Table of Contents

INTRODUCTION
- Overview
- Registering Your Copy
- WinBatch OnLine
- How WinBatch is Used
- What WinBatch Can Do
- About This Manual
- ORDERING INFORMATION
  - WILSON WINDOWWARE ORDER FORM
- Error Appendix

WINBATCH SETUP
- Installing and Configuring WinBatch
- License Numbers: Temporary and Permanent

USING WINBATCH

PARAMETERS
- Passing Parameters
- Displaying Passed Parameters in a Message Box.
- Passing Parameters Between WinBatch Script Files:

WINBATCH FUNCTIONS

NETWORK and Other EXTENDERS

UTILITIES
- Dialog Editor
- WinInfo
- WinMacro
- WinBatch FileMenu
- WinBatch PopMenu

FILENAME APPENDIX A

WINBATCH+COMPILER

How the Compiler Works
COMPILER INSTALLATION
COMPILER USAGE
    INTERACTIVE MODE
    BATCH MODE
NETWORK CONSIDERATIONS
RESTRICTIONS

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

WinBatch can automate Windows and Windows NT. WinBatch manipulates the Windows interface, Windows applications, and network connections.

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

WinBatch includes keystroke record and playback , and much more. WinBatch works from script files, so recorded events can be combined with advanced capabilities to automate operations impossible to record.

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

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

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

WinBatch utilities manipulate:

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

WinBatch has two components:

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

Separate versions of WinBatch are available for Microsoft Windows and for Windows NT. WinBatch is continually updated to function with future versions of Microsoft Windows.

It is easy to get started in Windows programming with WinBatch. Useful system utilities are produced quickly with WinBatch. All the things you couldnt do before in Windows are suddenly just a few minutes away.

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

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

Many WinBatch functions accomplish with one line operations that

take of pages of forms design, property setting and coding in other programming languages.

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

Registered users of WinBatch 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 must register your copy to obtain these benefits.

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

Registered users can share their WinBatch experience with other users on the Wilson WindowWare BBS. They can also share information on the Wilson WindowWare forums on America Online and CompuServe.

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

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

Internet Technical Support Articles: http://www.windowware.com/winware/techsupport

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

CompuServe forum: GO WILSON at Section 15.

America Online: keyword WINDOWWARE.

Wilson WindowWare BBS: (206.935.5198) requires N,8,1 logon parameters.

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.

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

With 269 general functions and commands, 64 networking functions, 74 physical constants, 24 operators, and 397 exception handling routines, 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.
- Make branching decisions based upon numerous factors.
- Call Dynamic Link Libraries.
- Act as an OLE 2.0 automation client.

And much, much more.

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

This User's Guide includes only topics and functions which are exclusive to WinBatch, or which behave differently in WinBatch. Also, there are additions and changes that have been made since the WIL Reference Manual went to press.

Network manipulation functions are dealt with in extensions 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 a **batch file** based implementation of WIL. A WinBatch batch file is a text file containing one or more lines of WIL functions and commands. PopMenu and FileMenu, two WinBatch utilities, are **menu file** based implementations.

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.

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

ALL-CAPS
: Used for filenames.

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

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

`Small fixed-width`
: Used for WIL sample code.

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

WinBatch software developed by Morrie Wilson.

Documentation written by Richard Merit, Tina Browning and Jim Stiles.

WinBatch is easy to install. You will find the necessary diskettes in your WinBatch package.

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

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

Insert your WinBatch disk into your A: or B: disk drive. From the **File Run** menu in **Program Manager, File Manager**, type A:\SETUP or B:\SETUP, depending on which floppy drive contains the WinBatch diskette. Follow the prompts from SETUP. SETUP will install the necessary files in a directory of your choice. You will be asked for additional diskettes and license numbers.

Note: If you want to install WinBatch over a network, copy the diskettes in the WinBatch package to a temporary directory on a server. Share that directory with read-only permission. Then attach to that directory from a workstation and run the installation from there. The WinBatch installation program, a Windows or Windows NT program, is called SETUP.EXE. Do not install WinBatch from a floppy drive shared over a network.

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 this WinBatch Users guide.

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

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

Keep 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. To make the registration screen appear, hold the shift key down while starting a WinBatch utility. Enter the license numbers into the screen that will pop up.

Creating WinBatch Script Files
Running WinBatch Utilities
Running WinBatch System Utilities

PARAMETERS

Passing Parameters
Displaying Passed Parameters in a Message Box
Passing Parameters Between WinBatch Script Files
WinBatch Functions

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 directory. Suitable icons for these scripts were added to the WinBatch group in the Windows Program Manager, or to the usual place programs are accessed in your version of Windows.

WinBatch script files must be formatted as plain text files. You can create them with WinEdit (Wilson WindowWares 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 in this manual 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 WindowWares 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.

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

WinBatch system utilities are very versatile. They can be run from icons in the Windows Program Manager.

- as automatic execution macros for Windows via the Run= line in the Windows Win.ini file.
- from macros in word processors and spreadsheets.
- from a command line entry such as the File Run... in the Windows Program and File Managers.
- by double clicking or dragging and dropping file names in the Windows File Manager.
- from menu items on the Windows control menu using WinMacro, an accessory program included with WinBatch.
- from other WinBatch scripts to serve as single or multiple agents, event handlers, or schedulers.
- from any Windows application or application macro language that can execute another Windows program. Software suite macro languages and application builders like Visual Basic and PowerBuilder are examples of these.

WinBatch utilities run like any other Windows programs. They can run from a command line, an icon in a shell program like the Program Manager in Windows 3.1 and Windows NT, or from a file listing such as the Windows and Windows NT File Managers.

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.

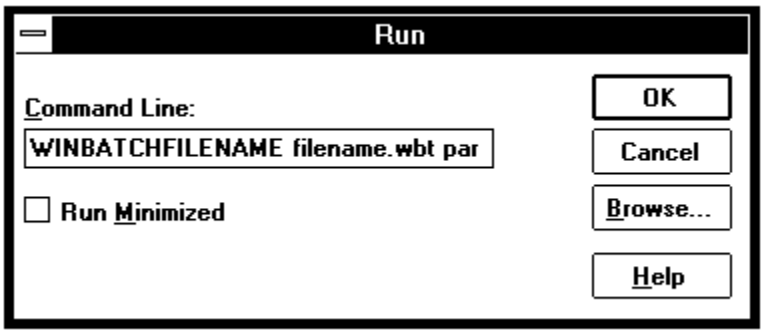
This capability can be used from the command line in the File Run menu items of both the Windows File Manager and the Program Manager. An example dialog is shown below.

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 Program Manager:



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, and the different Windows NT versions.

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

"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.

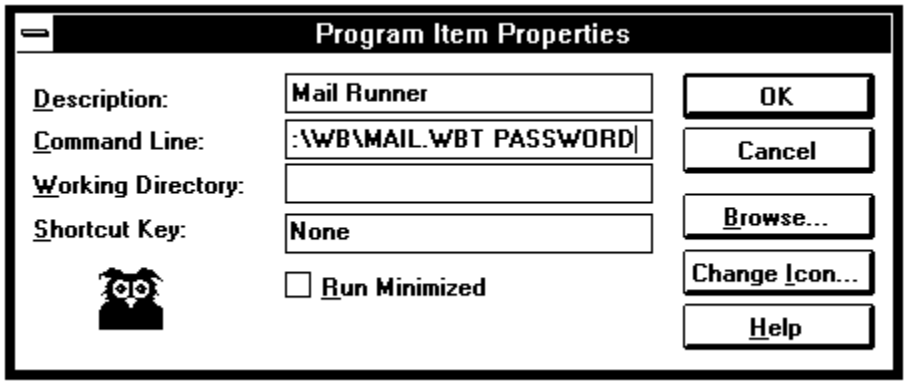
*In order to pass parameters to a WinBatch script file, you **must** run the WinBatch executable, itself, and it must be followed by the name of the WinBatch script file and any other desired parameters.*

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

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

"C:\WB\WBAT16I.EXE C:\WB\MAIL.WBT PASSWORD". (The actual command line entered does not include the quotation marks.)

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



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.

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.

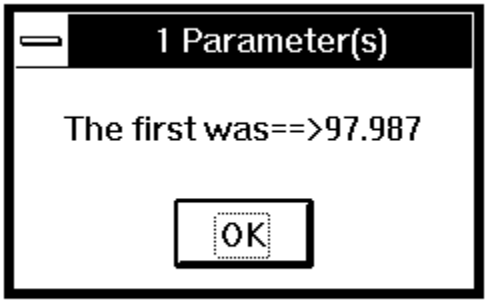
The Message function has this form:

```
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:



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

```
c:\www\wb50\wbat16i.exe c:\www\message.wbt 97.987
```

Note: Full path names were used for both the WinBatch executable file and for the WinBatch utility. Spaces separate the three parts of the command line.

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:

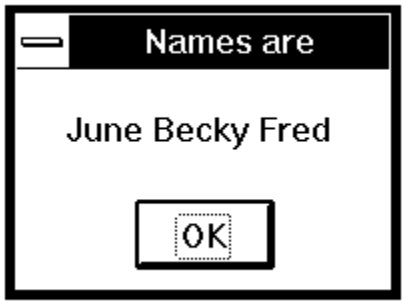
The first WBT calls a second WBT then passes three parameters.

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

TEST.WBT contains the following line:

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

which produces:



Function Reference Introduction

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

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

Function List

BoxOpen (title**,** text**)**
Opens a WinBatch message box.
BoxShut ( )
Closes the WinBatch message box.
BoxText (text**)**
Changes the text in the WinBatch message box.
BoxTitle (title**)**
Changes the title of the WinBatch message box.
CallExt (filename**,** parameters**)**
Calls another WBT file as a separate subprogram.

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.

The WinBatch 95/NT setup program uses WinBatch box functions to display the GUI part of the user interface. Additional "box" wbt files can be found in the 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)

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)*

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

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

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)*

Calls another WBT file as a separate subprogram.

Syntax:

CallExt (filename.wbt, parameters)

Parameters:

(s) filename.wbt the WBT file you are calling (the extension is required).
(s) parameters the parameters to pass to the file, if any, in the form "p1 p2 p3 ... pn".

Returns:

(i) always 0.

This function is used to pass control temporarily to a secondary WBT file. The main WBT file can optionally pass parameters to the secondary WBT file.

All variables are exclusive (**local**) to their respective files, so that neither WBT file "knows about" variables being used by the other.

The secondary WBT file should end with a **Return** statement, to pass control back to the main WBT file.
If a string of parameters is passed to the secondary WBT file, it will automatically be parsed into individual variables with the names **param1**, **param2**, etc. (maximum of nine parameters). The variable **param0** will be a count of the total number of parameters in the string.

Note: The CallExt function is not supported in compiled Exe's.

Example:

```
;This script is a short utility that politely asks for old and
;new names for a file.  Then it checks to be sure that a file of
;the new name does not already exist.

;To illustrate parameter passing, the old and new names are
;passed as parameters to a second script that actually does the
;renaming operation.  A final, and very polite, indeed, dialog
;informs that the deed has been done.

;In a practical version, most of this script would be combined
;and the script would take as passed parameters the old and
;new file names. Then with one CallExt function and two
;parameters, all the messages, and so forth, would work whenever
;needed.

;A CallExt() routine is another way to create new, and unique,
;functions.
```

```
;MAIN.WBT

old = AskLine("RENAME", "File to rename", "")
If !FileExist(old) Then Exit
new = AskLine("RENAME", "New name for %old%", "")
If FileExist (new)
  Message ("Rename aborted", "%new% already exists")
  Exit
Endif
CallExt("rename.wbt", "%old% %new%")
Exit

;RENAME.WBT
old = param1
new = param2
FileRename (old, new)
Message("New Filename", new)
Return
```

See Also:

Call, ParseData, Return *(all found in main WIL documentation)*

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.

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

Creates a push-button in a WinBatch box.

Syntax:

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

Parameters:

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

Returns:

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

Draws a button using standard Windows colors and fonts by specifying a unique "ID", text and coordinates. If an existing button "ID" is 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:

BoxesUp, BoxNew , BoxButtonKill, BoxButtonStat, BoxButtonWait

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:

BoxesUp, BoxNew, BoxButtonDraw, BoxButtonStat, BoxButtonWait

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:

BoxesUp, BoxNew, BoxButtonDraw, BoxButtonKill, 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:

BoxesUp, BoxNew, BoxButtonDraw, BoxButtonKill, BoxButtonStat,

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

Sets the background color for use with a WinBatch object.

Syntax:

BoxColor(box ID, color, wash color)

Parameters:

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

Returns:

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

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

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

Normal Color

BLACK="0,0,0"	DKGRAY="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"

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

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

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:

BoxesUp, BoxNew, BoxDrawLine, BoxDrawRect, BoxDrawText

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:

BoxesUp, BoxNew, BoxDrawCircle, BoxDrawRect, BoxDrawText

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:

BoxesUp, BoxNew, BoxDrawCircle, BoxDrawLine, 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)

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:

BoxesUp, BoxNew, BoxDrawCircle, BoxDrawLine, BoxDrawRect

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 **@HIDDDEN**.

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

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

Creates a WinBatch box.

Syntax:

BoxNew(box ID, coordinates, style)

Parameters:

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

Returns:

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

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

Style allows a selection from three different kinds of boxes.

0 No border
1 Border
2 Border and caption

Example:

```
;; sample script for BoxNew
BoxesUp("0,0,1000,1000", @normal)
BoxDrawText(1, "500,500,500,500", "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

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

Sets the text color for a WinBatch box.

Syntax:

BoxTextColor(box ID, color)

Parameters:

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

Returns:

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

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

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

BLACK="0,0,0"	DKGRAY="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 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, BoxTextFont, BoxColor, BoxPen

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.
(i) size — size of font, in virtual units.
(i) style — style flags for font.
(i) pitch & family — font pitch and family.

Returns:

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

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

Style (the following numbers may be added together):

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

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

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

Pitch:

0	Default
1	Fixed pitch
2	Variable pitch

Family:

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

Example:

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

```
f5="Book Antiqua"
fam=16
size=20

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

See Also:

BoxesUp, BoxNew, BoxTextColor

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

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 object(s). When it comes time to update those objects, a **BoxDataClear** will erase all items below the "TAG", and all remaining data space will again be available for reuse.

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

Removes commands from a WinBatch box command stack.

Syntax:

BoxDataClear(box ID, tag)

Parameters:

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

Returns:

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

This function removes all commands above "tag" from the command stack. "Tag" is not removed.
All buttons and Box commands after the tag are forever erased.

Example:

```
;; sample script for BoxDataClear
BoxesUp("100,100,900,900", @normal)
BoxColor(1,"255,255,0",0)
BoxDrawRect( 1, "0,0,1000,1000", 2)
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

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

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

Introduction
Novell 3.x Network Extender
Novell 4.x Network Extender
Basic Win 3.1 Network Extender
Multinet / WinForWrkGrp Network Extender
Windows 32 / Windows NT Network Extender

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 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.

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

This extender provides standard support for Novell 3.x networks. It may be used in addition with other extenders, such as the Windows for WorkGroups Multinet extender.

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

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

Other required DLL's: NWCALLS.DLL

This particular Dll, wwn3x16i.dll, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. Your system may require the use of a different Dll.

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

For more information and a list of functions see the Netware3 Extender Help file.

This extender provides standard support for Novell 4.x networks. It may be used in addition with other extenders, such as the Windows for WorkGroups Multinet extender, and the Novell 3.x extender.

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

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

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

Other required DLL's: NWCALLS.DLL, NWNET.DLL, NWLOCALE.DLL

This particular Dll, wwn4x16i.dll, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. Your system may require the use of a different Dll.

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

For more information and a list of functions see the Netware4 Extender Help file.

This extender 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. There are alternate extenders available for those products. In addition, some networks, like Novell, have better, more fully featured extenders available.

Additional Dlls required: NONE

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

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

This particular Dll, wwn3a16i.dll, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. Your system may require the use of a different Dll.

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

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

This extender 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. It is designed to work in conjunction with other extenders for other networks, such as extenders for Novell networks.

Additional Dlls required: NONE

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

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

This particular Dll, wwwn16i.dll, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. Your system may require the use of a different Dll.

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

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

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

This extender is only for 32 bit versions of Windows

32 Bit Intel Version

AddExtender("wwnet32i.dll")

32 Bit Dec Alpha Version

AddExtender("wwnet32d.dll")

32 Bit Mips Version

AddExtender("wwnet32m.dll")

32 Bit PowerPC Version

AddExtender("wwnet32p.dll")

Other required DLL's: none

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

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

Getting Started
Menu Commands
User Interface
Control Attribute Specifics

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

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

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

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

Note: The WIL Dialog Editor comes with an on-line help file (For the name of the help file see Filenames: Appendix A, as well as detailed instructions in the next section. Simply select the Help function in the Dialog Editor for detailed instructions on using the program.

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

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

The WIL Dialog Editor allows you to create dialog box templates for WIL using the WDL format. The Dialog Editor will write the WIL script statements necessary to create and display the dialog.

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

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

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

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

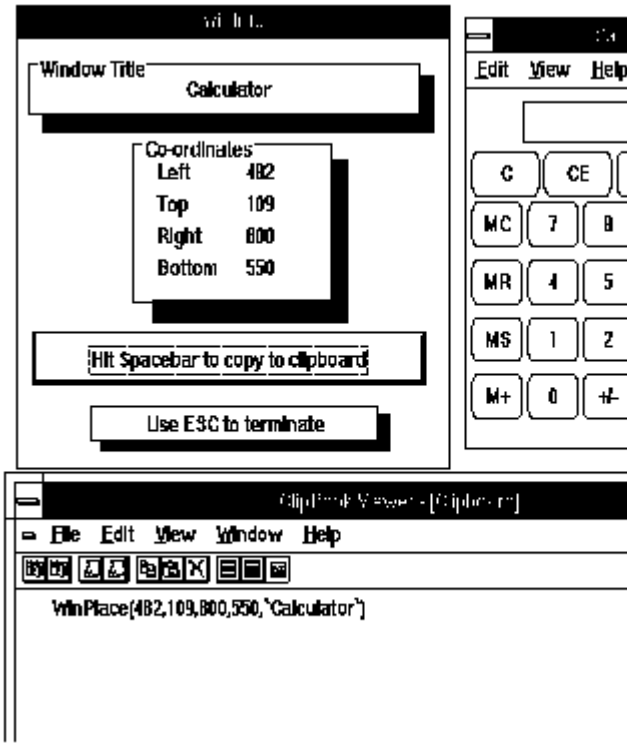
Using WinInfo

WinInfo is a handy window name and position grabber

The **WinInfo** utility (see Filename Appendix B 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 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**.

WinMacro is a standalone companion program included in the WinBatch package, which lets you create macro files and "attach" them to the control menu of any Windows application. These macros can then be executed, either by selecting them from the control menu, or through the use of a "hotkey." WinMacro also has the ability to "record" keystrokes, which can later be "played back" virtually anywhere in the Windows environment.

First, Record your keystrokes to create a .WBM file (macro script).

- Recording Keystrokes
  - Unrecordable Areas
  - SendKey
  - Options

Second run your script.

- Starting WinMacro
- Running Macros from the Control Menu
  - Macro Definition Files
  - Hotkeys
- WinMacro Example

Menu Utility for the Windows Explorer

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

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

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

System Requirements / Installation
Operation
Menu Files

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

Usage Tips, Known Problems and Limitations, etc.

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

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

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

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

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

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

WinBatch and Accessories

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.

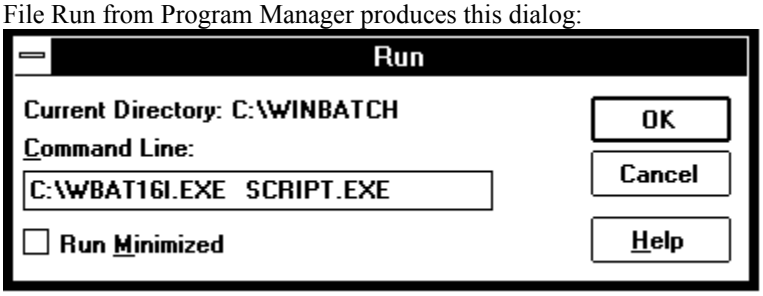
File Name Summary
File Naming Conventions
WinBatch DLLs
Names for the WinBatch DLLs

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 Windows Program Managers icons or File Run menu item, the file names of WinBatch has to be entered first and it must be followed by a space and the name of a script.

Example:

File Run from Program Manager produces this dialog:



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 launcher called WinMacro. It also has a window position and name grabber called WinInfo. Finally, WinBatch comes with a Dialog Editor. File names are used to run these.

WinBatch and Compiler Programs: File Names

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

WinBatch Required DLL's: File Names

(The ?? stands for a two letter code unique to a set of WinBatch installation disks.)

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

WinBatch Accessories: File Names

Environment	Dialog Editor	WinInfo	WinMacro
Windows 3.1, 3.11	WWDLG16I.EXE	WINFO16I.EXE	WWMAC16I.EXE
Windows 95/NT Intel	WWDLG32I.EXE	WINFO32I.EXE	WWMAC32I.EXE
Windows 95/NT DEC Alpha	WWDLG32D.EXE	WINFO32D.EXE	WWMAC32D.EXE
Windows 95/NT MIPS	WWDLG32M.EXE	WINFO32M.EXE	WWMAC32M.EXE
Windows 95/NT PowerPC	WWDLG32P.EXE	WINFO32P.EXE	WWMAC32P.EXE

Network Extenders: File Names

Environment	Novell 3.x	Novell 4.x
Windows 3.1, 3.11	WWN3X16I.DLL	WWN4X16I.DLL
Windows 95/NT Intel	WWN3X32I.DLL	WWN4X32I.DLL
Windows 95/NT DEC Alpha	WWN3X32D.DLL	WWN4X32D.DLL
Windows 95/NT MIPS	WWN3X32M.DLL	WWN4X32M.DLL
Windows 95/NT PowerPC	WWN3X32P.DLL	WWN4X32P.DLL

Environment	Microsoft	Basic
Windows 3.1, 3.11	WWWN16I.DLL	WWW3A16I.DLL
Windows 95/NT Intel	WWWN32I.DLL	WWW3A32I.DLL
Windows 95/NT DEC Alpha	WWWN32D.DLL	WWW3A32D.DLL
Windows 95/NT MIPS	WWWN32M.DLL	WWW3A32M.DLL
Windows 95/NT PowerPC	WWWN32P.DLL	WWW3A32P.DLL

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

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

WinBatch for Windows 3.1 running on a PC with an Intel, or compatible, microprocessor (the majority of installed PCs)will have the file name, WBAT16I.EXE. WinInfo is WINFO16I.EXE. The Dialog Editor is WWDLG16I.EXE. The WinBatch Compiler is WBC16I.EXE.

The First 4--5 Digits in the File Name

Program/Utility	Filename
WinBatch	WBAT
WinInfo	WINFO
Dialog Editor	WWDLG
WinBatch Compiler	WBC-

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 have Windows 3.1 and ordered the single-user version of WinBatch, the executable files you received are WBAT16I.EXE, WWDLG16I.EXE, and WINFO16I.EXE. You will only have files which are suitable to your platform needs.

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

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.

Names for the WinBatch DLLs

The WinBatch DLL names are made up of 3 parts.

The first three digits identify the DLL type.

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

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

XX = BG, or AK, (some combination of letters)

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

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

Here is are examples of a pair of DLLs for use on 16-bit versions of Windows on Intel 386, 486, and 586 class processors.

WBDAK16I.DLL

WBOAK16I.DLL

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).**
- **Insures 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 occassionally 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.
2701 California Ave SW #212
Seattle, WA 98116
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

Click below:

WILSON WINDOWWARE ORDER FORM

WILSON WINDOWWARE ORDER FORM

Name: _____

Company:_____

Address:_____

_____

City: _____ **St:**_____ **Zip:**_____

Phone: (_____**)**_____ **Country:**_____

Products

____ WinBatch 95 @ $99.95 : _____.____
For Windows 95/Windows NT

____ WinBatch 95 Compiler @$495.00 : _____.____
For Windows 95/Windows NT

____ WinEdit 95 @$99.95 : _____.____
For Windows 95/Windows NT

(all products include both 16 and 32 bit versions)

Upgrades

____ WinBatch to WinBatch 95 @ $30.00 : _____.____

WinBatch Compiler to
____ WinBatch 95 Compiler @$200.00 : _____.____

____ WinEdit to WinEdit 95 @ $30.00 : _____.____

License Numbers

WinBatch/WinEdit
ID __ __ __ __ __ __ __ Control __ __ __ __ __ __ __

WinBatch Compiler
ID __ __ __ __ __ __ __ Control __ __ __ __ __ __ __

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 previous page.

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 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 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)*

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

Table of Contents

Introduction
Getting Started

Menu Commands
- File
- Edit
- Help

User Interface
- Caption Box
- Control Attributes
  - Control Quick Reference
  - Altering Controls
- Save
- View the Script
- Decipher the Script

Control Attribute Specifics
- Setting Variables
- Push Button
- Radio Button
- Check Box
- Edit Box
- Fixed Text
- Varying Text
- File Listbox
- ItemSelect Listbox

Note:
ShowScript

Note:

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

ie.

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

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

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.

```
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")
```

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

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

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

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

Note: The WIL Dialog Editor comes with an on-line help file (For the name of the help file see Filenames: Appendix A), as well as detailed instructions in the next section. Simply select the Help function in the Dialog Editor for detailed instructions on using the program.

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

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

The WIL Dialog Editor allows you to create dialog box templates for WIL using the WDL format. The Dialog Editor will write the WIL script statements necessary to create and display the dialog.

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

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

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

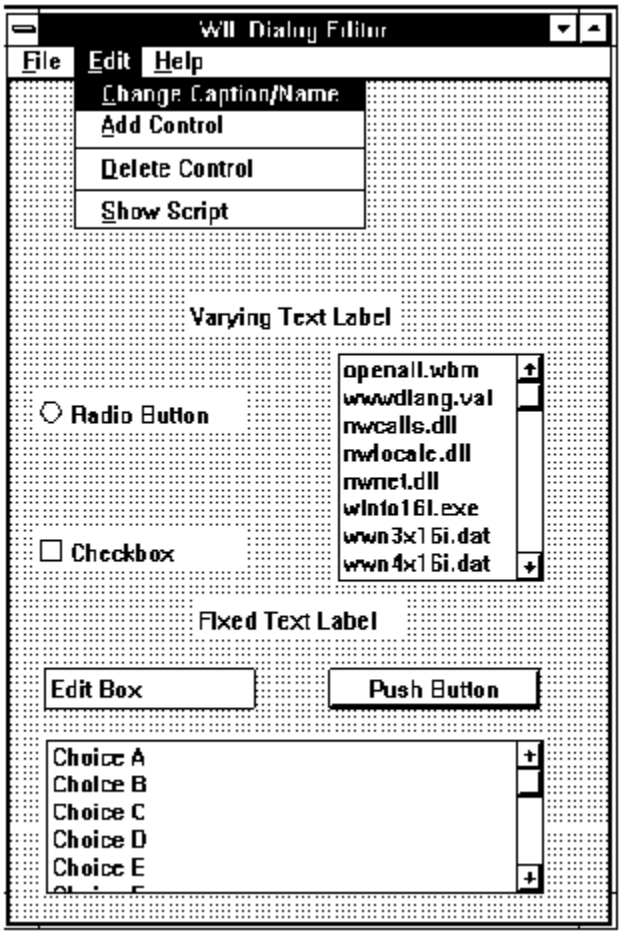
Using the Dialog Editor is easy. Once it is loaded, these hints offer a quick way to become comfortable with dialog box construction.

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

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

The editor will look like the following:

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



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

File
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.

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.

Index
Displays the Index of the On-line help information.

Menu Commands
Displays information about the WIL Dialog Editor menu commands.

How to use Help
Activates the Microsoft Windows Index to Using Help.

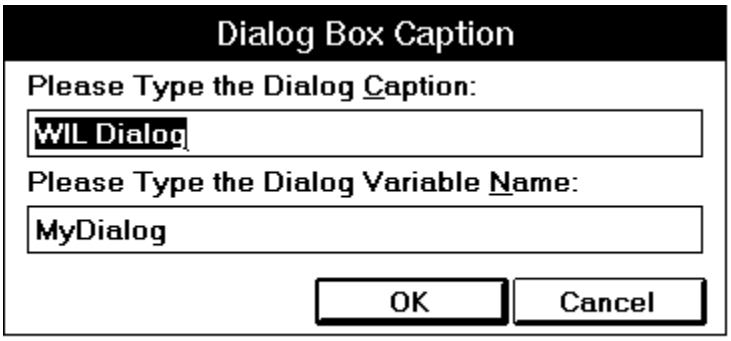
About
Displays the WIL Dialog Editor About dialog which includes the version number of the program.

Caption Box
Control Attributes
    Control Quick Reference
    Altering Controls
Save
View the Script
Decipher the Script

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

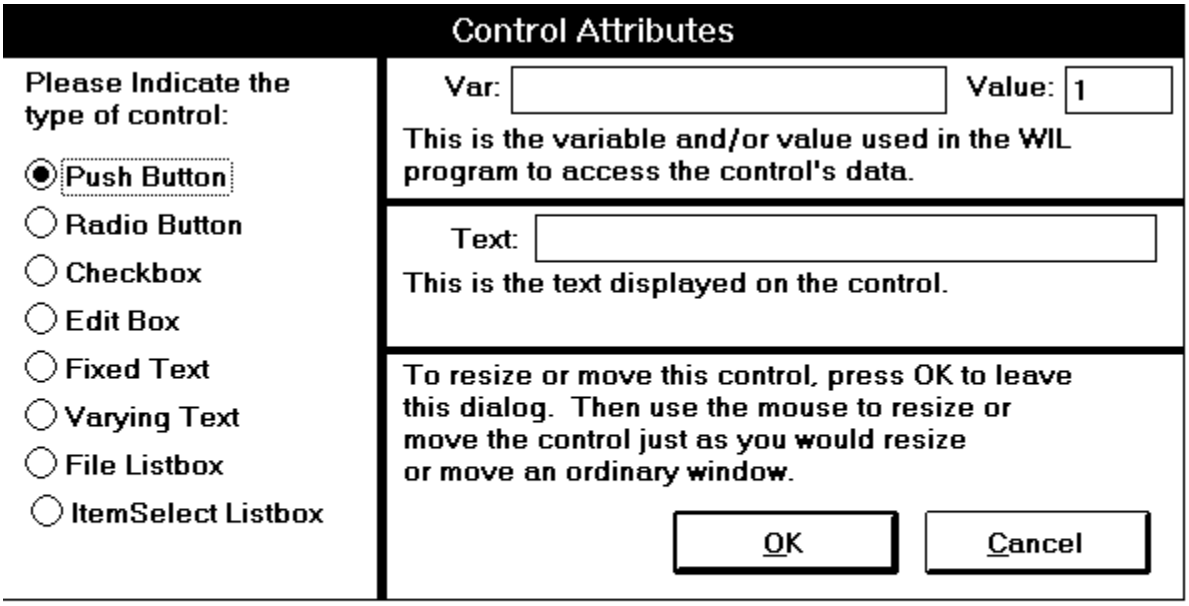
The **Dialog Caption** is the title of the dialog box as it appears in the title bar. The **variable name** is the name of the dialog as seen in the script.

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



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

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



The following table is a quick reference of what attributes are required for each control.

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

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

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

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

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.

Take a peek at the resulting script with the **File "ShowScript"** command to begin to get used to what WIL Dialog Scripts look like.

See: Decipher the Script

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

```
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")
```

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`
```

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")
```

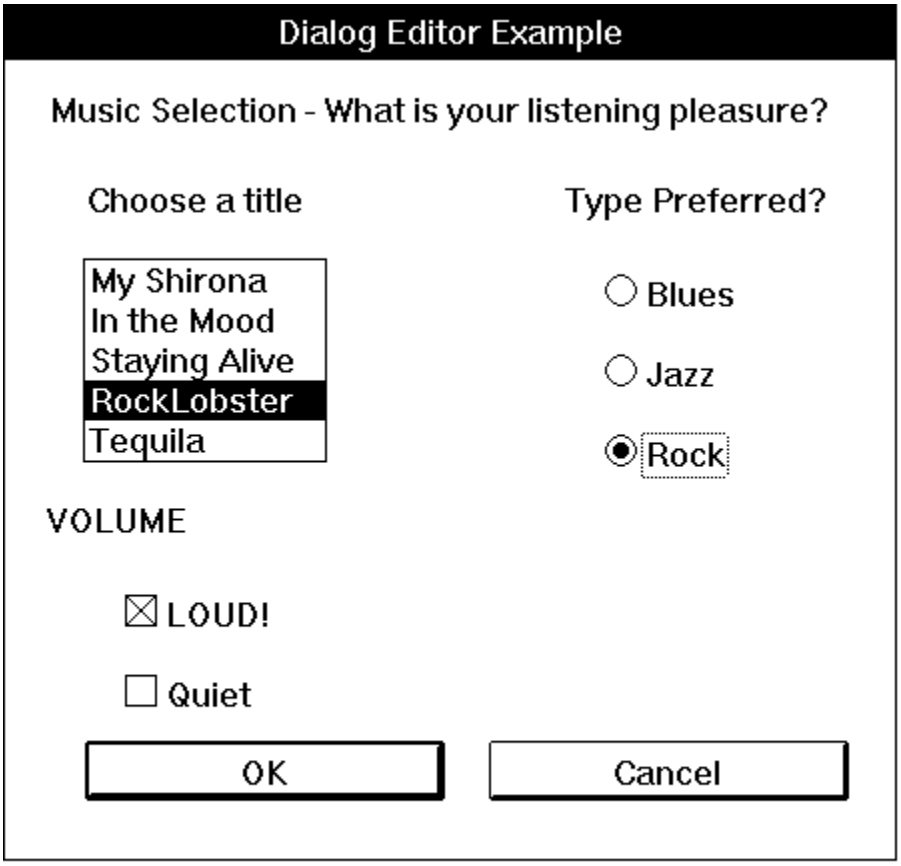
Put all the parts together and the completed script looks like the following.

```
ExampleFormat=`WWWDLGED,5.0`

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

Example01=`16,136,72,DEFAULT,PUSHBUTTON,DEFAULT,"OK",1`
Example02=`97,136,72,DEFAULT,PUSHBUTTON,DEFAULT,"Cancel",0`
Example03=`120,40,48,DEFAULT,RADIOBUTTON,music,"Blues",1`
Example04=`120,56,56,DEFAULT,RADIOBUTTON,music,"Jazz",2`
Example05=`120,72,56,DEFAULT,RADIOBUTTON,music,"Rock",3`
Example06=`24,104,112,DEFAULT,CHECKBOX,volume,"LOUD!",1`
Example07=`24,120,104,DEFAULT,CHECKBOX,volume2,"Quiet",2`
Example08=`8,88,64,DEFAULT,STATICTEXT,DEFAULT,"VOLUME"`
Example09=`9,6,164,DEFAULT,STATICTEXT,DEFAULT,"Music Selection - What is
```

```
                                                    your listening pleasure?"`
Example10=`16,40,48,40,ITEMBOX,tunes,DEFAULT`
Example11=`112,24,56,DEFAULT,STATICTEXT,DEFAULT,"Type Preferred?"`
Example12=`16,24,49,DEFAULT,VARYTEXT,song,"Choose a title"`

ButtonPushed=Dialog("Example")
```

Note:

Here is the completed dialog box.



Some of the Controls require extra knowledge or special handling.

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

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

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

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

To test a return value do the following:

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

"Cancel" or the value 0 will generally look for a label **:cancel**. If not found, it will exit.

For more information, see **Things to Know** in the **WIL Reference Manual**.

Used in situations to choose one item over another. You can have 9 choices per variable.
In using a **Radio Button**, the variable assigned is the same for each of the choices but the value is different.
For example, the script in a Dialog may look like:

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

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

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

Offers a choice of options. Any number may be marked or left unmarked. Each Check Box has its own specific information. Variable, Value and Text are different, allowing the user to choose more than one.

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

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

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

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

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 or the result of **FileItemize**.

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

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

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

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

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

```
"NOFILESELECTED"
```

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

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

Use **@tab,** a predefined constant, as the delimiter.

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

Note: When an ItemSelect Listbox is used, the dialog editor assumes that an item 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)
```

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

yesyesyesyesTRUEnono&AboutE&xitC&opyyesWinBatch + Compiler Help FileWBcompyes29/11/95

Table of Contents

WinBatch+Compiler
How the Compiler works
COMPILER INSTALLATION
COMPILER USAGE
    INTERACTIVE MODE
    BATCH MODE
NETWORK CONSIDERATIONS
RESTRICTIONS

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

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

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

- A small Windows EXE file.
- A 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
INTERACTIVE MODE
BATCH MODE

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.

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 Filenames Appendix A or information on filenames.

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 **File/Run** menu in **Program Manager** 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.

The compiler may be run in either interactive or batch mode. In interactive mode, the user is prompted to provide all necessary information via a popup dialog box. In batch mode, all required parameters are supplied via commandline arguments.

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 Compiler will not test WinBatch macro scripts. Each WinBatch macro script file should have a file extension of .WBT, .WBM, or .WIL.

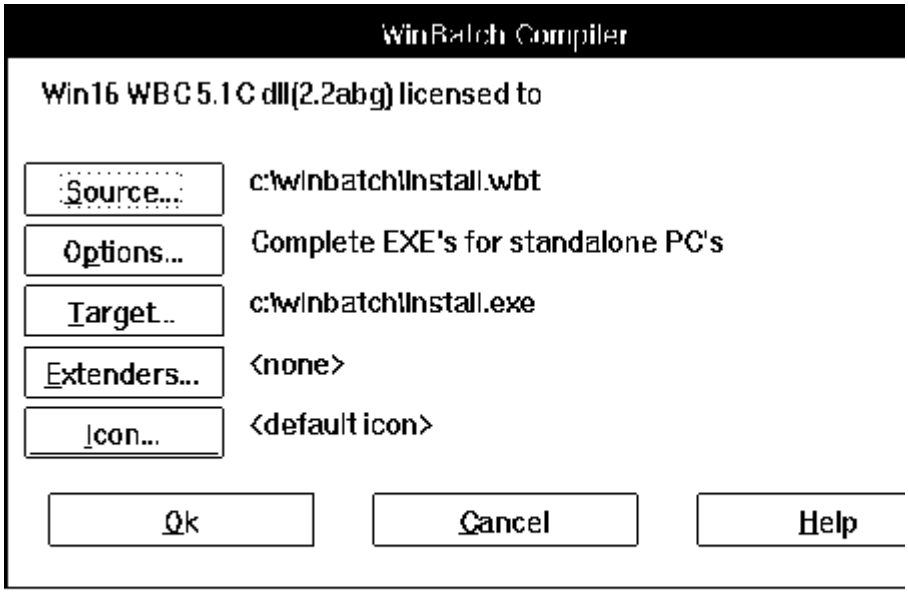
Running the Compiler in...

INTERACTIVE MODE

BATCH 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).

SOURCE
OPTIONS
TARGET
EXTENDERS
ICON



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

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

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.

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

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

(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.

(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. Thank You.)
   ```

   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 wll use the DLLs already installed on the computer.

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.

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 a 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.

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.

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.

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.

In batch mode all the information required to compile the program is passed to the compiler when it is initiated. The WinBatch WBT files and various other menu systems, including the program manager, can be configured to pass all required information in one operation.

Sample WinBatch code for an EXE compile

For a Standalone EXE compile without a default icon or network extenders.

```
Run("WBC-16i.exe", "1 Source.wbt Target.exe NONE NONE")
```

This particular compiler EXE, WBC-16i.exe, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. Your system may require the use of a different EXE. See Appendix A: Filenames in the WinBatch User's Guide for more information on filenames.

Command Lines

Below are several examples of the basic command line used to run in Batch Mode. These are the command lines which would be specified in the Program Manager command line, or in a File.Run dialog box.

More Command line info:

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

For a Standalone EXE compile without a default icon or network extenders.

```
Run("WBC-16i.exe", "1 Source.wbt Target.exe NONE NONE")
```

This particular compiler EXE, WBC-16i.exe, is for use on 16-bit versions of Windows on Intel 386, 486, and 586 type processors. Your system may require the use of a different EXE. See Filename Appendix A for more information on filenames.

Below are several examples of the basic command line used to run in Batch Mode. These are the command lines which would be specified in the Program Manager command line, or in a File.Run dialog box.

More Command line info:

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

Five parameters are required beyond the executable name of the compiler. Separate them with spaces.

Parameter 1: The first parameter is the number 1.

Parameter 2: The second parameter must be the source WBT file name. Example: source.wbt

Parameter 3: The third parameter must be the target executable utility file name. Example: utility.exe

Parameter 4: The fourth parameter must be either the name of an icon file or the word NONE. Example: balloon.ico

Parameter 5: The fifth parameter must be either the word NONE or the name of an extender DLL. Example: extender.dll

NOTE: NONE must be used in upper case.

Example:

A complete example with no icon or extender specified:

```
WBC-xxx.exe 1 source.wbt target.exe NONE   NONE
```

A complete example with an icon and an extender specified:

```
WBC-xxx.exe 1 source.wbt target.exe baloon.ico Ext.dll
```

About the Extenders: If you need to specify more than one Ext.dll, the string is delimited by a comma without spaces. Some extenders require more than one DLL. Interactive Mode will worry about this for you and include any extra DLLs. Look in the corresponding .DAT file for a list of extra DLLs the extender uses.

(for use where DLLs are already on the path, i.e.-on a network):

Four parameters are required.

Parameter 1: The first parameter is the number 2.

Parameter 2: The second parameter must be the source WBT file name. Example: source.wbt

Parameter 3: The third parameter must be the target executable utility file name. Example: utility.exe

Parameter 4: The fourth parameter must be either the name of an icon file or the word NONE. Example: icon.ico.

Note: NONE must be used in upper case.

Example:

```
WBC-xxx.exe 2 source.wbt utility.exe icon.ico

or

WBC-xxx.exe 2 source.wbt target.exe NONE
```

(used where the batch file is used as a subprogram called by a parent WinBatch executable program):

Encoded WinBatch utilities must be executed by another WinBatch executable, they are not stand alone executables. Whenever you have a Call statement in your script, the Called WBT must be encoded.

Note: The file extension changes from the source "WBT" to the target "WBC".

Three parameters are required.

Parameter 1: The first parameter is the number 3.

Parameter 2: The second parameter must be the source WBT file name. Example: source.wbt

Parameter 3: The third parameter must be the target file name. Example: utility.wbc

Example:

```
WBC-xxx.exe 3 source.wbt utility.wbc
```

Encrypted WinBatch utilities must be executed by the WinBatch interpreter, they are not stand alone executables. The encrypted option is rarely used because the capability of password protection is easily inserted into a compiled WinBatch utility.

Four parameters are required.

Parameter 1: The first parameter is the number 4.

Parameter 2: The second parameter must be the source WBT file name. Example: source.wbt

Parameter 3: The third parameter must be the target utility file name. In this case it takes the extension of WBE. Example: utility.wbe

Parameter 4: The fourth parameter must be the password for access to the compiled WinBatch utility. It is case sensitive.

Example:

```
WBC-xxx.exe 4 source.wbt utility.wbe password
```

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 in File Manager to a file server directory in the search path and set the DLL's as read-only. (see Filename Appendix B)

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 (see Filename Appendix B). 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.

The CallExt function is not supported in compiled Exes.

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

yesTRUEyesyesyesnono&PrintyesWINMACRO29/11/95

WinMacro is a standalone companion program included in the WinBatch package, which lets you create macro files and "attach" them to the control menu of any Windows application. These macros can then be executed, either by selecting them from the control menu, or through the use of a "hotkey." WinMacro also has the ability to "record" keystrokes, which can later be "played back" virtually anywhere in the Windows environment.

Table of Contents

WinMacro
- Recording Keystrokes
  - Unrecordable Areas
  - SendKey
  - Options
- Starting WinMacro
- Running Macros from the Control Menu
  - Macro Definition Files
  - Hotkeys
- WinMacro Example

FileMenu
- FileMenu System Requirements / Installation / Operation
- Menu Files
- Using the "all filetypes" FileMenu
- Creating/Modifying File-Specific Menus
- FileMenu.ini
- Usage Tips, Known Problems and Limitations, etc.

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

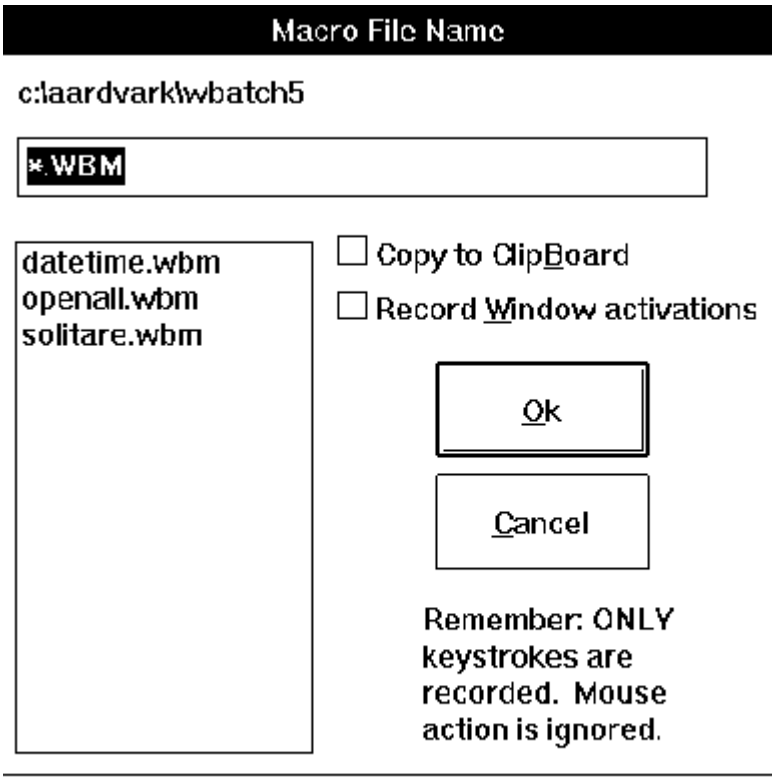
WinMacro is the keystroke recorder and program launcher included with WinBatch.

Keystrokes can be sent only to the active application. The SendKeys() function requires that the destination application have the current focus. This can be done with an appropriate WinActivate( ). The function SendKeysTo( ) does all this in one statement. Keystrokes cannot be sent to hidden or to full screen DOS applications.

To use the keystroke recorder, first start WinMacro if you havent already. Find the WinMacro icon and display its icon menu with a mouse click or an Alt Space with the keyboard.

Start recording by typing **Ctrl-Alt-Ins** from any window, or selecting **Begin Macro Record** from the WinMacro icon's menu.. You can try it immediately to get a feel for it. After recording a few sample keystrokes, click on the icon and select End Macro Record to end the recording process. (Control Shift Home and Control Shift End are the keyboard hot key combinations you can use to begin and end recording.)

WinMacro will present you with the following dialog box which contains a menu of existing WBM files.



If you want to overwrite an existing file, select its name from the menu; otherwise, enter a name for the file you wish to create in the edit box (a WBM extension will automatically be added), and press the **Enter** key or click on the **OK** button**.** At this point, the icon will begin flashing, indicating that you are in record mode.

Once you are in record mode, every keystroke you type will be recorded to your WBM file. Mouse movement and mouse clicks are *not* recorded. To end record mode, type **Ctrl-Alt-Ins** from any window**,** or click on the flashing WinMacro icon and select **End Macro Record** from the menu. The icon will stop flashing.

The recording will be saved in your WinBatch directory with a name you choose. A .wbm extension will be added. You can inspect it with a text editor. Note that much of the hard work has been done for you.

Once you have created a WBM keystroke macro file, you can assign it to a hotkey in a WDF file, using the steps outlined under the **Hotkeys** topic. Use WinBatch to run WBM files, the same way you do with WBT files.

There are two options which can be selected before you begin recording.

Copy to ClipBoard
If this is selected, then when you end macro recording the contents of the WBM file will be automatically copied to the clipboard.

Record Window Activations
If this is selected, then whenever you switch to a different window while recording is taking place, WinMacro inserts a corresponding "WinActivate" command into the WBM file.

WinMacro is unable to record keystrokes entered in Windows' **System Modal Dialog Boxes**. These include the dialog boxes in the MS-DOS Executive window, as well as dialog boxes generated by severe system errors. By the same token, WinBatch cannot play back keystrokes in these types of dialog boxes.

Sends keystrokes to the currently active window.

Syntax:

SendKey(char-string)

Parameters:

(s) char-string string of regular and/or special characters.

Returns:

(i) always 0.

Note1: **SendKey** will send keystrokes to the currently active window. For many applications, the related functions, **SendKeysChild**, **SendKeysTo** or **SendMenusTo** may be better alternatives.

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

Key	SendKey equivalent
~	**{~}** ; This is how to send a ~
!	**{!}** ; This is how to send a !
^	**{^}** ; This is how to send a ^
+	**{+}** ; This is how to send a +
{	{{} ; This is how to send a {
}	{}} ; This is how to send a }
Alt	**{ALT}**
Backspace	**{BACKSPACE}** or **{BS}**
Clear	**{CLEAR}**
Delete	**{DELETE}** or **{DEL}**
Down Arrow	**{DOWN}**
End	**{END}**
Enter	**{ENTER}** or ~
Escape	**{ESCAPE}** or **{ESC}**
F1 through F16	**{F1}** through **{F16}**
Help	**{HELP}**
Home	**{HOME}**
Insert	**{INSERT}** or **{INS}**
Left Arrow	**{LEFT}**
Page Down	**{PGDN}**
Page Up	**{PGUP}**
Right Arrow	**{RIGHT}**
Space	**{SPACE}** or **{SP}**
Tab	**{TAB}**
Up Arrow	**{UP}**

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

Alt	**!**

Control	^
Shift	+

To enter **Alt-S**:

```
SendKey("!s")
```

Note2: You should, in general, use lower-case letters to represent Alt-key combinations and other menu shortcut keys as that is the normal keys used when typing to application. For example "!fo" is interpreted as Alt-f-o, as one might expect. However "!FO" is interpreted as Alt-Shift-f-o, which is not a normal keystroke sequence.
To enter **Ctrl-Shift-F7**:

```
SendKey("^+{F7}")
```

You may also repeat a key by enclosing it in braces, followed by a space and the total number of repetitions desired.

To type 20 asterisks:

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

To move the cursor down 8 lines:

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

Examples:

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

In those cases where you have an application which can accept text pasted in from the clipboard, it will often be more efficient to use the **ClipGet** function:

```
Run("notepad.exe", "")
crlf = StrCat(Num2Char(13), Num2Char(10))
; copy some text to the clipboard
ClipPut("Dear Sirs:%crlf%%crlf%")
; paste the text into Notepad (using Ctrl-v)
SendKey("^v")
```

A **WIL** program cannot send keystrokes to its own **WIL** Interpreter window.

Note3: If your **SendKey** statement doesn't seem to be working (e.g., all you get are beeping noises), you may need to place a **WinActivate** statement before the **SendKey** statement to insure that you are sending the keystrokes to the correct window, or you may try using the **SendKeysTo** or **SendKeysChild** function.

See Also:

SendKeysTo, SendKeysChild, SendMenusTo, KeyToggleSet, SnapShot, WinActivate *(All found in main Wil Documentation)*

The Windows Control Menu is found at the upper left corner of every main (parent) window.

WinMacro wears another hat. It doubles as a handy script runner always available from any, or all, of your Windows applications. WinBatch utilities and applications can be run from the control menu of any or all applications. You can even add macros to applications that do not have their own internal language.

You can leave WinMacro running to add menu items to the control menu in every Windows program. This menu drops down when the space bar icon in the upper left corner of a Windows program window is activated either by a mouse click or by an Alt Space keystroke combination.

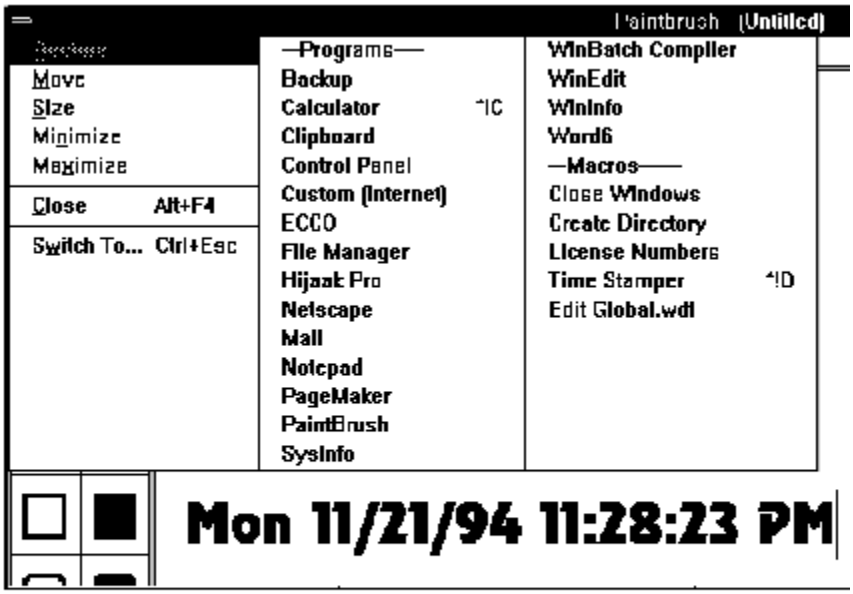
The example here shows how WinMacro can be used to run applications and macros. It is handy to run macros from within any application. Some general macros can work across applications and work within any one of them. For instance, this time and date stamp can be made from a WinBatch script containing nothing more than these three lines:

Example:

```
;TimeDate.WBT
;Sends time and date as keystrokes. Does not use clipboard. Does not
clear clipboard.
focus=WinGetActive()
now=TimeDate()
SendKeysTo(focus,now)
```

In this screen shot, the Time Stamper appears on the right. It is always available by keying the Ctrl-Alt-D key combination. Whether the application is a word processor, a personal information manager, a database, or even a paint program, the time and date is easily entered.

This easy access to system management utilities may be simple, but easy time-savers like this yield major benefits. Minutes saved for one person add up quickly when multiplied over the number of times used and the number of people doing the using.



To add items to the control menu, go to your WinBatch directory and use WinEdit (from Wilson WindowWare) or the Windows Notepad to open the file global.wdf. It will look like this:

Example:

```
Clipboard           : Clipbrd.exe
Control Panel  \ ^P : control.exe
WinEdit             c:\programs\we31\winedit.exe
Notepad             : Notepad.exe
Word6               :\programs\word6\winword.exe
```

Handy access to the sysinfo macro can be a great assistance to technical support efforts. Are there other time savers you could develop?

To add a WinBatch macro that will run on a Windows 3.1 computer, you will need to add a line like this one to the GLOBAL.WDF file. It will let the computer user easily access the sysinfo script you added to your WinBatch directory during the installation of WinBatch.

Example:

```
SysInfo    \^!+S       : c:\wbdir\wbat16.exe c:\wbdir*\sysinfo.wbt
```

Of course, the WinBatch directory, wbdir*, in the example will need to be changed to the directory you use for WinBatch. The \^!+S is optional. It specifies that the Control Alt Shift S keystroke combination can be used to launch this system information display macro.

You can also create .WDF files specifically for your applications. In this way you can add macros to applications that do not have a macro language.

For instance, adding several macros to PageMaker 4.0 can be done through creating a .wdf file called PM4.WDF. WinMacro will know when the executable file PM4.EXE is loaded and will attach the menu file to that application.

Note: You may need to do some detective work to uncover the executable file name for some applications. If you find that your menus do not attach to your application, check the properties for the icon used to launch that application from its Program Manager icon. The file name listed there should work for your WDF file name. The WinBatch directory includes an accessory program called wdf-exe.wbt. Run it and follow directions to uncover the name of an obscure executable file.

You can run WINMACRO.EXE just like any other Windows program, using your favorite Windows-program-starting method (keyboard, mouse, Program Manager, File Manager, MS-DOS Executive, Command Post, File Commander, WinBatch, etc.). However, if you will be using WinMacro on a regular basis, you may wish to have it load automatically when you start up Windows. You can do this by adding WINMACRO.EXE to the **Program Manager** Startup group**.** Drag and drop the exe or copy the current WinMacro icon into the Startup group. Consult your Microsoft Windows manual for more information.

WinMacro starts up as an icon, and remains active until you either close it or end your Windows session (whichever comes first).

WinMacro definition (**WDF**) files are plain ASCII files which you create and edit. They *must* have a WDF extension, and they *must* be located in the same directory as WINMACRO.EXE. A WDF file contains any number of definition lines, each of which represents an individual command. Each line has the following format:

Title [\ optional hotkey] : program to be executed

Title is the name which will appear on the application's control menu to identify the command. The **hotkey** is optional; if it is included, it must be preceded by a backslash (\). This is followed by a colon (**:**), and then the **program** which should be executed when the command is selected, with any required parameters. This can be any Windows or DOS EXE, COM, PIF, or BAT file, and you must include the appropriate file extension. If the program isn't located either in the current directory or on your DOS path, you must include a path specification for it. To run a WinBatch file, run WINBATCH.EXE, with the name of the WBT file as a parameter.

Let's create a WinMacro definition file, named GLOBAL.WDF:

```
Run Notepad                : notepad.exe
Play Solitaire      \ ^F9  : winbatch.exe solitare.wbt
```

(This second line assumes that you have created SOLITARE.WBT as part of the WIL tutorial. If not, just substitute any WBT file name).

GLOBAL.WDF is a special file name. When WinMacro starts up, it looks for this file. If present, WinMacro loads it, and attaches its contents to the control menu of every window currently running, as well as any windows that may subsequently be opened (the **control menu,** also known as the system menu, is the menu that you access by pressing **Alt-Space**, or by clicking the little box which appears at the left side of the title bar of almost all application windows).

Go ahead and start up WinMacro, then access the control menu of any open window. You should see that the two commands in your GLOBAL.WDF file have been attached to the control menu, and both are now available for your use. You can run these user-defined commands by selecting them from the menu. In addition, because you have defined a hotkey for the "Play Solitaire" command, you can run it from any window by pressing **Ctrl-F9**.

You can assign a hotkey to any WinMacro definition line. A hotkey consists of the **Ctrl** key plus any letter (**A - Z**) or function (**F1 - F16**) key. In addition, you can optionally use the **Alt** and **Shift** keys:

Key	Char
Ctrl	^
Alt	**!**
Shift	+

Here are some examples of valid key combinations:

Hotkey	Equivalent keystrokes
^F5	Ctrl-F5
^!F5	Ctrl-Alt-F5
^+F5	Ctrl-Shift-F5
^!+F5	Ctrl-Alt-Shift-F5
^D	Ctrl-D
^!D	Ctrl-Alt-D
^+D	Ctrl-Shift-D
^!+D	Ctrl-Alt-Shift-D

In addition to GLOBAL.WDF, you can create **application-specific** WinMacro definition files. They have the form **progname.WDF**, where "progname" is the name of the application's COM or EXE file. So, if you wanted to have a WDF file which would apply only to Notepad, you would name it NOTEPAD.WDF. Its contents would be attached *only* to *Notepad's* control menu, and its hotkeys would be active *only* when *Notepad* was the active window. WinMacro loads application-specific WDF files *after* GLOBAL.WDF, so if you have, for example, a NOTEPAD.WDF file, it's contents will be attached to Notepad's control menu *in addition to* (not instead of) GLOBAL.WDF. If you define the same hotkey in GLOBAL.WDF *and* NOTEPAD.WDF, the one in NOTEPAD.WDF will apply.

If you edit a WDF file while WinMacro is running, and want to see the changes reflected in the current menus, select **About/Reload** from the WinMacro icon's menu. All windows will be updated.

Let's create a macro for Solitaire which will cycle to the next deck back design (sound familiar?). First, WinMacro should be running. Next, start up Solitaire, and make sure that it is the current window. Now, activate keystroke record mode, as outlined above, and name the file SOLITARE.WBM. Once the WinMacro icon begins flashing, we're ready to record. Enter the following series of keystrokes:

Alt-G
C
Cursor right
Space
Enter

And end record mode. Now, create a WinMacro definition file named SOL.WDF, containing the following entry:

```
Change deck design      \ ^C   : winbatch.exe solitare.wbm
```

Finally, select **About/Reload** from the WinMacro icon's menu. Your new command is now available from the Solitaire control menu, or simply by typing **Ctrl-C** when the Solitaire window is active.

WBM files

If you look at a WBM file, you will see that it is nothing more than a series of one or more **SendKey** statements. For example, the SOLITARE.WBM file that we just created looks something like this:

```
; Recorded Macro D:\WINDOWS\BATCH\SOLITARE.WBM
SendKey(`!gc{RIGHT} {ENTER}`)
; End Recorded Macro
```

If you glance back at the SOLITARE.WBT file which appears at the end of the **Tutorial** section of the **WIL Reference Manual**, you will find a line which looks amazingly like the middle one above (~ has the same meaning as **{ENTER}**). This demonstrates that WBM files are simply WBT files in disguise.

So, why do we use different extensions for the two types of files? Consider, if you will, that a WBT file is a standalone program, which can be run from the Program Manager or File Manager. It starts up whatever other programs it needs, does its work, and cleans up after itself. A WBM file, on the other hand, is only a program fragment. When called, it sends a sequence of keystrokes to the active window, but it neither knows nor cares what window that may happen to be. In Solitaire, **Alt-G** selects the **Game** menu; in another program, it may trigger the **Goodbye** function. Needless to say, WBM files should be played back *only* in the window where they were recorded, and the easiest way to ensure this is to attach them to *application-specific* WDF files, as we have done here with Solitaire. That's why we distinguish them from regular WBT files.

However, because **SendKey** is a perfectly respectable WinBatch function and because WinMacro *does* generate **SendKey** statements it is quite useful to be able to record a WBM file, and later incorporate it into a full-fledged WinBatch file. Suppose that we had a one-line WinBatch WBT file like this:

```
RunZoom("sol.exe", "")
```

and we wanted to follow that with a **SendKey** statement to change the deck design every time the file was run. Instead of laboring over the WinBatch manual to find the cryptic symbols necessary to accomplish such a feat, we could simply use the WinMacro record feature to create a WBM file, as we did above, and then paste the resulting **SendKey** statement into the WinBatch WBT file:

```
RunZoom("sol.exe", "")
SendKey(`!gc{RIGHT} {ENTER}`)
```

You can also use your favorite editor to remove any accidental keystrokes you make when you are recording a WBM file.

Menu Utility for the Windows Explorer

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

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

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

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

System Requirements

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

Installation

FILEMENU is installed during the normal setup of WinBatch 95.

Operation

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

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

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

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

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

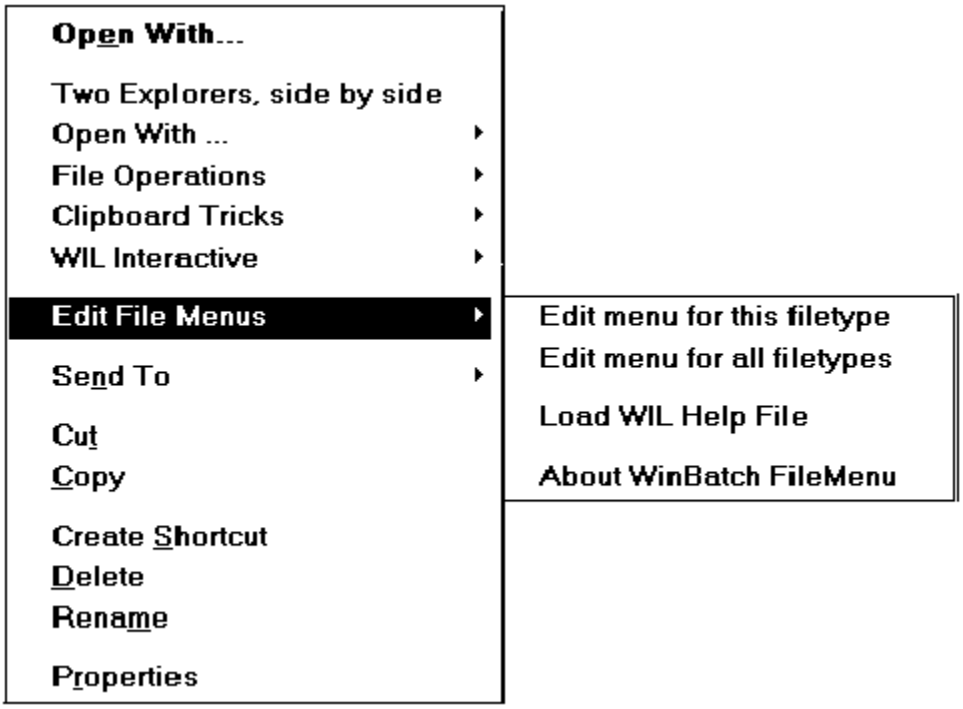
FILEMENU can add two menu files onto a file's context menu: the "all filetypes" menu, which is added to the context menu of every file, and a file-specific menu, whose entries depend on the type of file selected.

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

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

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

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



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

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

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

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

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

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

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

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

Note: Extensions can be longer than three characters.

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

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

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

Functions

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

CurrentFile
CurrentPath
CurrFilePath

The following functions are NOT supported:

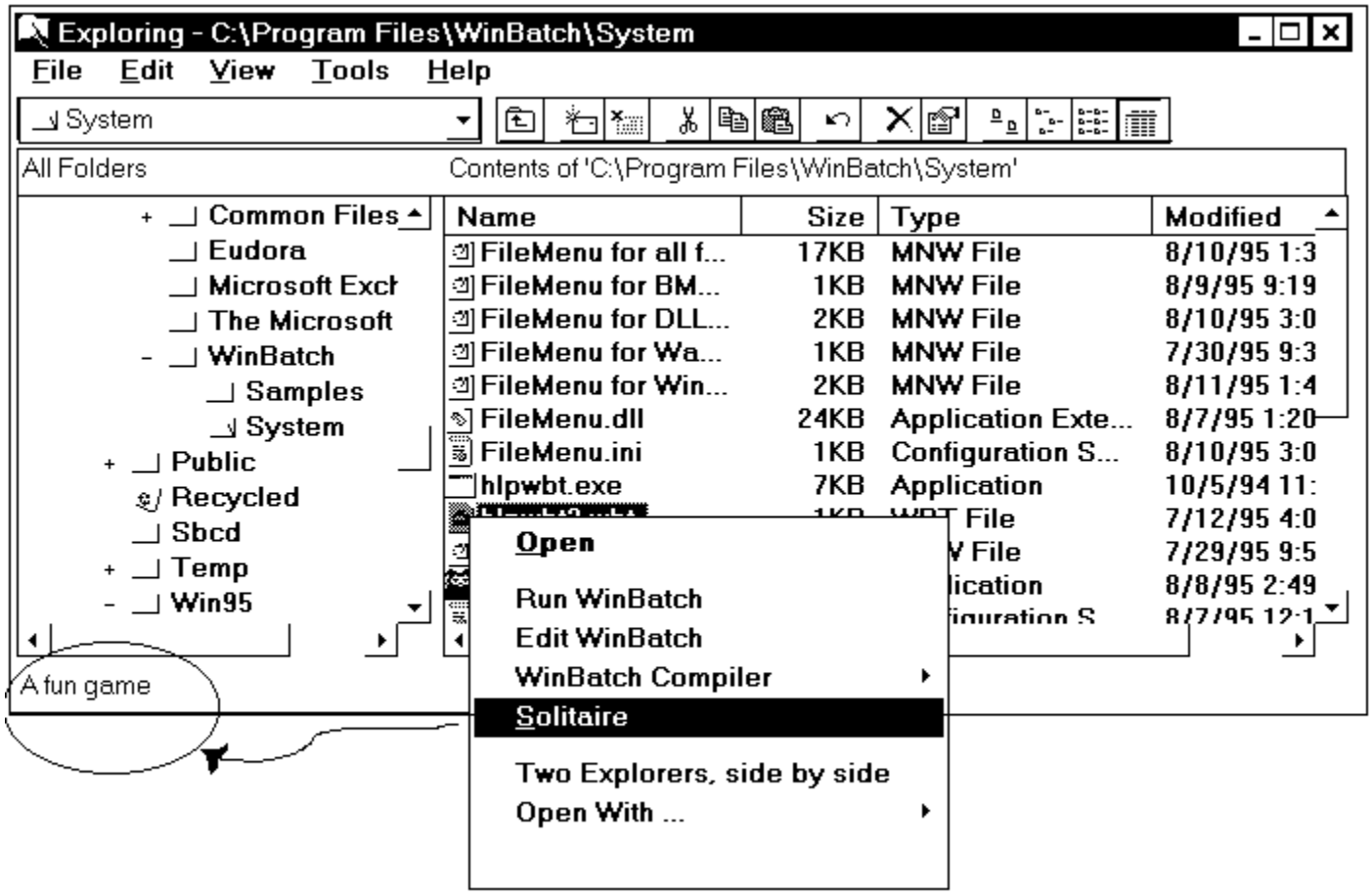
IsMenuChecked
IsMenuEnabled
MenuChange
Reload

Status Bar Comments

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

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

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



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

Hotkeys are not supported.

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

System Requirements
POPMENU requires Windows 95 or Windows NT.

Installation
To install POPMENU:

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

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

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

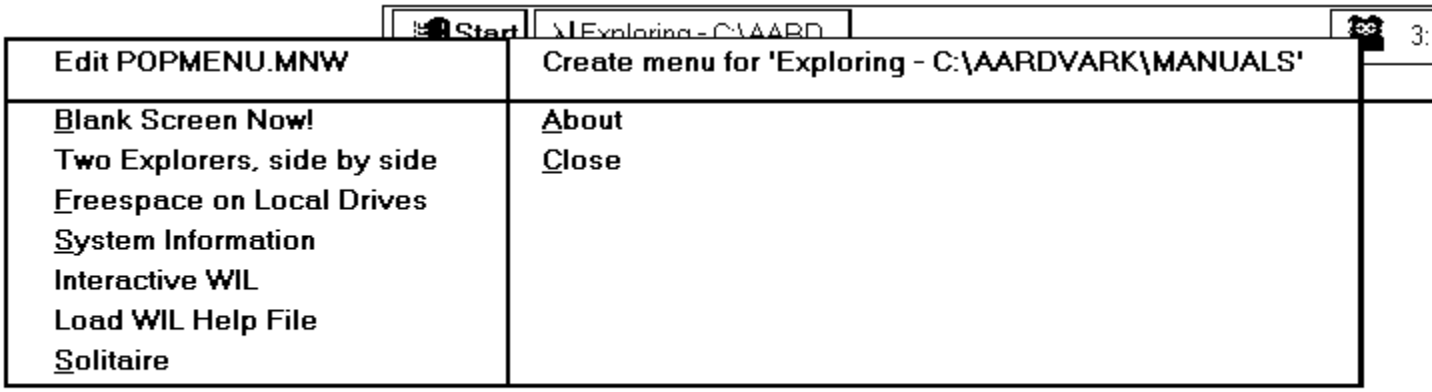
4. Set up your menu files (see "Menu Files", below), and place them in a directory on your path, or in your POPMENU directory. A sample menu file is included with the program. You can use it or adapt it to your requirements.

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

Operation
Start POPMENU by running POPMENU.EXE.

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

Close POPMENU by selecting "Close" from its menu.



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

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

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

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

POPMENU searches for menu files using the following sequence:

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

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

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

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

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

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

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

SkipGlobalMenu=1
Causes POPMENU not to load the global menu file. By default, the global menu file will be loaded.

SkipLocalMenu=1
Causes POPMENU not to load the window-specific local menu file. By default, the local menu file will be loaded.

SkipGlobalEdit=1
Causes POPMENU not to add a "Create/Edit menu" item at the top of the global menu. By default, the menu item will be added.

SkipLocalEdit=1
Causes POPMENU not to add a "Create/Edit menu" item at the top of the local menu. By default, the menu item will be added.

Functions

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

- BoxOpen
- BoxShut
- BoxText
- BoxTitle

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

- CurrentFile
- CurrentPath
- CurrFilePath
- IsMenuChecked
- IsMenuEnabled
- MenuChange
- Reload

Misc...

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

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

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

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

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

Hotkeys are not supported.

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

Status bar comments are not supported.