nono&PrintyesWINMACRO26/08/94

WinMacro

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

Starting WinMacro

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).

Macro Definition Files

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**.

Hotkeys

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.

Recording Keystrokes

Another feature of WinMacro is the ability to record keystrokes to a file, which can be played back at a later time. To do this, make sure that WinMacro is running, and then type **Ctrl-Shift-Home** from any window, or select **Begin Macro Record** from the WinMacro icon's menu. WinMacro will present you with 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-Shift-End** from any window**,** or click on the flashing WinMacro icon and select **End Macro Record** from the menu. The icon will stop flashing.

Once you have created a WBM keystroke macro file, you can assign it to a hotkey in a WDF file, using the steps outlined above. You use WinBatch to run WBM files, the same way you do with WBT files.

WinMacro Example

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.

Unrecordable Areas

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.

SendKey

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 +
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)*