Contents

Introduction

Gadfly Toolbox Reference

gLib

What is it?

This collection of macros began humbly enough during the initial release of *Word for Windows* as a group of useful toggle macros, a couple of window arrangement macros, and an envelope printing macro.

Many, if not most, of those macros came about either because I wanted them for my own use, or because someone on the *Word for Windows* forum of CompuServe suggested thus and such could not be done (and I took the bait). Many came about as a direct result of conversations with James Gleick, Barry Simon, and Robert Enns (the original other WinWord Gadfly Team members). I'd like, again, to thank them and all the other members of that remarkable forum, for their help and suggestions.

As the macros accumulated, and they became more and more complicated--both in terms of their functionality, and in terms of their "error handling"--these macros graduated from hobby to product. With the gathering of the orginal GJGMAx.ZIP files into GTOOLS.ZIP (and the addition of several new macros), they became the first ShareWare incarnation of my work.

Some new macros were kept separate, either because of functional specificity (like the printer related macros or the Current DDE macro GetAddress) or because they were so much larger and more "standa-alone-ish" than the others (like **ChooseDirectory** or **MacroKey**). This resulted in a bit of confusion.

With this release of what was **GTOOLS**, with a new name -- **GTOOLBOX** -- I am gathering the vast majority of my *Word for Windows* macros into a single package. And requesting a single registration price for all.

Encryption and Copyright

The macros in this package are not encrypted. This means you can easily modify their functionality to suite your particular preferences. And, if so inclined, you can learn quite a lot about WordBASIC by studying these macros.However, distributing any modified versions of these macros, in any manner, is strictly prohibited.

If you are interested in a site license, or a license to distribute modified/customized versions of these macros to your clients, please contact the author.

Pricing

New registrations

Registration Only	$39.95
Registration and Disk	$39.95 + $5.00 shipping

Upgrades from previous versions

There is no charge for registering GTOOLBOX if you have registered any **two** (or more) of the previously released packets:

GTOOLS2 (Gtools), **GPROE** (PrintRange), **GCDIR** (ChooseDirectory) or **GMKEY** (MacroKey)

If you have registered any **one** of the above packets, the upgrade to **gToolBox** is $10.00

Incremental updates

There may be incremental releases--to fix any bugs found in this release, to tweak performance, to add additional macros--between now and the next major release of *Word for Windows.* These "updates" will be free to all registered users.

ShareWare

This product is being distributed as ShareWare. This means you are under no obligation to pay for the product **unless** you continue to use it.

But if you do use (and learn from) these macros, please register. A great deal of effort went into their construction. Your registration will support the continuation of the Gadfly Macros series...

You are encouraged to distribute the package, as is, to other *Word for Windows* users.

ChangeQuoteDash

Purpose

ChangeQuoteDash allows you to easily toggle between Typewriter quotation marks and Typeset (Publishing) quotation marks. It also allows you to toggle between double hyphen and EMDash and single hyphen and ENDash.

Normally, when you type a quotation mark, *Word for Windows* inserts Character #34: "This is surrounded by normal quotation marks." And, similarly, when you insert a "single quotation mark", *Word for Windows* inserts Character #39, the Apostrophe: For instance:

"We're at the 'mercy' of our computers!" he said with chagrin.

The contraction apostrophe and the single quotation marks around 'mercy' are the same character.

Similarly, the opening and closing double quotation marks around the phrase are the same.

These marks are sometimes called Typewriter quotation marks.

Well, that's all you need if you are using a fixed font such as Courier or Prestige Elite. These fonts don't distinquish between Open and Close Quotation marks, However, proportionally spaced fonts, such as Times Roman, do.

Word for Windows provides a way to insert individuatl Publishing quotation marks with **InsertSymbol** command..

There is also a macro in the **newmacros.doc** called **SmartQuotes** which allows you to turn Publishing quotation marks on and off. (Note: ther is an improved version of SmartQuotes available on CIS under the named **GSMART.ZIP**. It's FreeWare.)

However, what if you don't always want Publishing Quotation marks? And what if you started a document without them and now you do want them?

ChangeQuoteDash is a SmartQuotes post processor.

A possible way, if you are like me, to use this macro, is before printing--that is, enter text without SmartQuotes installed, and then when you want a typeset style copy with Publishing characters, run this macro. If you don't save after printing, then the document will remain in typewriter format.

Conventions:

EmDash:

An EMDash should be represented, in Typewriter style, by two hyphens with not spaces before of after. Compare:

There is nothing--not a thing--wrong with this line, aside from silliness.

This line -- for what it's worth -- has spaces on either side of the dash.

EnDash:

An ENDash is not interchangeable with all single hyphens. It should be used in cases such as date ranges:

1990-1992

--would require an ENDash in Typeset style. Whereas compound words or hyphendation breaks would not.

For this reason, incidently, the option to change single hyphens to ENDashes requires user

intervention. You have to confirm each change.

Non-breaking hyphens

For the purposes of this macro, regular hyphens and non-breaking hyphens (entered with Ctrl+Shift+Hyphen) are equivalent. That is, a pair of non-breaking hyphens will be converted to an EMDash, and a single non-breaking Hyphen will be presented as an ENDash candidate. (Entering Non-breaking hyphens is covered on page 90 of the manual).

NOTE: since double hyphen pairs (Typewriter EMDash) should never be broken at a line end and a hyphen that would convert to an EMDash should never be separated from the two terms it separates, when you convert from Typeset back to Typewriter, the macro will use the non-breaking hyphen, not the normal hyphen.

Scope: (what it changes...)

ChangeQuoteDash makes the following assumptions about *what* you which it to change:

* If the selection, at the time of invoking the macro, is an insertion point (i.e. a blinking line), **ChangeQuoteDash** will operate on the entire document.
* If the selection is a solid block/selection (i.e. you have marked a section of text from the entire document), **ChangeQuoteDash** will operate **only on that selection.**

This is useful in the following scenario: You have a document, such as a manual, the majority of which is proportionally spaced (and therefore Typeset mark qualified). But there is a section of the manual that is formatted in Courier or LinePrinter--this section might be, for instance, a macro listing. In that section you do not want typeset marks. You want normal typewriter marks. Simply select the section of the document to toggle to typewriter and run the macro again.

ChangeQuoteDash: dialog box:

ChangeQuoteDash: dialog box:

This dialog box consists of three option buttons and a check box.

To typeset (publishing) characters

This button will change all normal inch/pound, typewriter quotation marks to the curly quotation marks used by typesetters.

It also automatically converts all hyphen and non-breaking hyphen pairs to an EMDash.

To Typewriter (normal) characters

This button will change all typeset type quotation marks back to the normal typewriter type.

It will also change all EMDashes to a pair of non-breaking hyphens, and all ENDashes to a single non-breaking hyphen.

Single Hyphen to EnDash

This option will begin a search and replace for all remaining hyphens and prompt the user if the found hyphen or non-breaking hyphen should be replaced with an ENDASH.

This last option **only** converts from hyphens to EnDashes. It should be run AFTER the first option (to typeset) and need not be run at all when converting back to typewriter).

Save to new file

If this box is check the macro will perform a **FileSaveAs** so that you can create a copy of your document before execution.

ChangeTool

One of the limitations of the built in **ToolsOptionsToolBar** command is that if you want to rearrange the icons in either NORMAL.DOT or a template you have to literally remove and replace each one.

This macro allows you to insert a new tool and move the rest to the right, replace a tool with another, replace a tool with a space, or remove a tool altogether.

ChangeTool: dialog box:

ChangeTool: dialog box:

The dialog box consists of six parts:

1) a listbox containing the **Current ToolBar**

2) to the right of this are two PushButtons and a checkbox which control the modification to be made; they are:

CheckBox:

> **Shift current toolsThis checkbox determines what will happen to the existing tools when either of the other modification buttons are selected.With Insert this shifts current tools to the right.With Delete this shifts current tools to the left.**

Buttons:

> **Insert new tool** - selecting this button will place the new tool at the current location.If Shift current tools is active (checked) the new tool will move existing tools to the right. If Shift current tools is inactive (not-checked) the new tool will replace the currently selected tool.
>
> **Delete selected tool** - selecting this button will remove the selected tool. If Shift current tools is active (checked) the existing tools will shift to the left. If Shift current tools is inactive (not-checked) the selected tool will be "deleted" and replaced by a Space.

3) an Editbox titled **Macro for tool**

> Type the macro to add to the ToolBar in this editbox. By default the proposed macro is [space].

4) To the right of this editbox is a group box containing one (if the current context is limited to NORMAL.DOT) or two (if the current document is based on a template and therefore there are two contexts: global and template) buttons.

> Selecting either of these buttons will display a list of the available macros.
>
> There is also a checkbox **Include commands**. This checkbox will list the built in Word for Windows commands as well as any user macros.
>
> Once the list of macros is displayed the selected macro is placed in the edit box.
>
> **Limitation**: the list of macros will not appear sorted. User macros are sorted according to their creation. The internal macro/commands are sorted according to some inscrutable whim on the part of the Word Development Team.

5) a ListBox of available icons.

> Since *Word for Windows* dialog boxes cannot contain bitmaps, there is no way to actually include the toolbar icons. Since the **ToolsOptionsToolBar** does not act like other dialog boxes (none of the ToolsOptions dialog boxes do), you can't use it to display and then grab the selected Icon. Therefore, the list box in this macro displays the number of the tool (-1 to 104) and a short description.

6) a group box that contains one (if this is a NORMAL document) or two (if it is a template document) option buttons to determine *where* to store the changes: in NORMAL.DOT or in the template.

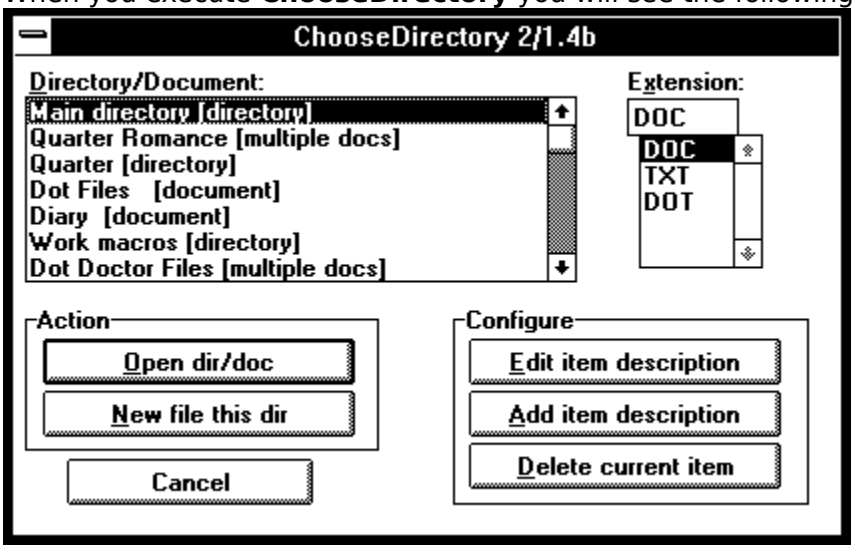
ChooseDirectory

ChooseDirectory was written to provide an easily customizable list of descriptions for documents, directories, and paths. Once configured properly it will present you with a list box of your most frequently accessed files and directories.

This version is specific to *Word for Windows* version 2.0. It has many enhancements, both in the interface and in the feature set.

The Dialog Box

When you execute **ChooseDirectory** you will see the following dialog box:



Directory/Document

This listbox contains descriptions, which you provide, that point to either a directory or a document or a path.. When you run **ChooseDirectory** for the first time, this list will be blank.

(The word contained in brackets is added automatically by **ChooseDirectory**. See below under Edit item description.)

Extensions

This combination box contains a list of extensions to use when you open a directory. The extension DOC is always a default. If you type in an alternate extension it will automatically be saved to the list and available next time you run **ChooseDirectory**.

(To remove an extension from the list, simply type the extension prefaced by an minus sign: e.g. **-TMP**.)

Open dir/doc

Selecting this option button executes a different function depending on what the currently selected description describes. The description can "point to" a directory, a single document, or a multiple path.

If it is a single path it will display the **FileOpen** dialog in that directory.

If it is a document, it will open that document.

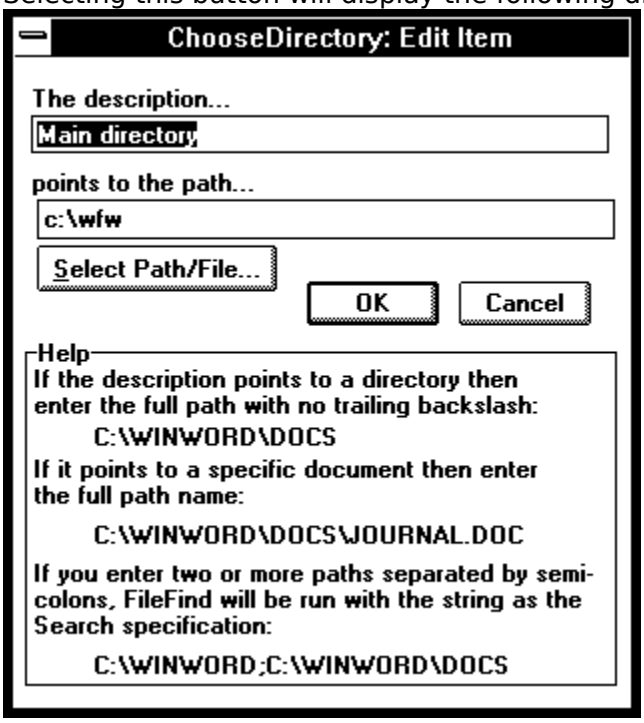
If it is a multiple path specification, it will run **FileFind** using that path specification as the SearchList.

New File this dir

Selecting this button simply changes to the directory pointed to by the description, and then executes **FileNew**.

Edit item description

Selecting this button will display the following dialog box:



If the path is a directory, [directory] will be added to the description.

e.g. C:\WINWORD

If the path is a document, [document] will be added to the description.

e.g C:\WINWORD\JOURNAL.DOC

If the path is a multiple path specification, [search path] will be added to the description.

e.g. C:\WINWORD;C:\DOCS

If the path points to multiple documents, [multiple docs] will be added to the description.

e.g. C:\WINWORD\JOURNAL.DOC+C:\DOCS\MY.DOC

Note: if the path specification points to multiple documents, then each of the documents will be opened.

There is a new facility on the Edit Item Description and Add Item Description dialog box. There is a button beneath the path edit box named "Select Path/File". Selecting this button will present you with the standard FileOpen dialog box. You can move around your hard disk, find the document you which to open.

To have ChooseDirectory work on just the path selected, be sure to remove the file name that is inserted (and the final backslash)

To do multiple paths remember to add a semi-colon between paths.

To do multiple documents remember to add a plus sign between file names.Add item description

Selecting this option does the same thing as **Edit item description** except instead of editing the currently selecting description/path, it adds a new description/path combination.

Delete current item

Selecting this option simply removes the currently selected description/path from the list. Note: there is no confirmation.

How it works
Installation Tips
Converting from Earlier versions of ChooseDirectory.
Advanced
Warning:

How it works

The macro uses a series of keywords in your WIN.INI file under [MS Word ChooseDirectory] to store alternate extensions, and path descriptions/specifications.

The alternate extensions are stored as follows:

xExt=*,TXT,XWS,DOT

When first run, this line will not exist. ChooseDirectory will prompt you for input. Note that the extensions are **just** that: no preceding period. Also note that the default extension (DOC) should not be entered in this line (though doing so won't harm anything).

The description/directories are stored in lines such as:

Dir0=Main Directory [directory|C:\WINDOWS\WINWORD

Dir1=My journal [document]|C:\WORD\JOURNAL.DOC

Dir2=Macro files [search path]|C:\WIN\WORD\MACROS

DirN

These lines are added as needed.

Installation Tips

To have **ChooseDirectory** run every time you start *Word for Windows*, simply either add the following line to your current **AutoExec** macro (create one if none exists):

ToolsMacro "ChooseDirectory", .Run

It is also recommended that you assign **ChooseDirectory** to both a menu and a key combination.

Converting from Earlier versions of ChooseDirectory.

Versions of **ChooseDirectory** intended for *Word for Windows* 1.1 used a series of keywords stored in the WIN.INI file under the heading [Microsoft Word]. *Word for Windows* 2.0 has a new heading, [Microsoft Word 2.0].

ChooseDirectory 2/x.x uses it's own heading: [MS Word ChooseDirectory]

If you want to start with the same directory, extension specifications with this version of ChooseDirectory as you had used with earlier, do the following:

Open Notepad.exe

Load WIN.INI

Search for the phrase [Microsoft Word].

Under that heading there should be the following Keywords:

dNum, xExt, and Dir0...DirN

Select all of those lines and move it to the heading [MS Word ChooseDirectory] (which already exists if you have run ChooseDirectory 2/1.02 even once.

Advanced

Specifying an extension

You can add a specific extension to a path.

For instance, if the description "My templates" pointed to a path in the following form:

C:\WINWORD\DOT*.DOT

Then upon execution this would run **FileOpen** on the C:\WINWORD\DOT directory, with *.DOT as the filter.

Note: the bracketed "type" description generated by **ChooseDirectory** will still read [document].

Similarly, you can add an extension to the items in a multiple path, as follows:

c:\wfw*.doc;c:\dot*.dot

Upon execution, FileFind will be run with this as the SearchPath. Note: the bracketed "type" description generated by ChooseDirectory will still read [search path].

Deleting an extension from the list

When you specify and extension, it is added to the list. To remove an extension from the list, run **ChooseDirectory** and type in the extension you wish to remove prefaced by a minus sign: e.g. **. -TMP**.

Sorting the extension list

Extensions will be displayed in the list in the following order: DOC will always come first; the remaining extensions will display in the order you entered them.

To re-arrange the alternative extensions you must edit the line xExt found in WIN.INI under the keyword [MS Word ChooseDirectory].

If can edit any of the [MS Word ChooseDirectory] items by using ToolsOptions and specifying MS Word ChooseDirectory as the application.

Warning:

The WIN.INI lines that contain the description/path specification **must** contain text on either side of the vertical bar (otherwise an error will be generated). This version of **ChooseDirectory** contains error checking to make it very difficult to incorrectly install these lines, however, nothing is fool proof... If you get an error when first running the macro, please check you WIN.INI (using NOTEPAD or SYSEDIT) and examine the section following [Microsoft Word]. Note: You can now also edit a specific section of WIN.INI in the ToolsOptionsWININI dialog box from within Word for Windows.

If you see a line such as the following --

Dir0=!c:\winword

Dir1=Main Directory|

-then you know there is a problem. Add the missing information (in the first case, a description, in the second case a path specification).

ControlRun

This macro is an adaptation of a macro posted by Doug Timpe for WfW 1.1. The only major change is the use of PushButtons instead of CheckBoxes.

It replaces completely the built in command named ControlRun. The advantage of this macro over the built in macro/command is that you can modify this macro to include frequently used program, and it has a text edit box into which you can enter any legal command.

ControlRun: dialog box:

ControlRun: dialog box:

By default the dialog box consists of a text edit box and four buttons.

You can enter any legal command into the edit box.

The PushButtons are

- **Clipboard**
- **Control panel**
- **Dialog editor**
- **Cancel**

You could modify this macro to include other frequently run programs.

CopyMacroActive

In my own work with macro development, this has proven one of the most useful tools in my NORMAL.DOT.

This macro will build a list of all the macros in the current context -- either the global macros or the macros in the reigning template -- and then build a second list of all the *other* templates that are currently loaded into *Word for Windows*, and allow you to choose a macro to copy and a destination template.

For example, if you are working on a letter in LETTER.DOT, and you have a document based on MEMO.DOT also loaded, running this macro will display a list of LETTER.DOT's macros and allow you to copy any or all of them into MEMO.DOT with a single click.

You can also encrypt the macro en route.

CopyMacroActive: dialog box

CopyMacroActive: dialog box

The dialog box consistes of two list boxes, two buttons, and one checkbox.

Macros in (templatename)

This list displays all of the macros in the active template at the time of executing **CopyMacroActive**.

Copy macro(s) to:

This second list box will display the possible destination templates (it gathers them by cycling through all open document windows and determining the rulling template. Therefore if there is only one document open the macro will exit).

Copy Selected

This button copies only the currently selected macro (and then moves the selection to the next macro in the list).

Copy All

This button will copy all the macros in the list to the selected destination template.

Encrypt

This check box, if active, will encrypt the macro during the copy process.

CreateOutlineDoc

This macro allows you to save only the heading text from a document to a second "outline" document.

Why? When you collapse a *Word for Windows* document into Outline View and then select the outline, you are actually selecting all the text and levels beneath the visible outline. If you were to cut and paste to a second, blank document you would be copying everything.

This macro allows you to create a document that reflects *only* the outline organization of its source document.

CreateOutlineDoc: dialog box

CreateOutlineDoc: dialog box

Outline Document Name

This is the name used to save the newly created outline document. By default the name will be the file name portion of the document being converted with the document extension OTL appended.

By default the OTL file will be saved in the same directory as the source document.

If you wish to change either of these defaults, type the filename desired, or the full path desired, in the edit box.

Outline Document Template

By default the new document will be based on the same template as the source document. If the source document is a template itself, then the destination document will be based on NORMAL.DOT.

If you have a standard outline template, you can enter it into the macro as the default. For example, if you have a template named OUTLINE.DOT used for all outlines, simply locate and change the line

```
dlg.Template = Template$
```

to

```
dlg.Template = "OUTLINE"
```

Keep Footnotes

Keep Annotations

Simply check the appropriate box to include footnotes and/or annotations in the new document.

Be aware that the footnotes numbering will not be the same as the numbering in the original document since the footnotes that are in the text are ignored.

Also, be aware that footnotes that do not have automatic numbering will be ignored.

Expand to level

You can specify how many levels of the outline you wish to preserve during the copy process.

DeleteIdleStyles

This macro is useful if you have overloaded a document with unused (and sometimes unknown...) styles.

This can happen when you convert from another word processor to *Word for Windows*, or when you combine documents from several different sources, or when you merge styles from another document or template (but don't want all the styles to remain in the current document).

Unfortunately, there is a limitation. There are several (thirty odd, I think) built in style names:

* annotation reference,annotation text,footer,footnote reference,footnote " + "text,header,heading 1,heading 2,heading 3,heading 4,heading 5,heading 6,heading 7,"heading 8,heading 9,index 1,index 2,index 3,index 4,index 5,index 6,index 7,index heading,"line number,Normal,Normal Indent,toc 1,toc 2,toc 3,toc 4,toc 5,toc 6,toc 7,toc 8,EnvelopeAddress,EnvelopeReturn

Often you will see that a document that starts out with only Normal, Normal Indent, and Heading 1-3 suddenly has a list containing **all** of the above style names.

There is no way to remove these styles from the style list once they appear. This is a *feature* in *Word for Windows*. Write Microsoft...

When you run this macro you will be given an opportunity to cancel.

Be careful **not to run** this macro unless you are sure that all the style names you wish to retain are actually in use in the document.

Another Note:

There is no reason (that I can think of) that you couldn't remove user styles from a template by following these steps:

* Load the DOT file directly.
* Examine the user styles in the template
* Apply those you wish to keep to separate paragraphs
* Run the macro

EncryptMacros

This macro will display a list of macros in the currently active template and either encrypt all the macros, or allow you to specify which of the macros to encrypt

WARNING: once a macro is encrypted, it cannot be unencrypted. BE SURE TO HAVE AN UNENCRYPTED VERSION OF ANY MACRO YOU ARE DEVELOPING BEFORE ENCRYPTING IT.

See EncryptTemplate

EncryptTemplate

This macro will ask you to choose a template, and then will encrypt every macro in that template.

Note: it does not pause for confirmation. When you load a template the macros contained in that template are encrypted. The macro does not, however, automatically save the newly encrypted template. So you do have the opportunity to discard the changes.

WARNING: once a macro is encrypted, it cannot be unencrypted. BE SURE TO HAVE AN UNENCRYPTED VERSION OF ANY MACRO YOU ARE DEVELOPING BEFORE ENCRYPTING IT.

See EncryptMacros

FiddleNotes

This is a simple macro that has four options:

* Change all footnotes to annotations
* Change all annotations to footnotes
* Delete all footnotes
* Delete all annotations

There is only one caveat: Footnote References and Annotation References must all have the same style. A buglet in *Word for Windows* allows for styles in such references to get out of kilter (since changing a reference style does not apply retro-actively).

Another macro in this package addresses this problem.

See ReInsertFootnotes

FileTemplate

Problem: automatically merging styles

Templates are the repository of several useful aspects of *Word for Windows*. If you aren't comfortable with the idea of a Template or the concept of Context, I strongly suggest you stare at the manual until it's clear.

Templates hold Macros, IconBar assignments, Key Assignments, Menu Assignments, Glossaries, and Styles.

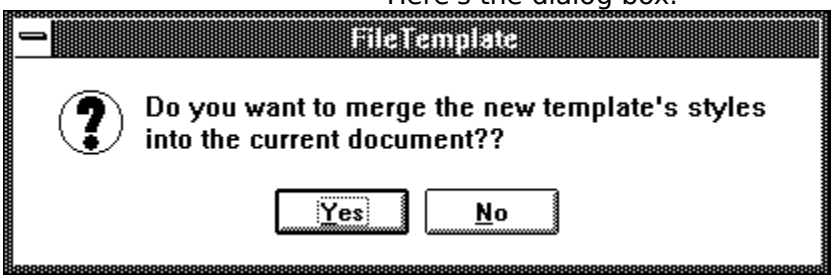
When you attach a template, or change the attached template, using the new **FileTemplate** command (used to be **FormatDocument.Template**...) you essentially change which of these assignments will now have sway over your document. That is, with the exception of styles.

Changing a template does not automatically change the style definitions in the current document.

The reason for this is that there are two separate lists of styles -- those contained in the Template (where you probably defined the style) and those contained in the document itself.

The included replacement for **FileTemplate** asks if you wish to change the current documents styles to match those contained in the template just selected for attaching...

Here's the dialog box.



Answering Yes to this prompt would perform the exact equivalent of selecting **FormatStyle, Define, Merge**, selecting the same template as that chosen in the FileTemplate dialog box, and selecting **To.**

For those of you who know the macro language, the short hand for this would be:

FormatStyle .FileName = Template$, .Merge, .Source = 1

Answering Yes, therefore, would *replace* any styles in the document with styles in the template of the same name. That is, Normal in the document would now look like Normal from the template. Styles that don't already exist in the document will be *added*.

See also SyncStyles

MacroKey

Overview:

Allows simple assignment and removal of key stroke combinations to macros (either global or template bound).

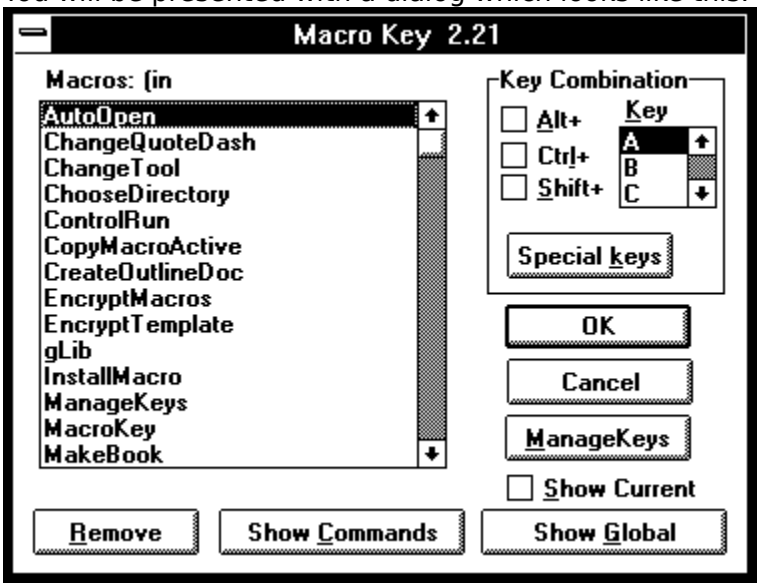
Enhancements over the built in MacroAssignToKey:

Allows the use of Alt-Char, Alt-Ctrl-Char, and Alt-Shift-Char. In the built in routine the first two combinations are disallowed. The last is reserved for *Word for Windows*'s use. Note that *Word for Windows* reserves these key sequences for it's own use in many cases, so preventing you from reassigning them is a form of protection. Use this routine cautiously, and backup all DOT files before you experiment.

Usage:

Main dialog box

You will be presented with a dialog which looks like this:



The line above the list of macros will display the current template or document/template containing the macros.

By default MacroKey displays the template macros (if there are any).

Macros

A Listbox of macros from the currently selected context. Select the macro you wish to assign a key to (or remove a key from).

Key Combination

Check the shift keys you wish to activate (Alt, Ctrl, Shift, in any combination), and then move to the Key list box and select the alphanumeric key you wish to assign to the selected macro.

Buttons

Show Global

Displayed only if the current context is a template (and therefore Global macros are not

visible) Selecting this button changes the list to display the macros from Normal.dot.

Show Template

This button is Displayed only if there *is* a template attached to the current document.

Show Commands

Selecting this button will display **all** of the built in command macros. Note, building a list of all macros -- user and internal -- take s fairly long time.

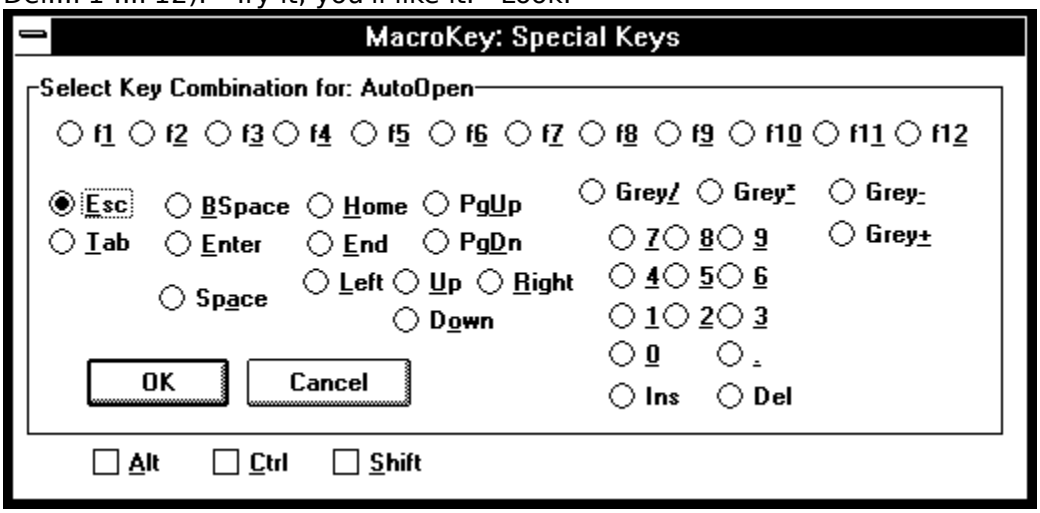
Manage Keys

This button will run a second macro (assuming it is installed) named **ManageKeys**. This macro will display a listbox of the current key assignments, and, optionally, allow you to create a document containing a table of those key assignments.

See ManageKeys

Special Keys

choosing this button will display a dialog box of all supported special keys (Esc through Del...F1 ...F12). Try it, you'll like it. Look:



Remove

this option checks to see if the currently selected macro has a keycode assigned to it. If so you are asked if you want to remove it. If not, a message displays at the bottom of the screen.

CheckBoxes

Show Current

This check box will rebuild the list of macros, displaying the current key assignment (if any). This takes a significant amount of time. This function only shows key assignments made by the user, not key assignments that are internal to *Word for Windows.*

Limitations:

Limitations:

MacroKey will warn you if you are about to use a key combination that is already assigned to a macro. However, it will only warn you if the key combination in question is **NOT** one of the default assignments. That is, it must be an assignments you have made, rather than one that *Word for Windows* assumes is the case.

Similarly, it does not warn you if you are about to use a key combination that is assigned to a style.

Illegal keys

It seems that any combination of keystroke with F1 will always call help. So it isn't fair game.

It seems that NumPad 5 (the numpad 5 that is activated by turning NumLock ON) does not accept shift characters. The NumPad 5 that is active in default state (NumLock OFF) does accept shift characters. The macro accounts for this oddity. Also note that the Unshifted NumLockOFF-5 cannot be assigned to a macro (this is the key, by the way, that is described as KeyCode 12 in the Technical Reference).

The direction keys, listed on the Special Keys dialog box, have some limitations.

No direction keys (Home, End, PgUp, PgDn, Right, Left, Down, Up) can be assigned to a macro.

No direction key plus Ctrl can be assigned to a macro.

No direction key plus Ctrl+Shift can be assigned to a macro.

The Alt key plus Up and Down are legal. The Alt key plus Left and Right are not legal.

MacroKey will not allow you to assign anything to Ctrl-Alt-Del (duh...)

Sorting

No sorting of the macros. The template macros will appear in the order you created them. The built in macros (if that option is checked) will appear in the order God created them.

Focus

MacroKey works best on a document, not on a template. That is, if you are editing a template directly there may be trouble. Base a blank document on the template and try again.

MakeBook

A slightly more complicated version of MakeWordBook, this macro takes the current selection and presents it for editing.

Instead of grabbing the current word if there is not selection it grabs the current line.

It presents the multi-word selection as a proposed bookmark, with spaces converted to underline marks

It also checks for illegal characters (punctuation marks and the like) and replaces them with underscores.

See MakeWordBook

MakeWordBook

This macro will first check to see if there is a block of text selected. If so, that selection will be inserted as a glossary name.

If there is no selected text (just an insertion point cursor), then this macro will take the current word and make it a glossary name.

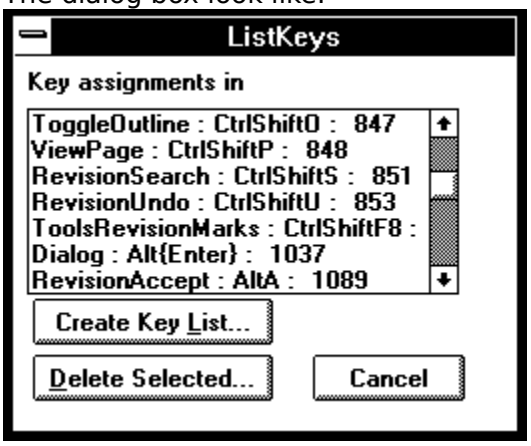
This macro will check for duplicate glossary names and append the instance count to the word if there is already a glossary using the word as the mark. It does not, however check for illegal punctuation marks in the current selection.

See MakeBook

ManageKeys

This macro will display the current key assignments for the active context. If you are currently working on a document based on NORMAL.DOT, it will display the "global" key assignments. If you are working on a document based on LETTER.DOT, it will display the key assingments specific to the template LETTER.DOT.

The dialog box look like:



The List Box

The list box displays three pieces of information:

The macro that has a key assignment

The mnemonic for the key combination

The keycode for the key combination

For instance, in the above list, the macro **NextWindow** is assigned to Ctrl+Tab, which is keycode 265.

Buttons:

Create Key List

This button will create a new document (based on NORMAL.DOT) and generate a table containing all of the key assignments.

Delete Selected

This button will remove the currently selected key assignment.

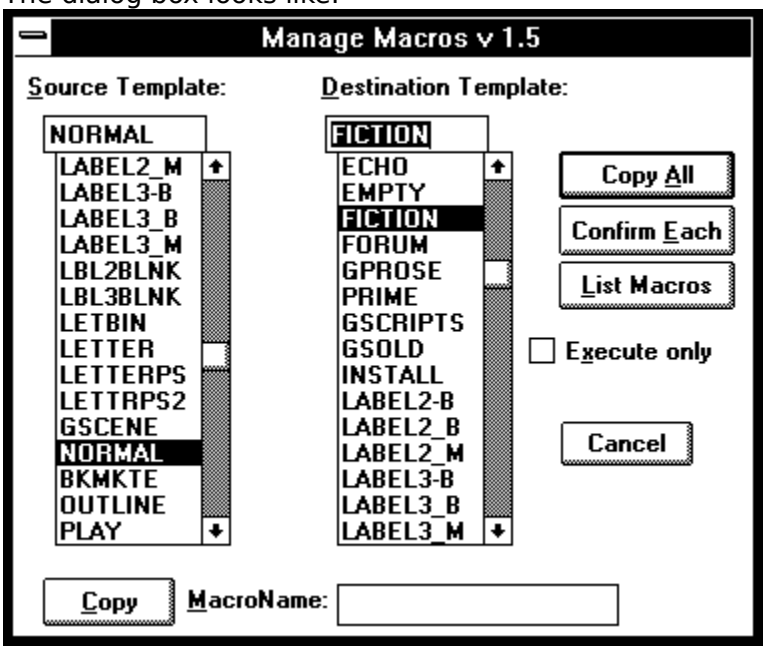
Note: It is possible for key assignments to appear in a template with not macro (apparently) associated with that key combination. If you see a line in the above list box that begins with a colon (and no macro name) you can pretty safely assume that somehow a bogus key assignment did not get nuked properly at some point. I'd delete it using ManageKeys.

ManageMacros

This is the orginal "copy macro" macro for *Word for Widnows* 2.0. A slightly different version exists in the *Using WordBASIC* companion disk.

It differs from **CopyMacroActive** in that you can specify both the source and destination templates (neither has to be currently open).

The dialog box looks like:



Souce Template

A list box of all the templates in your DOT-PATH.

Destination Template

The same list of templates

You must specify both a source and a destination template.

You can type a template name directly into the edit portion of the ComboBox.

The buttons available (after selecting the templates) are:

Copy All

This button will open the working templates and copy all source macros to the destination in one pass.

Confirm Each

This button will begin the process of copying all macros from source to destination but will pause at each macro for confirmation.

List Macros

This macro will display a list box of macros in the source template.

Copy

This button is only relevant if you have entered a macro into the editbox to the right. This is

useful if you wish to copy only one macro and you know the macro name.

You can also, optionally, encrypt the macro en route from source to destination.

Limitation

The templates do not display in the list in alphabetical order. The more templates you have, the slower the macro is to load.

In order for the templates to display sorted alphabetically, you must physically sort them on your hard disk with a utilitiy such as Norton's DiskSort.

See also CopyMacroActive

OpenMRUList

A simple macro that simply opens all the documents found under the File menu (the Most Recently Used List).

I would be useful as a startup command from Program Manager:

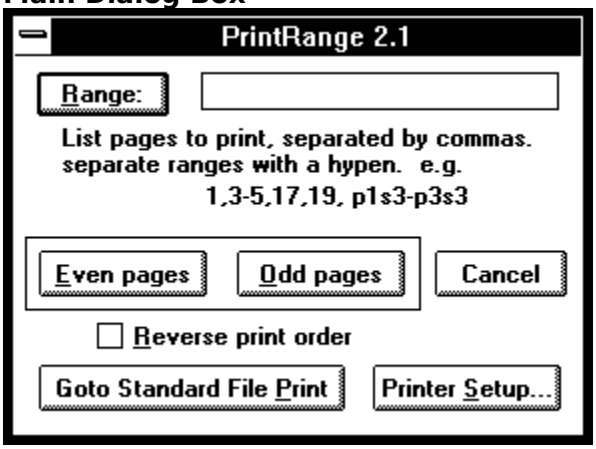
    C:\WINWORD\WINWORD.EXE /mOpenMRUList

--would start *Word for Windows* and open the files on the MRU list.

PrintRange

PrintRange is a simple macro that fills a major lack in *Word for Windows*

Main Dialog Box



Range

Selecting this button will print the comma delimited list of page numbers entered into the box to its right.

If you select Range then you will not be able to print Odd or Even pages.

Even Pages

Prints only the even pages in the current document.

Odd Pages

Prints only the odd pages in the current document.

Reverse Print Order

This check box is useful if you want to do "duplex" printing. Simple check this box, print all Even pages, then place the stack of pages back into the printer tray and print Odd Pages.

Goto Standard FilePrint

As a convenience, **PrintRange** allows you to call the standard **FilePrint** macro.

Printer Setup

Runs the standard Printer Setup macro to allow you to change the currently active printer.

Limitation:

At this time **PrintRange** is an either or proposition. That is, if you specify a Range you cannot also specify Odd or Even.

That may come in the next version.

ReInsertFootnotes

Note on Footnotes:

If you change the character style definition of a footnote or an annotation reference) instances of footnote reference inserted *prior* to he re-definition will not change. Subsequent footnote insertions will have the new attributes.

For example, if you begin inserting footnotes with the character definition of Red and Hidden, then change it to Blue and non-hidden, the already inserted footnotes will not change to the new style.

Another manifestation of this problem arises if you select a paragraph that has a footnote in it, and press Ctrl-SpaceBar to reset the character formatting.

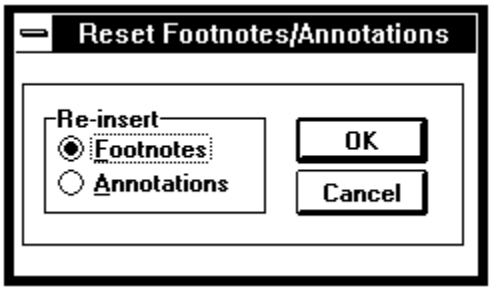
The included macro, ResetFootAnnote does not, like the other gFixes, replace a built in command. It is a new command which you can execute in order to reinsert a footnote or an annotation.

How it works

There are two modes of operation:

If the insertion point is currently positioned on a footnote reference or an annotation mark the macro will re-insert *that footnote or annotation only* and then stop.

If the insertion point is **not** on a footnote reference or an annotation mark, it will prompt you for which to re-insert document wide with a dialog box:



See also FiddleNotes

SyncStyles

This simple macro does the equivalent of FormatStyle.Option.Merge. It is useful if you want to make sure that the styles in a document are always also stored in the template; or if you want to reset the styles contained in the current document to the styles as they are defined in the template.

See also FileTemplate

SyncStyles: dialog box:

SyncStyles: dialog box:

Copy TO

If you select "Copy Document style TO template", you will, in essence, be merging the styles contained in the document to the current document template.

1. If a style exists in both the template and the document, the template definition is changed to match the current definition as it exists in the template.
2. If a style does not exist in the template, but does exist in the document, it is added to the style list of the template (that is, future documents based on that template will now have the additional styles...).

Copy FROM

If you select "Copy FROM Template styles into the current document", you will be merging all of the styles contained in the template into the current docuement.

The effect of these actions is two fold:

1. If a style exists in both the template and the document, that style is changed to match the template.
2. If a style does not exist in the document, but does exist in the template, it is added to the style list.

ToggleHidden

Toggle View Hidden -- without going through the ToolsOptionsView dialogue box. I suggest assigning this to Ctrl-Shif-H.

ToggleOutline

Toggles in and out of outline mode.

TogglePictures

Toggles the display of pictures and picture placeholders.

Word for Windows will scroll faster through a complex document if pictures are not displayed.

ToggleRevision

Toggle revision marks on and off.

ToggleStyleBar

Toggles the style bar on and off.

ToggleViewTools

There are seven "tools" that can be displayed on the *Word for Windows* screen:

* ToolBar
* Ribbon
* Ruler
* Status
* Vertical Scrollbar
* Horizontal Scrollbar
* Style Bar

The first three are controlled under the **View** menu. The last four are controlled under the **ToolsOptionsView** dialog box.

This macro present a single, simple dialog box of seven checkboxes. It has three buttons (not counting Cancel...):

On

Off

All On

Setting tolerance for toggle
Setting default minimum for on:

Setting tolerance for toggle

There is a line in the macro that determines how many options need to be set to on before the macro will assume you want the default action to be OFF.

That is, if three options are currently on, then make the default action for the push buttons to be Off.

If less than three are on, leave the default as On.

The line of code looks like:

```
If Total >= 3 Then St = 2 Else St = 1
```

Setting default minimum for on:

If you examine the macro you will see that there is also a section you can customize to specify which group of options should be forced to on *if everything is already off...*

That code looks like:

```
If Total = 0 Then    b = - 1 : vt = - 1 : v = - 1
```

ToggleWindow

Toggle the currently active document window split/zoomed.

WindowStack

Simply stacks the current document windows, leaving the title bars visible. There are two constants in the macro, HLap and VLap, which can be adjusted if the overlap does not suit your preferences.

The macro is not device dependent. It should work at all monitor resolutions.

WinSideBySide

Arranges the document Windows side by side. If there are more than two active windows you are prompted for which window to arrange next to the currently active document.

Overview

When you open **GTOOLBOX.DOC** you will automatically install the library of routines found in a macro called **GLIB**. This macro will be automatically copied to your **NORMAL.DOT.**

The reason for this is two fold:

1) *Word for Windows* version 2.0 allows one to call sub routines and functions contained in another macro. By placing several commonly used subroutines in one place, the other macros are smaller.

2) The subroutines and functions found in **GLIB** can be used by your own macros. In essence, it adds many new macro functions to WordBasic.

In order to use these functions in your own macros you simply must 1) make sure **gLib** is installed in your Normal.dot and 2) call the function prefaced by the library name: **gLib.**FunctionName**.**

Warning

If you modify any of the routines in the **gLib** library, be sure the increment the version number.

Subsequent installations of **gToolBox** (this version or a newer version) will check to see what the version number of the library is. If it is the same as the library to be installed nothing is coped.

If it is less than or gerater than the library to be installed the current library will be copied to another macro named **gLibBAK.**

Main dialog box

If you run **gLib** itself, you will see a listing of the subroutines and functions it contains:

gLib Reference

ActivatePartial(Doc$)

This subroutine can be used to supplement the built in command **Activate WindowName$.** The difference between the two is that this procedure only requires a partial name, whereas the built in command requires that you know precisely what's up there on the document bar.

e.g.

ActivatePartial("GTOOL")

Will cycle through each open window and stop at the first one whose title contains the letters "gtool" (not case-sensitive).

CheckLib

This function returns the version number of **gLib.**

```
If glib.CheckLib Then Print "Okay"
```

Chew$(Source$,Marker$)

A string function that returns the left portion of a string up to the occurence of the marker.

The source string is truncated. So, for instance:

```
S$ = "one|two|three|four"
m$ = "|"
For x = 1 to 4
      MsgBox glib.Chew$(s$,m$)
Next x
```

--would display four mesage boxes, containing one, two, three and four, in sequence.

CountChar(Source$, Char$)

A function that returns the number of occurences in a string of a specific character:

x = gLib.CountChar("one,two,three",",")

--would return 2.

fExist(FullPathName$)

This function returns -1 if the specified file was found; 0 if not.

fFileCount(FileSpec$)

This function counts the number of files in a directory:

```
DotCount = gLib.fFileCount("c:\winword\*.dot")
```

fFileName$(b$)

This function returns the filename portion of a full path name.

```
Name$ = gLib.fFileName$("c:\winword\normal.dot")
```

would return "normal".

fFileNameExt$(b$)

Returns the filename plus extension from a full path designation.

```
Name$ = gLib.fFileNameExt$("c:\winword\normal.dot")
```

would return "normal.dot".

fStr$(Num)

Returns a formatted string from an integer, stripping the padding Word places in front of a digit.

GetDocDir$

Returns the directory in which a document is stored.

If this function is called from within a macro pane it returns a null string.

GetFile(FullName$,Dir$,Default$)

Displays the FileOpen dialog box.

GetPath$(Source$)

Strips out a path from a full file/path specification

GetTemplate$

Returns the current document's template

gLibInstalled(Macro$)

This function is unique in the package. It should be used, not from within the **gLib** macro, but should be copied into any macro that calls a **gLib** routine.

What it does is use the same logic as **MacroExist** to determine if there is, in fact, a macro named **gLib** in the current Normal.dot.

This allows you to test at the top of a macro that requires **gLib** if the library is installed. It displays a warning message and request for re-installation.

If you look at almost any of the more complicated macros in **gToolBox** you will see an example of how this is used.

gMsg(Msg$,Title$)

Displays a message with an attention sign.

gQuery(Msg$,Title$)

Displays a query dialog box.

HasKey(Macro$,Context)

Returns whether or not a macro in the given context has a key assignment.

Inject$(Source$, New$, Place)

A string function to insert a string within a string at a given place.

```
    Name$ = gLib.Inject$(Name$("Guy Gallo", "J. ",4)
--would return "Guy J. Gallo"
```

IsMacroPane([MacroName$])

Returns -1 if the focus is currently on a macro editing window

MacroName$ is an optional parameter containing the name of the macro calling the function. This is so that the warning message will be smart and tell you the name of the macro causing trouble...

KeyDescription$(KeyCode)

This function returns a string mnemonic for the specified KeyCode.

lHelp(HelpFileName$)

Loads the specified HLP file.

```
glib.lHelp("gtools.hlp")
```

--would load the windows help engine and attempt to load the file GTOOLS.HLP.

(Both this and the following sub routine were adapted (very slightly) from a routine by Julianne Sharer of WexTech Systems.)

lcHelp(ContextNumber,HelpFileName$)

Loads the specified HLP file with a context number.

```
glib.lHelp(2,"gtools.hlp")
```

--would load the windows help engine and attempt to load the file GTOOLS.HLP and moves to the help page specifed by the context number 2.

ListMacros$(Context,All)

A simple fuction that displays a list of macros and returns the selected macro or a null string (if cancelled)

It will displays a list box of all macros in the specfied context (0 for global and 1 for the current template).

If All is 0 then only user macros will be displayed. If All is 1 then the built in commands will also be displayed.

Limitation: this routine does not sort the list of user macros. User macros are displayed in the order they were created.

The built in commands, for who knows what reason, are also not sorted alphabetically...

A sort routine (found in WOPR) could be added, but it would significantly slow things down.

LoopMsg(Message$)

This is an extremely useful function, to be used while writing macros. It's purpose is to display the value of any number of variables *during the execution of a loop.*

For example:

```
Loop = -1      'set the boolean variable Loop to true
For x = 64 to 255
      StringVar$ = StringVar$+ StringVar$ + Chr$(x)
      Index = x
      If Loop Then Loop = glib.LoopMsg("Index: " +Str$(x) + "; String: "+
StringVar$)
Next X
```

When the function displays the variables, you have two choices, **OK** and **Quit the loop.**

If you quit the loop the variable Loop is set to false, the loop continues on it's way without displaying a message...

MacroExist(Macro$)

Returns True or False (-1/0) if a macro exists.

Takes a string argument in the same form as IsExecuteOnly.

MacroExist([TemplateName:]MacroName)

e.g. MacroExist("NORMAL.DOT:GLIB")

If the template name is omitted the macro will look in the current document's template first and then in the global context.

NoSlash(Source$)

This function checks to see if the last character in a string is a backslash and if so, strips it off. This is useful for properly formatting a path specification for **ChDir.**

NukeTopMenu

Dangerous. Will rename all of the top menus to a space character.

To be used in conjuntion with **ResetTopMenus** and the internal command **RenameMenu.**

Replace$(Source$, Old$, New$)

This function replaces every occurence in a string of one string with another string.

```
    N$ = glib.Replace("h*e*l*l*o","*","_")
--would return the string "h_e_l_l_o"
```

ResetTopMenu

Resets the top menus to the default names. Used by a macro that calls **NukeTopMenu.**

Reverse$(Source$, Marker$)

This function rearranges a string along a given character or group of characters. This is useful, in combination with **Replace$()** when you want to change LastName,FirstName into Firstname LastName. For example:

```
Name$ = "Gallo,Guy"
Name$ = gLib.Reverse$(Name$,",")
Name$ = gLib.Replace$(Name$,",",Chr$(32))
```

After these three lines the variable Name$ would hold the string "Guy Gallo".

SameFormat

Returns true if every character in the selection has the identical character attributes.

If any character in the selection has a different character attribute it returns false.

SelectSameFormatRight

Extends the selection until encountering a different character format from the starting point.

This is used by **ReInsertNotes**.

This subroutine calls another function in **gLib**, named SameFormat

```
Sub SelectSameFormatRight     'Extends the selection until some part of the
character
                              'formatting changes
CharRight 1, 1
While SameFormat
      CharRight 1, 1
Wend
CharLeft 1, 1
End Sub
```

A corresponding **SelectSameFormatLeft** could easily be implemented as well.

Split(Source$, Marker$, First$, Second$)

This is a subroutine that will take a string and divde it in two at a specific point.

For example, if Source$ = "Guy+Gallo", the command

```
gLib.Split(Source$,"+",First$,Second$)
```

would return store "Guy" in the variable First$, and "Gallo" in the variable Second$.

Trim$(Source$, ZapChar$)

This function strips a character from both sides of a string. So, for instance:

```
Pad$ = "****Test****"
Unpad$ = gLib.Trim$(Pad$,"*")
```

--would return, in the variable Unpad$, the string "Test"

Wait(Seconds)

Not astronomically accurate... I think it's dependent on CPU speed... but still useful in debugging and testing macros...

Back in the early days of Word for Windows there was a forum on CompuServe where the pioneers gathered. A handful of the programmers and testers from Microsoft joined conversations and answered questions and fielded attacks and complaints. They signed their messages "WinWord Development Team". I took to signing mine "WinWord Gadfly Team."

An EM is a unit of measurement taken from the character box, as filled by the letter "M". An EMDash is on em wide. An ENDash is one-half of an EMDash.

For the proper usage of EM and ENDashes, see the Chicago Manual of Style, page 150.