

Contents

[Overview](#) | [Quick Start](#) | [Syntax Conventions](#)

Macro Mania® Commands:

- [Activate](#)
- [Beep](#)
- [Branch](#)
- [Clipboard...](#)
- [End](#)
- [Exit](#)
- [File...](#)
- [GetInput](#)
- [GoTo](#)
- [If-Then](#)
- [Let...](#)
- [Macro](#)
- [Minimize](#)
- [Mouse...](#)
- [Msg...](#)
- [OnTop](#)
- [Open](#)
- [Pause](#)
- [PlayWav](#)
- [Rem](#)
- [Repeat](#)
- [Repeat-Prompt](#)
- [Restore](#)
- [Run](#)
- [Send](#)
- [SendInput](#)
- [SendNow](#)
- [WinExit](#)

The [Run](#) and [Send](#) commands are used most often.

- [Important Tricks and Techniques](#)
- [How to Pay for this Program](#)
- [Benefits of Paying for this Program](#)
- [Technical Support](#)
- [Macro Mania License Agreement](#)
- [Our Reseller Program](#)
- [Revision History](#)

Macro Mania is a registered trademark

Overview

Macro Mania is a powerful, yet very easy-to-use macro program that works with *any* Windows program. Macro Mania will launch programs (or switch to currently running programs) and then send any keystroke to that program with one simple command, just as if you had manually started the program and typed in all the keystrokes yourself. Macro Mania takes the powerful concept of macros to the next level. Because it runs external to other programs, it can launch new or activate already running programs, transfer information between programs, use and manipulate variables, wait for one program to finish and then start another, schedule macros to run unattended, and much more.

If you are like many of the *Macro Maniacs* that continue to emerge, you will find yourself creating macros for all kinds of tasks that require repetitive tasks. Some of them will be permanent, crucial parts of your daily activities and others will be temporary macros you quickly create to save you a lot of time and keystrokes for one-time projects. Since its introduction in 1995, Macro Mania continues to receive positive comments from professional software reviewers as well as from *many* Windows users that believe, as we do, a PC should work for us and not vice versa.

We are sure you will find terrific time-savings tasks to assign Macro Mania. Once you learn to program macros for your tasks, you will save an enormous amount of time and likely find Macro Mania worth 100 times or more its price.

SEE ALSO:

[Quick Start](#)

Revision History

As evident by the Revision History, Macro Mania gets new features as fast as time and ideas permit. Typically minor revisions are released every few months and major versions about once a year. Just point your web browser to <http://www.nstarsolutions.com/mm> to make sure you have the latest version.

VERSION 9.0

Added the ability to display macros in either a *List Mode* or a *Button Mode*. The *List Mode* is convenient for anyone wishing to see a list of all the macros quickly, while the *Button Mode* is more visually appealing with the icons. You can easily toggle between either mode by pressing Ctrl+M. Macro Mania will remember which mode you were in before exiting the program and will position and size the window according to how you left it before exiting.

Added a feature where you can add space between each of the buttons so that the background color will show more and so you can achieve some visual separation between each of the macro buttons. (See *Button Settings* under the File pull-down menu to set up your button sizes and spacing options.)

Added some more examples to the pre-installed macros.

Added and replaced some icons that come pre-installed with Macro Mania.

Added the file being edited to display in the Macro Editor.

Rearranged the Help System a little to make the main screen more concise.

Fixed a minor bug where sometimes the cut, copy, and paste functions would not work right within the Macro Editor.

Fixed a minor bug where sometimes the macro window would not remember its last size and position if the program when the program was exited.

VERSION 8.2

Added the ability (option) to position the message box somewhere other than the default position (the middle of the screen) when using any of the Msg commands.

Added more sophisticated error messages when a macro tries to do something it can not do (e.g. if you point to a drive that does not exist when executing a file command, the error message shows the syntax of the macro command to help you diagnose errors with the macro).

Extended the quantity discount schedule to provide additional discounts for organizations that wish to purchase more than 500 licenses at a time.

VERSION 8.1.2

The "goto macro" command within the editor now remembers the last macro you went to. Added the ability to quickly jump to the "Find What" field within the Find/Search feature of the Macro Editor.

Fixed a bug where using Ctrl-V would duplicate the pasted information within the macro editor (oddly Shift-Insert or manually selecting "Edit", then "Paste" worked fine).

VERSION 8.1

Fixed some compatibility issues when Macro Mania was running under Windows 2000 -- the macro editor was not showing the macros after they were saved (thanks to Microsoft changing the Rich Text Format standards from the Windows 9x standards).

VERSION 8.0.1

Enhanced the Help System so that the macro commands are shown in color to make it much easier to read the macros and to make them more visually attractive (like when viewed in the Macro Editor of v8.x). The tool bar has also been revamped significantly to make it much smaller so that more of your actual macro can be viewed on the screen. Some new features have also been added -- such as the ability to quickly jump to the first or last macro, manipulate text (cut, copy, and paste) via the toolbar, and more.

Added an Overwrite indicator so one can easily determine if they are in insert or overwrite mode when editing macros.

Reworked the tab order in the Macro Editor to be a little more intuitive.

VERSION 8.0

The Macro Editor has been significantly enhanced with a more modern look, and new features have been added. Macro commands are now in color to help visually set them apart from other text. Remarks are in dark green, labels in purple, variables in red, and the rest of the commands are in blue. The tool bar has been revised with a newer look and feel and new features, such as the ability to quickly jump to the first or last macro, have been added.

VERSION 7.01

Fixed a small bug with the Command Wizard not highlighting the parts that need to be dynamically entered when it was being used.

Fixed a bug where the Ctrl-v command to paste text in the Macro Editor was not working correctly.

VERSION 7.0

Changed the way the Copy and Delete Commands work so that they now are more closely tied to the way the operating system behaves. For example, the prompts and status windows are like those seen when using the Windows Explorer. Also, if the file being copied is particularly large, a status bar (with a percentage complete indicator) is displayed. The Delete command now acts like the DOS DelTree command in that if you specify just a folder, all files and folders in and below it will be deleted and removed. There are also several other options available for these commands, such as toggling whether or not files will be put into the recycle bin, whether or not you should be prompted to overwrite or delete existing files, and much more.

Added several other Msg commands (now you can ask Yes/No questions, Display an Abort, Retry, Ignore prompt, store what option was selected in a variable to conditionally perform certain actions in your macros, and more).

Added the ability to search (and/or replace) text within one or more macros.

Fixed a minor bug where the icon display did not correctly scroll all the way to the last few icons (you had to use the file listbox rather than the displayed list of icons to view the final 3-7 icons).

Added a feature so that if you have made the main window of Macro Mania float on top, any branches will retain that property and float on top of other windows too.

Added the ability to change the background color of the main window by selecting Change Background Color from the File pull-down menu.

Fixed a bug so that if you are in insert mode in the Macro Editor and press enter, the editor behaves more as expected (a new line is created rather than it pulling up data from the following line).

Added the ability to manipulate the [Mouse](#)

Fixed a bug where sometimes a pause command would stop the macro if the pause was running and midnight occurred.

VERSION 6.2

Added the cross-referenced menu item [If ExistFile](#) to the *File* submenu on the Command Wizard.

Added a discussion about how one can check for the existence of a subdirectory using the [If ExistFile](#) command.

Wrapped the installation routine in a new and improved install engine.

VERSION 6.17

Added a more 3-D *look and feel* to some of the Windows -- just very minor cosmetic changes.

VERSION 6.16

Fixed a bug that, under certain conditions, turning off the invisible attribute of a macro button was difficult.

Fixed a bug where the REPLACE command was not recognizing variables with extensions that were 2 characters long.

Added a new look-and-feel feature in the Macro Editor where focus is automatically placed on the text of the macro right after it is saved.

VERSION 6.15

Fixed a bug for when certain days of the week were scheduled: the days were all being shifted to always start with Sunday.

Added a new intuitive feature so that when a time is scheduled, the hour or minute will automatically get set to a starting value.

VERSION 6.1

Added a {DUN} parameter to the [Run](#) Command so that one can easily launch a [Dial-Up Networking Session](#)

Added a {CP} parameter to the [Run](#) Command so that one can easily access the Windows [Control Panel](#) items.

Modified the interface to give it more of a 32-bit *look and feel* (e.g. turned off bold attributes of some of the fonts, etc.)

Changed it so that choosing cancel when a [GetInput](#) prompt is shown just makes the macro continue executing. (You can still have the macro abort if no input is given -- see the new detailed example at the

bottom of the screen where the [GetInput](#) Command syntax is shown in this help system.)

Added a more detailed [Year 2000 Compliant Statement](#) to the Help System.

Added a Yes/No prompt if you launch another instance of Macro Mania, giving you the chance to abort accidentally running another copy, yet also providing you the ability to intentionally run another instance. The prompt is intentionally avoided if you launch Macro Mania with any parameter (e.g. launch it with the EDIT parameter to have it go directly to the Macro Editor or launch it with a number because you want to launch it with the number corresponding to a macro number you wish to run as soon as Macro Mania loads, or you launch it with a file name to indicate you want to load another instance of Macro Mania with a certain group of macros.) This enables you to easily use Macro Mania itself to launch another Macro Mania instance in cases where doing so is implicitly intentional.

Added an example to the Help System demonstrating how one can easily toggle the scheduled state of a macro with another macro. See [Scheduling Macros With Other Macros](#).

Added the ability to either close just the Macro Editor or to exit Macro Mania entirely when in the Macro Editor.

Fixed a bug when the password options were set for opening or editing macro files with v6.x.

Removed some of the icons being distributed to eliminate any potential problems with certain ones being protected trademarks (e.g snoopy, aol...)

[VERSION 6.0](#)

Macro Mania was migrated from a 16-bit program to a 32-bit program. All macros created with previous versions of Macro Mania work fine, but a few important changes of interest to anyone used to any previous version should be noted. [Click here if you upgrading from a version of Macro Mania prior to v6.0...](#)

The global hotkeys have been extended so that more than just an Alt-[key] combination can be assigned to a hotkey. You can now set any combination of Alt, Ctrl, Shift, and WinKey, with function keys 1-12, numbers 0-9, and letters a-Z.

Global hotkeys may now be made invisible.

The ability to print out a list of all global hotkeys now exists.

Support for Long File Names now exists for all the [File](#) Commands as well as for Macro Mania data files (*.mm).

The Macro Editor now supports an *Insert Mode* so that you can easily type over existing text.

[VERSION 5.0](#)

A new *Command Wizard* has been added to make it much easier for you to insert macro commands. Just use the *Insert Command* pull-down menu from the Macro Editor to easily insert any macro command.

The Macro Editor interface has been enhanced to permit you to shrink and stretch it to better fit your screen size and likings (the size and position are remembered from session to session).

[VERSION 4.3](#)

Added the [Let\(Format\)](#) command so you can easily manipulate variables into different standard formats (e.g. currency, percentages, dates, etc.)

Added the [Let\(now\)](#) command so you can easily set a variable to the value of the current day and time (assumedly to extract whatever you need from that with the Let (Format) command).

Added the [Let\(Math\)](#) command so you can easily add, subtract, multiply, and divide the values of variables.

Added the ability to use variables with virtually any macro command that accepts parameters.

Added the ability to print a macro easily from the Add/Edit Window.

Added the ability to copy a macro to the clipboard from the Add/Edit Window with a menu option (no need to highlight and then select Edit, Copy).

VERSION 4.2

Added the ability to use variables with all the [File...](#) commands.

Added a discussion/example in this Help System about [adding a counter to a repeating macro](#).

Added the [Let\(Replace\)](#) command which enables you to replace text with other text in a variable.

VERSION 4.1

Added the ability to set [global hotkeys](#). You can now launch a macro while you are in any Windows program without having to first switch over to Macro Mania.

VERSION 4.0

Introduced several [Let Commands](#) that enable you to put text, the contents of files, and the contents of the clipboard into variables. You can then parse, evaluate, and change the contents of those variables with variations of the [Let](#) command.

Other commands such as [Send](#), [Msg-OK](#), etc will work with variables too!

Added some [If-Then](#) commands which enable you to compare variables, check for the existence of a file, and/or check for the existence of a Window and then execute a macro accordingly.

A new [GoTo](#) command has also been added so that you can jump to a particular part of a macro (assumedly to use with the [If-Then](#) Command).

Added several [File Commands](#) to enable you to copy, delete, move, rename, and even create/write to files as well as make, remove, and rename subdirectories.

Added the optional {[WAIT](#)} parameter to the [Run](#) command. If you include this parameter, Macro Mania will run the specified program, then halt execution of the rest of the macro until the program has terminated (either automatically or manually). This option is *especially* useful when shelling to programs that run and terminate automatically, such as DOS batch files, etc.

Added the ability for you to Order ([resequence](#)) macros. This is useful if, for example, you want to group

certain macros together, move hidden macros to the end, etc. This feature will *automatically* detect and adjust any macros that have a [macro](#) Command so that they point to the correct macro(s) after their sequence has changed.

The macro number is now displayed in the left corner of the status bar -- useful to get a visual overview of the macros and their positions when you wish to edit or reposition macros.

Fixed a minor bug with drive box not appearing on high resolution monitors when selecting icons.

Fixed a minor bug where the edit box did not detect a change if you pasted text using Shift-Insert.

Removed the "feature" of the Macro Editor Screen resizing itself according to the screen resolution -- this permits the same amount of information to be displayed in the editor, but it does not take up as much screen space.

If you pass in a parameter to tell Macro Mania what macro file to use, you do not have to include the full path. If a path is missing it looks in its current subdirectory, if a path is present it will use the environment path, and if the file is not found at all, then it just uses the DEFAULT.MM file. (See [Creating and Opening New Macro Files](#).)

Added a *Copy Macro* feature to the Macro Editor so you can easily copy an existing macro to a new macro (assumedly to allow you to copy then edit a macro that you may want to be similar, but not exactly the same as the one you are copying from).

Added a [MinimizeAll](#) sub-command to the [Minimize](#) command, which enables you to quickly and easily minimize all Windows currently open.

Added a Save As option in the Macro Editor so you can quickly save a macro file to a new file.

Fixed a bug with the passwords for opening or editing macros getting truncated to 5 characters.

[VERSION 3.53](#)

Added the [OnTop](#) command which enables you to toggle the *Always On Top* property (if the main window of Macro Mania floats on top of the other windows) in a macro.

[VERSION 3.52](#)

Enhanced the way the windows are displayed on high-resolution monitors.

Added better compatibility with International Date Settings that are not like (U.S.).

[VERSION 3.51](#)

Fixed a bug with scheduling a macro between midnight and 1:00 am -- it used to not save a macro scheduled in that time range correctly.

Made Macro Mania smarter about saving the size and position of a branch Window when it exits. If it is minimized during the exit it will not save the minimized size and therefore cause the Windows to be virtually unusable. This only happened under unusual exit circumstance with branch Windows, but that has been fixed.

Added some discussion about [DOS batch files](#) to the help system.

Added the feature/option to have the first button beep when it gets the focus. This is especially useful if you like to tab through your macros from the keyboard and would like an audio reference of when you have just put focus on (or passed) the first button.

VERSION 3.5

Added several commands that enable you to manipulate the contents of the Windows clipboard. Namely the [Clipboard-Restore](#), [Clipboard-Save](#), [Clipboard-Set](#), and [Clipboard-SetFile](#) commands.

Added the ability to [create](#) different macro files and then open them as needed. Useful if you want to categorize your macros (e.g. by user, functionality, what application they will be used with, etc.).

Added the [Open](#) command to open a different macro file from within a macro itself. Also added the ability to pass a macro file at startup and the ability to manually open another macro file via a selection on the pull-down menu.

Added the [Branch](#) command. This command allows you to specify another macro file which will pop up and allow you to interactively select an option. Once an option is selected, a macro will run and return to the original macro to execute further macro commands if needed.

Added the [Repeat-Prompt](#) command to give more control and easier access regarding setting repeat values in a macro.

Added the ability to make a macro button invisible from the Main Screen in Macro Mania. This is useful when you do not normally select a macro directly (e.g. it is used via a [Macro](#) command in another macro, it is only executed at [startup](#), etc.).

Enhanced Macro Mania so that the [macro](#) command no longer has to be the last command of a macro. Execution will now return to a macro that launches another macro. You can now easily create [subroutines and loops](#) within macros.

To [quickly edit](#) a macro, you can now click once on the *right* mouse button to automatically go to the edit screen to edit the macro your mouse pointer is on. Especially helpful if you have several macros and forget their sequence number.

Added a feature that *automatically* searches and adjusts any macros that have a [macro](#) command which points to macros that get resequenced when you delete macros.

Added a few more bells and whistles on the Macro Editor. For example, F3 will take you to the box so you can jump to a macro quickly.

Added a [password feature](#) you can set at two levels. You can now password protect your macro file entirely, password protect the macros from being view or edited via Macro Mania, or password protect both.

Added to the discussion of the [Run](#) command, which mentions some not so obvious features that will be sure to interest most of you. (See the NOTES section of the [Run](#) command for details.)

VERSION 3.1

Removed the opening screen if Macro Mania is a registered copy and it has been run minimized or with a parameter to have it run a specific macro at startup (an unregistered copy requires you to select OK at the opening screen). If Macro Mania is just being run normally (registered or not registered) a Splash Screen is displayed for just a couple seconds when Macro Mania first initializes.

Fixed a minor bug with the scheduler: if a date had been scheduled, one had to select *Not Scheduled*, save it, and then move to another macro or exit the macro setup for it to clear. (Now you only have to select *Not Scheduled* -- the need to move to another macro or exit the macro setup is no longer necessary to unschedule a macro.)

Added the ability to schedule macros to run only on certain days of the week (e.g. Monday, Tuesday, etc.).

Added the [PlayWav](#) command so that you can play a wav file from a macro for added pizzazz or to otherwise bring attention to a macro. (Kind of a fun to play with too!)

Added the [WinExit](#) command which can either exit Windows, restart Windows, or restart the computer -- fast and easy.

VERSION 3.0

Extended the ability of the [Activate](#) command to enable it to activate a program with only a partial match of its Title Bar.

Added the [GetInput](#) and [SendInput](#) commands. The [GetInput](#) Command can be used to request small amounts of dynamic information (file names, search strings, etc.). The [SendInput](#) Command will then send the information obtained from the [GetInput](#) command.

Reworked the examples that are installed. Based on the operating system being installed to (Windows 3.x or Windows 95), appropriate examples are installed to best demonstrate the usefulness of Macro Mania.

Enhanced the discussion of Windows 95 considerations in the Help System. (See the [Pause](#) or [Minimize](#) command for more details.)

VERSION 2.2

Added an optional date setting that can be used with the scheduler.

Added 2 more options to select from when toggling button sizes: extra small with icons and extra small without icons. (Now the footprint can be set to an even smaller size if needed.)

Added a new Installation routine.

VERSION 2.1

Due to popular demand, added the option to set/toggle the size of the buttons. The buttons can now be smaller (making the program's "footprint" much smaller).

Added the [Exit](#) command, which allows you to exit the macro anywhere within the macro.

Added the [End](#) command, which allows you to have "Macro Mania" easily end itself entirely. (For those of you who like to just run one macro, perhaps from another program, and then want "Macro Mania" to end.)

VERSION 2.0

Added a schedule feature enabling you to schedule macros for specific times or during any 15-minute, half-hourly, or hourly interval.

Removed the prompt "Exit Macro Mania - Yes/No" when the program is exited.

Added an option to expand the area where you edit the actual macro.

Bug fix: when the window was resized sometimes a few icons would get totally "lost" and the window had to be resized again to "find" them.

Bug fix: when Macro Mania was started minimized the initial screen minimized, but then the main icon window screen would return to its normal state.

Bug fix: when the program was closed from an icon the Window's environment still thought it was running if the task manager or Alt-Tab sequence was used.

Made a few underlying enhancements to fine tune performance.

VERSION 1.3

Added the ability to launch a particular macro when "Macro Mania" is first loaded.

Optimized the speed at which macro buttons are adjusted when you resize the "Macro Mania" screen.

VERSION 1.2

Fixed a problem that happens on some PC's when the [Pause](#) Command is used.

Removed some duplicate or infrequently used icons that are installed. This keeps the distribution file small. (Search for "Icons" in the Help system to find out how you can easily add your own icons.) Still, there are quite a few icons that come with Macro Mania!

Made sure the installation of new versions does not overwrite any macros from older versions unless the users "Okay's it". Too many happy users for us to risk messing that up. <smile>

VERSION 1.1

Fixed a small bug that caused the "Always On Top" characteristic to turn off when the program was minimized.

Added the [SendNow](#) Command, which allows you to send just about any conceivable combination/format of the current date and/or time to a program.

Added the [Beep](#) Command, which allows you to cause the PC to beep via its speaker. Useful when you need to bring attention to the PC, particularly after a long macro and/or when its time for you to type.

Activate

DESCRIPTION:

Brings an already running copy of a program to the front of the screen. (See the notes section below for further discussion.)

SYNTAX:

[Activate](#) [Title Bar]

Title Bar is the title bar name (caption) for the window you wish to activate (the *Macro Mania Help* title bar shown for this Help System is the Title Bar for this window). You may also use a [variable](#) for *Title Bar* (e.g. Activate {*var-p*}).

EXAMPLES:

[Activate](#) NOTEPAD - myfile.txt

[Activate](#) myfile.txt

NOTES:

In determining which program to activate, *Title Bar* is compared to the title string of each window that may be running. If there is no exact match, any program whose title string may contain *Title Bar* is activated. If there is more than one instance of the program named with *Title Bar*, one instance is arbitrarily activated. If no windows containing *Title Bar* are found, a warning message is displayed and you are given the opportunity to manually activate the window to allow the macro to continue or to cancel the macro entirely. Be sure to read the [Starting Programs](#) section in the Help System for other, actually better, ways to make sure a program is activated.

Title Bar does not have to be case sensitive to get a match.

SEE ALSO::

[Manipulating the State of Windows](#)

Beep

DESCRIPTION:

Sends a beep to any normal PC Speaker.

SYNTAX:

[Beep](#)

NOTES:

Useful for bringing attention to the computer. Particularly useful to put at the end of long macros or immediately before a pause to indicate you may begin typing.

SEE ALSO:

[PlayWav](#)

ChDir

DESCRIPTION:

Like the DOS command, this changes the working directory to the directory you specify. Issue this command if the program you wish to run using the [Run](#) command needs to find support files. (Issuing this command before the Run command is like specifying the *Working Directory* when you create a program item in the Windows Program Manager.)

SYNTAX:

[ChDir](#) [path]

path is a string expression that identifies which directory becomes the new default directory. You may also use a [variable](#) for *path* (e.g. [ChDir](#) {var-p}). *Path* must contain fewer than 128 characters and has the following full syntax:

[ChDir](#) [drive:] [\]directory[\directory] . . .

The *drive:* parameter is an optional drive specification; the *directory* parameter is a directory name. If you omit *drive:*, [ChDir](#) changes the default directory on the current drive.

NOTES:

As with DOS, [ChDir](#) may be abbreviated with [CD](#).

End

DESCRIPTION:

Causes the Macro Mania program to completely end.

SYNTAX:

End

NOTES:

This command is especially useful if you wish to run a macro, perhaps from another program, then want Macro Mania to stop running entirely. To simply exit an individual macro, see the [Exit](#) command.

Exit

DESCRIPTION:

Causes the macro to exit and quit running.

SYNTAX:

Exit

NOTES:

This command is especially useful for debugging your macros or dynamically changing where the macro should end without having to comment a lot of lines or delete part of a macro script you may want to use later. For debugging, if you are not sure where your macro is not working correctly, place this command a few lines above where you think the error is happening. Then gradually move it down one line at a time until the line with the error is identified.

To end the Macro Mania program entirely, see the End command.

GetInput

DESCRIPTION:

Gives a prompt within a macro, allowing for input of small amounts of dynamic data that can later be sent with the [SendInput](#) command (or [Send](#) command if a *variable* is used).

SYNTAX:

[GetInput](#) <{var-?}> [prompt]

{var-?} is optional -- however, when present, the contents of whatever is received is put into the specified [variable](#) where you can later use any of the Macro Mania commands that support variables with that variable.

Prompt is any text (approximately 255 characters maximum) that you want displayed when the Input Box is shown. (Usually *prompt* is a short description of what should be typed in the box.)

EXAMPLE:

[GetInput](#) {var-b} Please enter your first and last name.

NOTES:

This command is useful for getting file names, search strings, and other data that may change each time a macro is run.

Once you use the [GetInput](#) command, it is necessary to use either the [Run](#) or [Activate](#) command and later a [SendInput](#) command (or [Send](#) command if the contents was put in a variable) to send the information that was put into the Input Box.

By putting the input into several different variables, you can issue several [GetInput](#) statements in a row without the need to issue a [SendInput](#) command to prevent overwriting the previous contents. Thus, using variables with the [GetInput](#) command and then using the [Send](#) command to send the contents of the variable is much more powerful than using the [GetInput](#) and later the simple [SendInput](#) command. (For backward compatibility, the [SendInput](#) command is still available, but it is really not needed except for very simple cases.)

If no input is put into the box or the cancel button is selected, the macro will continue. (You should put the contents of what the user typed into a variable and then evaluate the response if you want to have the macro either continue or stop with either an [Exit](#), [End](#), or some other dialogue box such as the [Msg Command](#).

The following example shows how you can create a menu-like prompt to more easily facilitate the user entering predefined values and to execute a macro according to what is typed in the prompt. It also demonstrates how you can evaluate if the user just pressed Enter without entering anything or chose cancel (which then causes the reply put into the variable to be blank) so that you can abort the macro if you want to do so when that happens.

DETAILED EXAMPLE:

```
GetInput {var-a} 1 - c:\autoexec.bat | 2 - config.sys | 3 - c:\new.txt
Let {var-0} = {}
Let {var-1} = 1
Let {var-2} = 2
Let {var-3} = 3
IF {var-a} = {var-0} THEN Exit
IF {var-a} = {var-1} THEN Let {var-a} = {c:\autoexec.bat}
IF {var-a} = {var-2} THEN Let {var-a} = {c:\config.sys}
IF {var-a} = {var-3} THEN Let {var-a} = {c:\new.txt}
Run notepad.exe, {var-a}
```

Macro

DESCRIPTION:

Runs another macro you have programmed, allowing you to chain macros together.

SYNTAX:

Macro [n]

n is a number indicating the number of the macro you wish to run. Refer to the number displayed in the Macro Editor or on the status bar on the Main Screen. You may also use a variable for *n* (see example 2 below).

EXAMPLES:

Macro 3

Macro {var-p}

NOTES:

Macro Mania prevents you from using the Macro command to call itself, otherwise it would get into a continuous loop. It is possible to call a macro that, in turn, calls the macro that called it, so be sure you refer to the correct macro number and avoid getting yourself into such a continuous loop.

The macro command essentially allows you to create subroutines within your macros. By putting repeat commands in certain macros, you can also create nested loops, etc. too. If you only plan to use a macro as a subroutine from another macro, you can turn its visible property to off from where you select the icons for buttons, preventing a macro from showing up as a button on the main Macro Mania screen.

If your macro does not require a repeating subroutine, you may also use the Goto command to branch within a single macro and perhaps better keep your macro intact and easier to manage.

Minimize

DESCRIPTION:

Causes the Macro Mania window to minimize itself (become an icon). Use [MinimizeALL](#) to conveniently minimize all Windows programs that are currently running.

SYNTAX:

[Minimize](#) or [MinimizeALL](#)

NOTES:

This command is more helpful if used at the beginning of a macro, before a [Run](#) or [Activate](#) command has been issued. Use the [Restore](#) command to return the Macro Mania window to its *Normal* size (use the [Run](#) or [Activate](#) to restore other Windows).

It is not recommended you use the [MinimizeALL](#) command with a macro that repeats, as the macro will minimize all the windows every time it repeats. You could, however, use the command in a macro that then called a repeating macro with the [macro](#) command to prevent that effect.

It is recommended, however, you use the [Minimize](#) command (but not the [Restore](#) command) with a macro that repeats because Macro Mania will only minimize itself once and the macro will appear to run smoother.

SEE ALSO:

[Manipulating the State of Windows](#)

Pause

DESCRIPTION:

Use this command to pause a macros execution. Useful if you wish to give yourself time to manually type in some keystrokes or otherwise wish to prevent the macro script from immediately continuing.

SYNTAX:

Pause [n] <show>

n is a number indicating the number of seconds you wish the macro to pause for. You may also use a [variable](#) for n (see example 2 below).. The optional word [show](#) may follow the command to indicate that a small box counting down the number of seconds remaining should be displayed. The display box may be moved, and an option to immediately resume or cancel the macro is also available when the [show](#) parameter is used.

EXAMPLES:

Pause 3 Show

Pause {var-p} Show

RELATED TOPICS:

[Waiting for another program to finish](#)

Rem

DESCRIPTION:

Use before a command or text so that line is ignored when the macro is executed.

SYNTAX:

REM <command/text>

command/text is either a command you do not want to be used in your macro and/or some text, such as a comment within the macro to help you better identify what the macro is intended to do.

EXAMPLES:

REM Go to the bottom of the notepad.
=== Go to the bottom of the notepad ===

NOTES:

You may also use a single quote in place of the Rem statement.

Repeat

DESCRIPTION:

Causes a macro to repeat.

SYNTAX:

`Repeat [n]`

n is a number indicating the number of times the macro should repeat. You may also use a [variable](#) for n (see example 2 below).

EXAMPLES:

`Repeat 3`

`Repeat {var-r}`

NOTES:

This command should be put on the first line of your macros to help you remember its value. (You may also wish to include its value as part of your macro description.)

The repeat option can also be set dynamically by using the [Repeat-Prompt](#) command, or by using a variable that has been dynamically set elsewhere in the macro.

It is not recommended you use the [MinimizeAll](#) command with a macro that repeats, as the macro will minimize all the windows every time it repeats, causing the the programs you are using to be minimized and then activated again when you use the [Activate](#) or [Run](#) commands. In other words, although the macro will run fine, it will cause the macro to appear somewhat turbulent if it minimizes and then activates Windows repeatedly. (You could, however, use the command in a macro that then called a repeating macro with the [macro](#) command to prevent that effect.)

It is recommended you use the [Minimize](#) command (but not the [Restore](#) command) with a macro that repeats because Macro Mania will minimize itself only once and the macro will appear to run smoother.

See [Stopping a Macro](#) for strategies to help you stop macros if needed.

Restore

DESCRIPTION:

Makes the Macro Mania window normal, giving it the focus. Use [RestoreAll](#) to conveniently restore all Windows programs that are currently running.

SYNTAX:

[Restore](#) or [RestoreALL](#)

NOTES:

Using the restore command also puts the focus back to Macro Mania. Any keystrokes that are sent with the [Send](#) command will be sent to Macro Mania unless a [Run](#) or [Activate](#) command is used to put focus back to another program.

SEE ALSO:

[Minimize](#)

Run

DESCRIPTION:

Starts a program and/or brings an existing program to the front of the screen.

SYNTAX:

Run [program] <_program parameters> <{wait}> <{**Activate** [Title Bar]}>

program, the only required parameter, is the path and name of the executable file, or is the name of a file whose extension has been associated with a program in the Windows registry -- see *File Associating* in the Microsoft Windows Help for more information about file associations). If the path is not included, the current directory (which may be set with the [ChDir Command](#)), the Windows subdirectories, and the environment path are searched for a copy of the program. You may also use a [variable](#) for *program*. (See examples 4 and 6 below.)

program parameters are any valid parameters the program can accept when started (be sure to include the comma before any parameters are given). You may also use a [variable](#) for *parameters*. (See examples 5 and 6 below.)

{wait} is an optional parameter that, when included, will notify Macro Mania to halt execution of the rest of the macro until the program that is run is terminated. As also shown in example 3 below, include the special braces ({}) around the {wait} parameter when it is used. This option is very useful when shelling to programs that run and terminate automatically, such as DOS batch files, etc. See [Waiting for Another Program to Finish](#))

{Activate [Title Bar]} is an optional (though highly recommended) parameter that will first try to activate a Window based on its Title Bar and then run *program* if it is not found. It will also tell Macro Mania to wait until the Title Bar of *program* is ready before continuing the macro. Include the special braces { and } around the {Activate [Title Bar]} parameter when used. The brackets ([]) around *Title Bar*; however, are not used and are only shown to indicate that part of the syntax will change based on whatever window you want to activate. (See example 8 below.)

EXAMPLES:

1. **Run** c:\windows\notepad.exe
2. **Run** c:\windows\notepad.exe, mynotes.txt
3. **Run** c:\sample.bat {WAIT}
4. **Run** {var-2}
5. **Run** notepad.exe, {var-2}
6. **Run** {var-a}, {var-2}
7. **Run** c:\wp\myfile.txt
8. **Run** c:\windows\notepad.exe, myfile.txt {Activate myfile.txt}
9. **Run** c:\web\index.html
10. **Run** c:\My Documents\contract.doc

NOTES:

Be sure to read the [Starting Programs](#) section in the Help System for more important details.

Some programs know where to find their support files, and some do not and need a working directory specified to them. It is recommended that you use the [ChDir](#) immediately before the **Run** command so that the executable and/or all its support files are sure to be found.

SEE ALSO:

[Activating a Dial-up Networking Session](#)
[Accessing Control Panel Items](#)

Send

DESCRIPTION:

Sends keystrokes to a program as if you had typed them manually. You will be using the [Send](#) command a lot. Once you master how to use the [Send](#) command to its full potential, you can do virtually anything with your macros! You may wish to print this section for easy reference.

SYNTAX:

[Send](#) [keystrokes]

keystrokes are the keys you would press if you were sitting in a program and typing them manually. You may also use a [variable](#) for *keystrokes* (e.g. [Send {var-k}](#))

SYNTAX DETAILS:

The plus sign (+), caret (^), percent sign (%), tilde (~), and parentheses () have special meanings to the [Send](#) command (as noted further below). To specify one of these characters, enclose it inside braces. For example, to specify the plus sign, use {+}. Brackets ([]) have no special meaning to the Send command, but you must enclose them in braces as well, because in other programs brackets do have special meaning. To send brace characters, use {} and {}.

To specify characters that are not displayed when you press a key (such as Enter or Tab) and keys that represent actions rather than characters, use the codes shown below:

<u>Key</u>	<u>Code</u>
Backspace	{BACKSPACE} or {BS} or {BKSP}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Clear	{CLEAR}
Del	{DELETE} or {DEL}
Down Arrow	{DOWN}
End	{END}
Enter	{ENTER} or ~
Esc	{ESCAPE} or {ESC}
Help	{HELP}
Home	{HOME}
Ins	{INSERT}
Left Arrow	{LEFT}
Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Right Arrow	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
Up Arrow	{UP}
F1 - F16	{F1} - {F16}
SpaceBar	{ } << 1 space between the brackets
Variable	{var-?} , where ? is a letter between a-z or number from 0-99. When using the Send command with variables, unlike with other text and the special commands listed above, you can only issue one variable at a time. However, by either using multiple Send commands or merging the variables, this should impose no problem at all. (See the Let command for more information about using and manipulating variables.)

To specify keys combined with any combination of Shift, Ctrl, and Alt keys, precede the regular key code

with one or more of the following codes:

<u>Key</u>	<u>Code</u>
Shift	+
Ctrl	^
Alt	%

To specify that Shift, Ctrl, and/or Alt should be held down while several other keys are pressed, enclose the keys code in parentheses. For example, to have the Shift key held down while e and c are pressed, use `+(ec)`. To have Shift held down while an e is pressed, followed by a c being pressed without Shift, use `+ec`. (Note that some applications interpret capital letters as Shift-Letter, which may mean something different to that application. Although this is rare, you may wish to get into the habit of using lower case letters that invoke commands within an application unless you are sure it does not matter to the application whether it is lower case or not.)

To specify repeating keys, use the form {key number} (putting a space between key and number). For example, {LEFT 42} means press the Left Arrow key 42 times; {h 10} means press h 10 times.

The macro will not continue until the [Send](#) Command is complete. This feature ensures that the macro does not get ahead of itself. If you are sending a lot of keystrokes, you may wish to break them up into several [Send](#) commands so that the program that is receiving the keystrokes has time to process them before another group of keystrokes are sent.

<u>EXAMPLES:</u>	<u>DESCRIPTION:</u>
Send Wow, this is easy!	Self explanatory
Send ^{home}	Ctrl + the Home key
Send ^Fhome	Ctrl + F, then the letters/word home
Send +{end}	Shift + the End key
Send ^B	Ctrl + the letter B
Send %Fx	Alt + F, then x by itself
Send {F10}	The F10 function key
Send {tab}	The Tab key
Send 123{enter}	123 and then Enter
Send {r 22}	The letter r 22 times
Send {left}	The Left Arrow key
Send {up 7}	The Up Arrow key 7 times

LIMITATIONS: The [Send](#) command can not send keystrokes to a program that is not designed to run in Microsoft Windows (e.g DOS programs). [Send](#) also can not send the Print Screen (PRTSC) key to any program.

SEE ALSO:

[Making Sure Keystrokes Get Received.](#)

SendInput

DESCRIPTION:

Sends the most recently entered text retrieved from the [GetInput](#) command.

SYNTAX:

[SendInput](#)

EXAMPLE:

[GetInput](#) Enter the file to open.

[Run](#) notepad.exe

[Send](#) %Fo

[SendInput](#)

[Send](#) {enter}

NOTES:

If no text is in the buffer from the [GetInput](#) command, the macro asks if you wish to abort the macro.

Answering *Yes* causes the macro to stop, *No* causes the macro to skip the current [SendInput](#) command.

Like the [Send](#) command, an [Activate](#) or [Run](#) command must be issued before the [SendInput](#) command.

A [GetInput](#) command should also precede the [SendInput](#) command.

The [SendInput](#) command sends input obtained from the [GetInput](#) command if no specific [variable](#) was used to store the input. If a variable was used with the [GetInput](#) command, use the [Send](#) command to send the contents of the specified variable (e.g. [Send](#) {[var-p](#)}).

SendNow

DESCRIPTION:

Sends the current (system) date and/or time as specified with the *format* parameter. (Like the [Send](#) command, this command should only be issued after you have issued either the [activate](#) command or [run](#) command.)

SYNTAX:

[SendNow](#) [format]

The following table shows the characters you can use to create user-defined date/time formats (examples are at the bottom):

c	Send the date as dddd and send the time as t t t t t, in that order. Only date information is sent if there is no fractional part to the date serial number; only time information is sent if there is no integer portion.
d	Send the day as a number without a leading zero (1-31).
dd	Send the day as a number with a leading zero (01-31).
ddd	Send the day as an abbreviation (Sun-Sat).
dddd	Send the day as a full name (Sunday-Saturday).
dddddd	Send a date serial number as a complete date (including day, month, and year) formatted according to the Short Date setting in the International section of the Windows Control Panel. The default Short Date format is m/d/yy.
dddddd	Send a date serial number as a complete date (including day, month, and year) formatted according to the Long Date setting in the International section of the Control Panel. The default Long Date format is mmmm dd, yyyy.
w	Send the day of the week as a number (1 for Sunday through 7 for Saturday.)
ww	Send the week of the year as a number (1-53).
m	Send the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is sent.
mm	Send the month as a number with a leading zero (01-12). If m immediately follows h or hh, the minute rather than the month is sent.
mmm	Send the month as an abbreviation (Jan-Dec).
mmmm	Send the month as a full month name (January-December).
q	Send the quarter of the year as a number (1-4).
y	Send the day of the year as a number (1-366).
yy	Send the year as a two-digit number (00-99).
yyyy	Send the year as a four-digit number (100-9999).
h	Send the hour as a number without leading zeros (0-23).
hh	Send the hour as a number with leading zeros (00-23).
n	Send the minute as a number without leading zeros (0-59).
nn	Send the minute as a number with leading zeros (00-59).
s	Send the second as a number without leading zeros (0-59).
ss	Send the second as a number with leading zeros (00-59).

- ttttt** Send a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is sent if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.
- AM/PM** Use the 12-hour clock and send an uppercase AM with any hour before noon; send an uppercase PM with any hour between noon and 11:59 PM.
- am/pm** Use the 12-hour clock and send a lowercase AM with any hour before noon; send a lowercase PM with any hour between noon and 11:59 PM.
- A/P** Use the 12-hour clock and send an uppercase A with any hour before noon; send an uppercase P with any hour between noon and 11:59 PM.
- a/p** Use the 12-hour clock and send a lowercase A with any hour before noon; send a lowercase P with any hour between noon and 11:59 PM.
- AMPM** Use the 12-hour clock and send the contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; send the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string sent matches the string as it exists in the WIN.INI file. The default format is AM/PM.

EXAMPLES:

Format	What is sent if the current day is December 9, 2015 and the time is 8:50 pm.
m/d/yy	12/9/15
m/d/yyyy	12/9/2015
d-mmmm-yy	9-December-15
d-mmmm	9 December
mmm-yy	December 15
hh:mm AM/PM	08:50 PM
h:mm:ss a/p	8:50:35 p
h:mm	20:50
h:mm:ss	20:50:35
m/d/yy h:mm	12/9/15 20:50

A Simple Example

Maybe the best way to explain the easy syntax that Macro Mania uses is with an example...

Notice how you can add remarks to your macros so they are easier to read by just adding a single quote or the rem command before the text you want to use as a remark.

```
Run Notepad.exe {Activate Untitled - Notepad}
== The next line maximizes the notepad window ==
Send %{ }x
Send Macro Mania is powerful, yet very easy to use!
== The next line sends the Enter key twice ==
Send {ENTER 2}
Send You can send virtually any keystroke to a program.
Send {ENTER 2}
' == The next command makes the macro pause for 5 seconds (the show parameter is optional) ==
Pause 5 Show
The next send command sends Alt-H (which gets the Help pull-down menu, then an A so the About window will be activated.
Send %Ha
```

To experiment with this macro, feel free to Copy and Paste it into a new macro of your own. See [Quick Start](#) to learn how to create a new macro. For more detailed examples, be sure to review the sample macros that come pre-installed with Macro Mania. Notice how potentially powerful this macro can be, and yet this example uses only 3 commands ([Run](#), [Send](#), and [Pause](#)).

Starting Programs

The first part of a macro needs to start the program that will be receiving the keystrokes sent from the macro. There are two commands you can use to start a program, either the [Run](#) Command, or the [Activate](#) Command.

The [Run](#) Command is the most powerful and effective way to start programs, especially when the {[Activate...](#)} parameter is used. That parameter makes the [Run](#) Command a combination of the [Run](#) Command, [Activate](#) Command, [If ExistWindow](#) Command and in many cases eliminates the need for a [Pause](#) Command; thus, it gives your macro maximum speed and versatility. When that parameter is used, Macro Mania will first check all the main windows of the programs already running to see if their title bar matches the Title Bar in the parameter you provided and, if found, it will activate that window. If no window with Title Bar is found, Macro Mania will launch the program file you provided. Additionally, because some programs take a few seconds to load, Macro Mania will wait until the program has loaded and Title Bar is found before continuing. Thus, you probably will not even have to issue a [Pause](#) Command (whose length may have to vary from system to system anyway) after the [Run](#) Command to make sure the program has had a chance to launch because this is done for you. The whole process is much faster and smoother than if you were to manually try to do all these commands separately and helps you ensure you are activating the correct window as fast as possible.

EXAMPLE:

[Run](#) c:\windows\notepad.exe, myfile.txt {[Activate](#) myfile.txt}

The above example will search to see if a program with myfile.txt in its caption is already running (note that only a partial match has to be made) and will activate it if found. If it is not found, it will then launch notepad.exe and open the file myfile.txt since that is the parameter it was given.

HOW PROGRAMS LAUNCH:

Some programs will allow more than once instance of themselves to be running at the same time; while others will not. In fact, some programs are smart enough to detect if they are already running and will just switch to the already running instance of themselves if you try to run them again. Other programs may not be so smart (or intentionally permit you to run multiple instances of them -- Windows notepad is a good example of that type of program since it is not inconceivable that you may want to run multiple instances of notepad to edit/view different files at the same time). So, you may leave the {[Activate...](#)} parameter off if you explicitly wish to run another instance of a program that will allow you to do so.

Depending on what you want to accomplish, you may wish to explicitly start a new instance of the program even if another instance is already running (assuming that particular program will allow multiple instances of itself to be run at the same time). If that is the case, then that is a situation where you will not be wanting to use the {[Activate...](#)} parameter. On the other hand, if you want to use any instance of a program that may already be running, but are not sure if the program will be running and if the program is one that detects itself running, the added parameter helps you write macros that will execute smoothly and error-free.

Tricks and Techniques

- [Helpful Hints for Beginners](#)
- [A Simple Example](#)
- [Accessing the Macro Editor](#)
- [Making Sure Keystrokes Get Received](#)
- [Scheduling Macros](#)
- [Hotkeys](#)
- [Faster Keystrokes](#)
- [Accessing the Windows Control Panel](#)
- [Activating the Windows Start Button](#)
- [Activating the Windows Desktop](#)
- [Manipulating the State of Windows](#)
- [Launching Macros](#)
- [General Keyboard Navigation Tips](#)
- [Recording Keystrokes](#)
- [Waiting for another program to finish](#)
- [Launch a Particular Macro at Startup](#)
- [Adding Icons to the icon list](#)
- [Activating Buttons on Other Programs](#)
- [Combine a Search Feature with a Macro](#)
- [Saving Space in Macros](#)
- [Sending a Space \(spacebar\)](#)
- [Stopping a Macro](#)
- [Creating and Opening New Macro Files](#)
- [Password Protecting Your Macros](#)
- [DOS Batch Files](#)
- [Resequencing Macros](#)
- [Activating Child Windows](#)

- Compatibility with other Programs
- Adding a Counter to a Repeating Macro
- Subroutines within macros (looping, etc.)
- Activating a Dial-up Networking Session

How to Pay for this Program

As evident by the [revisions](#), Macro Mania gets new features as fast as time and ideas permit. Typically minor revisions are released every few months and major versions about once a year. Just point your web browser to <http://www.nstarsolutions.com/mm> to make sure you have the latest version. If you have purchased a previous version of Macro Mania and wish to upgrade, see the [Upgrade Policy](#). Also make sure you have reviewed and agree with the [License Agreement](#).

The following information is also presented at our web site, which offers the most convenient way to purchase Macro Mania. Please see <http://www.nstarsolutions.com/mm/purchase>

You must purchase a license for each PC you will use Macro Mania on (multiple licenses may be purchased for a significantly discounted price and are offered in convenient multi-user packs). There are two types of licenses.

1. **Personal Use License (Product #8183)** - may only be used by an individual *not* using the program for a business, academic, or government purpose. That is, you only intend to use it at home for private use. Your unlock code and license will be assigned to an individual and only 2 *Personal Use Licenses* may be purchased by any one person. The price is **\$45/License**.

2. **Professional Use License (Product #8184)** - the most common type of license purchased, this type of license should be purchased when the program will be used in a professional environment (such as in a business, academic, or government environment). The unlock code will be assigned to the organization name provided at the time of purchase and will reflect the total number of licenses purchased. The price schedule for this type of license is as follows:

Quantity	Price	Price per License
Single User License	\$198	\$198
2-user License	\$348	\$174 (12% discount)
5-user License	\$850	\$170 (14% discount)
10-user License	\$1,500	\$150 (24% discount)
25-user License	\$3,375	\$135 (32% discount)
50-user License	\$6,250	\$125 (37% discount)
100-user License	\$11,000	\$110 (45% discount)
250-user License	\$23,750	\$95 (52% discount)
500-user License	\$42,500	\$85 (57% discount)
1000-user License	\$70,000	\$70 (65% discount)

Quantity discounts will only be applied during a single time of purchase. For example, you may not purchase 5 licenses now and then receive a 10-user discount to purchase 5 more license several weeks from now. Prices are subject to change without notice. Our web site will always have the latest price schedule available.

You can make your purchase using one of several convenient methods:

If you have a Discover, MasterCard, Visa, or American Express card, you can contact *NorthStar Solutions* via any of the methods shown below, or you can use the *Mail* method to send a check or money order. Regardless of how you choose to order, please be sure to note the **Product #** (shown above) and how many copies you need so you can immediately be given the appropriate [registration code](#).

INTERNET -- fast, easy, and secure!

<http://www.nstarsolutions.com/mm/purchase>

PHONE -- Monday-Friday, 9am-7pm, CST

- * Operators can take *orders* only, see the [Technical Support](#) section for technical support.
- * Be sure to provide [Product #](#) (shown above with the license definitions) when placing your order.

1 800-699-6395 (From the U.S. only.)
1 785-539-3743

FAX -- 24 hours
1 785-539-3743

MAIL -- U.S. Funds only

You may register with a check or money order.

Make them payable to "**NorthStar Solutions**" and send them to:

NorthStar Solutions
1228 Westloop Pl, #204
Manhattan, KS 66502

For fastest delivery of your [registration code](#), provide either an e-mail address (preferred) or a fax number where it can be sent. Otherwise one will have to be sent via snail mail.

The following information will be needed:

- * The program you are buying (please use the [Product #](#) - shown above with the license definitions).
- * Your credit # and its expiration date (if using credit card).
- * Your mailing address.
- * Your e-mail address (so NorthStar Solutions can send you a message confirming your order and so we can contact you easily with any important follow-up information, upgrade announcements, etc.).

The online purchase form at <http://www.nstarsolutions.com/mm/purchase.htm> conveniently prompts you for all the information needed to place your purchase.

Our Reseller Program

Many consultants/resellers have asked us about a reseller program. These people realize that if people or organizations are introduced to Macro Mania, they will quickly realize the tremendous savings in labor and associated costs that the program can provide, and they will certainly want to purchase it. In other words, Macro Mania is one of those products that appeal to a very wide market and whose features can almost sell itself. With a little effort to introduce Macro Mania to other people and organizations, and the ability to take advantage of our discounts for quantity purchases, a reseller can make a healthy profit with Macro Mania!

Our reseller program permits you to purchase licenses and then resell those licenses in any quantity you wish so long as the total licenses that you resell do not exceed the total licenses that you purchase from NorthStar Solutions. You may use our suggested retail price, or you may use whatever pricing schedule you develop to help you generate a profit (see below for ways you can easily make a profit reselling Macro Mania).

There are two basic ways you can easily make a profit as a reseller of Macro Mania:

1) You purchase multiple licenses at once and get the quantity discount, but sell the licenses at their normal *per-license* price so that you make a profit on each license. The more licenses you purchase, the larger your quantity discount and, thus, the larger your profit. This strategy is best done if you first prepurchase licenses and then sell them as separate licenses to more than one person or organization. The reason is because an organization that purchases multiple licenses from you may eventually discover what the actual price discounts are -- and may not be happy to find you overcharged them (except if you are using strategy 2 below -- which can easily justify a markup by you -- and be hidden in the cost of your services).

2) If you are selling multiple licenses to the same organization, your profit may not necessarily be in selling the licenses you have purchased; but in your valuable consulting services. If you are going to introduce them to the program, provide them with help installing it, give them assistance using it (writing macros), etc., then your consulting services can be very useful to the organizations that purchase Macro Mania from you. In fact, as you likely realize, any organization that purchases Macro Mania is wanting to save time and labor for various tasks they perform and your services in addition to Macro Mania itself can help them achieve that. With your knowledge of Macro Mania which will help your customers quickly maximize the tremendous benefits of using the program, you should be able to command a very healthy fee for your valuable add-on services. In the long run, both your services and Macro Mania will save your customers a lot of time and money, so it ends up being a *win-win situation* for all.

Of course, another strategy would be to blend the above two strategies depending on who purchases the licenses from you, how many are purchased, etc.

As a reseller, you will be responsible for a certain level of technical support. This support will be for general installation and basic use of the program (such as writing and executing macros). NorthStar Solutions will provide low-level support for issues beyond just writing and executing macros, such as bug fixes and changes to Macro Mania itself.

All prices and terms subject to change without notice. Contact NorthStar Solutions for the latest information about our Reseller Program using any the methods noted in the [Technical Support](#) section.

Upgrade Policy

If you have already bought a license of Macro Mania, all *minor* updates (e.g. v8.0 to v8.1) are free and approximately 50% will be discounted for any *major* updates (e.g. v8.x to v9.x), up to the same number of licenses you have already purchased.

The following information is also presented at our web site, which offers the most convenient way to purchase a Macro Mania upgrade. Please see <http://www.nstarsolutions.com/mm/purchase/upgrade.htm>

You must purchase a license for each PC you will use Macro Mania on (multiple licenses may be purchased for a significantly discounted price and are offered in convenient multi-user packs). There are two types of licenses.

1. **Personal Use License Upgrade (Product #8185)** - may only be used by an individual *not* using the program for a business, academic, or government purpose. That is, you only intend to use it at home for private use. Your unlock code and license will be assigned to an individual and only 2 *Personal Use Licenses* may be purchased by any one person. The price is **\$45/License**.

2. **Professional Use License Upgrade (Product #8186)** - the most common type of license purchased, this type of license should be purchased when the program will be used in a professional environment (such as in a business, academic, or government environment). The unlock code will be assigned to the organization name provided at the time of purchase and will reflect the total number of licenses purchased.

The price schedule for a *Professional Use License Upgrade* (with a quantity discount and 50% upgrade discount included) is as follows:

Quantity	Price	Price per Upgrade License
Single User License Upgrade	\$99	\$99 (50% discount)
2-user License Upgrade	\$174	\$87 (56% discount)
5-user License Upgrade	\$425	\$85 (57% discount)
10-user License Upgrade	\$750	\$75 (62% discount)
25-user License Upgrade	\$1,688	\$67.50 (66% discount)
50-user License Upgrade	\$3,125	\$62.50 (68% discount)
100-user License Upgrade	\$5,500	\$55 (72% discount)
250-user License Upgrade	\$11,875	\$47.50 (76% discount)
500-user License Upgrade	\$21,250	\$42.50 (78% discount)
1000-user License Upgrade	\$35,000	\$35 (82% discount)

NOTE: You may only upgrade as many licenses as you already have purchased previously at the full price. For example, if you have purchased 5 licenses, you may not purchase 20 licenses at the upgrade price. You may, however, purchase 5 license at the upgrade price and 15 more at the regular price. Quantity discounts will only be applied during a single time of purchase. For example, you may not purchase 5 licenses now and then receive a 10-user discount to purchase 5 more license several weeks from now. Prices are subject to change without notice. Our web site will always have the latest price schedule available. See <http://www.nstarsolutions.com/mm/purchase>

You can purchase upgrade licenses just as you would purchase regular licenses, except the product #s are slightly different (as shown above with the *Personal Use* vs *Professional Use* definitions). When you are ready to make your purchase, note the product #s and follow the same procedures discussed in the section [How to Pay for this Program](#). The most convenient way to purchase an upgrade is to go to our web site at <http://www.nstarsolutions.com/mm/purchase/upgrade.htm>

Need Help? Send e-mail to support@nstarsolutions.com if you are uncertain of the price and

would like help figuring it up.

Technical Support

Contact us using any of the methods listed below, and we will try to help. (Please make sure you have read the Help provided with this program first. The [Important Tricks and Techniques](#) section has answers to many common questions.)

When contacting us, please be sure to explain as many relevant details about the problem you are experiencing as possible and provide what version you have. You may easily contact us via any of the methods noted below (E-mail contact is preferred because it is fast, inexpensive, and accurate. We check e-mail several times a day and can often respond very quickly (and in more detail, if needed).

E-MAIL:

support@nstarsolutions.com

FAX:

785 539-3743

MAIL:

NorthStar Solutions

1228 Westloop Pl, #204

Manhattan, KS 66502-2840

Run Command

Starts a program or brings an existing program to the front of the Windows environment using its {[Activate...](#)} parameter.

Activate Command

Brings an already running window to the front of the screen.

Restore Command

Use after issuing the [Minimize](#) command to make the Macro Mania window a "normal" size.

Minimize Command

Causes Macro Mania to minimize itself (to the task bar). This is useful if you wish to see what is happening as the macro executes, or plan to immediately start doing something manually in a program after a macro is finished. You may also use the [MinimizeALL](#) command to minimize all the windows before you run or activate the particular window you want to work with.

Quick Start

The easy-to-use commands and syntax for *Macro Mania* will have you creating useful macros in minimal time. Once you learn the [2 important commands](#) of *Macro Mania*, you will have the ability to create macros for *all* your Windows programs! After you master those commands, you can quickly add to your macro vocabulary to create even more powerful, time-saving macros. In addition to using the *Command Wizard* available in the Macro Editor (as described below), you are encouraged to take a look at the examples that install with Macro Mania. If you read through those examples and generally understand what they are doing, you will be well on your way to mastering Macro Mania and creating powerful, time-saving macros of your own in very little time!!!

To View or Edit Existing Macros:

From the main *Macro Mania* window shown when you start the program, select *Add/Edit Macros* from the *File* pull-down menu to go to the Macro Editor. (Within the Macro Editor you can navigate to view or edit any macro that has already been created.) If you want to automatically view a given macro as soon as the Macro Editor window is displayed, you can easily do so: If in *Button Mode* (where the buttons are showing on the main screen), just *right* click the macro button or if in *List Mode*, left click once on the description (to put focus on that item in the list) and then *right* click the description. Regardless of how you get to the Macro Editor, once you are there you can navigate through all the macros using the convenient navigation tools or hotkeys (shown in the Macro Editors pull-down menus) to Goto (Ctrl+G), Find (Ctrl+F), or Move (Ctrl+P or Ctrl+N) to a particular macro.

To Create Your Own Macros:

1. Within the Macro Editor (see above about how you can get to the Editor) select *Add a New Macro* from the *File* pull-down menu (or press Ctrl+A).
2. Select either an icon or hotkey for the macro you are about to create.
3. Type in a description for your macro in the description box.
4. Begin typing your macro script. A convenient *Command Wizard* is available to you: all the Macro Mania commands can easily be inserted via the *Insert Command* pull-down menu on the Macro Editor Screen. The *Command Wizard* will help you get familiar with the commands available and help ensure you use correct syntax. In general, you will probably want to start with the [Run](#) command to start a program, then use the [Send](#) command to send keystrokes to your program(s). See [A Simple Example](#) for an easy example that you can copy to get acquainted with *Macro Mania*.
5. Once you have typed in your macro, select the *Save this Macro* from the *File* pull-down menu (or press Ctrl+S), and then you are ready to test your new macro by either closing the Macro Editor and selecting the macro you just made, or using [F5] within the Macro Editor.

Special Rules for the Send Command

The plus sign (+), caret (^), percent sign (%), tilde (~), and parentheses () have special meanings to the [Send](#) command. To specify one of these characters, enclose it inside braces. For example, to specify the plus sign, use {+}. Brackets ([]) have no special meaning to the Send command, but you must enclose them in braces as well, because in other programs brackets do have special meaning. To send brace characters, use {{} and {}}.

Msg

DESCRIPTION:

Displays a message and one or more buttons (e.g. Yes/No, Ok/Cancel, etc.) that should be selected as a response to the message. The macro will continue executing after a selection has been made (except when cancel has been selected, in which case the macro stops and exits any operations following the **Msg** Command). The results of the last response are always stored in a special variable called **{var-mr}** -- remember *mr* stands for *message response*. You can evaluate and manipulate the contents of that variable just as with any other variable. (See the Let Command for more details about evaluating and manipulating variables.)

SYNTAX:

Msg-<buttons> [Message] <{xPos = n, Ypos = n}>

Examples of different syntax options are noted further below. Message is any text you wish to have displayed in the message box. You may also use a variable for message. The **{xPos = n, Ypos = n}** parameter is optional and can be used to position the message box on the screen (useful if you prefer to not have it pop up in the middle as it does by default). The coordinates are the position of the top, left corner of the message box. The <buttons> part of the **Msg** Command is *required* and may be any of the following:

BUTTONS:

Ok
OkCancel
OkCancel-2
YesNo
YesNo-2
YesNoCancel
YesNoCancel-2
YesNoCancel-3
RetryCancel
RetryCancel-2
AbortRetryIgnore
AbortRetryIgnore-2
AbortRetryIgnore-3

DISPLAY (Default button)

Ok (Ok)
Ok & CANCEL (Ok)
Ok & CANCEL (Cancel)
Yes & No (Yes)
Yes & No (No)
Yes, No, & Cancel (Yes)
Yes, No, & Cancel (No)
Yes, No, & Cancel (Cancel)
Retry & Cancel (Retry)
Retry & Cancel (Cancel)
Abort, Retry, & Ignore (Abort)
Abort, Retry, & Ignore (Retry)
Abort, Retry, & Ignore (Ignore)

EXAMPLES:

Msg-Ok Select OK after the download has completed.

Msg-Ok This box gets positioned at the very top, left corner of the screen. {xPos = 0, Ypos = 0}

Msg-OkCancel {var-4}

Msg-OkCancel {var-4} {xPos = 100, yPos = 100}

Msg-YesNo-2 The file was not downloaded. Select Yes to continue or No to stop.

If **{var-mr}** = NO Then Exit

Msg-AbortRetryIgnore No files were found on drive A.

If **{var-mr}** = ABORT Then Exit

If **{var-mr}** = RETRY Then Goto Top

Otherwise the macro just continues

NOTES:

The **Msg** Command is useful for giving the user instructions, and/or stopping a macro while another program is executing and/or for getting simple responses from a user so that you may jump to another part of a macro with the Goto Command. (It is also useful to debug your macros since it can display the values of variables and stop at certain points in the macro.)

The **Msg** Command requires user input: use the **Pause** command to have a macro pause, but then continue *automatically* after a default time period has elapsed. The **Pause** command can also display a **RESUME** and **CANCEL** button by using the **Show** parameter, but it can not display a custom message as with the MSG Command. Another option to continue automatically is to set up a loop the checks for the existence of a particular window that should appear. For example, the following macro waits for a window to appear called Connected To (or it times out after 30 seconds):

```
Let {var-1} = 1
:WaitForConnect
Let {var-2} = MATH {var-2} + {var-1}
Pause 1
IF {var-2} = 30 THEN Exit
IF EXISTWINDOW Connected To THEN GoTo ContinueMacro ELSE GoTo WaitForConnect
:ContinueMacro
```

DESCRIPTION:

Stops the macro and displays a message with an OK and CANCEL button. If OK is selected, the macro continues, if CANCEL is selected the macro stops and exits.

SYNTAX:

Msg-OKCancel [message]

Message is any text you wish to have displayed in the message box. *Message* is any text you wish to have displayed in the message box. You may also use a **variable** for *message* (see example 2 below).

EXAMPLES:

Msg-OKCancel Select OK after the download has completed or CANCEL to abort this macro.

Msg-OKCancel {var-4}

Scheduling Macros

The interface for scheduling a macro can be found on the Macro Editor Window and should be easy enough to figure out. Here are a couple of important notes and strategies ...

Macro Mania must be running in order for this function to work. It also must be sitting at the regular, main icon window (the scheduler is not activated while you are in the Macro Editor).

You may wish to use the **Pause** command with the **show** parameter as the first command in any macro that is scheduled so that you can be warned that it is about to begin and you will be able to let it resume or cancel it. At any rate, you will want to stop what you are doing while the macro executes so that you do not interfere with it or otherwise accidentally put the focus onto the wrong program and cause unintentional things to occur.

Scheduling Times:

Macros can be scheduled to execute for a specific time interval. For example, to have the macro run at hh:00, hh:15, hh:30, and hh:45, select the Every 15 Minutes Option. If you want it to run every 2 minutes, select the corresponding option and the macro will run any time the minute is an even number (hh:02, hh:04, hh:06, etc.).

Any schedule you set for a macro is saved so that the next time you start Macro Mania that schedule is followed. Use the optional date setting if you only want the macro to run once (rather than every day at the scheduled time).

You can also define a specific time for a macro to run. Use regular time and define the AM or PM setting with the appropriate option button. For example, to have a macro run at 3:10 PM, key in **3** in the hour box, **10** in the minute box, and then select the **PM** option button.

SCHEDULING DAYS:

The interface to schedule days is very easy-to-use (just point-and-click) and can be found by selecting the *day* button from the scheduling interface on the Macro Editor Window. By default, if you only schedule the *time* portion of a macro, the macro will run every day for that time (or time interval) you set. However, you can have macros run on specific days of the week or specific dates by scheduling them to do so. If you select one or more days of the week (e.g. Monday, Tuesday, etc), the specific date (e.g. June 14, 1996) will be ignored. If you wish to have the macro run on a specific date, select that day from the calendar.

Benefits of Paying for this Program

After your registration payment has been received, you will be given a [registration code](#) to make your current copy (or copies) fully-registered and legal. As a result ...

- * There will be no more reminder screens and the initial splash screen will no longer be displayed.
- * Your name (or your company name) will be displayed on the About window. (No more shamefully displaying that you are merely evaluating the program.)
- * You will have done the right thing (both legally and morally).
- * You will be notified via Email of new releases.
- * You will be eligible for free upgrades to any minor release within the same version you have purchased and will be given the opportunity to purchase *major* updates (up to the same number of licenses you have already paid for) at a significant discount (currently a 50% discount). See [Upgrade Policy](#) for more details.
- * Although technical support is not often necessary, you will naturally be given more priority regarding technical support than someone that has not yet paid for the program.

(Registration Code)

Given to you after you register your copy (or copies) of Macro Mania, this code will make your copy a fully-registered version.

This code is entered in the "Enter Registration Code" window which can be found under the Help pull-down menu on the main Macro Mania window (the window with all your macros and their corresponding buttons).

If you provide us an e-mail address, it can be given to you immediately after your purchase has been processed.

PlayWav

DESCRIPTION:

Plays a *.wav file on any PC set up and equipped with a sound card.

SYNTAX:

PlayWav [path][wavfile]

wavfile is the file to be played and *path* is where that file exists. You may also use a [variable](#) for *[path]* *[wavfile]* (see example 2 below).

EXAMPLES:

PlayWav c:\windows\tada.wav

PlayWav {var-34}

NOTES:

If the wav file and/or path is not found, the macro will simply continue (without playing the wav file).

SEE ALSO:

[Beep](#)

WinExit

DESCRIPTION:

Allows you to reboot the computer, restart Windows, or exit Windows.

SYNTAX:

[WinExit](#) [n]

where [n] is either 1 or 2 as follows:

1 - shut down the system

2 - reboot the system

EXAMPLE:

[WinExit](#) 2

NOTES:

Macro Mania gracefully shuts down all Windows programs as if you manually closed Windows before closing other programs. If a program can not be closed gracefully, such as a running DOS program, then Windows will not exit and you will be prompted to close the active application before exiting.

Macro Mania is a Windows program and, as a result, Macro Mania is unloaded when this command is run. Therefore, there is no point to have any commands following the [WinExit](#) command in your macro, as they will never be executed. (You could, however, put Macro Mania in your Windows Startup Group and pass it a macro number as a parameter to have Macro Mania begin executing a macro immediately upon starting Windows. See [Launch a Particular Macro at Startup](#).)

It may be useful to have a [Msg Command](#) right before the [WinExit](#) command so you have the opportunity to abort the macro if desired.

Manipulating the State of Windows:

A window needs to be running either *Normal* or *Maximized* (not *Minimized* as an icon) for it to accept keystrokes. (When you think about it, this is true whether the keystrokes are being sent from Macro Mania, or you are working with a program and typing manually.)

For that reason, the [Activate](#) command will detect if a window is minimized and, upon detection of that, automatically restore the window to its *normal* size for you. If you are an advanced user and/or know you will remember to change the window state yourself within your macro, this feature may be overridden so that you can explicitly do this yourself (such as when you would rather that not happen because you plan to just close the window by sending an *Alt-Space*, *C* key combination (or *Alt-F4*) to the window (e.g. [Send %{}c](#) or [Send %{}F4](#)), or you plan to *maximize* the size of the window and do not want it to bounce between a restored and maximized state, etc.). To turn the feature on/off at any time, select the *Restore Minimized Window on Activate* from the File pull-down menu on the main screen for Macro Mania. Because many people forget to restore or maximize a window before trying to send it keystrokes, the default is for the feature to be on.

If you wish to explicitly change the state of a windows (change whether it is Maximized, Minimized, or Normal) you can easily do this by sending *Alt-space* (translated as *Send %{} }*) and the appropriate key. To get a better idea how this works, just press *Alt-space bar* now and then pay attention to the top, left control box of this particular window.

As you can see, the following keystrokes manipulate the state of a window quite easily:

[Send %{}r](#) - Restore the window
[Send %{}x](#) - maXimize the window
[Send %{}n](#) - miNimize the window

SEE ALSO:

[Minimize](#)
[Restore](#)

Launch a Particular Macro at Startup:

You can launch a particular macro when Macro Mania is first started by following the command with the number of the macro you want it to run automatically. For example, to run macro 3 you would use: "[path]MACROM.EXE 3" to automatically start your 3rd macro (where [path] is the location of MACROM.EXE).

Note that while adding new macros will not affect the position of your macros since new macros are simply appended to your macro list, deleting macros in front of existing macros causes them to be resequenced by -1. For example, you have 10 macros and delete the 7th, now the 8th becomes the 7th, the 9th becomes the 8th, and so on. Just bear this in mind whenever you delete macros so you can make any needed adjustments.

Adding Icons to the icon list:

Macro Mania installs nearly 200 icons you can choose from; however, if you want to add to the list of your choices, you easily can! Just copy any valid icon file(s) to the *icons* subdirectory beneath the subdirectory where you installed *Macro Mania*. (This is done automatically for you if you select an icon from a directory other than the default icon directory.)

There are many icon libraries available and you can probably locate one where you found this program. Be sure to find a library that has actual icons files (*.ico) that can be copied to the icon subdirectory, not icon libraries that are contained in DLLs -- unless you have a utility that can extract the icon files from the DLL.

The Macro Mania Icon Selector can read up to 900 icons, so please do not get too carried away <smile>.

Activating Buttons

Most buttons in a program can be activated by sending a space or, perhaps more obvious and common, an {enter}; however, be sure that the focus is on the button you wish to activate. A better way to activate a button is to use its hotkey if it has one. Hotkeys are the Alt-letter combination that invokes that button and can be identified because the letter is underlined. (This discussion is true, at least, if the program follows normal Windows conventions.)

Combine the search feature of the program with a macro

For example, suppose you have 50 expressions that say A = B, but they are not all the same, some say C = D, some say R = Y, etc. Using the *Find* command of your program, you can search for the equal sign (=) and then send the keystrokes that you would do in order to move the value on the left of the equal sign to the right of the equal sign and vice versa.

Here is a macro that would do that, assuming you have already done a search for the equal sign and F3 causes a find next to occur.

```
REM Repeat this 49 times
Repeat 49
REM Run (or activate) the program (in this case Visual Basic)
Run c:\vb\vb.exe
REM Search for the next equal sign
Send {F3}
REM Delete the equal sign and space, then cut what is on its right
Send {del 2}+{end}%Et
REM Add the equal sign and paste back what was on the left
Send {home}%Ep=
```

Save Space in Your Macros

It is not recommended you use your macros to type every character of a particularly big group of text (several paragraphs). By using the [Clipboard-Setfile](#) command, you can quickly put the contents of a large group of text into the clipboard and then paste it where you need. This will also improve performance for the macro in general, as pasting is faster than sending one character at a time with the Send command. This will also enable you to more easily edit the actual text in whatever text editor you choose (possibly to run spell check, format paragraphs more easily, etc.)

Sending a space (spacebar)

Either of the following two syntax methods will send a space:

Syntax 1: `Send` << two spaces follow *Send*

Syntax 2: `Send { }`

Since brackets are ignored in the `Send` command, the second syntax method is helpful for identifying that a space is there and is a better method.

Stopping a macro

Once a macro is in motion, it may be difficult stop it, which is why we always emphasize saving your work and testing your macros carefully. However, if you do have a macro that you might need to stop (like when you use the [Repeat](#) Command and are not sure of the exact number of times you need it to repeat, etc.) you can explicitly add a command in the macro which would allow you to stop the macro at certain points:

One option would be to place a [Msg Command](#) in the macro. This would allow you the option to cancel it or press Enter to have it resume.

Another option would be to use the [Pause](#) command with the [Show](#) parameter and a small delay (2-3 seconds), then you would be able to cancel the macro, press resume, or it would count down in a couple of seconds and resume automatically.

Yet another option would be to quickly put focus onto Macro Manias main window when you know that the [Send](#) command is about to be issued in the macro. Because Macro Mania smartly identifies that keystrokes can not be passed to itself, if keystrokes get sent to it, it will cause an error and stop the macro.

For more control over repeating macros, be sure to use the [Repeat-Prompt](#) command which allows you to dynamically set how many times a macro repeats.

(Important Commands)

Although Macro Mania sports a variety of commands to add power to your macros, [Run](#) and [Send](#) are the two most important commands that will help you create useful macros fast and easy. As you become a more experienced *Macro Maniac*, you can add to your macro vocabulary and create more sophisticated macros. Also note that the syntax for creating macros is fairly simple, and reference is just a few steps away in this help system.

For added convenience, you can access all the commands available to you via the *Insert Command* pull-down menu of the Macro Editor. The *Command Wizard* will help you get a general overview of the commands available to you and help you keep proper syntax.

Clipboard-Set

DESCRIPTION:

Sets the contents of the Windows clipboard with text you specify.

SYNTAX:

[Clipboard-Set](#) [text]

text is a single string of characters you want to put in the Windows clipboard. You may also use a variable for text (see example 2 below).

EXAMPLES:

[Clipboard-Set](#) Blessed is the man who finds wisdom, the man who gains understanding, for she is more profitable than silver and yields better returns than gold. - *Proverbs 3:11* (NIV)

[Clipboard-Set](#) {var-77}

NOTES:

This command is useful for putting small amounts of simple text strings into the clipboard for future paste commands to other programs. However, using the [Send](#) command will accomplish this just as effectively, though the information may be sent slightly slower. If a large amount of text, or text which includes carriage returns, is needed, see the [Clipboard-SetFile](#) command for a better way to put such information in the clipboard.

If text is omitted, the contents of the clipboard is cleared.

The length of the text you wish to put in the clipboard can not exceed any limitations for the Windows clipboard.

Clipboard-SetFile

DESCRIPTION:

Sets the contents of the Windows clipboard with the contents of the file you specify.

SYNTAX:

`Clipboard-SetFile [file]`

file is the full path and file name for a text file that you want put in the Windows clipboard. You may also use a [variable](#) for *file* (see example 2 below).

EXAMPLES:

`Clipboard-Setfile c:\windows\readme.txt`

`Clipboard-Setfile {var-33}`

NOTES:

The file should be a pure text file. Unlike the [Clipboard-Set](#) command, this command enables you to also put text with Carriage Returns in the clipboard, since Carriage Returns are copied to the clipboard as they appear in the file.

For this command to work, the length of the file can not exceed 65,535 bytes. Of course, that is probably more than the limit for how much the clipboard can hold anyway. The length of the file can not exceed the limitations of the Windows clipboard for this command to work.

If the file is not found, an error occurs and the macro will terminate.

Clipboard-Restore

DESCRIPTION:

Restores the contents of the Windows clipboard which was saved with the [Clipboard-Save](#) command.

SYNTAX:

[Clipboard-Restore](#)

NOTES:

This is useful if you do not want the clipboard commands of Macro Mania to permanently overwrite the contents of the Windows clipboard after a macro has finished executing. The Clipboard-Save and Clipboard-Restore commands must be used in the same macro for Macro Mania to associate them correctly.

The [Clipboard-Save](#) command must have been used prior to this command, or the contents of the clipboard will be cleared entirely.

Clipboard-Save

DESCRIPTION:

Saves the current contents of the Windows clipboard so that a [ClipBoard-Restore](#) command may be used later in a macro to restore the clipboard contents.

SYNTAX:

[Clipboard-Save](#)

NOTES:

This is useful if you do not want the clipboard commands of Macro Mania to permanently overwrite the contents of the Windows clipboard after a macro has finished executing. The [Clipboard-Save](#) and [Clipboard-Restore](#) commands must be used in the same macro for Macro Mania to associate them correctly.

Repeat-Prompt

DESCRIPTION:

Prompts you to enter the number of times you want a macro to repeat. See the [repeat](#) command for more information about repeating macros.

SYNTAX:

[Repeat-Prompt](#) <text>

text is an optional parameter you can add to customize the prompt. If *text* is omitted, a generic prompt with the title of the macro will be used.

EXAMPLE:

[Repeat-Prompt](#) Enter the number of times to copy information from Program A to Program B.

NOTES:

The prompt will only appear once (the *first* time it is encountered) during the execution of a macro. Unlike the [Repeat](#) Command, the [Repeat-Prompt](#) Command allows you to more dynamically set the number of times a macro executes.

SEE ALSO:

[Stopping a Macro](#)

Branch

DESCRIPTION:

Allows you to specify another macro file which will pop up and allow you to interactively select an option. Once an option is selected, a macro will run and return to the original macro to execute further macro commands if needed.

SYNTAX:

Branch [macro file]

macro file is the name of the Macro Mania data file that you want displayed. You may also use a [variable](#) for *file* (see example 2 below).

EXAMPLES:

Branch choices.mm

Branch {var-z}

NOTES:

Macro Mania will suspend all macro activity until an option is selected from a branch option box. Once an option is selected, the macro for that option is executed and control returns back to the original macro that called the branch. You can cancel a branch by selecting *Exit* from the *File* pull-down menu and the option to cancel or resume the main macro entirely will be presented.

You may nest **Branch** commands from other branch option boxes. Although most Macro Mania commands may be used from a branch, which certainly includes the 2 [important commands](#), the [Open](#) command may not be used from a branch option box.

Use the [Open](#) command to open a different macro file entirely without just branching to it.

See [Creating New Macro Files](#), for more information about how to create branch files.

As you are testing your Branches, note that their positions and sizes can be manipulated and will automatically be stored for the next time that branch is used (the same feature that is applied to the main Macro Mania screen).

Creating New Macro Files

Saving New Macro Files:

You may find it useful to group your macros into different categories (e.g. by function, user, application they are used for, etc.). Macro Mania allows you to do this by creating new macro files in addition to the default DEFAULT.MM file. Creating new macro files is also necessary if you wish to use the [Branch](#) command to provide a branch option box in your macros.

To create a new file, you can either select *New* from the *File* pull-down menu of the Macro Editor and then enter the name for the new file, or save the current macro file you are working with to a new name with the *Save As* option under the *File* pull-down menu. (Note: It is recommended you use the MM extension for macro files to keep their extensions consistent. The *Open* feature when you manually select macro files to open will default to looking for all files that end with .MM.)

Opening Other Macro Files:

Although Macro Mania usually tries to locate and execute its default file, DEFAULT.MM, you can pass it the file as a parameter when first running *macrom.exe* for it to automatically open a different macro file when it first loads. For example, c:\progra~1\macrom~1\macrom.exe c:\progra~1\macrom~1\eric.mm will automatically open eric.mm rather than DEFAULT.MM when Macro Mania first loads.

Note that you should pass the full path and macro name to tell macro mania where to locate the macro file; however, if no path is found, it looks in its own subdirectory, then if the file is still not found, it opens DEFAULT.MM file. (You can also use the [Open](#) command to have a macro open another macro file, or you can open another file manually by selecting *Open* from the *File* pull-down menu.)

Open

DESCRIPTION:

Opens another macro file for Macro Mania to use as its main macro file.

SYNTAX:

Open [macro file]

macro file is the name of the Macro Mania data file that you want to use. You may also use a [variable](#) for *n* (see example 2 below).

EXAMPLES:

Open jeff.mm

Open {var-m}

NOTES:

See [Creating New Macro Files](#) for more information.

Fast access to edit a macro

From the main *Macro Mania* window shown when you start the program, select *Add/Edit Macros* from the File pull-down menu to go to the Macro Editor. (Within the Macro Editor you can navigate to view or edit any macro that has already been created.) If you want to automatically view a given macro as soon as the Macro Editor window is displayed, you can easily do so: If in *Button Mode* (where the buttons are showing on the main screen), just *right* click the macro button or if in *List Mode*, left click once on the description (to put focus on that item in the list) and then *right* click the description. Regardless of how you get to the Macro Editor, once you are there you can navigate through all the macros using the convenient navigation tools or hotkeys (shown in the Macro Editors pull-down menus) to Goto (Ctrl+G), Find (Ctrl+F), or Move (Ctrl+P or Ctrl+N) to a particular macro.

If you have several macros and do not remember their sequence number, you can easily exit the Macro Editor, find the next macro you wish to edit, and repeat this procedure rather than try and search/scroll for them from within the editor.

General Keyboard Navigation

The following are some navigation tips that apply to most Windows programs. Since many Windows program(mer)s follow a set of rules about how a Windows application is supposed to behave, these rules will likely apply (though we obviously have no control if a program(mer) does not follow these conventions). While this is not a definitive discussion on the topic, we think it covers the most common keyboard conventions/shortcuts available in most Windows applications.

Navigating Folder Tabs - once one of the tabs has the focus, the left and right arrow keys will usually move to the tab to the left and right of the current tab.

Expanding/Collapsing List Branches - the left and right arrow keys can also be used to collapse and expand list branches where that feature is available (usually indicated with a little plus (+) or minus (-) sign next to the branch). The arrow keys are more friendly to macros than the plus and minus keys which often require a numeric keypad keystroke.

Tab - Moves from object to object in (hopefully) a logical pattern (often left to right, top to bottom like one would read a book). **Shift-Tab** will often move the cursor in the opposite direction of tab. Naturally an exception to this is with large text fields where a tab needs to act like a tab (usually to quickly indent a certain number of spaces).

Ctrl-Left and Ctrl-Right - Usually allows you to jump a word at a time left or right, respectively.

Home and End - Move the cursor to the far left or far right of a field, respectively within an object that holds text. In a list box, it often moves to the very top (home), or very bottom (end) entry.

Ctrl-Home and Ctrl-End - In large notepad entry fields, move the cursor to the top, left corner or bottom, right corner respectively.

Holding the **Shift** key down while moving the cursor within a field has special meaning: it *highlights* the text it crosses similar to dragging your mouse pointer across text (allowing for subsequent cut, copy, and paste operations). Using the shift key with some of the conventions mentioned above, you can highlight text within a field just as if you had taken the mouse and dragged across the text. For example, to highlight the entire portion of a large text field, you can first move to the top of the field (Ctrl-Home), then while holding the Shift key, move to the very bottom of the field (Ctrl-End). Translated into a macro where a ^ represents the Ctrl key and a + represents the Shift key, it would look like this: `Send ^{Home}+^{End}`. To highlight just the current line the cursor is on, you would use: `Send {home}+{end}`

Often letters that are underlined mean that the command can be quickly executed by holding down the Alt key and pressing that letter at the same time. For example, a button or menu that had the word Help can be invoked by holding down the Alt key and pressing H, translated as `Send %H` in macro syntax.

Other Conventions/Shortcuts:

F1 - Invoke the Help system

Alt-F4 - Quit a program

Ctrl-X - Cut highlighted text

Ctrl-C - Copy highlighted text

Ctrl-V - Paste text currently in clipboard.

Shift-Insert - Another Paste shortcut

Del - Delete character to the *right* of the cursor (or a currently highlighted section)

BackSpace - Delete character to the *left* of the cursor (or a currently highlighted section)

PgUp - Scroll one screen up

PgDown - Scroll one screen down

Alt-Space - Display the control box of a Window

Note that programs often have their own shortcuts built into them, and you are encouraged to take advantage of them if you can. For example, in word processors, Ctrl-B often means to toggle the **bold** attribute of the font, Ctrl-U to toggle underline, Ctrl-I toggles the *italic* attribute, etc.

Password Protecting Your Macros

To offer *some* security over your macros, you can define a password that must be used to either open a macro file, view/edit your macros via Macro Mania, or both. This password protection scheme is not hack proof in that someone with reasonable knowledge of computer files can view and/or edit your macros, but it does offer protection from casual nosiness or prying eyes.

Actually, the only reason it was added is because some have mentioned they would like to create macros for others without the others going in and messing them up, in which case you can set the password level to prevent someone from accessing the Macro Editor screen. To prevent anyone but people with the password to be able to open (and consequently run) macros, you can password protect the macro file entirely.

The password and level of use is set up via the *Set Password Options* found under the File pull-down menu on the Macro Editor.

IMPORTANT NOTE: As mentioned above, this password protection scheme is not absolutely hack proof, but it should offer some reasonable level of protection for preventing people from easily viewing, editing, and/or using your macros. However, if you use Macro Mania to send passwords, etc. you should note that someone with a little more than casual experience with Windows may be able to view your macros via a way other than using the Macro Mania interface. If you think that information in your macros could become compromised, you should not hard code that information into your macros. Consider using the GetInput and SendInput commands in your macros to dynamically send password information, etc. from your macros.

Subroutines within macros (looping, etc.)

There may be times when you want only a certain portion of a macro to loop. That is, you have something that needs to be done repeatedly, but you do not want the whole macro to be done repeatedly. You can do this easily with Macro Mania by combining a couple of its features:

Create a separate macro that does what you need to be done repeatedly. Then by using the [Repeat](#) (or [Repeat-Prompt](#)) Command, you can control how many times that part repeats. You are essentially creating a looping subroutine. Then simply use the [Macro](#) Command to call that repeating macro, and only that portion will repeat, not the entire macro. If you are concerned about the repeating macros getting in the way, etc. you do not need to be -- just make them Invisible by checking that option in the Icon box when you set them up. That way they'll be available to your other macros, but not available (or in the way) as a selection from the Macro Mania Main Screen.

Also, since your macros are *automatically* adjusted when you delete or reposition macros, you will not have to worry about keeping track of their sequence, etc. -- it is all done for you. (See [Resequencing Macros](#).)

DOS Batch Files

Sometimes there is nothing as good as a DOS batch file for accomplishing what you need to do. With DOS batch files, you have a whole arsenal of commands that allow you to copy, delete, and move files, make and remove subdirectories, etc. DOS batch files can branch (*goto*), receive and/or check for conditional parameters, and much more. The commands in DOS are quite powerful, and they have a *lot* of development and testing time invested in them. Just as an example, go to a DOS prompt and type *xcopy /?* to get an idea of all the features and parameters for just that one command!

Batch files can check for the existence of files (*if exist* and *if not exist*), display prompts and solicit user input (see the DOS *Choice* command), run programs, etc. This help section is not intended to be a DOS tutorial, but we do want to point out this often overlooked resource -- which can easily be used in combination with macros to accomplish those kinds of tasks.

To use DOS batch files with Macro Mania, simply create your batch file and then use the Macro Mania Run Command to launch the batch file. Use the optional {wait} parameter with the *Run* command to halt execution of the macro until your batch file has finished. (You may need to edit the properties of the DOS shell the first time the batch file is run if you want to toggle whether it should automatically close when it is finished, if it should run in a full window, etc. To access the properties, click on the top, left corner of the DOS window and select *properties* from the menu).

So why have we included some DOS-like commands in Macro Mania? One ugly limitation of the DOS interface is it is a DOS interface. That is, you lose the true *Windows look-and-feel* when shelling out to DOS, especially within a fancy Windows macro. Also, some people entered the computer scene after the DOS prompt was something usual to work with and may not be as familiar with DOS as others. However, if you are not afraid of the DOS prompt and need the power, do not look over the tried and true commands that come with DOS.

SEE ALSO:

[Waiting for Another Program to Finish](#)

Helpful Hints for Beginners

THE COMMAND WIZARD

The macro editor screen has a pull-down menu entitled *Insert Command*. All of the macro commands are available from this *Command Wizard* and can both help you get a general overview of all the commands available to you as well as help you maintain proper syntax.

PRACTICE WITH A FEW SIMPLE MACROS:

Macro Mania is very powerful and relatively simple to use. The nice thing is that once you master it, you can use it for *any* windows program without having to relearn new syntax for every program (and most programs do not have the ability to create macros anyway). However, you may wish to start off gradually, creating a few simple macros to get an understanding of the basic concept. Also note that there are only two [important commands](#) that are truly essential to creating useful macros with Macro Mania.

GET FAMILIAR WITH THE SEND COMMAND:

The [Send](#) command is the core part of Macro Mania, offering a full host of options to ensure you can do virtually anything, send any keystroke, and be as productive as possible without a lot of repetitive typing. We recommend that if you print anything, you print the syntax of the [Send](#) command in this help screen so you will have easy access to the syntax while you create your macros. Be sure to pay careful attention to the [special rules](#) of the [Send](#) command.

SAVE YOUR WORK:

Before executing a macro, especially a new one, save your work in all your running applications so that if a macro you write does something you did not quite expect, you can return to the original state of the program easily.

MANUALLY PERFORM THE STEPS AND WRITE THEM DOWN:

Before trying to create a macro, perform the task manually and carefully record every key that you press. Another way to write macros is to perform a few keystrokes, switch over to Macro Mania to record them into your macro, then switch back and perform a few more keystrokes, switch back, etc. (See [Recording Keystrokes](#)).

DEBUGGING YOUR MACROS:

Macro Mania sends keystrokes fast -- sometimes too fast for you to detect where a macro may be doing something unexpected or missing a step. For debugging: if you are not sure where your macro is not working correctly, place the [exit](#) command a few lines above where you think the error is happening. Then gradually move it down one line at a time until the line with the error is identified. (It may also be useful to break your [Send](#) commands into smaller ones for this purpose.)

ALWAYS TEST YOUR MACROS:

Test your macros, especially before issuing the [Macro](#) Command that calls other macros, or the [Repeat](#) command which causes the macro to repeat itself. If a macro is not working properly, you definitely want to find out BEFORE you have issued the macro several times.

MAKE MACROS THAT ARE EASY TO READ AND FOLLOW:

Macro Mania allows you to put in extra spaces, add comments, and issue commands repeatedly -- so try to make your macros easy to read. For example, while the following two macros do the same thing, the

2nd one better explains what is going on with the macro...

EXAMPLE 1:

Minimize

Run write.exe {Activate WordPad}

Send %oP%Ac{enter}%Of%S12{enter}

Send ^B

Send Acme Technologies{ENTER}1228 Westloop PI, Ste 204{ENTER}Manhattan, KS 66502{ENTER 3}

Send ^{HOME}+{END}^I

Send ^{END}

Send %oP%AL{enter}

Send %Of%S10{enter}Dear :{left}

EXAMPLE 2:

Minimize

Run write.exe {Activate WordPad}

' == Center the text ==

Send %oP%Ac{enter}

' == Resize the font a little larger ==

Send %Of%S12{enter}

' == Make the text bold ==

Send ^B

' == Type in the letter head ==

Send Acme Technologies{ENTER}1228 Westloop PI, Ste 204{ENTER}Manhattan, KS 66502{ENTER 3}

' == Go back and italicize the company name ==

Send ^{HOME}+{END}^I

' == Go to bottom ==

Send ^{END}

' == Left-justify the paragraph

Send %oP%AL{enter}

' == Resize a little smaller than the letterhead ==

Send %Of%S10{enter}

' == Begin Letter ==

Send Dear :{left}

<p>Remember, once you master using macro mania, it is the only syntax you will need to remember to create macros for many, if not all, of your windows programs! A little time invested in learning Macro Mania will save you a lot of time in the long run!!!</p>

Recording Keystrokes

Macro Mania does not record your keystrokes as you enter them. Actually, we always found that process awkward when using the old Windows recorder program -- one wrong keystroke and you had to start all over. Also, since Macro Mania does so much more than just send keystrokes, it would be even more cumbersome to try and record keystrokes when mixed with all the other features you can use with your macros.

However, translating your keystrokes into a macro is very easy and this should not present any problems. When you create a macro, you can either remember the keystrokes needed, or if it is a long sequence, simply start editing the macro, then flip over to whatever application is getting the keystrokes and perform a few, flip back to the editor to write the macro reflecting what was typed, etc. The process is really easy and takes very little time. Once you have done this a couple times you will appreciate how easy it is, and that macros can be edited, portions copied and reused for other macros, etc. As you become more experienced, you will be creating and editing macros very quickly and easily without any cumbersome recording mechanism.

Launching Macros

To launch a macro, simply click once on the button or use the tab key on your keyboard to tab to the macro you want, then press Enter or Space to launch it. The description of the macro will be shown in the bottom bar of the main Macro Mania Window when that button is highlighted (not necessarily chosen). You should use descriptions to best identify what a macro does and you can further aid yourself by selecting icons to match.

Another way to launch a macro is to use its assigned [hotkey](#).

Related Topics:

[Launch a Particular Macro at Startup](#)

[Adding Icons to the icon list](#)

Resequencing Macros

When you add macros, the new macro is just appended to the end of the macro file/sequence. However, you may find it useful to sequence your macros in a certain order. For example, you may wish to put macros in a logical order, group like macros together, and/or put all hidden macros at the end of the sequence.

To resequence a macro, just determine what new position you want to put it in (use the macro number indicator at the top of the editor screen or on the status bar display of the main window), then go to the macro you wish to resequence, select the *Set Sequential Order # of Macro* from the File pull-down menu in the Macro Editor and enter the new position you want it to be at.

This will not only put the macro in the new position, but it will *automatically* adjust any macros that use the [macro](#) command. Otherwise they could end up pointing to the wrong macro if a macro they point to is resequenced to another position (and, thus, assigned a different macro number).

Waiting for another program to finish

If you use the {wait} parameter with the [Run](#) command, Macro Mania will halt executing the rest of the macro until the program has terminated (either by itself automatically or manually by a user). This is especially useful with batch files, etc. that you want to run and that terminate themselves automatically.

IMPORTANT NOTE: This feature will not work if the [Run](#) command merely activates a program that has been running already. That is, the [Run](#) command needs to launch the program that the macro is waiting on as a new instance. (If needed, you could ensure the program is closed using the [If ExistWindow](#) and issuing the commands to close the window, then run it and, thus, this would make sure it was run new, not just activated to the front.)

If you are running a program in a DOS shell, the program may not exit automatically unless you have selected the *Close on Exit* option. You can set that property by going to the programs Control Box (right click on the top status bar where the program title is given or left click the very top, left corner of the status bar where the icon is located), select *Properties*, then make sure the *Close on Exit* box is checked.

SEE ALSO:

[DOS Batch Files](#)

Activating child Windows

Child Windows are those small Windows that are contained within the larger (parent) window of a program. You can activate these smaller windows by using a Alt-[minus sign]. Translated in Macro Mania syntax as follows:

[Send](#) %-

Note that you may also tab through the child windows by sending Alt-[F6] (Send %{F6}) or sometimes by using the left and right arrow keys once the control box menu (the little pull-down menu in the top, left corner) has been activated in a child window with either Alt-Alt-[minus sign] or Alt-[spacebar].

Related Topics:

[Manipulating the State of Windows](#)

OnTop

DESCRIPTION:

Enables you to toggle the *Always On Top* property (if the main window of Macro Mania floats on top of the other windows) in a macro.

SYNTAX:

OnTop = [True | False]

EXAMPLE:

OnTop = True

NOTES:

Especially useful if need to watch what is happening in a macro and then need to manually intervene to make the macro continue via a [Msg-OK](#) or [Msg-OKCancel](#) message box. By setting [OnTop](#) = True, the message box will float on top so that you can watch what is happening and yet be ready to select the appropriate item in the message box when ready without having to Alt-Tab back to Macro Mania.

Activating the Windows Desktop

In order to activate the desktop, just use the [MinimizeAll](#) command. This will put focus onto the desktop, and then you can navigate to the icon of your choosing using arrow movement, home, end, etc.

SEE ALSO:

[Activating the Windows Start Button](#)

[Activating a DUN Session](#)

Activating a DUN Session

With the special {DUN} parameter of the **Run** Command, you can have Macro Mania quickly launch a Dial-Up Connection -- and optionally send additional keystrokes necessary for the connection.

SYNTAX:

Run {DUN} [*ConnectionName*]

Where *ConnectionName* is the name of the DUN connection you want to use.

EXAMPLE:

MinimizeALL

Run {DUN} *My Connection* {**Activate** Connect}

Send %P

Send *MyPassword*{enter}

Note: *My Connection* and *MyPassword* are whatever yours are.

Also, if you are already connected and want to disconnect quickly, you can do so with a macro like this:

Send {DUN} *My Connection*

Pause 1

Send %c

NOTES:

The {**Activate** Connect} parameter is used when *connecting*, but should not be used when *disconnecting* (it does not find the window correctly when disconnecting; but, fortunately, the DUN session knows to pull itself to the front of the screen if there is already a connection).

Accessing Control Panel Items

With these special parameters of the **Run** Command, you can have Macro Mania quickly access the Control Panel items indicated:

Accessibility Options

Keyboard: **Run** {CP} access.cpl,,1
Sound: **Run** {CP} access.cpl,,2
Display: **Run** {CP} access.cpl,,3
Mouse: **Run** {CP} access.cpl,,4
General: **Run** {CP} access.cpl,,5

Add/Remove Programs

Install/Uninstall: **Run** {CP} appwiz.cpl,,1
Windows Setup: **Run** {CP} appwiz.cpl,,2
Startup Disk: **Run** {CP} appwiz.cpl,,3

Date and Time Options

Date/Time: **Run** {CP} timedate.cpl

Display Options

Background: **Run** {CP} desk.cpl,,0
Screen Saver: **Run** {CP} desk.cpl,,1
Appearance: **Run** {CP} desk.cpl,,2
Settings: **Run** {CP} desk.cpl,,3

Joystick Options

Joystick Properties (Joystick: **Run** {CP} joy.cpl

Mouse/Keyboard/Printers/Fonts Options

Mouse Properties: **Run** {CP} main.cpl @0
Keyboard Properties: **Run** {CP} main.cpl @1
Printers: **Run** {CP} main.cpl @2
Fonts: **Run** {CP} main.cpl @3

Mail and Fax Options

Microsoft Exchange Profiles: **Run** {CP} mlcfg32.cpl

Microsoft Mail Postoffice Options

Microsoft Workgroup Postoffice Admin: **Run** {CP} wgpocpl.cpl

Modem Options

General: **Run** {CP} modem.cpl

Multimedia/Sounds Options

Audio: **Run** {CP} mmsys.cpl,,0
Video: **Run** {CP} mmsys.cpl,,1
MIDI: **Run** {CP} mmsys.cpl,,2
CD Music: **Run** {CP} mmsys.cpl,,3
Advanced: **Run** {CP} mmsys.cpl,,4
Sounds Properties: **Run** {CP} mmsys.cpl @1

Network Options

Configuration: **Run** {CP} netcpl.cpl

Password Options

Change Passwords: [Run {CP} password.cpl](#)

Regional Settings

Country: [Run {CP} intl.cpl,,0](#)

Number: [Run {CP} intl.cpl,,1](#)

Currency: [Run {CP} intl.cpl,,2](#)

Time: [Run {CP} intl.cpl,,3](#)

Date: [Run {CP} intl.cpl,,4](#)

System/Add New Hardware Options

General: [Run {CP} sysdm.cpl,,0](#)

Device Manager: [Run {CP} sysdm.cpl,,1](#)

Hardware Profiles: [Run {CP} sysdm.cpl,,2](#)

Performance: [Run {CP} sysdm.cpl,,3](#)

Add New Hardware Wizard: [Run {CP} sysdm.cpl @1](#)

Clipboard Commands

The following commands enable you to quickly and easily retrieve and set text into the Windows clipboard.

- [Clipboard-Set](#)
- [Clipboard-SetFile](#)
- [Clipboard-Restore](#)
- [Clipboard-Save](#)

Related Commands:

- [Let \(ClipBoard\)](#)

If-Then

DESCRIPTION:

Allows conditional execution of a macro based on evaluation of [variables](#).

SYNTAX:

If *condition* Then *ThenCommand* [Else *ElseCommand*]

[condition](#) may be the comparison of two variables such as:

If {var-a} = {var-b} Then...

If {var-c} <> {var-d} Then...

If {var-e} > {var-f} Then...

If {var-g} < {var-h} Then...

If {var-g} => {var-h} Then...

If {var-g} =< {var-h} Then...

or [condition](#) can check for the existence of a file as follows:

If ExistFile [FileName] Then...

where *FileName* is the path and file to check for. You may also use a [variable](#) for *FileName* (e.g. [If ExistFile {var-p} Then...](#)). You can also use this command to check for the existence of a subdirectory by using the file name *nul*. For example, if you wanted to check if the subdirectory *c:\test* existed, you would use the command [If ExistFile c:\test\nul Then...](#)

or [condition](#) can check for the existence of a Window as follows:

If ExistWindow [Title Bar] Then...

where [Title Bar] is the caption of the window, and may be either the main window or a child window of a program (the *Macro Mania Help* title bar shown for this Help System is the Title Bar for this window).

The *Title Bar* may be a full or partial match (use the full title if you want it to match exactly or only a partial title if you think parts of the name could change). *Title Bar* is not case sensitive

You may also use a [variable](#) for *Title Bar* (e.g. [If ExistWindow {var-p} Then...](#)).

EXAMPLE 1:

[If ExistWindow](#) myfile.txt THEN Activate Notepad ELSE Run notepad.exe, myfile.txt

EXAMPLE 2:

rem This macro waits for a window entitled "Connected To" to appear, or times out after 30 seconds.

Let {var-1} = 1

:WaitForConnect

Let {var-2} = MATH {var-2} + {var-1}

Pause 1

IF {var-2} = 30 THEN Exit

IF EXISTWINDOW Connected To THEN GoTo ContinueMacro ELSE GoTo WaitForConnect

:ContinueMacro

or [condition](#) can check for the existence of a string (another variable) within a variable as follows:

If {var-?} CONTAINS {var-a} Then...

[ThenCommand](#) and [ElseCommand](#) are any valid Macro Mania commands (except another [If-Then Command](#)). The [Else](#) part is optional. When omitted and the condition is *false*, the macro just continues to the next macro line.

EXAMPLES:

```

IF {var-b} = {var-c} THEN GoTo FINISH
IF {var-c} > {var-d} THEN BRANCH c.mm ELSE Branch d.mm
IF {var-r} <> {var-s} THEN Exit
IF {var-1} => {var-2} THEN GoTo CheckVar3 ELSE Exit
IF {var-3} =< {var-4} THEN Exit
IF EXISTFILE c:\autoexec.bat THEN Branch edit.mm
IF EXISTFILE {var-f} THEN Branch edit.mm
IF EXISTWINDOW NorthStar - Orders THEN GoTo NS
IF EXISTWINDOW {var-4} THEN GoTo NS
IF ExistWindow2 NorthStar THEN GoTo NS
IF ExistWindow2 {var-g} THEN GoTo NS
IF {var-s} Contains {var-19} THEN GoTo Kansas

```

NOTES:

The *equal* (=) and *not equal* (<>) conditions compare the variables as *strings*, which means you can compare text or numbers. However, note that, for example, 1,234 <> 1234. Use the [Let \(Number\)](#) command to convert strings to numbers. The *greater than* (>), *less than* (<), *equal or greater than* (=>), and *equal or less than* (=<) imply you are evaluating numbers and you do not need to convert the strings to numbers when using any of those conditions. If a string does not have any number in it, the string will equal 0.

When using *equal* (=) and *not equal* (<>) conditions, the comparison is case sensitive; thus, *NorthStar* <> *Northstar*. If you do not want the comparison to be case sensitive, first set the variables to all one case using either the [Let \(UCase\)](#) or [Let\(LCase\)](#) command, then make the comparison.

The *greater than* (>), *less than* (<), *greater than or equal* (>=), and *less than or equal* (<=) conditions compare numbers only, which means, as expected, 1,234 > 1233, 1234 < 1,235, 1.09 < 1.30, 123 <= 123, 123 >= 122.9, etc.

Be careful when comparing strings. Mainly, be careful that the string you are comparing is what you think it is. For example if you highlight a line of text in a word processor by going to the beginning of the line and then use the down arrow, it is possible that you will also get a Carriage Return, Line Feed (CR, LF) as part of the text. This is different than highlighting a line of text by going to the beginning of the line and then to the end of the line (not just dropping down). You can confirm this by experimenting yourself and seeing the results of when you paste the text somewhere. The first method will paste the text and then drop down to another line while the second method will leave the cursor at the end of the text. So, if you are comparing a line of text and do not want the CR, LF, you should use the second method of highlighting a line of text.

If you are using the *Contains* condition, the search *is case sensitive*, so if the string contains hello and you search for Hello, the condition will not be true. If you wish for the search to not be case sensitive, just use either the LCase or UCase sub-command of the Let command to make both variables in the command all one case before issuing the comparison.

The command must all be on the *same* line (or naturally wrapped around with the word-wrap). In other words, *do not press enter to break up the If-Then command*.

As noted in the syntax, the *ThenCommand* or *ElseCommand* may be any valid Macro Mania command except another If-Then command. If you need to check for a condition and then check for another condition, just use the Goto command to branch to another If-Then command in your macro.

Note that except for the [ExistFile](#) conditions, the *condition* part of the command uses variables. You need only use the [Let](#) command previously in your macro to make those variables mean something, such as setting a variable equal to text, the contents of a file or the clipboard, etc.

The [ExistWindow](#) is useful to first check for a Window that may indicate if you need to use the [Run](#) command and then send keystrokes to close the program. If the program is not running, there is no need to run it just to close it again.

The IF-THEN command can bring a great deal of power to your macros. Programmers will appreciate the fact you can now write flags to files, then read in those flags with a macro and then execute the macro accordingly. In fact, you could use a basic flag of just creating a file, any file, and then using the ExistFile condition noted above. If a file exists, the macro does one thing, if it does not exist, the macro does another thing. To take it one step further, you could put the file into a variable using the [Let \(File\)](#) command. Then you can evaluate or search for an expression in the variable with this If-Then command and have the macro act accordingly.

RELATED COMMANDS:

While you can use any valid Macro Mania command with the IF-THEN command (except another IF-THEN command), the following commands are likely to be used most often:

[Let](#)

[Goto](#)

[Branch](#)

[Exit](#)

GoTo

DESCRIPTION:

Causes execution of the macro to jump to another place (label) within the macro.

SYNTAX:

GOTO [label]

where label is a valid label in the macro as defined below.

EXAMPLE:

Let {var-b} = {1234567}

Let {var-c} = Clipboard

IF {var-b} = {var-c} THEN GoTo FINISH

.

.

:FINISH

.

.

Msg-OK Macro is now complete.

NOTES:

A label can be any alphanumeric string and always begins with a single colon (:). Spaces are also permitted.

Do not include the colon (:) in the GoTo command, only include the colon at the beginning of the label itself.

The [label] is not case sensitive, so :LABEL 32 = :Label 32.

If [label] is not found, an error occurs and the macro stops.

As shown in the example above, the Goto Command is especially useful when combined with the IF-THEN command.

Let

DESCRIPTION:

This command has several sub-commands that enable you to save text to variables and to manipulate the contents of those variables.

Setting a Variable Equal to Something:

- [Let \(Equals\)](#)
- [Let \(ClipBoard\)](#)
- [Let \(File\)](#)
- [Let \(Now\)](#)

Changing the Case:

- [Let \(LCase\)](#)
- [Let \(UCase\)](#)

Changing the Value:

- [Let \(Format\)](#)
- [Let \(Math\)](#)
- [Let \(Number\)](#)
- [Let \(Replace\)](#)

Removing/Trimming Spaces:

- [Let \(LTrim\)](#)
- [Let \(RTrim\)](#)
- [Let \(Trim\)](#)

Parsing Strings:

- [Let \(Left\)](#)
- [Let \(Right\)](#)
- [Let \(Mid\)](#)
- [Let \(Remove\)](#)

Concatenating/Merging:

- [Let \(Merge\)](#)

SEE ALSO:

The [IF-THEN](#) command can be used to evaluate the value of variables.

Important General Notes for the Let Command

Let (Equals)

DESCRIPTION:

Set the contents of a variable equal to the contents of another variable or text.

SYNTAX:

Let {var-?} = {var-?}

or

Let {var-?} = text

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-?} = {Macro Mania saves a lot of time!}

NOTES:

The special brackets ({}) must be used when identifying variables.

The special brackets ({}) are optional when using text, but are needed if the text is supposed to have leading or trailing spaces (otherwise the spaces are automatically trimmed off). The special brackets are also necessary if, by some chance, you wanted to use any of the special cases, such as letting a variable equal the word *Clipboard* or the word *NOW*, not the special meaning it has with the Let (Clipboard) or Let (Now) sub-command. Thus, Let {var-?} = Clipboard is not the same as Let {var-?} = {Clipboard}. The first example will make it equal the *word* clipboard, the second example will make it equal the contents of whatever text is in the Windows clipboard.

SEE ALSO:

Important General Notes for the Let Command.

Let (ClipBoard)

DESCRIPTION:

Sets the contents of a variable equal to whatever text is in the Windows clipboard

SYNTAX:

Let {var-?} = CLIPBOARD

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-5} = CLIPBOARD

SEE ALSO:

The various [clipboard commands](#) for ways to set the contents of the clipboard.

Important General Notes for the Let Command.

Let (File)

DESCRIPTION:

Sets the contents of a variable equal to the contents of a file.

SYNTAX:

Let {var-?} = FILE [FileName]

where *FileName* is the path and name of the file to read.

EXAMPLE:

Let {var-f} = FILE c:\windows\readme.txt

NOTES:

If the file is not found, the contents is set to null. Use the *ExistFile* condition of the [IF-THEN](#) command if you need to check for the existence of the file first.

The length of the file can not exceed 65,535 bytes.

SEE ALSO:

Important General Notes for the Let Command.

Let (Now)

DESCRIPTION:

Sets a variable to the value of the short date and time formats set in the Windows *Control Panel*.

SYNTAX:

Let {var-?} = Now

where ? is any letter from a to z or any number from 0 to 99

EXAMPLE:

Let {var-d} = Now

NOTES:

You can use the [Let \(Format\)](#) command to extract certain parts from the date and/or time after you have set a variable equal to the current date/time with the *Let (Now)* command.

SEE ALSO:

[Important General Notes for the Let Command.](#)

Let (Format)

DESCRIPTION:

Formats a variable according to instructions contained in a format expression.

SYNTAX:

Let {*var-?*} = Format {*var-?*} *fmt*

where *?* is any letter from a to z or any number from 0 to 99 and *fmt* is any of the words in **bold** below (descriptions follow). Note: Examples are at the end of this Help Section and may make the format command more clear.

DATES:

General Date - Display a date and/or time. For real numbers, display a date and time. (e.g. 4/3/93 05:34 PM); If there is no fractional part, display only a date (e.g. 4/3/93); if there is no integer part, display time only (e.g. 05:34 PM).

Long Date - Display a Long Date, as defined in the International section of the Control Panel.

Medium Date - Display a date in the same form as the Short Date, as defined in the International section of the Control Panel, except spell out the month abbreviation.

Short Date - Display a Short Date, as defined in the International section of the Control Panel.

Long Time - Display a Long Time, as defined in the International section of the Control Panel. Long Time includes hours, minutes, seconds.

Medium Time - Display time in 12-hour format using hours and minutes and the AM/PM designator.

Short Time - Display a time using the 24-hour format (e.g. 17:45)

NUMBERS:

General Number - Display the number as is, with no thousand separators.

Currency - Display number with thousand separator, if appropriate; display negative numbers enclosed in parentheses; display two digits to the right of the decimal separator.

Fixed - Display at least one digit to the left and two digits to the right of the decimal separator.

Standard - Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator.

Percent - Display number multiplied by 100 with a percent sign (%) appended to the right; display two digits to the right of the decimal separator.

Scientific - Use standard scientific notation.

Yes/No - Display No if number is 0, otherwise display Yes.

True/False - Display False if number is 0, otherwise display True.

On/Off - Display Off if number is 0, otherwise display On.

The following table shows the characters you can use to create user-defined date/time formats and the meaning of each:

c	Send the date as dddd and send the time as t t t t t, in that order. Only date information is sent if there is no fractional part to the date serial number; only time information is sent if there is no integer portion.
d	Send the day as a number without a leading zero (1-31).
dd	Send the day as a number with a leading zero (01-31).
ddd	Send the day as an abbreviation (Sun-Sat).
dddd	Send the day as a full name (Sunday-Saturday).
dddddd	Send a date serial number as a complete date (including day, month, and year) formatted according to the Short Date setting in the International section of the Windows Control Panel. The default Short Date format is m/d/yy.
ddddddd	Send a date serial number as a complete date (including day, month, and year) formatted according to the Long Date setting in

the International section of the Control Panel. The default Long Date format is mmmm dd, yyyy.

w Send the day of the week as a number (1 for Sunday through 7 for Saturday.)

ww Send the week of the year as a number (1-53).

m Send the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is sent.

mm Send the month as a number with a leading zero (01-12). If m immediately follows h or hh, the minute rather than the month is sent.

mmm Send the month as an abbreviation (Jan-Dec).

mmmm Send the month as a full month name (January-December).

q Send the quarter of the year as a number (1-4).

y Send the day of the year as a number (1-366).

yy Send the year as a two-digit number (00-99).

yyyy Send the year as a four-digit number (100-9999).

h Send the hour as a number without leading zeros (0-23).

hh Send the hour as a number with leading zeros (00-23).

n Send the minute as a number without leading zeros (0-59).

nn Send the minute as a number with leading zeros (00-59).

s Send the second as a number without leading zeros (0-59).

ss Send the second as a number with leading zeros (00-59).

t t t t t Send a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is sent if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.

AM/PM Use the 12-hour clock and send an uppercase AM with any hour before noon; send an uppercase PM with any hour between noon and 11:59 PM.

am/pm Use the 12-hour clock and send a lowercase AM with any hour before noon; send a lowercase PM with any hour between noon and 11:59 PM.

A/P Use the 12-hour clock and send an uppercase A with any hour before noon; send an uppercase P with any hour between noon and 11:59 PM.

a/p Use the 12-hour clock and send a lowercase A with any hour before noon; send a lowercase P with any hour between noon and 11:59 PM.

AMPM Use the 12-hour clock and send the contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; send the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string sent matches the string as it exists in the WIN.INI file. The default format is AM/PM.

EXAMPLES:

NUMBER FORMATS:

Format	What is sent if the value is 1257.1582
--------	---

General Number	1257.158
Currency	\$1,257.16
Percent	125715.80%
Fixed	1257.16
Standard	1,257.16
Scientific	1.26E+03
Yes/No	Yes
True/False	True
On/Off	On

DATE AND TIME FORMATS:

Format	What is sent if the current day is December 9, 1995 and the time is 8:50 pm.
--------	--

m/d/yy	12/9/15
m/d/yyyy	12/9/2015
d-mmmm-yy	9-December-15
d-mmmm	9 December
mmmm-yy	December 15
hh:mm AM/PM	08:50 PM
h:mm:ss a/p	8:50:35 p
h:mm	20:50
h:mm:ss	20:50:35
m/d/yy h:mm	12/9/15 20:50

SEE ALSO:

[Let \(Now\)](#) sets a variable to the current time/date where you could then use the *Let (Format)* command to extract the date or time parts you want.

[Important General Notes for the Let Command.](#)

Let (LCASE)

DESCRIPTION:

Changes the case of all characters in a string to lower case (no capital letters).

SYNTAX:

Let {var-?} = LCASE {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-8} = LCASE {var-8}

NOTES:

You can let another variable equal the new value, or just manipulate the contents of the same variable as shown in the example above.

SEE ALSO:

Important General Notes for the Let Command.

Let (UCase)

DESCRIPTION:

Changes the case of all characters in a string to upper case (all capital letters).

SYNTAX:

Let {var-?} = UCASE {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-8} = UCASE {var-8}

NOTES:

You can let another variable equal the new value, or just manipulate the contents of the same variable as shown in the example above.

SEE ALSO:

Important General Notes for the Let Command.

Let (LTrim)

DESCRIPTION:

Trims off any extra spaces on the left side of a variable.

SYNTAX:

Let {var-?} = LTRIM {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-8} = LTRIM {var-8}

NOTES:

You can let another variable equal the new value, or just manipulate the contents of the same variable as shown in the example above.

SEE ALSO:

Important General Notes for the Let Command.

Let (RTrim)

DESCRIPTION:

Trims off any extra spaces on the right side of a variable.

SYNTAX:

Let {var-?} = RTRIM {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-8} = RTRIM {var-8}

NOTES:

You can let another variable equal the new value, or just manipulate the contents of the same variable as shown in the example above.

SEE ALSO:

Important General Notes for the Let Command.

Let (Left)

DESCRIPTION:

Returns part of a string from the left side of a variable and the number of characters specified.

SYNTAX:

Let {var-?} = LEFT {var-?} n

where ? is any letter from a to z or any number from 0 to 99, and n is the number of characters to read beginning at the far left side of a string.

EXAMPLE:

Let {var-t} = LEFT {var-c} 145

NOTES:

If n exceeds the length of the contents in a variable, all characters are returned.

SEE ALSO:

Important General Notes for the Let Command.

Let (Trim)

DESCRIPTION:

Trims off any extra spaces on both sides of a variable.

SYNTAX:

Let {var-?} = TRIM {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-32} = TRIM {var-32}

NOTES:

You can let another variable equal the new value, or just manipulate the contents of the same variable as shown in the example above.

SEE ALSO:

Important General Notes for the Let Command.

Let (Right)

DESCRIPTION:

Returns part of a string from the right side of a variable and the number of characters specified.

SYNTAX:

Let {var-?} = RIGHT {var-?} n

where ? is any letter from a to z or any number from 0 to 99, and n is the number of characters to read beginning at the far right side of a string.

EXAMPLE:

Let {var-c} = RIGHT {var-c} 75

NOTES:

If n exceeds the length of the contents in a variable, all characters are returned.

SEE ALSO:

Important General Notes for the Let Command.

Let (Math)

DESCRIPTION:

Adds, subtracts, multiplies, or divides the values of variables.

SYNTAX:

Let {var-?} = MATH {var-?} <operator> {var-?}

where ? is any letter from a to z or any number from 0 to 99, and operator is either a plus (+), minus (-), multiplication (*), or division (/) sign.

EXAMPLE:

Let {var-g} = MATH {var-19} * {var-33}

NOTES:

Only two variables can be used at a time. Use the Let (Math) command multiple times if you need to perform several calculations to set a variable. This command is useful if you want to [add a counter](#) to a macro or convert minutes to seconds for the pause command, etc, etc.

SEE ALSO:

Important General Notes for the Let Command.

Let (Mid)

DESCRIPTION:

Returns part of a string from a variable from any given starting point and optionally reads a certain number of characters.

SYNTAX:

Let {var-?} = MID {var-?} start [length]

where ? is any letter from a to z or any number from 0 to 99, and start is the starting position and [length] is the optional number of characters to read.

EXAMPLE:

Let {var-g} = MID {var-c} 3 10

NOTES:

If [length] is omitted or exceeds the number of possible characters to return, all characters beginning from the *start* position are returned. The start value should be equal or greater than 1, if it exceeds the length of the contents in a variable, no characters are returned.

SEE ALSO:

Important General Notes for the Let Command.

Let (Remove)

DESCRIPTION:

Removes a specified number of characters from the *right* side of a string

SYNTAX:

Let {var-?} = REMOVE {var-?} n

where ? is any letter from a to z or any number from 0 to 99, and n is the number of characters to remove.

EXAMPLE:

Let {var-d} = REMOVE {var-c} 10

NOTES:

If n exceeds the number of possible characters to remove, all characters are removed. You can use the [Let \(Mid\)](#) sub-command to remove characters from the left side of a string.

SEE ALSO:

Important General Notes for the Let Command.

Let (Merge)

DESCRIPTION:

Concatenates the values of 2 or more variables into a single variable string.

SYNTAX:

Let {var-?} = MERGE {var-?} + {var-?} + {var-?} +...

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-g} = MERGE {var-44} + {var-45} + {var-46}

NOTES:

This sub-command is especially useful if you want to write several variables to a single line with the [WriteFile](#) command, since the WriteFile command will automatically add a Carriage Return, Line Feed when writing a file.

SEE ALSO:

Important General Notes for the Let Command.

Let (Number)

DESCRIPTION:

Returns only the numbers in a string.

SYNTAX:

Let {var-?} = NUMBER {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-c} = NUMBER {var-c}

NOTES:

This sub-command is useful if you wish to use the [If-Then](#) command to compare a numeric string with another numeric string and want, for example, 1,234 = 1234.

SEE ALSO:

The [Let \(Format\)](#) command can also change variables to predetermined numeric values.

Important General Notes for the Let Command.

Let (Replace)

DESCRIPTION:

Replaces text with other text in a variable.

SYNTAX:

Let {var-?} = REPLACE {var-?} WITH {var-?} IN {var-?}

where ? is any letter from a to z or any number from 0 to 99.

EXAMPLE:

Let {var-a} = {My dog is brown}

Let {var-b} = {my}

Let {var-c} = {Our}

Let {var-d} = REPLACE {var-b} WITH {var-c} IN {var-a}

Msg-OK {var-d}

NOTES:

The keywords LET, REPLACE, WITH, and IN are necessary for correct syntax.

To simplify the process, the string being searched for is not case sensitive. The string it is being replaced with, however, will be exactly as it is in the variable (it will retain its case.)

SEE ALSO:

Important General Notes for the Let Command.

Important General Notes for the Let Command

Be sure to use the special brackets ({}) to identify the variable. This also applies to when variables are used in other commands. For example, **Send {var-32}** or **Msg-OK {var-h}**.

Use a space on each side of the equals (=) sign when assigning values to a variable. This is necessary for the syntax to work correctly and makes the macro easier to read. Thus, you should use **Let {var-a} = Hello** rather than **Let {var-a}=Hello**.

Variables must follow the format presented in the syntax (**{var-?}**), where ? is any letter from a to z or number from 0 to 99. The a-z part is not case sensitive and uses the 26 letters of the U.S. alphabet. Thus, 126 variables are available.

Except for the *Let (Equals)* sub-command, all the other sub-commands use the text within a variable, not text itself. If needed, first put text into a variable with the *Let (Equals)* sub-command to later use it with any of the other sub-commands.

When using variables with other commands, you can not use multiple variables in the same command. However, the **Let (Merge)** command will enable you to first concatenate the values of multiple variables into a single variable.

The contents of the variables are kept in memory as long as Macro Mania is running and are carried across macros and even across branches and other macro files when opening new macro files. If you need to clear the contents of a variable (or want to be sure it is clear), let the contents equal blank (e.g. **Let {var-a} = { }**).

File Commands

- [Copy](#)
- [Delete](#)
- [MkDir](#)
- [Rename](#)
- [RmDir](#)
- [WriteFile](#)

Related File Commands:

- [Let \(File\)](#)
- [Clipboard-SetFile](#)
- [IF ExistFile... THEN](#)

Copy

DESCRIPTION:

Copies one or more files.

SYNTAX:

`Copy` [/options] [SourceFiles] **To** [Destination]

where SourceFiles is the full path and file specification (similar to the DOS copy command, (wildcards such as *, ?, and ~? are permitted.), and Destination is the full path to the subdirectory where SourceFiles should be copied to. SourceFiles and/or Destination may also be [variables](#) see example 2 below).

/options may be any one or more of the following optional parameters:

/NORECYCLE	Do not put any overwritten files in the recycle bin
/RECURSE	Recurse folders if *.* is used (like <i>xcopy /s</i> in DOS)
/RENAME	Rename destination files if any by the same name already exist.
/SILENT	Do not show any status dialogue
/SIMPLE	Do not show files in the status dialogue
/Y	Do not prompt to overwrite files
/YMKDIR	Do not prompt if a folder/directory needs to be made for the destination

EXAMPLES:

`Copy` c:\dir1*.txt **To** c:\dir2
`Copy` c:\dir\test.txt **To** {var-8}
`Copy` /Y c:\dir1 **To** c:\dir2
`Copy` /recurse /y c:\dir1 **To** c:\dir2
`Copy` c:\dir1*. * **To** c:\dir2\dir3

NOTES:

Remember to include the **TO** part of the command. (It is a required part of the command and it helps break up the command visually, especially when a file/directory has spaces in its name.)

The options may come in any order and are not case sensitive. However, when using any of the options, be sure to immediately (no spaces) precede the option with the required slash / (as shown in the examples above).

If you need to check for the existence of a file, use the *ExistFile* condition of the [If-Then](#) command.

Macro Mania does not have a *Move* command since you can easily make a macro that uses both the *Copy* command and the [Delete](#) command (or you can use the [Rename](#) command to move files on the same drive).

Delete

DESCRIPTION:

Deletes a file.

SYNTAX:

Delete </options> [file/folder]

where *file/folder* is the full path and file name for a file or folder that you want to delete. The *file* may also be a *variable* following these rules (see example 2 below).

</options> may be any one or both of the following optional parameters:

/NORECYCLE	Do not put deleted files in the recycle bin
/CONFIRM	Confirm when deleting

EXAMPLE:

Delete c:\test.txt

Delete /NoRecycle c:\test.txt

Delete {var-f}

Delete /Confirm /Norecycle c:\dir1*.gif

Delete /confirm C:\FolderXYZ

NOTES:

CAUTION: This command DELETES the files/folders specified. Be sure to make backups of any files if needed.

The options may come in any order and are not case sensitive. However, when using any of the options, be sure to immediately (no spaces) precede the option with the required slash / (as shown in the examples above).

If you need to check for the existence of a file, use the *ExistFile* condition of the *If-Then* command.

Leave off any file names to delete a folder (IMPORTANT: That will delete all folders and files below it like the DOS DelTree command and should be used with care).

Rename

DESCRIPTION:

Changes the name of either a file or directory.

SYNTAX:

Rename OldName As NewName

where *OldName* and *NewName* are the old name and new name of the file or directory, respectively. The *OldName* and/or *NewName* may also be [variables](#) following these rules (see example 2 below).

EXAMPLE:

Rename c:\tmp\test.txt As c:\tmp\test.doc

Rename c:\tmp\test.txt As {var-b}

NOTES:

Remember to include the AS part of the command. (It is a required part of the command and it helps break up the command visually, especially when a file/directory has spaces in its name.)

The Name statement is similar to the operating system RENAME command, but it can also be used to change the name of an empty *directory* (an error occurs if the directory being renamed is not empty). For example, to rename the directory c:\tiger to c:\lion, use RENAME c:\tiger AS c:\lion. Also, Rename can move a file from one directory to another or a directory (on the same drive).

The arguments *OldName* and *NewName* should each contain a file name and an optional path (or just path if renaming an empty directory). If the path in *NewName* exists and is different from the path in *OldName*, the command moves the file to the new directory and renames the file if necessary. If *NewName* and *OldName* have different paths and the same file name, it moves the file to the new directory and leaves the file name unchanged.

If *OldName* is not found, an error occurs and you are given the option to Abort the macro, Retry the copy operation, or Ignore the error and continue with the rest of the macro. Use the *ExistFile* condition in the [If-Then](#) command if you want to avoid this error message entirely.

If *NewName* already exists as a file, an error occurs. Use the *ExistFile* condition in the [If-Then](#) command and/or [Delete](#) if you want to avoid this error.

Both *NewName* and *OldName* must be on the same drive.

Using Name on a file currently open produces an error. You must close an open file before renaming it.

If you need to check for the existence of a file, use the *ExistFile* condition of the [If-Then](#) command.

Rmdir

DESCRIPTION:

Like the DOS command, this will remove a directory you specify.

SYNTAX:

RMDIR [path]

path is a string expression that identifies which directory you wish to remove and should include the drive specification as part of the path. If you omit the drive, Rmdir will search the current drive for the directory. The *path* parameter may also be a variable following these rules (see example 2 below).

EXAMPLE:

RMDIR c:\test\dir1

RMDIR {var-p}

NOTES:

RMDIR may be abbreviated with *RD*.

Like the DOS RD command, you must delete/remove all files and subdirectories in the specified directory before you can remove it. The Delete command also removes directories, but it can also delete files in the directories automatically.

MkDir

DESCRIPTION:

Like the DOS command, this will make a directory you specify.

SYNTAX:

MKDIR [path]

path is a string expression that identifies which directory you wish to make and should include the drive specification as part of the path. If you omit the drive, Mkdir uses the current drive. The *path* parameter may also be a [variable](#) following these rules (see example 2 below).

EXAMPLES:

MKDIR c:\test\dir1

MKDIR {var-p}

NOTES:

MKDIR may be abbreviated with *MD*.

This command is also capable of creating several nested directories at once. For example, even if c:\test does not exist, if you issue **MD c:\test\dir1**, the command will create c:\test and then c:\dir1.

The Mkdir command does not currently support long file names (at least entirely). If you specify a directory name of 9 or more characters, it will truncate it to the first 8 characters. For example, if you issue **Mkdir c:\thisismydirectory** it will create a directory called *c:\thismydi*.

WriteFile

DESCRIPTION:

Writes/Appends the contents of text or a variable to a file.

SYNTAX:

WRITEFILE [file] {text}

where *file* is the full path and file name for a file that you want to write to, {var-?} is any valid variable you have set using the [Let](#) command, and {text} is any text you wish. The *file* and/or *text* may also be a [variable](#) following these rules (see example 2 below).

EXAMPLES:

WRITEFILE c:\test.txt {Manhattan, KS 66503}

WRITEFILE c:\test.txt {var-t}

NOTES:

WriteFile will automatically create [file] if it does not already exist, or if it exists it will append directly to it. If you need to make sure the file is new, you may delete it first using the [Delete](#) command.

WriteFile automatically adds a Carriage Return, Line Feed (CRLF) to the end of the string it writes to the file. To write one contiguous string on the same line, use the MERGE sub-command of the [Let](#) command to first concatenate the string before writing it to the file.

If the *File* to be written to is on drive a or b, then a status window will display during the operation (since the operation is likely to take a couple seconds longer when writing to a diskette).

If you need to check for the existence of a file, use the *ExistFile* condition of the [If-Then](#) command.

License Agreement

MACRO MANIA - PRODUCT LICENSE INFORMATION

NOTICE TO USERS: CAREFULLY READ THE FOLLOWING LEGAL AGREEMENT. USE OF THE SOFTWARE PROVIDED WITH THIS AGREEMENT (THE "SOFTWARE") CONSTITUTES YOUR ACCEPTANCE OF THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT INSTALL AND/OR USE THIS SOFTWARE. USER'S USE OF THIS SOFTWARE IS CONDITIONED UPON COMPLIANCE BY USER WITH THE TERMS OF THIS AGREEMENT.

1. LICENSE GRANT. NorthStar Solutions grants you a license to use one copy of the version of this SOFTWARE on any one system for as many licenses as you purchase. "You" means the company, entity or individual whose funds are used to pay the license fee. "Use" means storing, loading, installing, executing or displaying the SOFTWARE. You may not modify the SOFTWARE or disable any licensing or control features of the SOFTWARE except as an intended part of the SOFTWARE's programming features. When you first obtain a copy of the SOFTWARE, you are granted an evaluation period of not more than 30 days, after which time you must pay for the SOFTWARE according to the terms and prices discussed in the SOFTWARE's documentation, or you must remove the SOFTWARE from your system. This license is not transferrable to any other system, or to another organization or individual. You are expected to use the SOFTWARE on your system and to thoroughly evaluate its usefulness and functionality before making a purchase. This "try before you buy" approach is the ultimate guarantee that the SOFTWARE will perform to your satisfaction; therefore, you understand and agree that there is no refund policy for any purchase of the SOFTWARE.

2. OWNERSHIP. The SOFTWARE is owned and copyrighted by NorthStar Solutions. Your license confers no title or ownership in the SOFTWARE and should not be construed as a sale of any right in the SOFTWARE.

3. COPYRIGHT. The SOFTWARE is protected by United States copyright law and international treaty provisions. You acknowledge that no title to the intellectual property in the SOFTWARE is transferred to you. You further acknowledge that title and full ownership rights to the SOFTWARE will remain the exclusive property of NorthStar Solutions and you will not acquire any rights to the SOFTWARE except as expressly set forth in this license. You agree that any copies of the SOFTWARE will contain the same proprietary notices which appear on and in the SOFTWARE.

4. REVERSE ENGINEERING. You agree that you will not attempt to reverse compile, modify, translate, or disassemble the SOFTWARE in whole or in part.

5. NO OTHER WARRANTIES. NORTHSTAR SOLUTIONS DOES NOT WARRANT THAT THE SOFTWARE IS ERROR FREE. NORTHSTAR SOLUTIONS DISCLAIMS ALL OTHER WARRANTIES WITH RESPECT TO THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES OR LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY MAY LAST, OR THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION.

6. SEVERABILITY. In the event of invalidity of any provision of this license, the parties agree that such invalidity shall not affect the validity of the remaining portions of this license.

7. NO LIABILITY FOR CONSEQUENTIAL DAMAGES. IN NO EVENT SHALL NORTHSTAR SOLUTIONS OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, SPECIAL, INCIDENTAL OR INDIRECT DAMAGES OF ANY KIND ARISING OUT OF THE DELIVERY, PERFORMANCE OR USE OF THE SOFTWARE, EVEN IF NORTHSTAR SOLUTIONS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL NORTHSTAR SOLUTIONS' LIABILITY FOR ANY CLAIM, WHETHER IN CONTRACT, TORT OR ANY OTHER

THEORY OF LIABILITY, EXCEED THE LICENSE FEE PAID BY YOU, IF ANY.

8. GOVERNING LAW. This license will be governed by the laws of the State of Kansas as they are applied to agreements between Kansas residents entered into and to be performed entirely within Kansas. The United Nations Convention on Contracts for the International Sale of Goods is specifically disclaimed.

9. ENTIRE AGREEMENT. This is the entire agreement between you and NorthStar Solutions which supersedes any prior agreement or understanding, whether written or oral, relating to the subject matter of this license.

Making Sure Keystrokes Get Received

There are generally four basic (and easily fixable) reasons a program might not be accepting keystrokes from a macro:

- 1) The window that you want to get the keystrokes has not been given the focus with either the [Activate](#) or [Run](#) command.
- 2) Step 1 above has been done, but then focus has been put onto another window: you manually set focus to another window while the macro stopped or paused, or another macro command such as Msg-Ok, GetInput, etc., put focus on Mania Mania). You should always be sure focus is on the window you want to get the keystrokes by first using a [Run](#) or [Activate](#) command before using the Send command.
- 3) The program is running minimized (see [Manipulating the State of Windows](#)).
- 4) The program getting the keystrokes needs time to catch up with Macro Mania....

Sometimes it is necessary to put in a short [Pause](#) command between an Activate or Run command and a subsequent [Send](#) command to give the program receiving keystrokes time to get ready. Although most of the time adding a Pause is not necessary, if you experience that a program is not getting keystrokes (and you are sure you have addressed the issues noted above), then it may be necessary to add a small delay (sometimes one second is all it takes) before using the Send command. An example being worth a thousand words:

```
Run myapp.exe
Pause 1
Send {down 3}{home}+{end}%Ec
```

Although most applications are smart enough to start getting the keystrokes from the keyboard buffer after they launch, some seem to ignore the keyboard buffer and thus never receive what the macro has sent. On the same topic, it is often better to break up long Send commands into several smaller ones. This is effective because Macro Mania will not begin processing the next line until the previous keystrokes have been sent and accepted by the receiving program. The following example shows what we mean:

```
Send abcdefg^G{tab}{up 4}qrstuv
```

is changed to

```
Send abcdefg
Send ^G
Send {tab}{up 4}
Send rstuv
```

See Also:

[Compatibility](#)

Hotkeys

In addition to being able to [launch macros](#) by selecting a button from Macro Mania, you can use a *Hotkeys*. Hotkeys enable you to quickly invoke a macro by enabling you to just press the key combination you assign. For example, if you assign Alt-A, you can then just press Alt-A and the macro will be executed without you having to highlight it with the mouse or tab to it via the keyboard.

Hotkeys can be assigned to a macro by choosing *Select Hotkey* from the Macro Editor. Local hotkeys are invoked by using an Alt-[hotkey] combination. For global hotkeys, you can assign any of the special keys (Alt, Ctrl, Shift, or the Windows 95 *WinKey* plus any 1 function key, a number, or any letter A-Z.)

[Local Hotkeys](#) will be shown underlined on the button display and require that Macro Mania be the program with the focus. In other words, with a local hotkey you simply need to switch over to Macro Mania (Alt-Tab) and then use the Alt-[hotkey] combination to invoke the macro. This allows you to review the macros a bit before selecting them if needed, and it reserves the global hotkeys for times when they are more appropriate.

[Global Hotkeys](#) will be shown on the button display with the hotkey character and an abbreviated character as follows: a - Alt | c - Ctrl | s - Shift | w - Winkey. You may also remind yourself of the corresponding hotkey assigned to a button by holding the mouse over the key for a moment to have a tiny tooltip window appear. Unlike local hotkeys, global hotkeys are not underlined, nor do they require that Macro Mania have the focus (thus, they are global to all Windows programs). In other words, you do not have to first switch over to Macro Mania to launch a macro that has been assigned a *global* hotkey. Global Hotkeys are especially useful if you have a certain text you always type. Examples include *signature* text you might send to add your name, title, etc. while composing e-mail in an e-mail program and/or a letter in a word processor. When invoking hotkeys to send text, often just a couple Send commands is all you need for that macro to work effectively! Global hotkeys enable you to create more generic macros that work with more than one application if all you are doing is sending text to the current program you are in when the hotkey is invoked; and, of course, they give you the ability to quickly launch macros without switching over to Macro Mania first.

NOTES: Some hotkeys may conflict with hotkeys of other programs, so be careful when assigning them.

Function Keys: Can only be assigned to global hotkeys.

Global Hotkey Example:

If you just want the macro to do is send text to the current program you are in, all you need to use is the Send command to send the text. For example, suppose you are in your favorite word processor and you want to send a certain set of text to it each time you press a hotkey (without having to switch over to Macro Mania first). The following macro is all that is needed (provided you are already in the program that is supposed to get the keystrokes):

```
Send Sincerely,{enter 4}  
Send John Hancock, President{enter}  
Send Continental Congress
```

NOTE: Global hotkeys are the one exception where you do not have to tell Macro Mania where to send the keystrokes because with a global hotkey, you can (and should if you are using the Send Command) be in another program to invoke them. Thus, if you invoke a global hotkey from another application and focus has not explicitly been put to another program with either the Activate or Run command, any keystrokes that get sent will be sent to the program that currently has the focus.

Compatibility

Macro Mania is used by literally thousands of people that have found it an extremely powerful tool with a wide variety of uses. With that in mind, it is not unreasonable to assume that it has been used on hundreds, possibly thousands, of different programs and environments. We have tested and verified Macro Mania is compatible with any Windows program that follows basic Windows conventions. In fact, we have not found a single Windows program it does not work with. However, here are some notes on a couple popular programs worth mentioning:

Microsoft Word * - invoking the hotkeys within Microsoft Word is at least sometimes case sensitive. For example, the Format pull-down menu in Word is invoked by sending Alt-o. However, sending Alt-O (translated as Send %O in Macro Mania syntax) does not have the same results as Alt-o (Send %o). We have not seen this peculiarity in any other program, but it is probably best to recognize that case may be important when invoking hotkeys within a program, especially when using *Microsoft Word*.

America Online for Windows ** - it appears the development team at AOL has modularized the program to a large extent (probably to help accomodate them easily updating certain components on your system quickly while you are online) and, therefore, the program acts a little differently than any other program we have ever encountered. For example, we have found that in most programs, if a child window in a program is already active for that program, it is only necessary to activate the main program -- then that window which already has the focus will be ready to accept keystrokes sent from a macro. However, that is not always the case with *America Online for Windows* software. In fact, an odd thing is you can sometimes activate a child window directly with the Activate command (as opposed to having activate the main parent window to activate the child window) almost as if it were a separate program itself. In general, Macro Mania will send keystrokes to *America Online for Windows*, but you may need to experiment with this to ensure the keystrokes are going to the window you want them to go to. (This seems to be especially true for the child Windows that pop up and ask for passwords, etc.)

While on the subject, we have found that *America Online for Windows*, unlike any of the many other programs tested, is not very friendly in regards to navigating it from the keyboard. It seems their developers want everyone to be slaves to the little rodents attached to the PC -- to the detriment of someone wanting to create a fast and easy macro, or someone that simply wishes to navigate without having to lift their hands from the keyboard just to invoke a simple function (which is especially true if you are using a laptop where there may be a limited pointing device available or for someone that would prefer to navigate quickly via the keyboard because they are adept at doing so). Anyway, just a warning that you may wish to start using Macro Mania with another program before you tackle a more awkward one like *America Online for Windows*.

Other than the peculiarities with those two programs, we have not heard of or seen any other idiosyncrasies (again, out of a large number of programs we have tested Macro Mania with and with many thousands of people using Macro Mania). Still, hopefully mentioning these here will save us from having to explain them in a technical support dialogue.

* *Microsoft Word* is a registered trademark of Microsoft Corp.

** *America Online for Windows* is a registered trademark of America Online, Inc.

Adding a Counter to a Repeating Macro

The following macro examples shows how you can use the [Let \(Math\)](#) command to easily increment a counter within a repeating macro (using [variables](#)) that can then be used as part of another variable:

```
Repeat 4  
Let {var-1} = {1}  
Let {var-a} = MATH {var-a} + {var-1}  
Let {var-b} = {file}  
Let {var-c} = {.txt}  
Let {var-d} = MERGE {var-b} + {var-a} + {var-c}  
Msg-OK {var-d}
```

(Variables)

Variables are set and manipulated using the Let commands. Variables extend the power of macros considerably and permit you to "program" sophisticated macros that dynamically change the data you are working with and/or perform certain functions in a macro based on the values of variables. See the Let command for more details about how to use variables in your macros.

Year 2000 Compliance Statement

The following statement was issued 1/15/1998 and now that we have passed two major dates the software industry has flagged as having potential problems (1/1/2000 and 2/29/2000), we are as confident as ever that no problems exist concerning Macro Mania and date issues. Not one single incident or problem has been reported to us (despite literally thousands of Macro Mania users worldwide).

Assuming the underlying operating system and hardware components are year-2000 compliant, Macro Mania versions 6.0 and above (see the 3rd paragraph below for versions prior to 6.0) are year-2000 compliant. That is, Macro Mania versions 6.0 and above will run without any errors on systems where the date may be in the year 2000 or beyond.

The only issue of real concern for Macro Mania is its scheduling feature, which uses the format of mmddyyyy. You can easily test this compliance -- you will see the macro will launch as expected even if the date is in the year 2000 or beyond.

Versions of Macro Mania prior to v6.0 will also run fine on systems beyond the year 2000; however, the only nuance is that the scheduling feature stored the date as mmddyy. All that really means is that if there was a macro scheduled to run on a specific calendar date and that particular macro was not updated or deleted within the next 100 years, it would run again in another 100 years -- so obviously there is no practical concern even if, by some really odd chance, one never upgraded Macro Mania to version 6.0 or above for 100 years.

If your organization requires an official Year 2000 Compliance Statement beyond the one included here, please send a self-addressed, stamped envelope to the following address and we will be glad to provide one:

NorthStar Solutions
1228 WestLoop Pl, #204
Manhattan, KS 66502

Upgrading from a Version Prior to 6.0

NOTE: If you already have version 6.0 or later, this section is really of no interest to you.

Beginning with version 6.0, Macro Mania is written for a 32-bit Operating System (Windows 95 or above). The potential capabilities that the 32-bit environment opens up for future versions of Macro Mania are exciting. The main goal with the first 32-bit version of Macro Mania, version 6.0, was to make it a 32-bit version while keeping all the previous features available and ensuring backward compatibility with any macros created with previous versions. This goal was accomplished, however, a few things have changed ...

The Run Command:

Macro Mania now gives you more control over your macros in that the way the Run command works has changed. The Run command used to search all the running programs to see if the program was running already, then it would pull up that instance of the program if it found one already running. However, because of the way the 32-bit Windows Operating System works, the Run Command now just behaves as if you issued a run statement from the Run option of the Windows Startup Menu. That is, the Run Command makes no assumptions about whether or not you want to pull up an already running program or want to run a new instance. You can, however, easily check for an already running program first with a newly added parameter for the Run command. In fact, this new parameter has other advantages too and will likely be very welcomed once you become acquainted with it. **For more details, please be sure to read the [Starting Programs](#) section in the Help System.**

Syntax Changes:

The syntax for any commands using the underscore (_) has changed to using a dash (-). For example, MSG_OK is now MSG-OK, Clipboard_SAVE is now Clipboard-SAVE, and so on. This is not a real big deal, but since we were converting the macro format a little anyway and since we always regretted using the underscore character (which takes two keystrokes to type) when the dash would do, this seemed like a good time to make this modification. Do not worry about backward compatibility, however, as any macros you have created previously will have this syntax changed automatically for you when you first load any old macro files. In fact, we have made this especially friendly in that any character (a space, a dash, the underscore, or whatever) will not matter so long as there is one character between the base part of the command and the trailing part of the command. The default (used in the documentation and the *Command Wizard*), however, is a dash since it helps to visually keep the command contiguous and yet requires only one keystroke.

For simplification purposes, The Activate2 and IF EXISTWINDOW2 commands have been eliminated because they are now combined with their base commands (Activate and IF EXISTWINDOW). It was not necessary to break them like before since you can either search for a partial match or, if you need to find the exact caption, use a longer string to be more exact.

Other Notes of Interest:

The [MinimizeAll](#) Command now works much smoother and faster and may be something you add to some of your macros if you are not already using it. (The syntax is exactly the same.)

An irregularity we noticed is the command *Run explorer.exe* no longer just activates the main Windows 9x Desktop as it used to. However, to easily accomplish the task of minimizing all the windows and putting focus on the desktop (presumably to quickly navigate to a shortcut), the MinimizeAll command will do this very nicely. If any of your macros had the command Run explorer.exe, that command will automatically be substituted with the MinimizeAll command when Macro Mania converts old macros to its new format.

A new and much improved strategy for activating a DUN session is discussed in the section [Activating a Dial-up Networking Session](#).

To stay with Windows 9x conventions, the default installation directory for Macro Mania is now `%program`

files\Macro Mania. If you had your own icon library, you may need to move it to the new *icons* subdirectory below the new default directory if any icons you had previously associated with a button were not one that came with the official distribution archive.

Syntax Conventions

The syntax conventions used in this Help System follow usual and customary conventions of software documentation. Thus...

Parameters in brackets indicate they are a *required* part of the command, but that they will change. Parameters between the less than and greater than sign indicate they are *optional* parameters that may be added to the command.

EXAMPLE:

Pause [n] <show>

n is a required parameter (in this case an integer telling the macro how many seconds to pause) and show is an optional parameter (in this case the show parameter, when present, tells the pause command to show a window with the number of seconds it will pause for). Thus, you could have any of these commands:

Pause 3
Pause 2 show
Pause 77
Pause 120 show

Note that neither the brackets nor the greater than and less than signs are used in the actual command.

Faster Keystrokes

Although Macro Mania works extremely fast when it types information (much faster than even the fastest typer), it does only send one key at a time when playing back keystrokes -- which can take a few seconds even for Macro Mania if you are sending a lot of text or your system is hard at work performing other tasks too. If you have a particularly long section of text you wish to send, using the [Clipboard Commands](#) will greatly speed the process.

For example, you could first put all the text you wish to send to the screen into the Windows clipboard with the Clipboard-Set Command, then send the appropriate keystrokes to paste the text (e.g. [Send](#) + {insert}).

The following example demonstrates a macro that first saves the current contents of the clipboard with the Clipboard-Save Command, then puts some text into the Windows clipboard, then issues the keystrokes to paste the Windows clipboard into any standard screen that accepts text, and finally restores the clipboard back to whatever it previously was (so you do not have to worry about overwriting the contents of the clipboard just to use a macro):

MINIMIZE

[Run](#) notepad.exe {Activate Untitled}

[Pause](#) 1

[REM](#) Maximize the Window, then turn on the WordWrap

[Send](#) %{ }x

[Send](#) %Ew

[REM](#) Send a few sentences and pause. Notice that as long as I do not press the ENTER key and just allow the word wrap to work, that the command continues to the next line.

Clipboard-SAVE

[Clipboard-SET](#) Notice that if you put text in the clipboard with the Clipboard-SET command and then paste it to the application, the process goes much faster than sending one key at a time. Thus, it is recommended you use that strategy if sending a lot of text.

[Send](#) %Ep

[Send](#) {ENTER 2}

[Clipboard-SET](#) Now that I have had the macro paste the above text and essentially press Enter twice, I can set another paragraph (this one) and paste it too. Really quite a simple process -- and very fast!!!

[Send](#) +{insert}

[Clipboard-RESTORE](#)

Mouse

Please note that a limitation with mouse movement is that it relies on the exact position of a window in order for it to be useful. If a window is resized or moved, the macro could become unusable (possibly even causing unwanted things to happen), and either the window has to be repositioned to exactly where it was before, or the macro has to be reprogrammed. (Macro Mania is intended to make tasks faster and easier, not vice versa!) The strength and intent of Macro Mania, therefore, is sending *keystrokes*. Since Macro Mania can send any keystroke, you should be able to reliably perform most, if not all, tasks by simply using keystrokes and are, therefore, encouraged to use keystrokes to perform tasks when possible.

However, you may occasionally find a program that just does not have a way to easily set focus to certain fields on the screen: likely the result of a short-sighted programmer except possibly in situations where the mouse is the only reasonable strategy for navigation (such as drawing programs, etc.). Thus, the following commands will help you create macros for programs that must use the mouse:

■ Mouse Move...

In addition to manipulating where the mouse pointer is located with the Mouse Move Command, you can invoke the mouse buttons using the following commands (since these commands are self-explanatory, no additional syntax or examples are provided):

- Mouse LeftClick - combines Mouse LeftDown with Mouse LeftUp
- Mouse MiddleClick - combines Mouse MiddleDown with Mouse MiddleUp
- Mouse RightClick - combines Mouse RightDown with Mouse RightUp
- Mouse LeftDown
- Mouse LeftUp
- Mouse MiddleDown
- Mouse MiddleUp
- Mouse RightDown
- Mouse RightUp

MOUSE MOVE

DESCRIPTION:

Locates the cursor to a position on the screen using pixel coordinates.

SYNTAX:

`Mouse Move xPos, yPos`

where xPos is the x coordinate on the screen and yPos is the y coordinate on the screen.

EXAMPLE:

`Mouse Move 400, 300`

NOTES:

One of the easiest ways to make a good guess about where you need to locate the cursor is to remember that the syntax uses screen pixels. For example, if you have your screen resolution set at 800 x 600 pixels, you know that the middle of your screen would be at the coordinates 400, 300. You will likely need to test your macro a few times (before issuing any click commands) to make sure you are locating the cursor to the right position. (Your screen settings are located via the Control Panel: Settings >> Control Panel >> Display >> Settings).

The following grids will give you an idea of some general coordinates depending on the screen resolution:

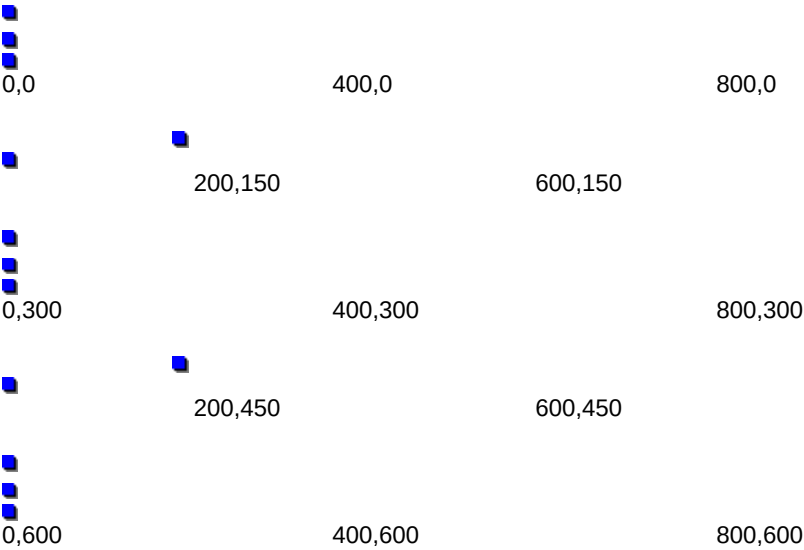
[640 x 480 pixels](#)

[800 x 600 pixels](#)

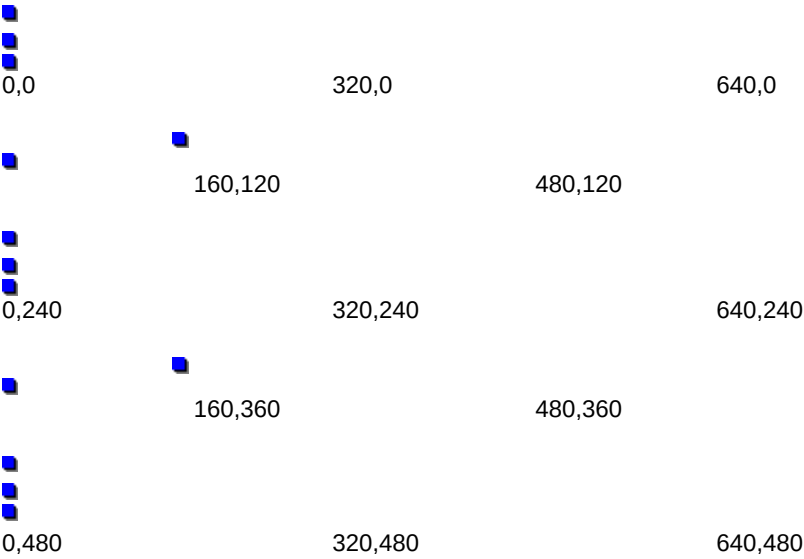
[1024 x 768 pixels](#)

[1280 x 1024 pixels](#)

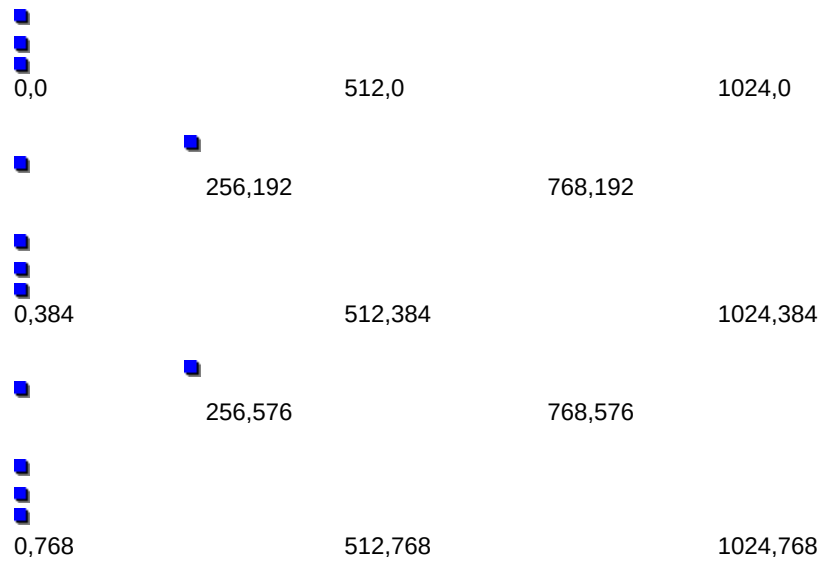
800 x 600 pixels



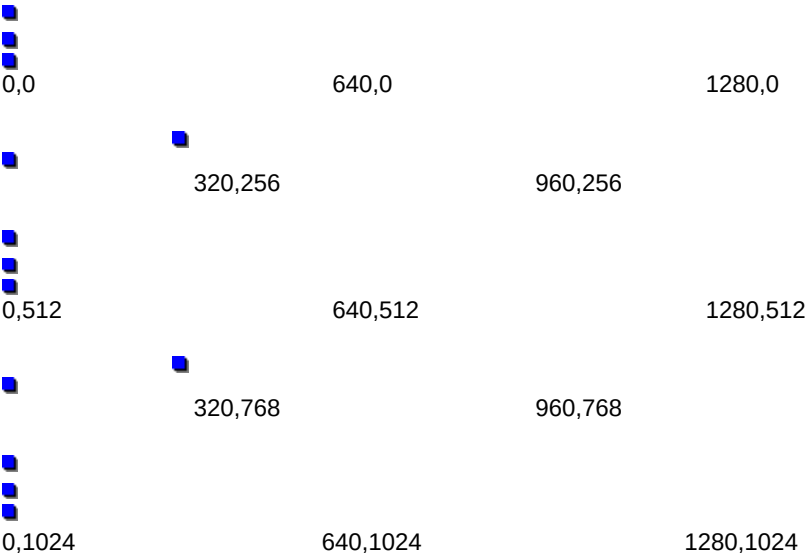
640 x 480 pixels



1024 x 768 pixels



1280 x 1024 pixels



Activating the Windows Start Button

Almost anything you need to do from the Windows *Start* Button can be done directly. For example, if you want to launch a program, then the [Run](#) command can be used. If you want to access the Windows Control Panel, there are [special ways](#) to do that too. The Windows shut down series is also available directly with a Macro Mania command ([WinExit](#)). Nonetheless, if you really need to get to the *Start* button, here is a code snippet that does the job:

```
MinimizeALL
```

```
Send {home}
```

```
Send {tab}
```

```
Send { }
```

SEE ALSO:

[Activating the Windows Desktop](#)

[Accessing the Windows Control Panel](#)

