClear ",
            @FALSE, 1)
BoxCaption(1, "WinBatch BoxDataClear Example")
BoxDataTag(1, "tag1")

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")
BoxDrawText(1, "0,240,1000,1000", "BoxDataClear - 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, 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)
BoxDrawText(1, "0,200,1000,1000", "WinBatch Box Example - BoxDataTag ",
            @FALSE, 1)
BoxCaption(1, "WinBatch BoxDataTag Example")
BoxDataTag(1, "tag1")

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")
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

WIL 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, TAPI, and other important Application Program Interface 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). 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. Up to 10 extender Dlls may be added. 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 exender in each WIL script that requires it.

To use a WIL extender, at the top of each script in which you use network commands add the appropriate extender with the AddExtender command.

```
  AddExtender(extender filename)
```

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

Some of the WIL extender DLL's, that are not included on the distribution disks, can be downloaded from our website at http://www.winbatch.com

- NetWare Extender
- Win32 Network Extender
- Basic Win 3.1 Network Extender
- Multinet / WinForWrkGrp Network Extender

NetWare Extenders

- WIL Extenders
- Win32 Network Extender
- Basic Win 3.1 Network Extender
- Multinet / WinForWrkGrp Network Extender

These Windows Interface Language network extenders provide standard support for Novell networks.   They may be used in addition with other extenders, such as the Win32 Network extender or with each other.

The NetWare functions can be used within WIL scripts or can be compiled into WIL executables. NetWare 3x and NetWare 4x extenders are available for both Windows 3.1 and Windows 95/98.   Windows NT requires the 32 bit NetWare 3x or 4x extenders be used in conjunction with the NetWare Client 32.

Selecting the appropriate extender for use with your system can be a little tricky.   Both NetWare 3 and NetWare 4 have their own extenders and own set of functions.   In addition, the extender must match the Windows operating system.

These WIL extenders provide standard support for Novell 3.x networks.

Windows 3.1 and Windows for WorkGroups

```
AddExtender("wwn3x16i.dll")
```

Other required DLL's: nwcalls.dll

Windows 95/98 and Windows NT require separate extenders. For Windows 95/98 with a Windows 3.1 NetWare Client (not Windows NT).

```
AddExtender("wwn3z32i.dll")
```

Other required DLL's: nwcalls.dll, wwn3z16i.dll

Windows 95/98 and INTEL versions of Windows NT running NetWare Client 32

```
AddExtender("wwn3x32i.dll")
```

Other required DLL's: nwcalls.dll

This   dll is designed to work with NetWare Client 32.   NetWare Client 32 can be downloaded from: *http://netwire.novell.com/home/client/client32*

These WIL extenders provide standard support for Novell 4.x networks.

Windows 3.1 and Windows for WorkGroups

```
AddExtender("wwn4x16i.dll")
```

Other required DLL's: nwcalls.dll, nwnet.dll, nwlocale.dll.

Windows 95/98 and INTEL versions of Windows NT running

NetWare Client 32

```
AddExtender("wwn4x32i.dll")
```

Other required DLL's: nwcalls.dll

This dll is designed to work with NetWare Client 32. NetWare Client 32 can be downloaded from: *http://netwire.novell.com/home/client/client32.*

For more information and a list of functions see the NetWare Extender Help File.

Win32 Network Extenders

These Windows Interface Language Network extenders provide standard support for computers running 32 bit versions of Windows, such as Windows NT and Windows 95/98.

There are 3 separate extenders for Windows 95-98 client /95-98 server (WWW9532i.DLL), Windows 95-98 client /NT server (WWW9X32i.DLL) and Windows NT (WWWNT32i.DLL).

For Windows 95-98 client /95-98 server use…
(same extender for both 95 and Windows 98)

```
AddExtender("WWW9532i.DLL")
```

Other required DLL's: none

For Windows 95-98 client /NT server use…

```
AddExtender("WWW9X32i.DLL")
```

Other required DLL's: RADMIN32.DLL, RLOCAL.DLL

For Windows NT client / NT server use…

```
AddExtender("WWWNT32i.DLL")
```

Other required DLL's: none

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

- WIL Extenders
- NetWare Extender
- Basic Win 3.1 Network Extender
- Multinet / WinForWrkGrp Network Extender

Basic Network Extender

This particular Dll, **wwn3a16i.dll**, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors.   It provides basic links to networks (like Novell) for the Windows 3.1 environment.

It is not designed for Windows for WorkGroups, Windows 95, or for other versions of Windows.   Your system may require the use of one of the alternate extender Dlls available for those products.   In addition, some networks, like Novell, have better, more fully featured extenders available.

For Windows 3.1 use…

```
AddExtender("WWW3A16i.dll")
```

Other required DLL's: none

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

- WIL Extenders
- NetWare Extender
- Win32 Network Extender
- Multinet / WinForWrkGrp Network Extender

Multinet Network Extender

This particular Dll, wwwn16i.dll, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. It is designed for versions of Windows 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 and are designed to work in conjunction with other extenders for other networks, such as the extenders for Novell networks. Your system may require the use of one the alternate Dlls.

For Windows for WorkGroups use…

```
AddExtender("WWWN16i.dll")
```

Other required DLL's: none

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

- WIL Extenders
- NetWare Extender
- Win32 Network Extender
- Basic Win 3.1 Network Extender

AddExtender(filename)

Installs a WIL extender Dll.

Syntax:

AddExtender(filename)

Parameters:

(s) filename	WIL extender Dll filename.

Returns:

(i)	**@TRUE** if function succeeded. **@FALSE** if function failed.

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, TAPI, and other important Application Program Interface 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). 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.

Use this function to install extender Dlls as required. Up to 10 extender Dlls may be added. The total number of added items may not exceed 200 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 in each WIL script that requires it.

The documentation for the functions added are supplied either in a separate manual or disk file that accompanies the extender Dll.

Example:

```
; Add vehicle radar processing dll controlling billboard visible to
; motorists, and link to enforcement computers.
; The WIL Extender SPEED.DLL adds functions to read a radar speed
; detector(GetRadarSpeed) , put a message on a billboard visible to
; the motorist (BillBoard), take a video of the vehicle (Camera), and
; send a message to alert enforcement personnel (Alert) that a
; motorist in  violation along with a picture id number to help
; identify the offending vehicle and the speed which it was going.
;
AddExtender("SPEED.DLL")
BillBoard("Drive Safely")
While @TRUE
     ; Wait for next vehicle
     while GetRadarSpeed()<5    ; if low, then just radar noise
            Yield                     ; wait a bit, then look again
     endwhile
     speed=GetRadarSpeed()             ; Something is moving out there
     if speed < 58
            BillBoard("Drive Safely")  ; Not too fast.
     else
```

```
                    if speed < 63
                              BillBoard("Watch your Speed")      ; Hmmm a hot one
                    else
                              if speed < 66
                                BillBoard("Slow Down")     ; Tooooo fast
                              else
                                BillBoard("Violation  Pull Over")
                                pictnum = Camera()  ; Take Video Snapshot
                                Alert(pictnum, speed); Pull this one over
                              endif
                    endif
          endif
endwhile
```

See Also:

DllCall *(found in main WIL documentation)*

WIL Dialog Editor

- Getting Started
- Using the Dialog Editor
- Dialog Box Caption
- Control Attributes

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

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.

It provides a convenient method of creating dialog box templates for use with the **Dialog** function.

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

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.

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
- Dialog Box Caption
- Control Attributes

Using the Dialog Editor is easy. These steps offer a general overview for using the Dialog Editor as well as a quick way to become comfortable with dialog box construction.

1. **Run** the Dialog Editor.
2. Familiarize yourself with three standard menus in this program; **FILE**, **EDIT**, and **HELP**.
3. Give your **dialog box a name**.
4. Decide the **size** of your dialog box.
5. **Add a control**, i.e. an OK button.
6. **Save** your dialog box.
7. **Use your dialog box** within another script

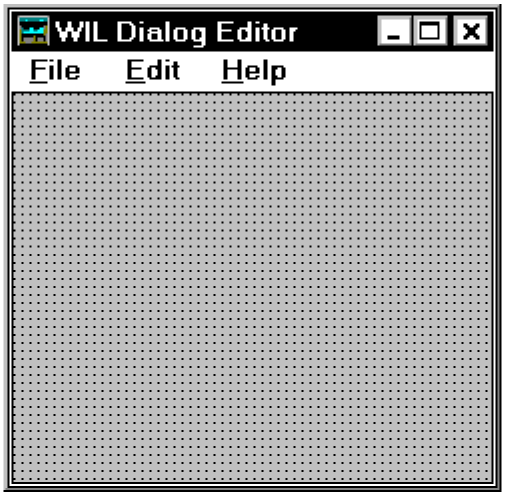
Run the Dialog Editor

Launch the dialog editor executable. There is both a 16 bit and a 32 bit version of the Dialog Editor.

For 16 bit Windows use: wwdlg16i.exe

For 32 bit Windows use: WIL Dialog Editor.exe

The editor will look like the following:



- Getting Started
- Using the Dialog Editor
- Dialog Box Caption
- Control Attributes
- Menu Commands

Menu Commands

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

- Getting Started
- Using the Dialog Editor
- Dialog Box Caption
- Control Attributes

File

- Getting Started
- Using the Dialog Editor
- Edit
- Help

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

- Getting Started
- Using the Dialog Editor
- File
- Help

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.

GUI Font

Uses the new style font in the dialog for Windows 95/98 and NT 4.0. If running Windows 95/98 or NT 4.0 the GUI font is used as default.

note: Intcontrol(52,0,1,0,0) can change font to be displayed as system font.

System Font

Uses the System font in the dialog for Windows 3.x and NT 3.5. If running Windows 3.x or NT 3.5 the Sytem font is used as default.

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

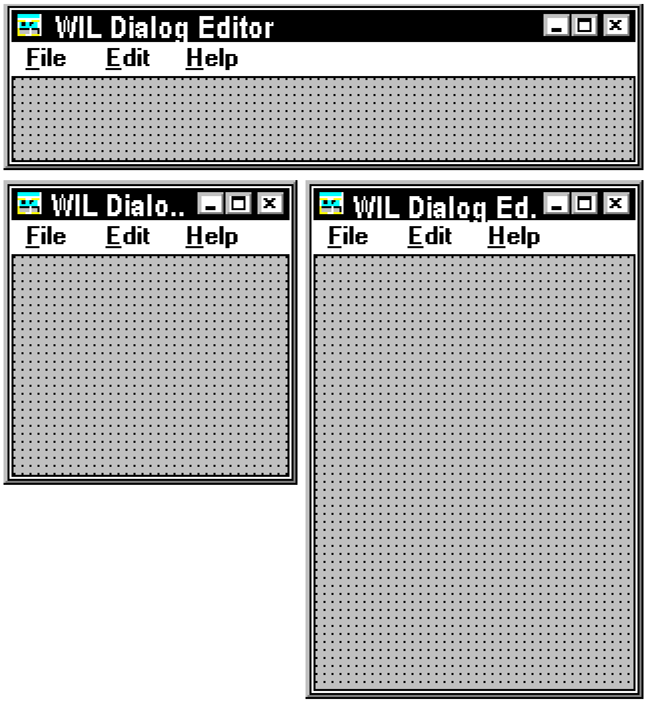
Displays the WIL Dialog Editor About dialog which includes the version number of the program.

- Getting Started
- Using the Dialog Editor
- File
- Edit

Size the Dialog Box

- Getting Started
- Using the Dialog Editor
- Dialog Box Caption
- Control Attributes

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



Using the Dialog Editor

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

The WIL Dialog Editor allows you to create dialog box templates for WIL using the WDL* format. As you visually design your dialog box on the screen, the Dialog Editor writes the WIL script statements necessary to create and display the dialog. **Save** the template as either a .WDL files or to the Windows Clipboard.

Take a peek at the resulting script with the **File ShowScript** command to begin to get used to what WIL dialog scripts look like. For more information on what the code means see **Decipher the Script**.

Finally, **incorporate the code** created with the Dialog Editor into a WIL script.

* WDL is the extension automatically given to WIL scripts created with the Dialog Editor.

- Getting Started
- Running WDL Scripts
- Saving WDL Scripts
- Decipher the Script
- Dialog Box Caption
- Control Attributes
- Menu Commands

Running WDL Scripts

- Getting Started
- Saving WDL Scripts
- Decipher the Script
- Dialog Box Caption
- Control Attributes

The Dialog Editor saves dialog boxes with an extension of .WDL which is pre-associated with the WIL interpreter.   There are several ways to utilize finished dialog boxes.

Manually, .WDL files can be launched from the Windows 95/98/NT Explorer by double-clicking on the filename or by entering the filename into the Windows 95/98 "Start…Run.." menu option.

Within WIL scripts, .WDL files can be initiated using either the Run or the Call command.   Remember to **set your variables** either in the WIL script or at the top of the .WDL code.

```
Run("example.wdl", "")
;or
Call("example.wdl", "")
```

With a little copying and pasting, a dialog box can be completely integrated into a WIL script.   The Dialog Editor **File**/**Save to Clipboard** feature allows an easy way to put the code into the clipboard.   Use the keyboard command 'Control v' to paste the code into the script.   The entire code can be placed at the point in the script at which it will be executed.

However, if the dialog box needs to be accessed multiple times, it is more efficiently utilized as a sub-routine.   A **GoSub** command will send the WIL processing to the dialog sub-routine and return to the point of origin. (See the WIL manual or on-line help for more information on **GoSub**)   Alternatively, execution of a dialog box will occur whenever the dialogs final statement is encountered in the script.

```
ButtonPushed=Dialog("mydialog")
```

Example:

```
miniFormat=`WWWDLGED,5.0`

miniCaption=`Mini Example`
miniX=100
miniY=42
miniWidth=132
miniHeight=73
miniNumControls=4

mini01=`40,56,51,DEFAULT,PUSHBUTTON,DEFAULT,"GO",1`
mini02=`6,22,51,DEFAULT,RADIOBUTTON,station,"am",1`
mini03=`6,38,51,DEFAULT,RADIOBUTTON,station,"fm",2`
mini04=`28,4,75,DEFAULT,STATICTEXT,DEFAULT,"Just a quickie example"`

ButtonPushed=Dialog("mini")
Message("Times Run", "1")
```

```
ButtonPushed=Dialog("mini")
Message("Times Run", "2")

ButtonPushed=Dialog("mini")
Message("Times Run", "3")
```

Saving WDL Scripts

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.

- Getting Started
- Running WDL Scripts
- Decipher the Script

ShowScript

Here is an example of what a WIL Dialog Editor script looks like. For information on what it all means, see **Decipher the Script**

- Using the Dialog Editor
- Running WDL Scripts

```
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")
```

Decipher the Script

- Using the Dialog Editor
- Control Attributes
- Running WDL Scripts
- Saving WDL Scripts
- ShowScript

The Dialog Editor follows a specific format when creating your script. For example, here is a dialog box script we created.

The first line sets the format and specifies the version of the Dialog Editor being used.

```
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.

```
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=`84,41,38,DEFAULT,RADIOBUTTON,music,"Rock",3`
```

When the first line in the example above is broken down, the parts are as follows.

Code	**Definition**
Example	Dialog Variable Name
01	Control Number
27,113,76,DEFAULT	Coordinates of the control
PUSHBUTTON	Control Type
"DEFAULT"	Variable name
OK	Text
1	Value

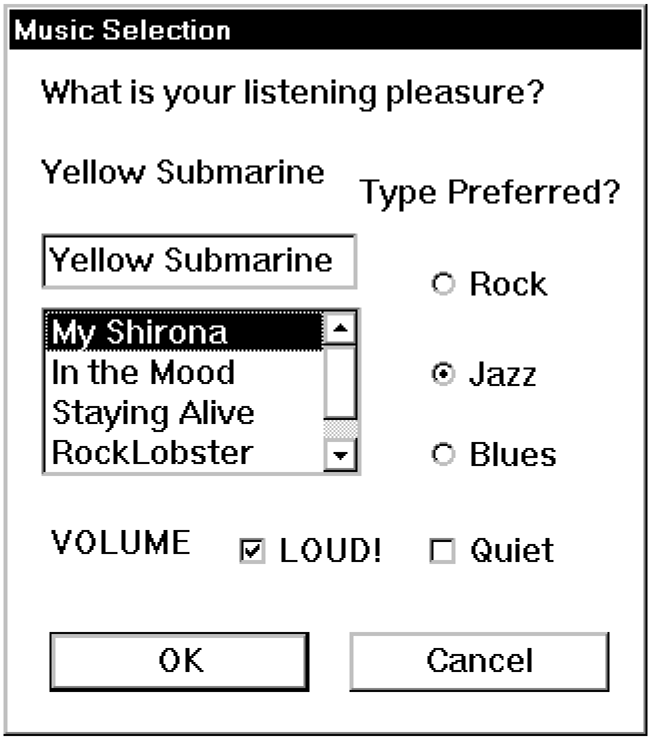
Each Dialog script will end with the following line, making it easy to test the PushButton return values.

```
ButtonPushed=Dialog("Example")
```

Assembled with the **variables**, the completed script looks like the following.

```
;set variables
;the list for the item box.
tunes="My Shirona%@tab%In the Mood%@tab%StayingAlive%@tab%RockLobster%@tab%Tequila"
;the contents of the varytext.
song="Yellow Submarine"
music=2  ;sets this radiobutton as default
volume=1 ;pre-selects checkbox.

ExampleFormat=`WWWDLGED,5.0`

ExampleCaption=`Music Selection`
ExampleX=120
ExampleY=50
ExampleWidth=129
ExampleHeight=138
ExampleNumControls=13

Example01=`8,116,52,DEFAULT,PUSHBUTTON,DEFAULT,"OK",1`
Example02=`68,116,52,DEFAULT,PUSHBUTTON,DEFAULT,"Cancel",0`
Example03=`84,75,38,DEFAULT,RADIOBUTTON,music,"Blues",1`
Example04=`84,59,38,DEFAULT,RADIOBUTTON,music,"Jazz",2`
Example05=`84,41,38,DEFAULT,RADIOBUTTON,music,"Rock",3`
Example06=`46,94,38,DEFAULT,CHECKBOX,volume,"LOUD!",1`
Example07=`84,94,38,DEFAULT,CHECKBOX,volume2,"Quiet",2`
Example08=`8,94,38,DEFAULT,STATICTEXT,DEFAULT,"VOLUME"`
Example09=`6,4,112,DEFAULT,STATICTEXT,DEFAULT,"What is your listening pleasure?"`
Example10=`6,51,65,40,ITEMBOX,tunes,DEFAULT`
Example11=`70,24,56,DEFAULT,STATICTEXT,DEFAULT,"Type Preferred?"`
Example12=`6,20,60,DEFAULT,VARYTEXT,song,"Choose a title"`
Example13=`6,36,64,DEFAULT,EDITBOX,song," "`

ButtonPushed=Dialog("Example")
```

Here is the completed dialog box.



Music Selection

What is your listening pleasure?

Yellow Submarine

Type Preferred?

Yellow Submarine

My Shirona
In the Mood
Staying Alive
RockLobster

○ Rock

● Jazz

○ Blues

VOLUME ☑ LOUD! □ Quiet

Setting Variables

Any information which is needed by the Dialog Box Controls should defined 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.

- Using the Dialog Editor
- Control Attributes
- Running WDL Scripts

In the **script example**, the variables are set at the top.

```
;;set variables

;;the list for the item box.
tunes="My Shirona%@tab%In the Mood%@tab%StayingAlive%@tab%RockLobster"

;;the contents of the varytext.
song="Yellow Submarine"
music=2  ;sets the radiobutton as default
volume=1 ;pre-selects checkbox.
```

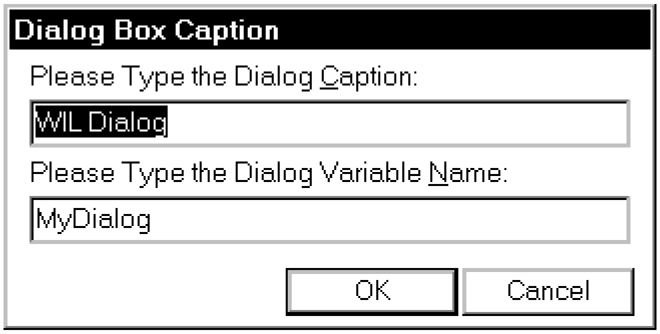
Dialog Box Caption

Dialog boxes have both an internal and external name. 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 display the caption box, double click with the left mouse button on the workspace background, (not on a control), or from the **Edit** menu select **Change Caption/Name**.

- Getting Started
- Using the Dialog Editor
- Control Attributes
- Running WDL Scripts
- Saving WDL Scripts
- Decipher the Script



Dialog Box Caption

Please Type the Dialog Caption:

WIL Dialog

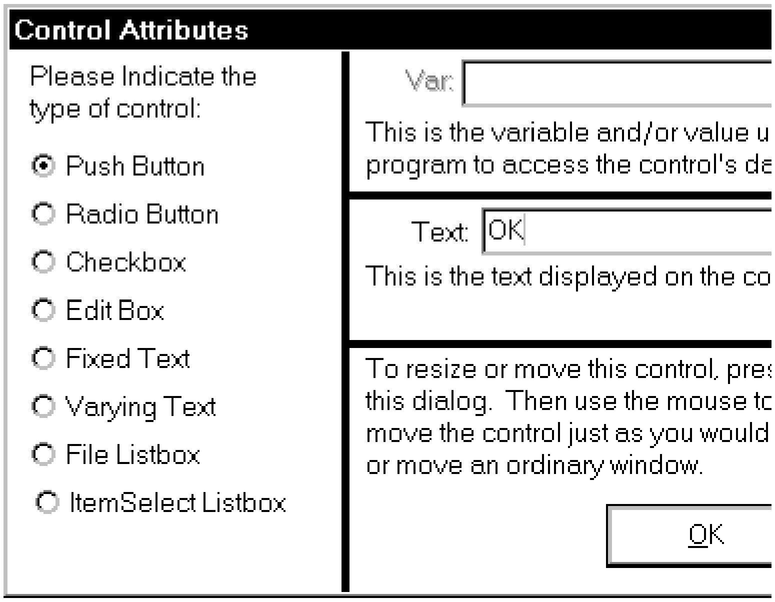
Please Type the Dialog Variable Name:

MyDialog

Control Attributes

The Dialog Editor has a variety of controls which can be selected to create a customizable user interface.

To add a control, double click with the right mouse button where you want the control or from the **Edit** menu, select **Add Control**. Fill in the information in resulting dialog box about the control.



Control Attributes

Please Indicate the type of control:

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

Var:

This is the variable and/or value u
program to access the control's de

Text: OK

This is the text displayed on the co

To resize or move this control, pres
this dialog. Then use the mouse tc
move the control just as you would
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 and select the OK button.
After a control has been created in your dialog box you can move it, size it or delete it.

> 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 over the control and press the delete key.

Some of the Controls require extra knowledge or special handling.

- Using the Dialog Editor
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Push Button

When creating **Push Buttons**, each button must have a separate value. To keep your programming consistent, we recommend assigning the value of 1 to your "OK" button equivalent and the value of 0 to your "Cancel" button equivalent.

The Dialog Editor adds a line to the end of your script which helps to test return values.

```
Buttonpushed=Dialog"MyDialog"
```

To test the return value do the following:

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

"Cancel" or the value 0 will generally look for a label **:cancel**. If no such label is found, the script will exit. For more information on "Cancel", see the Windows Interface Language manual or on-line WIL help file.

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Radio Button

Radio Buttons are used to select one item over another.

The variable assigned to the Radio Button is the same for each of the choices but the values are different. The Dialog Editor can have a maximum of 9 choices per variable.

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 but the text and the values are different.

Note: Radio Button cannot have a value of 0.

To test the return value, the variable can be placed in an If structure.

```
If music == 1
     Message("Music", "Let's play the blues.")
else
     Message("Music", "Let's play the Jazz.")
endif
```

Don't limit yourself to using if/endif statements. The **Switch** structure provides a more efficient way to test multiple values. For more information on **Switch**, see the Windows Interface Language manual or on-line WIL help file.

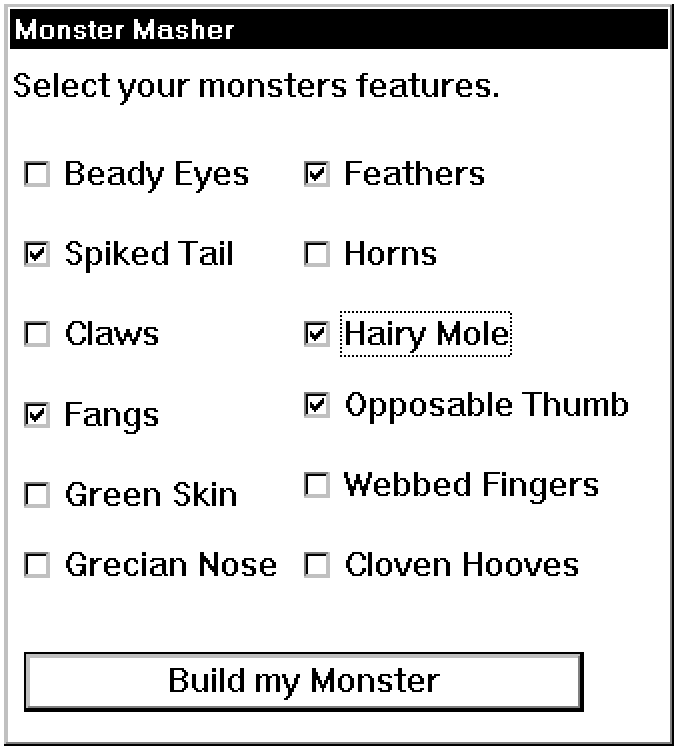
- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Check Box

The Check Box offers a way to present a variety of options. Each Check Box has its own specific information.   Variable, Value and Text are all different, allowing the user to select more than one.   Any number may be marked or left unmarked.

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Example:



Monster Masher

Select your monsters features.

☐ Beady Eyes   ☑ Feathers

☑ Spiked Tail   ☐ Horns

☐ Claws   ☑ Hairy Mole

☑ Fangs   ☑ Opposable Thumb

☐ Green Skin   ☐ Webbed Fingers

☐ Grecian Nose   ☐ Cloven Hooves

Here's the script which created this dialog box.

```
MonsterFormat=`WWWDLGED,5.0`

MonsterCaption=`Monster Masher`
MonsterX=63
MonsterY=79
MonsterWidth=128
MonsterHeight=141
MonsterNumControls=14

Monster01=`3,38,51,DEFAULT,CHECKBOX,tail,"Spiked Tail",1`
Monster02=`59,38,51,DEFAULT,CHECKBOX,horn,"Horns",1`
Monster03=`3,54,51,DEFAULT,CHECKBOX,claw,"Claws",1`
Monster04=`3,70,51,DEFAULT,CHECKBOX,teeth,"Fangs",1`
Monster05=`3,22,51,DEFAULT,CHECKBOX,eye,"Beady Eyes",1`
```

```
Monster06=`3,6,104,DEFAULT,STATICTEXT,DEFAULT,"Select your monsters features."`
Monster07=`6,121,112,DEFAULT,PUSHBUTTON,DEFAULT,"Build my Monster",1`
Monster08=`59,54,51,DEFAULT,CHECKBOX,mole,"Hairy Mole",1`
Monster09=`3,86,51,DEFAULT,CHECKBOX,flesh,"Green Skin",1`
Monster10=`3,102,51,DEFAULT,CHECKBOX,nose,"Grecian Nose",1`
Monster11=`59,86,62,DEFAULT,CHECKBOX,hands,"Webbed Fingers",1`
Monster12=`59,102,60,DEFAULT,CHECKBOX,foot,"Cloven Hooves",1`
Monster13=`59,22,51,DEFAULT,CHECKBOX,wings,"Feathers",1`
Monster14=`59,70,67,DEFAULT,CHECKBOX,thumb,"Opposable Thumb",1`

ButtonPushed=Dialog("Monster")
```

To test the return value, the variables can be placed in If structures.

```
If mole && claw == 1
     Message("Monster", "If you scratch my back, I'll bleed to death.")
endif
If eye || thumb
     Message("Not a monster", 'Jack is a psychopath, "Here's Johnny!"')
endif
```

Logical operators can be used when checking for the return of several variables; && is logical *and*, || is logical *or*. For more information on operators, see the Windows Interface Language manual or on-line WIL help file.

Edit Box

Edit Box creates a dialog in which a choice can be entered by default and then altered by the user.

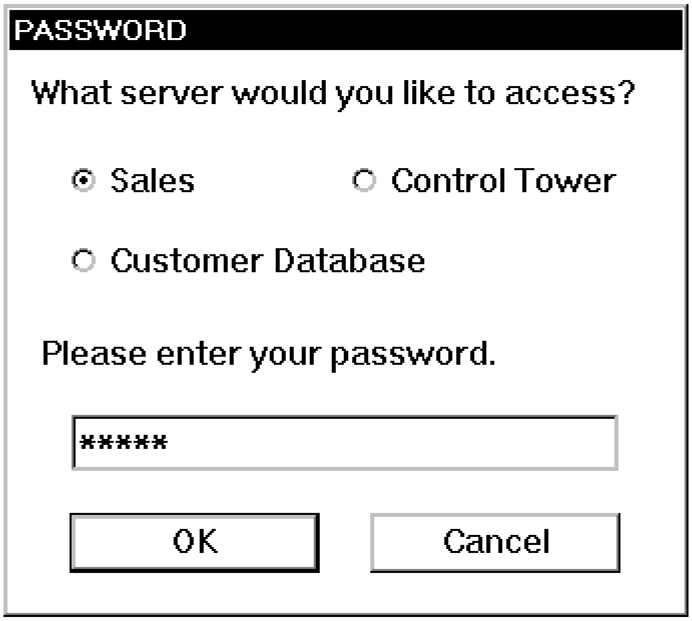
Variable names that begin with "PW_", will be treated as password fields causing asterisks to be echoed for the actual characters that the user types.

Note: The **AskPassword** function is a simple way of asking for a password. When a dialog must include additional information, the Dialog Editor can be used to create a custom box.

Here's an example of how to incorporate a password field into a dialog box.

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Example:



PASSWORD

What server would you like to access?

● Sales ○ Control Tower

○ Customer Database

Please enter your password.

*****

This box was created by the following script.

```
:top
;set the password variables
PW_pass1=""
PW_pass2=""
passwordFormat=`WWWDLGED,5.0`

passwordCaption=`PASSWORD`
passwordX=54
passwordY=70
passwordWidth=118
```

```
passwordHeight=82
passwordNumControls=4

password01=`3,59,51,DEFAULT,PUSHBUTTON,DEFAULT,"OK",1`
password02=`62,59,51,DEFAULT,PUSHBUTTON,DEFAULT,"Cancel",0`
password03=`3,25,110,DEFAULT,EDITBOX,PW_pass1,""`
password04=`3,8,110,DEFAULT,STATICTEXT,DEFAULT,"Please enter your new password."`
;this statement initializes the box.
ButtonPushed=Dialog("password")
```

Minor changes can be made to the dialog box without having to include the entire dialog box code.

```
;;initialize the box again with a couple of minor changes.
password03=`3,25,110,DEFAULT,EDITBOX,PW_pass2,""`
password04=`3,8,110,DEFAULT,STATICTEXT,DEFAULT,"Please verify your new password."`
ButtonPushed=Dialog("password")
```

To test the return value, the variables can be placed in If structures.

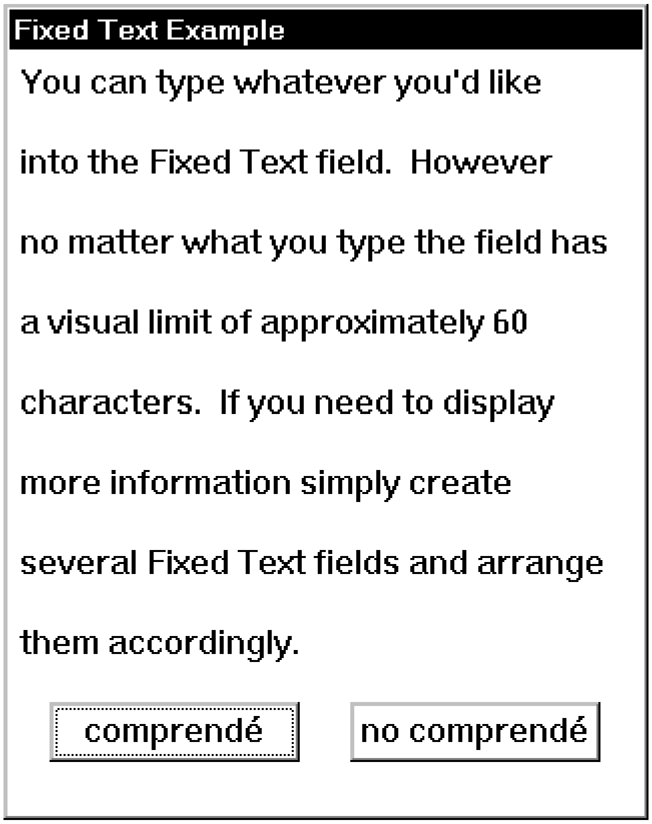
```
;test the return values
If PW_pass1 == PW_pass2
        Message("Password", "Thanks, your new password is in effect")
        Fail=0
else
        Message("Password Failed", "They didn't match, please try again.")
        Fail=1
endif
If Fail == 1 then goto top
```

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, its display capability is limited to 60 characters.

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Example:



Fixed Text Example

You can type whatever you'd like

into the Fixed Text field. However

no matter what you type the field has

a visual limit of approximately 60

characters. If you need to display

more information simply create

several Fixed Text fields and arrange

them accordingly.

This box was created by the following script.

```
fixedtextFormat=`WWWDLGED,5.0`

fixedtextCaption=`Fixed Text Example`
fixedtextX=119
fixedtextY=68
fixedtextWidth=130
fixedtextHeight=155
fixedtextNumControls=10

fixedtext01=`3,4,107,DEFAULT,STATICTEXT,DEFAULT,"You can type whatever you'd like "`
fixedtext02=`3,33,121,DEFAULT,STATICTEXT,DEFAULT,"no matter what you type, the field
has"`
```

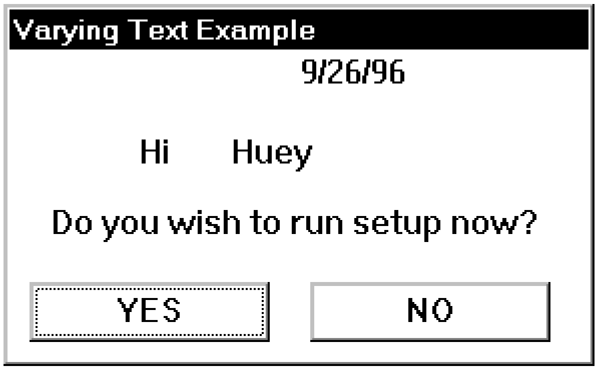
```
fixedtext03=`3,62,120,DEFAULT,STATICTEXT,DEFAULT,"characters.  If you need to
display"`
fixedtext04=`3,76,115,DEFAULT,STATICTEXT,DEFAULT,"more information, simply create "`
fixedtext05=`3,91,120,DEFAULT,STATICTEXT,DEFAULT,"several Fixed Text fields and
arrange "`
fixedtext06=`3,19,112,DEFAULT,STATICTEXT,DEFAULT,"into the Fixed Text field.
However,"`
fixedtext07=`3,48,115,DEFAULT,STATICTEXT,DEFAULT,"a visual limit of approximately 60
"`
fixedtext08=`3,105,67,DEFAULT,STATICTEXT,DEFAULT,"them accordingly."`
fixedtext09=`8,131,51,DEFAULT,PUSHBUTTON,DEFAULT,"comprendé",1`
fixedtext10=`68,131,51,DEFAULT,PUSHBUTTON,DEFAULT,"no comprendé",0`

ButtonPushed=Dialog("fixedtext")
```

Varying Text

Varying Text is used to display data which may change, like a date or a password.

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Example:



Varying Text Example

9/26/96

Hi Huey

Do you wish to run setup now?

This box was created by the following script.

```
;Grab the date for display in the dialog box.
dadate=Timedate()
wkday=ItemExtract(1, dadate, " ")
date=ItemExtract(3, dadate, " ")
day=StrCat(wkday, " ", date)

; Grab the users name for display in the dialog box.
user= Environment("USER")

varytextFormat=`WWWDLGED,5.0`

varytextCaption=`Varying Text Example`
varytextX=110
varytextY=78
varytextWidth=119
varytextHeight=64
varytextNumControls=6

varytext01=`38,16,48,DEFAULT,VARYTEXT,user,"xxx"`
varytext02=`4,46,48,DEFAULT,PUSHBUTTON,DEFAULT,"YES",1`
varytext03=`0,16,36,DEFAULT,STATICTEXT,DEFAULT,"                Hi "`
varytext04=`8,30,100,DEFAULT,STATICTEXT,DEFAULT,"Do you wish to run setup now?"`
varytext05=`60,46,48,DEFAULT,PUSHBUTTON,DEFAULT,"NO",0`
varytext06=`59,1,56,DEFAULT,VARYTEXT,date,"day"`

ButtonPushed=Dialog("varytext")
```

File Listbox

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect 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 filemask. i.e.

```
  wbtfiles="C:\WINBATCH\*.WBT"
```

This box can be tied with the variable to an Edit Box or to Varying Text. When the user chooses a file, it will be displayed in the Edit Box. Varying Text will display the actual variable.

For multiple selections or to display pre-defined lists, use the Dialog Editor's Item Listbox option.

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.

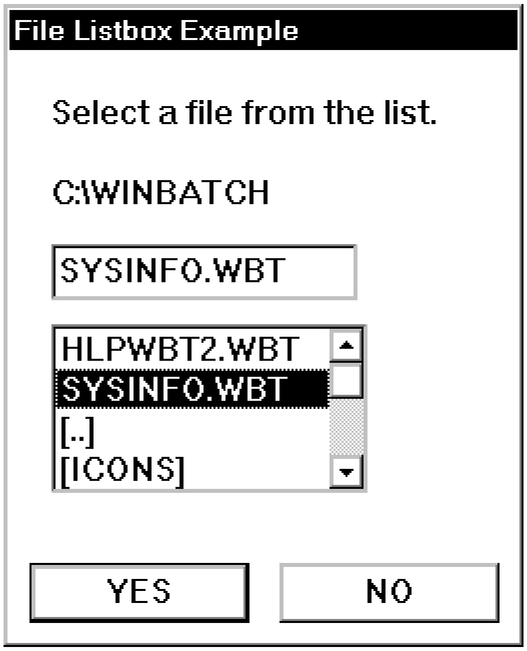
```
  IntControl(4, 0,0,0,0)
```

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

```
    "NOFILESELECTED"
```

For more information on **IntControl**, see the Windows Interface Language manual or on-line WIL help file.

Example:



File Listbox Example

Select a file from the list.

C:\WINBATCH

SYSINFO.WBT

In the example code below, notice that the variable 'wbtfiles' is used for the control attributes Varying Text, Edit Box and File Listbox.

```
wbtfiles="C:\WINBATCH\*.WBT"

filelistFormat=`WWWDLGED,5.0`

filelistCaption=`File Listbox Example`
filelistX=184
filelistY=50
filelistWidth=105
filelistHeight=120
filelistNumControls=6

filelist01=`8,24,84,DEFAULT,VARYTEXT,wbtfiles,"xxx"`
filelist02=`4,102,44,DEFAULT,PUSHBUTTON,DEFAULT,"YES",1`
filelist03=`8,54,65,43,FILELISTBOX,wbtfiles,DEFAULT`
filelist04=`8,8,84,DEFAULT,STATICTEXT,DEFAULT,"Select a file from the list."`
filelist05=`54,102,44,DEFAULT,PUSHBUTTON,DEFAULT,"NO",0`
filelist06=`8,38,51,DEFAULT,EDITBOX,wbtfiles,"ooo"`

ButtonPushed=Dialog("filelist")

Message("File Selected", wbtfiles)
```

ItemSelect Listbox

The ItemSelect Listbox allows the user to choose an item from a list box. This option is similar to the WIL commands **AskItemList**, and **ItemSelect**.

Set a variable to display a list of items. Use **@tab,** a predefined constant, as the delimiter. I

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

By default, the ItemSelect Listbox allows multiple selections. To disable this feature use IntControl 33.

```
IntControl(33, 0, 0, 0, 0)
```

For more information on **IntControl**, see the Windows Interface Language manual or on-line WIL help file.

- Control Attributes
- Push Button
- Radio Button
- Check Box
- Edit Box

- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

WinInfo

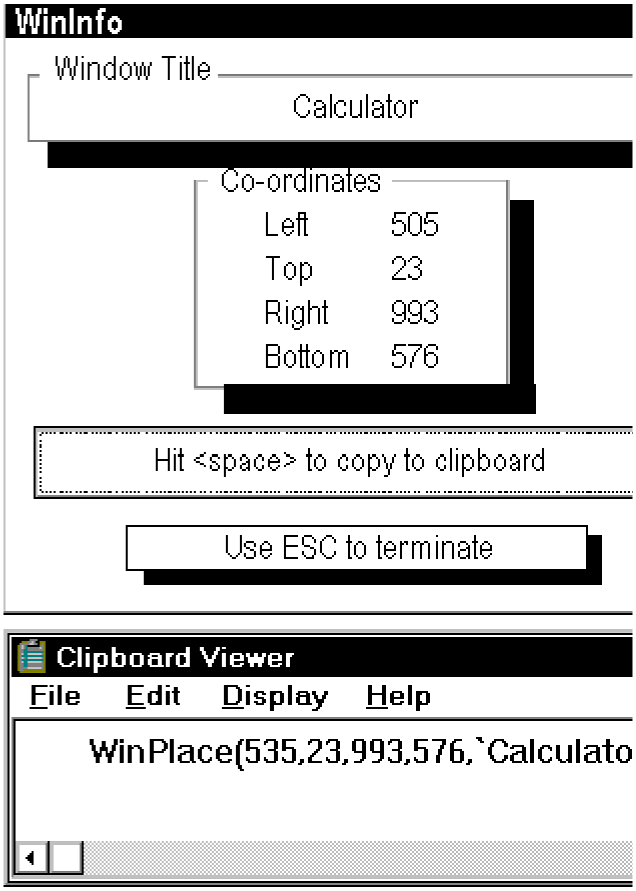
WinInfo can grab a window position settings from windows displayed on your monitor.

WinInfo is a handy window name and position grabber

The **WinInfo** utility (see the Filename Appendix 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. It puts the text into the Clipboard, from which you can paste it into your WIL program.

WinInfo 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 users 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.



WinInfo

Window Title

Calculator

Co-ordinates

Left	505
Top	23
Right	993
Bottom	576

Hit <space> to copy to clipboard

Use ESC to terminate

Clipboard Viewer

```
WinPlace(535,23,993,576,`Calculato
```

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

FileMenu

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

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

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

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

WinBatch Help File

System Requirements

FileMenu requires a version of Windows supporting the Windows Explorer, such as Windows 95/98, NT 4.0.

Installation

FileMenu is installed during the normal setup of WinBatch 95.

Operation

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

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

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

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

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

- FileMenu
- Menu Files
- Usage Tips, Known Problems and Limitations, etc.

WinBatch Help File

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.** 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 further in the Windows Interface Language reference manual and in the WIL on-line help file under the topic Menu Files.

- FileMenu
- Using the "all filetypes" FileMenu
- Creating/Modifying File-Specific Menus
- FileMenu.ini

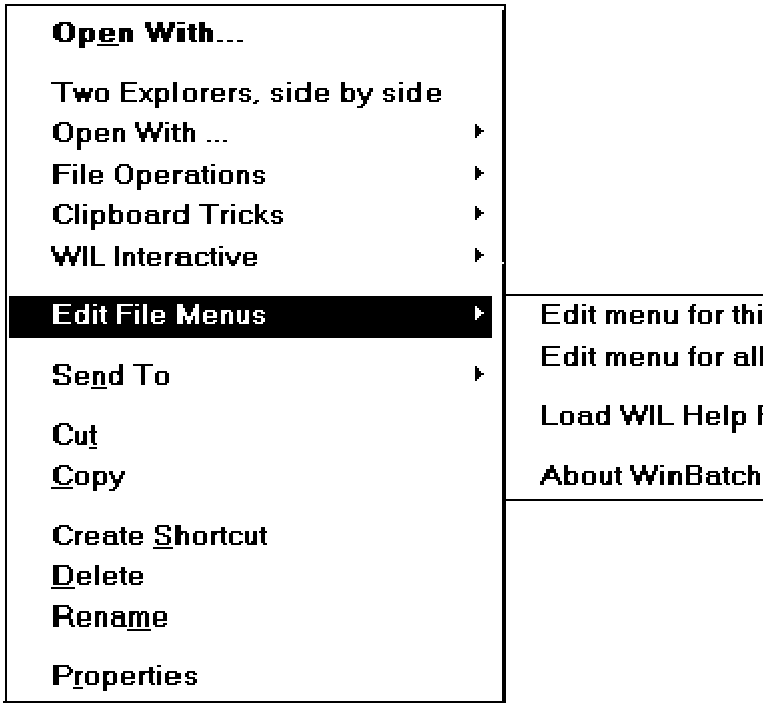
WinBatch Help File

Using the "all filetypes" FileMenu

- FileMenu
- Menu Files
- Creating/Modifying File-Specific Menus
- FileMenu.ini

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

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



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

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

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

WinBatch Help File

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 clicked on with the right mouse button.

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

- FileMenu
- Menu Files
- Using the "all filetypes" FileMenu
- FileMenu.ini

WinBatch Help File

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 in your **FileMenu** directory. Or, you can specify a full path in Filemenu.ini.

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

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

Note: Extensions can be longer than three characters.

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

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

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

- FileMenu
- Menu Files
- Using the "all filetypes" FileMenu
- Creating/Modifying File-Specific Menus

WinBatch Help File

Usage Tips, Known Problems and Limitations, etc.

- FileMenu
- Menu Files

Functions

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

CurrentFile
CurrentPath
CurrFilePath

The following functions are NOT supported:

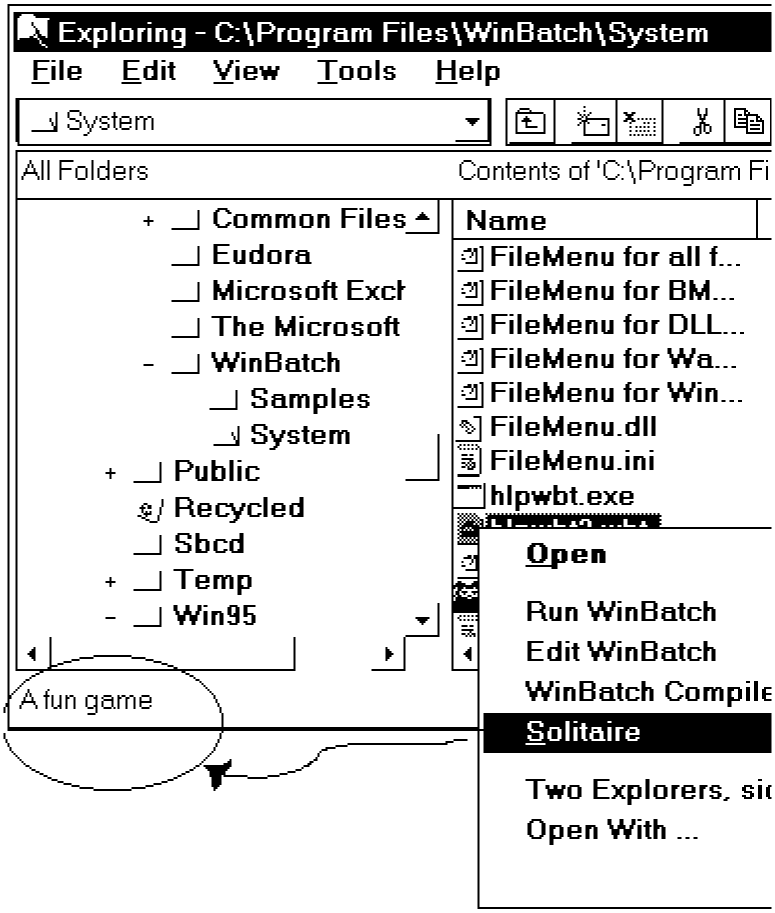
IsMenuChecked
IsMenuEnabled
MenuChange
Reload

Status Bar Comments

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

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

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



Misc....

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

Hotkeys are not supported.

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

PopMenu

PopMenu is a WinBatch 95/98/NT desktop interface to Windows batch files written in WIL, the Windows Interface Language. **PopMenu** batch files are used to automate PC operations and application specific procedures. (**FileMenu**, the other WinBatch menu utility, is used in manipulating files in the Windows Explorer.)

PopMenu appears as an icon on the Windows 95/98/NT Task Bar. This bar extends along one edge of the Windows 95/98/NT desktop and includes the "START" Button. A click on the **PopMenu** icon brings up a menu of WIL batch files. Samples are included, but you can completely modify these to meet your needs.

- PopMenu
- System Requirements / Installation / Operation
- Menu Files
- INI Settings
- Usage Tips, Known Problems and Limitations, etc.

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

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

WinBatch Help File

System Requirements

PopMenu requires Windows 95/98 or Windows NT.

Installation

To install **PopMenu**:

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

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

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

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

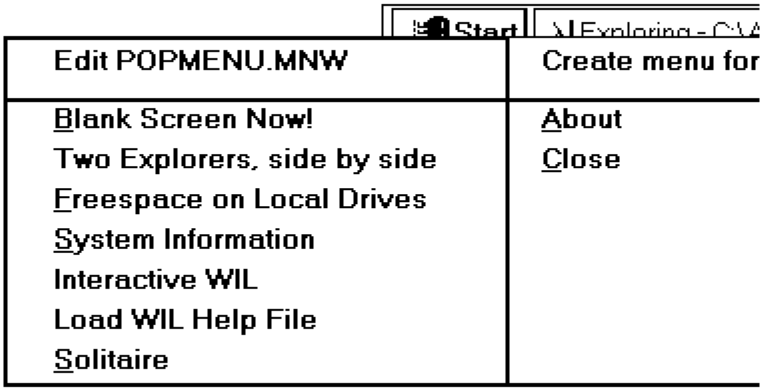
Then, run PopMenu.exe. You should see the PopMenu icon appear in the task bar.

Operation

Start **PopMenu** by running PopMenu.exe.

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

Close **PopMenu** by selecting "Close" from its menu.


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

WinBatch Help File

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

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

- PopMenu
- System Requirements / Installation / Operation
- Menu Files
- INI Settings
- Usage Tips, Known Problems and Limitations, etc.

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.

- PopMenu
- System Requirements / Installation / Operation
- Menu Files
- INI Settings
- Usage Tips, Known Problems and Limitations, etc.

WinBatch Help File

Usage Tips, Known Problems and Limitations, etc.

Functions

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

BoxOpen

BoxShut

BoxText

BoxTitle

The following optional WIL menu functions are NOT supported by **PopMenu**:

CurrentFile

CurrentPath

CurrFilePath

IsMenuChecked

IsMenuEnabled

MenuChange

Reload

Misc...

You can only run one copy of **PopMenu** at a time.

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

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

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

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

Hotkeys are not supported.

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

Status bar comments are not supported.

- PopMenu
- System Requirements / Installation / Operation
- Menu Files
- INI Settings
- Usage Tips, Known Problems and Limitations, etc.

Filename Appendix

- File Name Summary
- File Naming Conventions
- WinBatch DLLs
- Names for the WinBatch DLLs

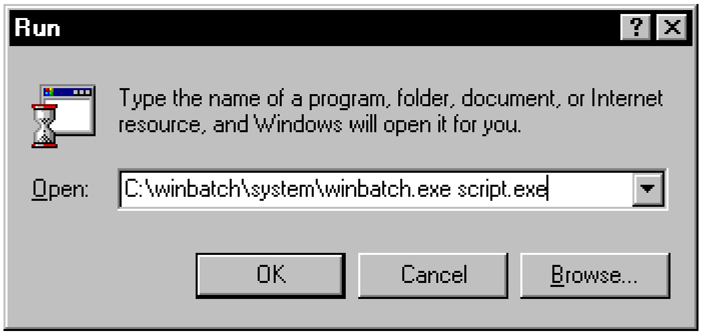
There are several different platforms which WinBatch and its utilities may be run on. When a file name is generated, it is made up of four or five characters which specify WHAT the file is, three characters which specify which platform the PC is running under and an .EXE or .DLL file extension.

File 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 Run commandline, the file names of WinBatch has to be entered first followed by a space and the name of a script.

   Example:

   "Run" from theTask Bar produces this dialog:



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

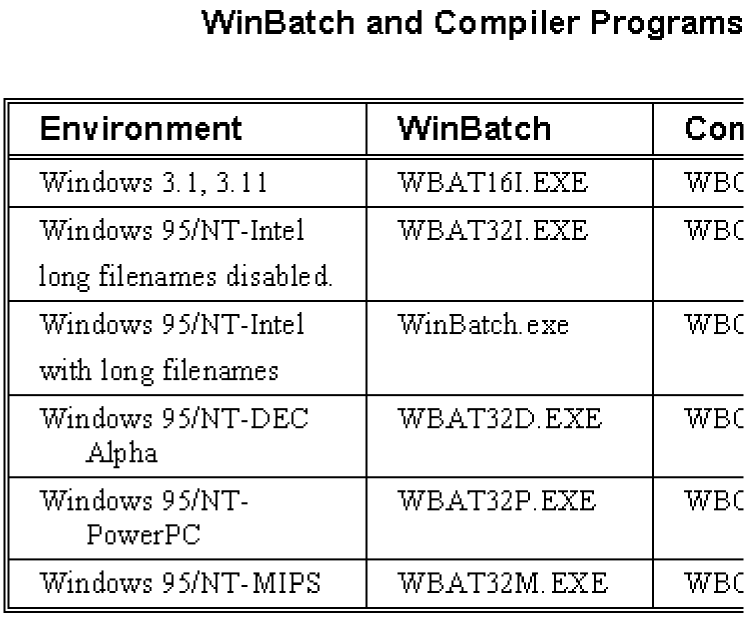
Open: C:\winbatch\system\winbatch.exe script.exe

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 WinInfo. Finally, WinBatch comes with a Dialog Editor. File names are used to run these.

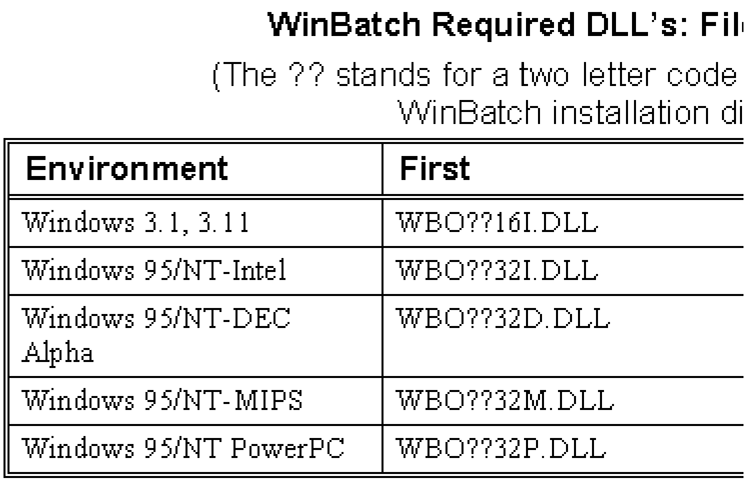
File Name Summary

- Filename Appendix
- File Naming Conventions
- WinBatch DLLs



WinBatch and Compiler Programs

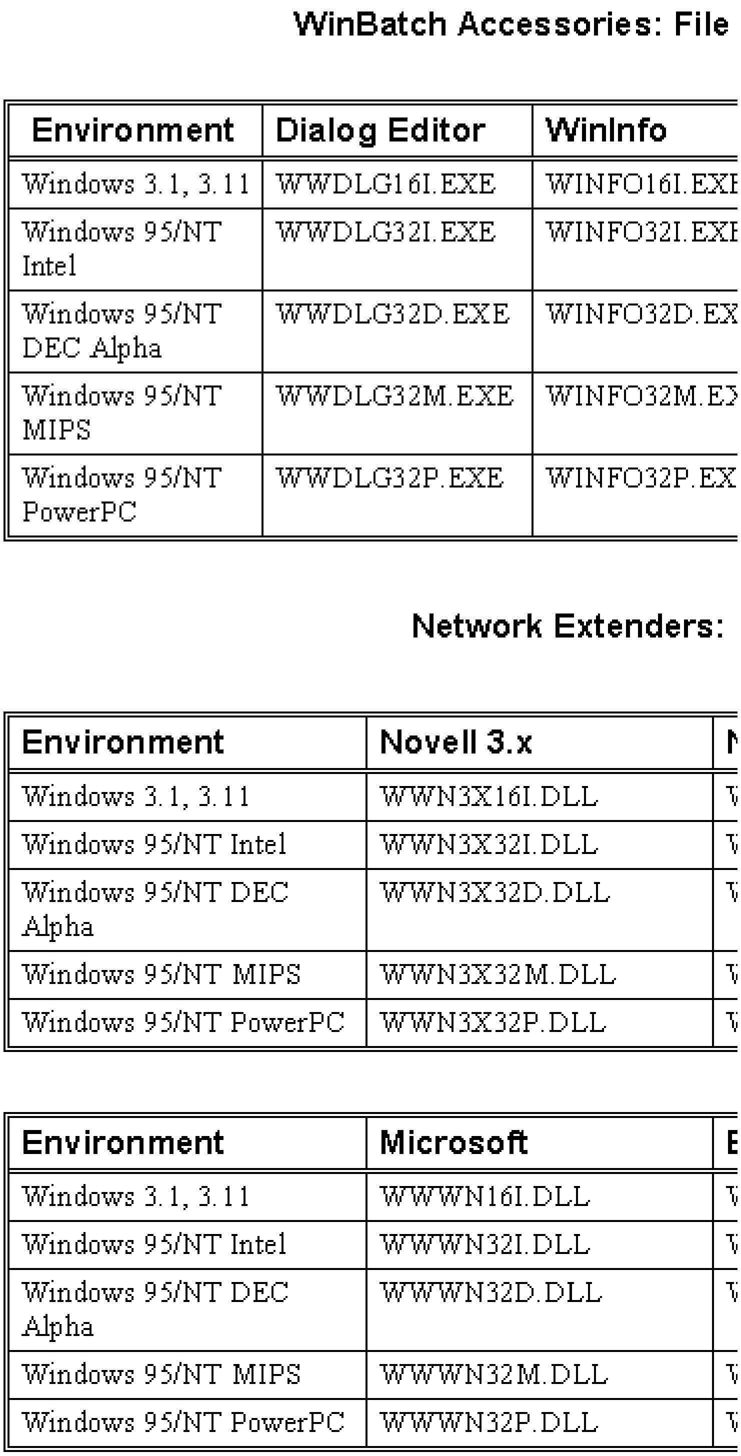
Environment	WinBatch	Con
Windows 3.1, 3.11	WBAT16I.EXE	WBC
Windows 95/NT-Intel long filenames disabled.	WBAT32I.EXE	WBC
Windows 95/NT-Intel with long filenames	WinBatch.exe	WBC
Windows 95/NT-DEC Alpha	WBAT32D.EXE	WBC
Windows 95/NT-PowerPC	WBAT32P.EXE	WBC
Windows 95/NT-MIPS	WBAT32M.EXE	WBC



WinBatch Required DLL's: Fil

(The ?? stands for a two letter code
WinBatch installation di

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



WinBatch Accessories: File

Environment	Dialog Editor	WinInfo
Windows 3.1, 3.11	WWDLG16I.EXE	WINFO16I.EXE
Windows 95/NT Intel	WWDLG32I.EXE	WINFO32I.EXE
Windows 95/NT DEC Alpha	WWDLG32D.EXE	WINFO32D.EX
Windows 95/NT MIPS	WWDLG32M.EXE	WINFO32M.EX
Windows 95/NT PowerPC	WWDLG32P.EXE	WINFO32P.EX

Network Extenders:

Environment	Novell 3.x	N
Windows 3.1, 3.11	WWN3X16I.DLL	W
Windows 95/NT Intel	WWN3X32I.DLL	W
Windows 95/NT DEC Alpha	WWN3X32D.DLL	W
Windows 95/NT MIPS	WWN3X32M.DLL	W
Windows 95/NT PowerPC	WWN3X32P.DLL	W

Environment	Microsoft	E
Windows 3.1, 3.11	WWWN16I.DLL	W
Windows 95/NT Intel	WWWN32I.DLL	W
Windows 95/NT DEC Alpha	WWWN32D.DLL	W
Windows 95/NT MIPS	WWWN32M.DLL	W
Windows 95/NT PowerPC	WWWN32P.DLL	W

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

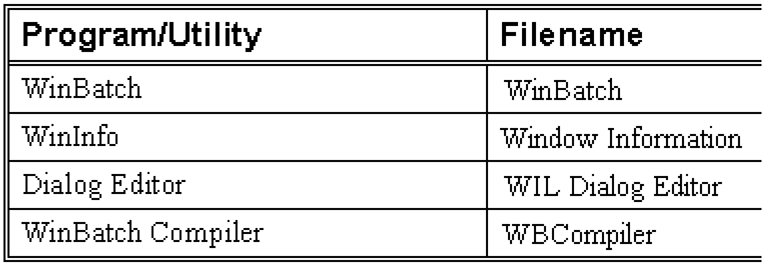
File Naming Conventions

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

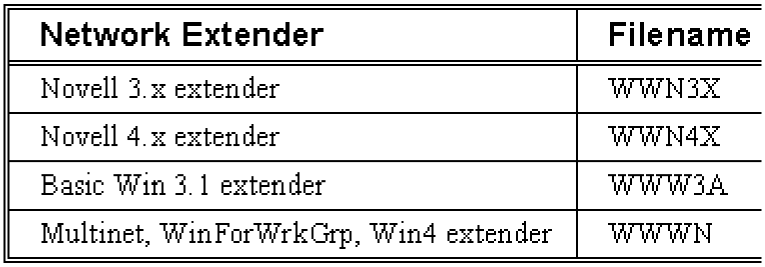
- Filename Appendix
- File Name Summary
- Names for the WinBatch DLLs

WinBatch 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 Digits in the File Name

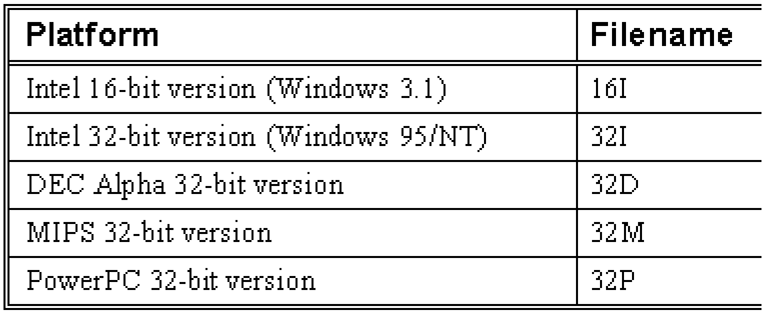


Program/Utility	Filename
WinBatch	WinBatch
WinInfo	Window Information
Dialog Editor	WIL Dialog Editor
WinBatch Compiler	WBCompiler



Network Extender	Filename
Novell 3.x extender	WWN3X
Novell 4.x extender	WWN4X
Basic Win 3.1 extender	WWW3A
Multinet, WinForWrkGrp, Win4 extender	WWWN

The Second 3 Digits 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 are using the single-user version of WinBatch, the executable files are WINBATCH.EXE, WIL DIALOG EDITOR.EXE, and WINDOW INFORMATION.EXE.

Note: Not all of the possible combinations above will exist.

WinBatch DLLs

A WinBatch utility needs two DLLs to function: a WBO DLL and a WBD DLL.

For WinBatch to find and use them, they must be either in the directory holding the WinBatch utility, or on a DOS or network search path.   They can be copied there manually, or automatically with the Large EXEstandalone option of the Compiler.

When a script is compiled with the Large EXE option, all the necessary   DLLs will be added to the executable utility.   When it runs, these DLLs are extracted and saved in the directory where the WinBatch utility is run.

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

Small WinBatch executables will need to find the WinBatch DLLs. They can be in the current directory, or on the DOS path or search path.   The easiest way to get them 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.

- Filename Appendix
- File Name Summary
- Names for the WinBatch DLLs

Names for the WinBatch DLLs

- Filename Appendix
- File Name Summary
- WinBatch DLLs

The WinBatch DLL names are made up of 3 parts.

The first three characters identify the DLL type.

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

The second two characters 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 = BQ, or AK, (some combination of letters)

The final three characters 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 386, 486, and 586 class processors.

WBDBQ32I.DLL
WBOBQ32II.DLL

Error Messages

10102, "10102: WinBatch - Unrecognized ParentProcess request code"
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: Unrecognized Request"

WinBatch+Compiler

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 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.

- How the Compiler Works
- Compiler Installation
- Compiler Usage
- Network Considerations
- Restrictions
- #include

How the Compiler Works

Compiler users frequently call and say, "I don't understand! What is it doing?" We've done our best to explain the Compiler in detail, in both the WinBatch help file and in the WinBatch User's Guide. Not surprisingly, comprehension seems to expand like waistbands after Thanksgiving dinner when the Compiler is explained in plain, simple English.

- WinBatch+Compiler
- Compiler Installation
- Compiler Usage
- #include

English version minus technical verbiage:

The Compiler gives you the ability to compile your scripts into executables which can be launched on PC's without WinBatch. The two standard executable options are Large for Standalone and Small for networked PC's.

When you place a Large EXE on a PC and run it, the EXE looks for the DLL's it needs to run. It looks in the current directory and on the path. If the DLL's are not found in either of these places, it writes the DLL's to the current directory. If the directory is write protected, an error will occur.

A Small EXE doesn't have the ability to write DLL's. The DLL's must be on the machine either in the path or in the current directory before it can execute. A Small EXE can use DLL's placed on the machine by a Large EXE.

Any extender DLL's you are using, plus the interpreter dll, Wbxxxyyy.dll, will be installed. See Filename Appendix for information on filenames.

Compiler Installation

WinBatch and the Compiler install from one set of diskettes in your WinBatch+Compiler package. The installation program is itself a Windows application, so make sure Windows is running.

Insert your disk into your A: or B: disk drive. From the Start…Run menu or your favorite shell, type A:\SETUP or B:\SETUP, depending on which floppy drive contains the Compiler diskette. Follow whatever instructions SETUP gives you. SETUP will create the necessary files in a directory of your choice.

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 your WinBatch User's Guide.

- WinBatch+Compiler
- How the Compiler Works
- Compiler Usage

Compiler Usage

- WinBatch+Compiler
- How the Compiler Works
- Compiler Installation

The compiler may be run 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 batch file interpreter to create and test a WinBatch script file. The WinBatch script file cannot exceed 64K in filesize. The Compiler will not test WinBatch macro scripts. Each WinBatch macro script file should have a file extension of .WBT or .WIL.

Running the Compiler:

- Interactive Mode

Notes about the compiler:

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

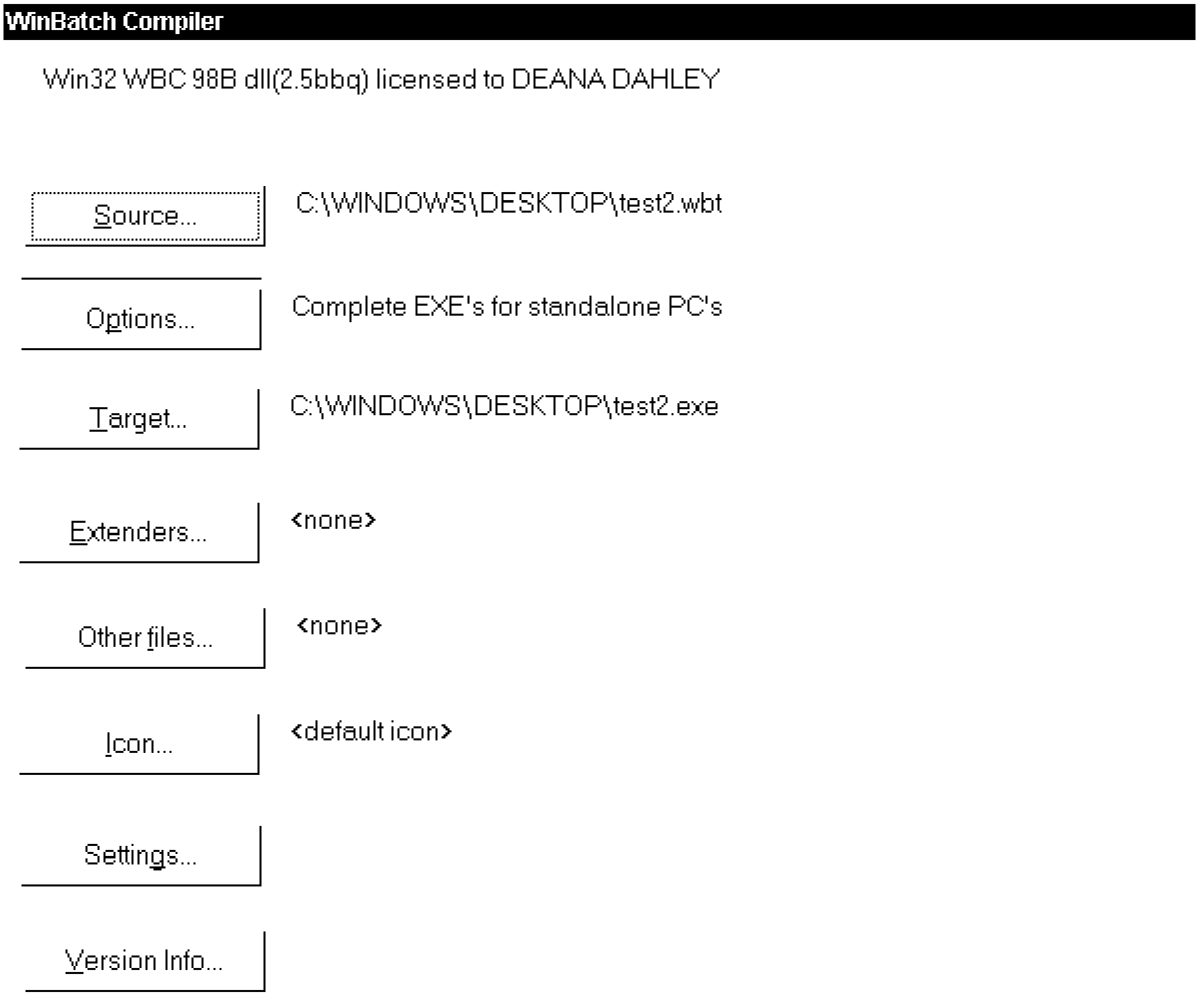
The compiler also creates a configuration file for each source file you compile. It will be placed in the same directory as the source file, and will have the same base name with an extension of ".CMP". For example, if you compile "C:\UTIL\TEST.WBT", it will create a configuration file named "C:\UTIL\TEST.CMP".

Interactive Mode

Start the compiler by double-clicking the compiler icon or the Compiler.EXE file name. (or by choosing the appropriate item in any menu system you may be using).   A dialog box will be displayed asking for input.

The compiler also supports "Drag and Drop" compiling. Select the Winbatch source file (WBT or WIL File) and drag it over the Compiler 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.

- WinBatch+Compiler



WinBatch Compiler

Win32 WBC 98B dll(2.5bbq) licensed to DEANA DAHLEY

Source... C:\WINDOWS\DESKTOP\test2.wbt

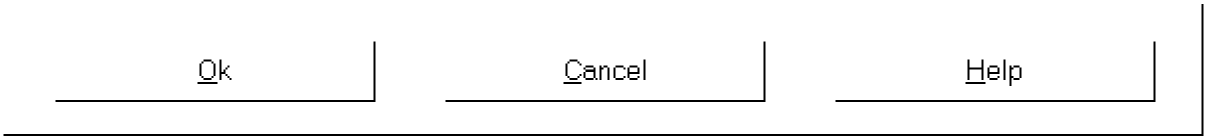
Options... Complete EXE's for standalone PC's

Target... C:\WINDOWS\DESKTOP\test2.exe

Extenders... <none>

Other files... <none>

Icon... <default icon>



Click on a button above to learn more….

Select the type of compile desired (large EXE, small EXE, encoded or encrypted) from the OPTIONS button.   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 standard WinBatch product prior to the compile step.

SOURCE

The SOURCE button displays a File Selection Box. Select your file or key in 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: Keeping 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 with have an extension of EXE. If your WinBatch utility has the same name as the program you want to run from the WinBatch utility, you have a problem you must resolve. The result of this situation is that your utility will run itself. 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 utility 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.

Note 3: Running an application ONLY from a WinBatch utility:

You can prevent users from running an application from outside of a WinBatch utility. A WIL Run() function can run an executable file name like this:

```
Run(excel*lib,)
```

The application can be renamed to excel.lib, an action that will prevent it from being run under Windows. Setting excel.lib to be read only, especially if it is located on a network server with full security capabilities, will make this operation more secure.

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

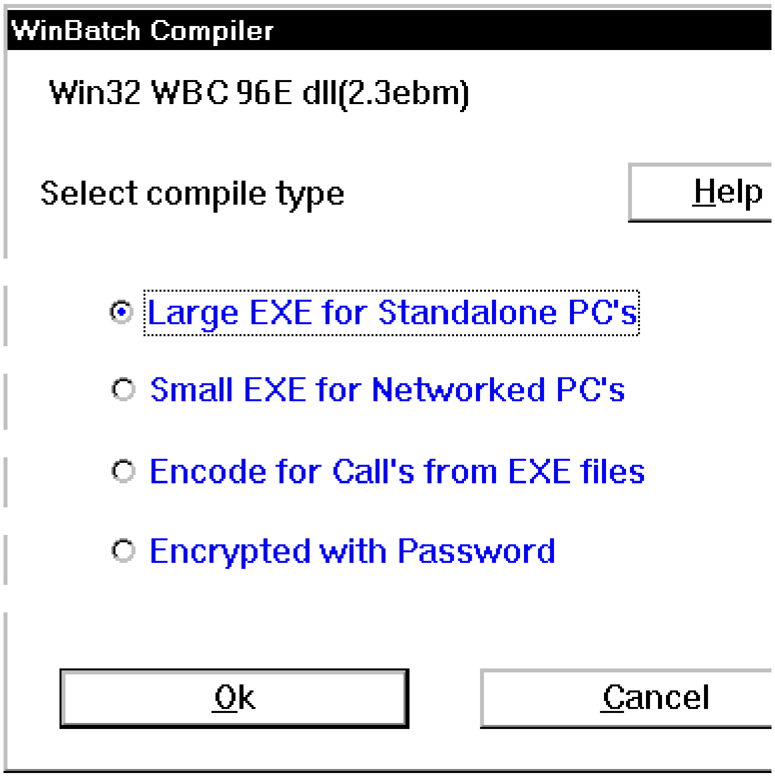
- WinBatch+Compiler

OPTIONS

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

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler



WinBatch Compiler

Win32 WBC 96E dll(2.3ebm)

Select compile type

● Large EXE for Standalone PC's

○ Small EXE for Networked PC's

○ Encode for Call's from EXE files

○ Encrypted with Password

Large EXE 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. 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 computers PATH and the compiled EXE will find them there and run. The Compiler has a small EXE option that takes advantage of this.

The DLLs need to be placed on the PATH only once. Subsequent EXE files installed on this same machine can be compiled under the Small EXE option.

If Network commands have been used, you will need to compile the Network Extender DLLs into the EXE. This is explained more specifically in the section, EXTENDERS

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".

- Large EXE for Standalone PC's
- Small EXE   for Networked PC's
- Encode for Call's from EXE files
- Encrypted with Password

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

Small EXE for Networked PC's
(without accessory files)

This option is suitable for network file server installation, or for distribution with separate DLL files. DLLs external to the WinBatch utility that uses them must be available in order to run small utilities.

When a small WinBatch utility is run, it will look in the Windows directory and the directories in the environment PATH variable for the DLLs. The WinBatch DLLs and network extender DLLs must be 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 standalone utility installed.

Hint: You can **automatically install** the DLLs on the PATH in a computer.

1. Create a large executable containing only a single statement:

   ```
   Display(1,WinBatch,WinBatch installed.)
   ```

   You can change this statement as you like.
2. Compile this as a large EXE with all the DLLs your scripts are ever likely to need.
3. Copy it into a directory on the path, for example the Windows System directory, and run it from there.

The DLLs will be installed once and for all. Any subsequent batch files run on that computer can be compiled as Small Exes. They will use the DLLs already installed on the computer.

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".

- Large EXE for Standalone PC's
- Small EXE for Networked PC's
- Encode for Call's from EXE files
- Encrypted with Password

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

Encode for Call's from EXE files

This option creates an encoded WBT file. The standard WinBatch product or a compiled EXE file is needed to access and run the encoded file. Encoded WBT files provide the following:

- Source code 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, your Target filename will have 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, go into your EXE and change the Call statement to reflect the new filename .WBC. Recompile the EXE.

- Large EXE for Standalone PC's
- Small EXE for Networked PC's
- Encode for Call's from EXE files
- Encrypted with Password

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

Encrypted with Password

This option encrypts a WBT file and uses a default Target extension of .WBE. The WinBatch interpreter (WBAT16I.EXE, or version specific WinBatch file) is needed to access the encrypted file. During the compilation, a password is provided to the compiler. 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.

- Large EXE for Standalone PC's
- Small EXE for Networked PC's
- Encode for Call's from EXE files
- Encrypted with Password

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

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 filename and path will generally be generated from the SOURCE filename and path.

Note: Your Target exe should not be the same name as the EXE file launched from within the compiled WBT. If you use the same name, Windows will ignore the path in the run command and run what it recognizes as the current exe, the compiled WinBatch executable, again.

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

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 selected extenders will be displayed in the WinBatch Compiler Dialog box next to the EXTENDERS button.

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

OTHER FILES

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

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

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 an ICONS subdirectory of your WinBatch directory.

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

SETTINGS

The SETTINGS button displays dialog for configuration settings. The "Modify Configuration Settings" dialog accepts 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.

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

VERSION INFO

The VERSION INFO button displays dialog that allows you to input Version string information. These are the version strings that will be displayed in the Properties dialog of the compiled EXE. The selected modifications are not displayed in the WinBatch Compiler Dialog box next to the SETTINGS button.

- SOURCE
- OPTIONS
- TARGET
- EXTENDERS
- OTHER FILES
- ICON
- SETTINGS
- VERSION INFO

- WinBatch+Compiler

#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 conatin **#include** directives). However, the WinBatch script and all files being included, when merged together, cannot exceed 64K in size (after comments and whitespace are optimized by the WinBatch compiler).

- For Standalone (Large) EXE compiles
- For Compiles of Small EXES
- For Compiles of Called Wbt's (Encode)
- For Encrypted WinBatch Wbt's

- Interactive Mode
- WinBatch+Compiler

Network Considerations

If you plan to put the compiled files 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 DLL's from the Compiler directory to a file server directory in the search path and set the DLL's as read-only.

3) Whenever the compiler, or any compiled WBTs with the Standalone option selected, are run, they will search the entire PATH for the required DLLs. If the DLLs are not found, they will be created in the user's WINDOWS directory. If you skipped item 2 immediately above, you will want to hunt these files down and remove them when you get around to actually doing item 2.

See the Filename Appendix for information on file and dll names.

- WinBatch+Compiler
- How the Compiler Works
- Compiler Usage

Restrictions

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 capability of this sort, please call Customer Service.

- WinBatch+Compiler
- Compiler Usage
- Network Considerations