

Macro Scheduler

Windows Automation Made Easy

© 2008 MJT Net Ltd

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Web:

<http://www.mjtnet.com/>

Email:

support@mjtnet.com

Phone:

+44 8700 684 523

+1 360 519 5383

Fax:

+44 709 214 3231

+1 866 788 7502

Table of Contents

Part I Using Macro Scheduler	2
1 Getting Started	2
2 Menu Commands	2
3 Toolbar	3
4 Creating Scripts	4
5 Using Variables	5
6 Command Locator	5
7 Code Snippets	6
8 Code Folding and Bookmarks	6
9 Scheduling Scripts	7
10 Scheduler Service	8
11 Advanced Scheduling	9
12 Trigger	10
13 AutoLogon	10
14 Logging	11
15 Encryption	11
16 Hot Keys	12
17 Groups	12
18 Recording Macros	12
19 Playing Scripts And Macros	13
20 Creating Desktop Shortcuts	14
21 Using the Debugger	14
22 Dialog Designer	15
23 Command Line Options	16
24 Help and Resources	17
 Part II Menu Commands	 20
1 New Macro	20
2 Macro Properties	20
3 Import Macros	20
4 Hide	21
5 Exit	21
6 Delete	21
7 Remove from Group	21
8 Rename	21
9 Select All	21
10 New Group	21
11 Delete Group	21
12 Group Properties	22

13	Sort	22
14	Font	22
15	Grid Lines	22
16	List	22
17	Details	22
18	Large Buttons	22
19	Toolbar	23
20	Search Bar	23
21	Create Exe	23
22	View Log File	24
23	Desktop Shortcut	24
24	View System Windows	24
25	Options	24
26	Run Now	25
27	Record	26
28	Stop	26
29	Contents	26
30	Check for Update	26
31	Support Forums	26
32	About	26
Part III Scripting Windows For Beginners		28
1	Introduction	28
2	Requirements	28
3	The Problem	28
4	Using Macro Scheduler	28
5	First Steps	29
6	Building The Script	29
	Run Notepad	31
	Wait for the window "Untitled – Notepad"	31
	Send our lines of text	32
	Press ALT-FA	33
	Wait for the window "Save As"	33
	Send our filename	33
	The Script So Far	35
	Printing the Document	36
	Closing Notepad	37
7	The Complete Script	37
8	Where To Go From Here	38
Part IV Command Reference		41
1	Complex Expressions	41
	Arithmetic Operators	41
	Logical Operators	41
	String Operators	42
	Relational Operators	42

Arithmetic Functions	42
String Functions	43
Conditional Functions	43
Examples	44
2 Clipboard Commands	44
GetClipboard	44
PutClipboard	45
WaitClipboard	45
3 Console App Funtions	45
SOWrite	45
SOWriteln	45
4 Database Commands	45
DBCclose	45
DBConnect	46
DBExec	46
DBQuery	47
5 Date/Time Commands	48
Day	48
DayOfWeek	48
GetDate	49
GetTime	49
Hour	50
Min	50
Month	50
Sec	50
Year	51
6 DDE Commands	51
DDEPoke	51
DDERequest	51
7 Dialogs	52
CloseDialog	52
Dialog	52
EndDialog	55
GetDialogAction	55
ResetDialogAction	56
SetDialogObjectFont	56
SetDialogObjectFocus	57
SetDialogObjectVisible	57
Show	57
8 File Handling	58
AppendFile	58
ChangeDirectory	58
CopyFile	58
CountFiles	59
CreateDir	59
DeleteFile	59
EditIniFile	60
FileDate	60
FileSize	60
FileTime	61
GetFileList	61
IfDirExists	61
IfFileChanged	62
IfFileExists	62
MoveFile	63

ReadFile	63
ReadIniFile	63
ReadLn	64
RenameFile	64
WriteLn	65
9 FTP/HTTP/Telnet/Email Commands	65
FTPDelFile	65
FTPGetDirList	66
FTPGetFile	67
FTPMakeDir	68
FTPPutFile	69
FTPRemoveDir	70
FTPRenameFile	71
HTTPRequest	72
RetrievePOP3	73
SMTPSendMail	74
TelnetClose	75
TelnetConnect	75
TelnetSend	76
TelnetWaitFor	76
10 Conditional Commands (If...)	77
If	77
IfDirExists	78
IfFileChanged	78
IfFileExists	79
IfWindowOpen	79
11 Image Recognition	80
CompareBitmaps	80
FindImagePos	80
GetScreenRes	81
ScreenCapture	82
WaitScreenImage	82
12 Keyboard Commands	83
Ascii	83
CapsOff	83
CapsOn	83
NumOff	83
NumOn	84
Press	84
Release	85
ScrollOff	86
ScrollOn	86
SendText	86
WaitKeyDown	87
13 Messages	87
Ask	87
Input	88
Message	88
MessageModal	89
14 Miscellaneous	89
Assigned	89
BlockInput	90
DateStamp	90
ExportData	90
GetCheckBox	91
GetControlText	91

GetEnvVar	92
GetListItem	92
GetPixelColor	93
GetRectCheckSum	93
GetTreeNode	94
LibFree	95
LibFunc	95
LibFuncW	97
LibLoad	97
PlayWav	97
PushButton	97
Random	98
RGB	98
SelectMenu	99
SetCheckBox	99
SetControlText	100
SetEnvVar	100
TimeStamp	100
WaitCursorChanged	101
WaitPixelColor	101
WaitRectChanged	101
15 Mouse Commands	102
GetCaretPos	102
GetCursorPos	102
LClick	103
LDbClick	103
LDown	103
LUp	103
MClick	104
MDbClick	104
MDown	104
MouseMove	104
MouseMoveRel	105
MouseOver	105
MUp	106
RClick	106
RDbClick	106
RDown	107
RUp	107
Toolbar	107
16 Numeric Functions	108
Add	108
Let	108
Sub	109
17 Registry Functions	109
RegistryDelKey	109
RegistryDelVal	110
RegistryReadKey	110
RegistryWriteKey	110
18 Running Programs/Files	111
ExecuteFile	111
RunProgram	111
19 Script Control	112
Sub Routines	112
Gosub	112
SRT	113

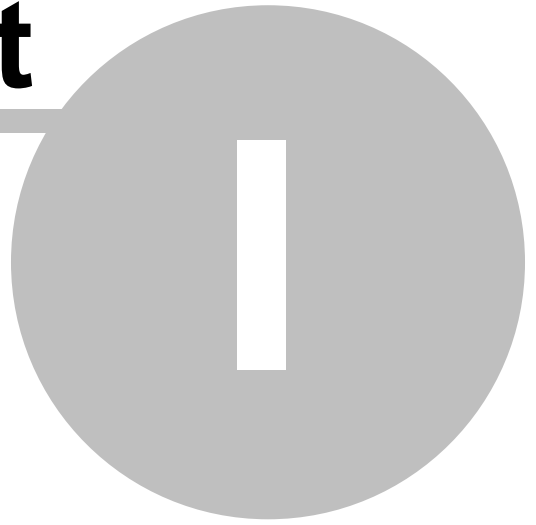
END	114
Exit	114
Goto	115
If	115
Include	116
Label	116
Let	117
Macro	118
OnEvent	118
Remark	120
Repeat	120
Until	121
Wait	121
WaitReady	121
20 String Handling	122
Base64	122
ConCat	122
Crypt	123
Length	123
MidStr	124
Position	124
Separate	124
StringReplace	125
21 Text Capture	125
GetTextAtPoint	125
GetTextInRect	125
GetWindowTextEx	126
WaitScreenText	126
GetControlText	127
GetWindowText	127
22 VBScript Commands	128
VBEND	128
VBEval	128
VBSTART	129
VBRun	130
23 Window Functions	131
CloseWindow	131
FindWindowWithText	132
GetActiveWindow	132
GetWindowHandle	133
GetWindowList	133
GetWindowPos	134
GetWindowProcess	134
GetWindowSize	135
GetWindowText	135
IfWindowOpen	136
MoveWindow	136
ResizeWindow	137
SetFocus	138
ShutDownWindows	138
WaitWindowChanged	139
WaitWindowClosed	139
WaitWindowOpen	140
WindowAction	141
24 Using Variables	141
User Defined Variables	141

Arrays	142
Explicit Variable Resolution - VAREXPlicit	143
Ignoring Spaces - IGNORESPACES	143
System Variables	144
Variable Explorer	145

Index	147
--------------	------------


Using Macro Scheduler

Part

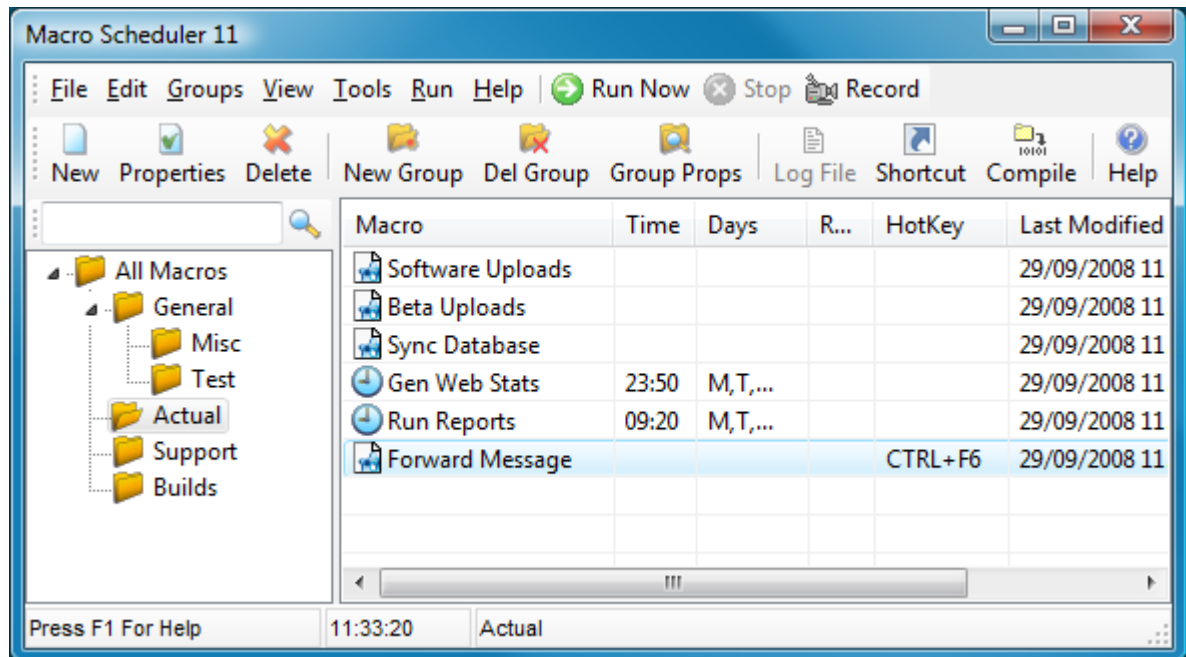


1 Using Macro Scheduler

1.1 Getting Started

When Macro Scheduler starts it places its icon -  - in the system tray (next to the clock). The first time Macro Scheduler is run it starts minimized. To view Macro Scheduler Double click on this icon, or right click on the icon to display the pop up menu and select 'Show'.

If you like you can prevent Macro Scheduler from starting minimized. To do this select Tools/Options and uncheck 'Start Minimized'. See [Options](#) ^[24].



This is the main control center for Macro Scheduler. From here macros can be started, recorded, deleted and edited.

For further information on creating macros see [Creating Scripts](#) ^[4] and [Recording Macros](#) ^[12]

Or see [Menu Commands](#) ^[2] or [Toolbars](#) ^[3] for an explanation of the menu options and toolbar buttons.

Access to the functions is through the toolbar buttons, the system menu or by double clicking on the appropriate macro, or right clicking on it to display a pop up menu.

1.2 Menu Commands

File

[New Macro](#) ^[20]
[Macro Properties](#) ^[20]
[Import Macros](#) ^[20]
[Hide](#) ^[21]
[Exit](#) ^[21]

Edit

[Delete](#) ^[21]

[Remove from Group](#) ^[21]
[Rename](#) ^[21]
[Select All](#) ^[21]

Groups

[New Group](#) ^[21]
[Delete Group](#) ^[21]
[Group Properties](#) ^[22]

View

[Sort](#) ^[22]
[Font](#) ^[22]
[Grid Lines](#) ^[22]
[List](#) ^[22]
[Details](#) ^[22]
[Large Buttons](#) ^[22]
[Toolbar](#) ^[23]
 Search Bar

Tools

[View Log File](#) ^[24]
[Create Exe](#) ^[23]
[Desktop Shortcut](#) ^[24]
[View System Windows](#) ^[24]
[Options](#) ^[24]

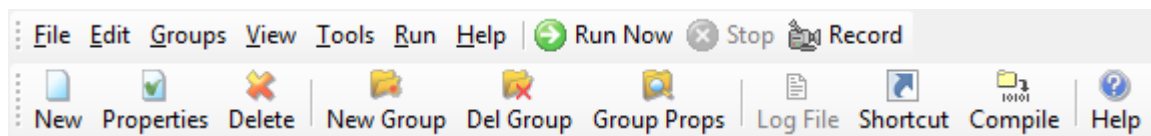
Run

[Run Now](#) ^[25]
[Record](#) ^[26]
[Stop](#) ^[26]

Help

[Contents](#) ^[26]
[Check for Update](#) ^[26]
[Support Forums](#) ^[26]
[About](#) ^[26]

1.3 Toolbar



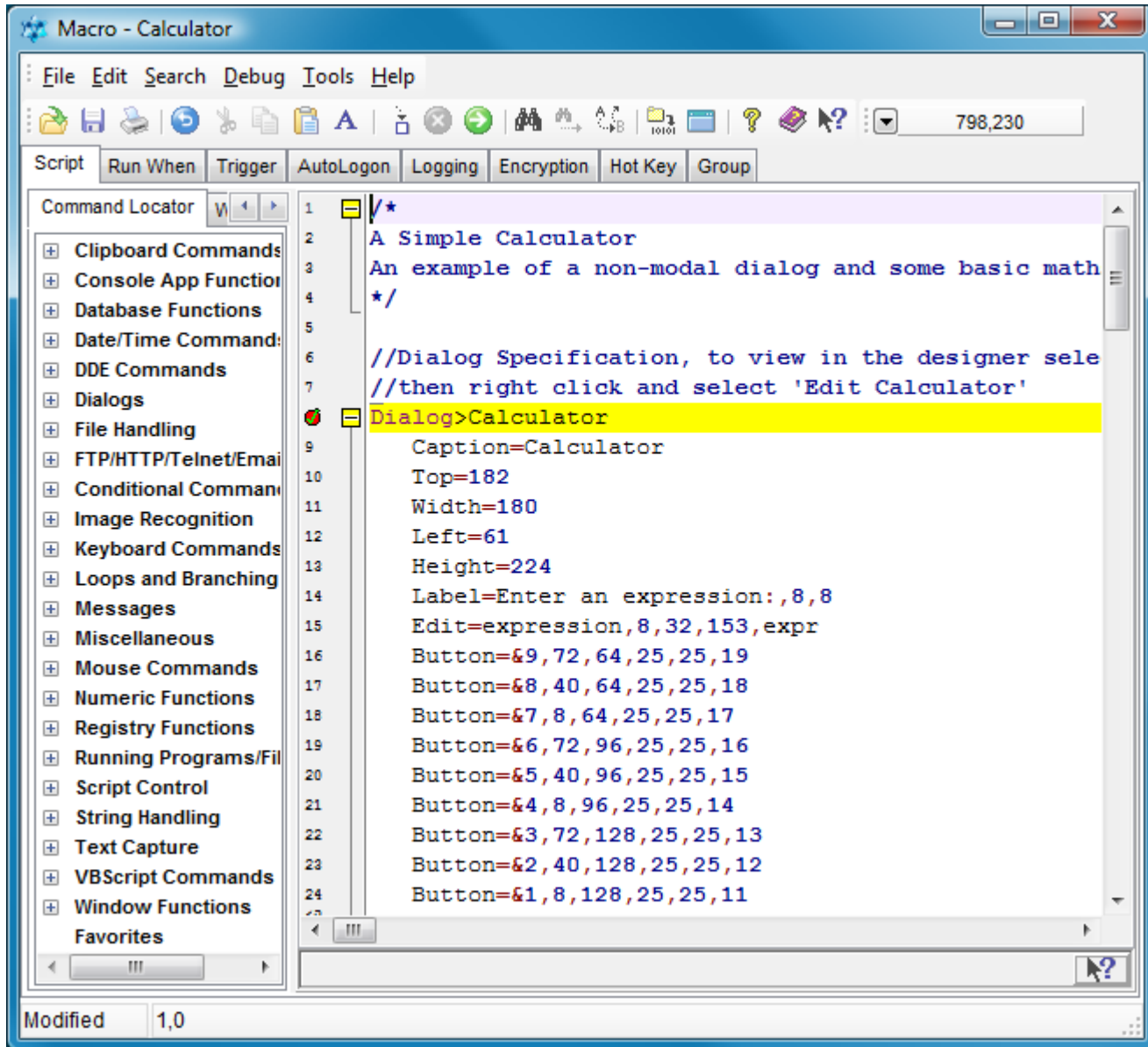
The top row of the toolbar contains the menu and buttons to [Run](#) ^[25], [Stop](#) ^[26] and [Record](#) ^[26] macros.

Buttons from Left to Right on the second line are:

[New Macro](#) ^[20], [Macro Properties](#) ^[20], [Delete Macro](#) ^[21], [New Group](#) ^[21], [Delete Group](#) ^[21], [Group Properties](#) ^[22], [View LogFile](#) ^[24], [Desktop Shortcut](#) ^[24], [Create Exe](#) ^[23], [Help](#) ^[26]

1.4 Creating Scripts

To create a new script select 'File/New Macro', or click the 'New' button on the [Toolbar](#)³. To Edit an existing script, select it from the list by clicking on it and then select 'File/Macro Properties' or double click on it. Having done this you will be presented with the following window:



In this example an existing script was selected to be edited. If 'New' had been pressed then this form would appear blank.

Code can be inserted and edited directly into the script editor. Use the [Command Locator](#)⁵ to quickly find a function to insert. Pressing F1 in the editor will present the help topic for the highlighted function or, if no text is highlighted, the function in the current line. To test the script click the 'Run' button.

To save the macro press the Save toolbar button. If you have not yet given the macro a name you will be asked for one. Press Save to save any changes made to the macro script or properties.

The script editor provides a full featured programmer's editor with code folding, bookmarks, the

[Debugger](#)^[14] and other script building options including the [Dialog Designer](#)^[15] and Code Snippets. See also [Code Folding and Bookmarks](#)^[6].

The Import button allows you to load in a script that has already been created. This is useful if you have a number of Macro Scheduler installations and want to make use of a script created on a different PC for example.

The numbers at the top right of this window show your mouse cursor position. Use these to determine the correct parameters when using the MouseMove command. The button to the left of the numbers reveals a drop down menu which allows you to toggle between absolute coordinates and relative coordinates. When set to relative, the numbers show you the relative coordinates to the window the cursor moves over. The other option allows you to attach a small tag to the mouse cursor that shows the coordinates and follows the cursor around. This means that if the script window becomes concealed by another application you can still see the cursor position. The last option of the drop down menu is used to add the pixel colour to the display as well as the cursor position. This is helpful for the [WaitPixelColor](#)^[10] command.

For help with other tabs of this window see: [Scheduling Scripts](#)^[7] (Run When), [Trigger](#)^[10], [AutoLogon](#)^[10], [Logging](#)^[11], [Encryption](#)^[11], [Hot Keys](#)^[12], [Groups](#)^[12]

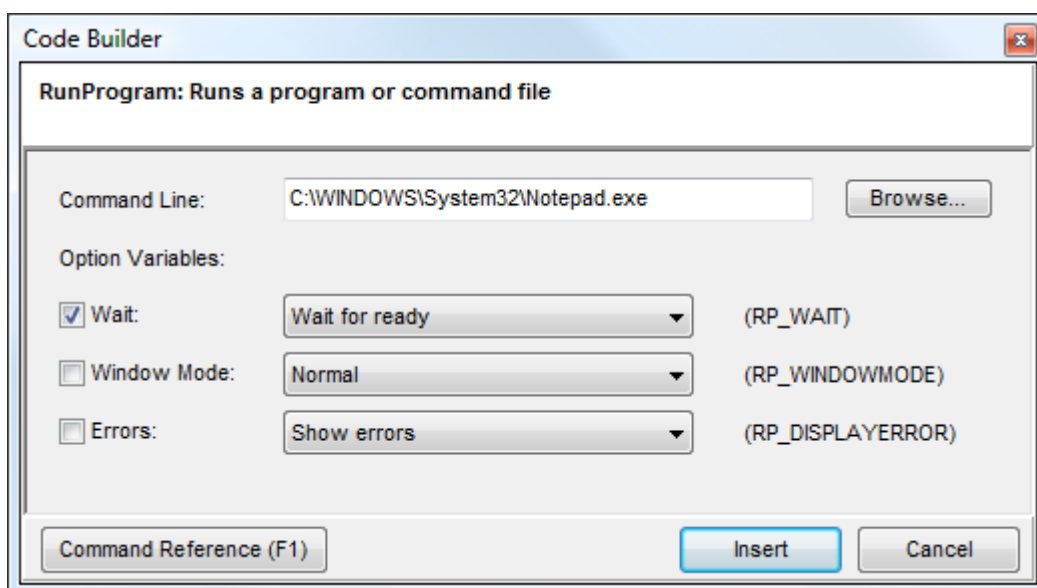
To get help with the MacroScript language see: [Scripting Windows for Beginners](#)^[28], Command Reference, [Using Variables](#)^[14], [Complex Expressions](#)^[41]


1.5 Using Variables

Please see the following topics under the Command Reference chapter:

[User Defined Variables](#)^[14]
[Arrays](#)^[142]
[Explicit Variable Resolution - VAREXPPLICIT](#)^[143]
[Ignoring Spaces - IGNORESPACES](#)^[143]
[System Variables](#)^[144]
[Variable Explorer](#)^[145]
[Debugger - Watch List](#)^[14]

1.6 Command Locator



The Command Locator is available under the [Script tab of the Macro Properties](#)  window. It organises all the script commands into convenient categories, and in the Script Editor also allows commands to be added and removed from a Favorites category.


Expand a category to see the list of commands. To insert the selected command into the editor, double click the command. The chosen script command will appear in the Code Builder dialog where you can enter the parameter values and options required for that command. Click Insert to insert the code into the editor.

In the Script Editor you can right click on a command and add it to your favorites category using the pop up menu options.

To get help on the command press F1.

1.7 Code Snippets

With Code Snippets you can store pieces of frequently used code which can quickly be inserted into a script as you are editing it.

To view Code Snippets click the Code Snippets tab in the left hand pane (the Builder Pane) of the [Editor](#) .

Right click on the Code Snippets window to reveal the popup menu with options to create categories, create a new snippet, edit, delete and rename snippets. Insert a snippet into your script code by selecting the Insert Snippet option from the popup menu or by double clicking the snippet.

Before a snippet can be created you must first create a category. Once a category has been created you can right click on it and select New Snippet. In the box that appears, enter a name for the snippet and then enter the snippet code. Enter or paste the code into the snippet box and click Save.

Once you have created a snippet it will be available in the editor for you to insert into any script.

1.8 Code Folding and Bookmarks

The Macro Scheduler script editor supports various advanced editing features such as Code Folding and Bookmarks.

Code Folding

Recognised code blocks marked by the editor with a vertical line in the margin showing their extent, and a collapse/expand icon at the first line. You can collapse or expand the code block by clicking on this icon. Collapsing a block of code hides it temporarily from view.

You can also fold any code of your choice by placing the text CODEBLOCK and ENDCODEBLOCK before and after the code. E.g.:

```
CODEBLOCK
Run>Notepad.exe
WaitWindowOpen>Notepad.exe
Send>Hello World
ENDCODEBLOCK
```

Bookmarks

Bookmarks allow you to mark a line so that you can get back to it quickly. Right click on a line and select Create Bookmark. You will be prompted for the bookmark text to use and the bookmark will be created. An icon will appear in the gutter marking the code and the line will be highlighted. You can add as many bookmarks as you wish. To quickly get back to a line right click to invoke the context menu, or select the main Search menu and click Goto Bookmark. Choose the bookmark from the list.

Bookmarks are persistent. When you save your script the bookmarks are saved with it so that when the script is later reopened the bookmarks are reinstated.

1.9 Scheduling Scripts

Once you have created your macro you will probably want to execute it. Macros can be run at any time from the main window, from Windows shortcuts, from the command line or to a specified schedule.

To set up a schedule, select the appropriate script and choose to edit it to invoke the [Macro Properties Form](#). Then select the tab marked 'Run When' to display the following options.

The screenshot shows the 'Macro - Run Reports' dialog box with the 'Run When' tab selected. The 'Days' section has checkboxes for Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday. The 'Time' section has 'At' set to 9:20 and 'Repeat Every' set to 0 minutes. The 'Monthly' section has a checkbox for 'On the' followed by 0 of Each Month. An 'Advanced Options' button is visible. The status bar at the bottom shows 'Modified 1,0'.

Mark off the days on which you want the macro to run and enter a time. The time must be entered in 24 hour notation. If you want the macro to be repeated enter an appropriate value in the 'repeat every' box. If you don't want it repeated simply leave this set to zero. The repeat every box allows you to specify up to 1440 minutes (a full day).

To make the macro run on a monthly basis check the monthly box and enter the day of the month

on which you want the macro invoked.

If you choose to run a macro on the 5th of the month and also check the Friday box, the macro will run every Friday AND on the 5th of the month regardless of what day the 5th is.

In the 'Advanced' scheduling settings you can specify what should happen when Macro Scheduler restarts. This can be useful if you have Macro Scheduler start every time you reboot your computer. For instance, you can set Macro Scheduler to continue to repeat when the computer restarts. You can also determine what should happen if a schedule was missed while the computer was turned off, and you can specify a window title so that the macro starts when that window appears. See [Advanced Scheduling Options](#)^[9].

If you have created a schedule for a macro, but wish to temporarily stop scheduling without changing the schedule details, you can do so from the pop up menu of the script list on the main Macro Scheduler window.

For information about scheduling macros when Macro Scheduler is not running see [Scheduler Service](#)^[8].

A Note About Screensavers

Screensavers usually stop successful detection of other windows. Consequently if a script that needs to setfocus or wait for windows to appear is run while a screensaver is active it may not work correctly.

To get round this Macro Scheduler temporarily disables screensaving just before it runs a script and re-enables screensaving when the script completes. It also attempts to determine if a screensaver is currently active and if so closes it down. However, there are many different implementations of screensavers which operate in different ways, making their detection and close down a rather unreliable process. To try to ensure that Macro Scheduler is successful in closing an active screensaver it briefly moves the mouse back and forth before running the script.

This should work in most cases. However, if you find that Macro Scheduler is unable to stop your screensaver, then the most reliable method of making sure that a scheduled macro runs properly is to simply disable screensaving altogether.

You can also use [AutoLogon](#)^[10] to unlock a locking screensaver.

1.10 Scheduler Service

The Scheduler Service is not available in Windows 95/98/Me/Vista.

Macro Scheduler 9.0 comes with an optional Scheduler Service which can be used to schedule macros when Macro Scheduler is not running and even when Windows is logged out.

Normally, when Macro Scheduler is running it handles the scheduling of macros itself. This means that you can continue to have multiple instances of Macro Scheduler running together, scheduling different sets of macros (if installed in such a way). One main instance of Macro Scheduler can be assigned the Scheduler Service so that when it is exited the Scheduler Service is started and takes over the scheduling of macros.

Enabling the Scheduler Service

To enable the Scheduler Service go to [Options](#)^[24] (Tools/Options) and select "Keep Scheduling on Exit (Starts Scheduling Service)." If this option is grayed out then another copy of Macro Scheduler already has the service allocated. To remove that allocation start that other copy and deselect this option.

Running Macros under User Login

In Windows, interactive services run under the Local System account. This means that by default

they can't necessarily access the same resources that you can when logged in under your own account, because the two accounts are different. Application settings and network resources accessible by your account may not be accessible by the local system account. E.g. the local system account will not be able to access your email in Outlook because Windows needs to be logged in as you to do that. To get round this you can tell the service to run a macro under a specific account by providing the username and password for that account in the [AutoLogon](#)^[10] settings for that macro. The Service will then impersonate that user account and start the macro process with it's credentials. If you have enabled this option and your macro still fails to run correctly it could be that your account has security restrictions that prevent the system from impersonating it- or certain objects that the macro needs to use do not allow an impersonated logon access.

Unlocking or Logging in to Windows

When Windows is locked (e.g. by a screensaver) or logged out it is not usually possible to manipulate the user interface of applications because they exist on a different "desktop" to the "login desktop". Mouse and keyboard events will not get to the target windows and in many cases windows and applications have not even been created yet. However, with [AutoLogon](#)^[10], Macro Scheduler 8.0 is able to unlock the workstation or log in to Windows, perform the script and then log out again. Use the [AutoLogon](#)^[10] settings to set the user account which the macro should be logged in for.

AutoLogon on Windows XP

AutoLogon cannot work with the new XP "Welcome" screen and will only work with the classic logon prompt. To revert to the classic logon prompt, choose "User Accounts" from the control panel, then click "Change the way users log on or off". Uncheck "Use the Welcome screen" and click "Apply Options".

Preventing a Macro being started by the Scheduler Service

You may want to use the Scheduler Service when Macro Scheduler is not running but prevent it from starting specific macros. To do this go into [Macro Properties](#)^[4] for that macro and then [Advanced Scheduling](#)^[9] and uncheck "Allow Scheduler Service to Run this Macro"

Limitations of the Scheduler Service

The Scheduler Service is unable to run macros that require a password to be entered when run from a schedule. Since the Scheduler Service is designed primarily to run macros unattended when the machine is logged out this should not present a problem. When the machine is logged in Macro Scheduler can schedule macros as normal.

1.11 Advanced Scheduling

The Advanced Schedule options dialog is accessed from the [Run When](#)^[7] tab of the [Macro Properties](#)^[4] window. Options are:

When Repeating

These options are used when you have set a repeat interval on the main schedule. The default option is repeat continuously. What this means is that even after the system restarts the schedule will continue to repeat (as long as the current day is a day chosen in the main schedule). The second option will tell Macro Scheduler not to keep repeating continuously, but only to repeat until the specified time. At that point the original schedule time is reset to the original start time.

Recovery

Here there are three options. The default is to do nothing except run the macro at the scheduled start time, as normal. If the second option is set, then when Macro Scheduler restarts it checks to see if a macro should have run on that day. If a macro was supposed to run on that day and at a time prior to the system restarting, the macro will be run immediately. The third option will simply allow you to run the macro on startup regardless. Note that the third option will not effect the schedule time in any way - it simply runs the macro.

Scheduler Service

If you have enabled the scheduler service in [options](#)^[24] to schedule your macros when Macro Scheduler is not running you can determine whether this macro should be started by the scheduler service here. Uncheck Allow Scheduler Service to Run this Macro if you do not want the scheduler service to start this macro when Macro Scheduler is not running. This option has no effect if the scheduler service is not enabled in [Options](#)^[24].

1.12 Trigger

Under the Trigger tab of the Macro Properties dialog you can specify a condition which should trigger the macro. Trigger types include

Window Event

- Window open
- Window closed

File Event:

- File exists
- File does not exist

Folder Event

- Folder exists
- Folder does not exist
- New file in folder
- File deleted from folder

Macro Scheduler needs to be running for these triggers to be actioned.

More triggers can be implemented by building them into the script itself. Events and conditions can be coded into continuously looping macros so that they themselves detect conditions and act on them - e.g. screen changes, window changes, object caption changes, event log entries, ini file values, registry values, and almost anything you can think of.

1.13 AutoLogon

AutoLogon can be used to automatically unlock or log on the workstation if it is locked when the macro is scheduled. When the macro has finished running the workstation is locked again. The AutoLogon tab can also be used to set the user account under which the macro should be run by the optional [Scheduler Service](#)^[8].

AutoLogon is not supported in Windows Vista, Windows 95, Windows 98 or Windows Me.

Unlock Computer To Run This Script

Check "Unlock Computer To Run This Script" if you want Macro Scheduler to automatically unlock/log on to run the script if the workstation is locked at the time. Enter a valid Windows username and password.

XP Users: AutoLogon can only work with the classic logon screen. If you are using Windows XP be sure to disable the "Welcome" screen and "Fast User Switching" to revert to the classic logon screen.

When Started By Scheduler Service Run as User Below

Scripts that are started by the [Scheduler Service](#)^[8] (enabled in [Options](#)^[24]) when Macro Scheduler is not running are run under the Local System account by default. This is because interactive services must run under the Local System account to work. However, if your script needs to have access to settings, network resources and/or applications specific to your account it needs to run as your user or the script may fail as it will not be running in your security context. To

make the [Scheduler Service](#)^[8] run the macro under your account check this box and, if not already entered, enter a username and password. This option has no effect when Macro Scheduler is running or if the [Scheduler Service](#)^[8] has not been activated in [Options](#)^[24]. While Macro Scheduler is running your account is logged in anyway. This option also has no effect if "Allow Scheduler Service to Run this Macro" has been unchecked in [Advanced Scheduling](#)^[9] options.

This option is checked automatically when "Unlock Computer to Run This Script" is selected as most of the time you'd want the macro to run under the account it logged in as! However, you do not have to use "Unlock Computer to Run This Script" to select this option. E.g. you may not want to automatically log in if logged out but you do want the macro to run as this user when Macro Scheduler is not running and you are already logged in.

1.14 Logging

To set up a log file for a script select the tab marked 'Logging' on the [Macro Properties Window](#)^[4].

To enable logging check 'Log Progress of Macro'.

You can have Macro Scheduler purge the log file before each run by ticking the second box.

Enter a file name for the log file or select an existing one by using the browse button. If you like you can use one file for more than one macro.

In Log Level you can specify whether to log just after each script step is executed, just before, or before and after.

1.15 Encryption

The majority of us probably won't ever have to use the encryption facility. However, if you need to use Macro Scheduler to automate a process which involves sending passwords to other applications or to send other sensitive information, then you would want to ensure that only the right people can edit the script and see the secrets.

Macro Scheduler allows you to set a password for a script which must then be used to edit it. The script file itself is scrambled so that if it is edited in any way it wouldn't make any sense.

For encryption options select the Encryption tab of the [Macro Properties](#)^[4] dialog.

Simply check the "Encrypt This Macro" box and provide a password.

The password must be entered twice to ensure it is entered correctly.

Next time you try to edit the macro you will be asked for the password.

Using the last two options you can specify whether the password should be entered when the macro is run. The first of these two options will ensure that when a macro is started from the [main window](#)^[2], [command line](#)^[16], or [Macro](#)^[118] command, it will only run if the correct password is entered. The second option will further secure the macro for scheduled events, so that a scheduled macro will need the password to be entered before it will start.

To disable encryption on a macro that has been encrypted, edit it, select the Encryption tab and then uncheck "Encrypt This Macro".

1.16 Hot Keys

Each script can be assigned a hot key to allow the script to be launched from a keyboard shortcut.

To assign a hot key select the tab marked 'HotKey' from the script settings window and select the appropriate keys from the drop down lists.

Press OK to save the settings. No matter what program you are working with, as long as Macro Scheduler is running, the chosen key combination will now launch your macro.

1.17 Groups

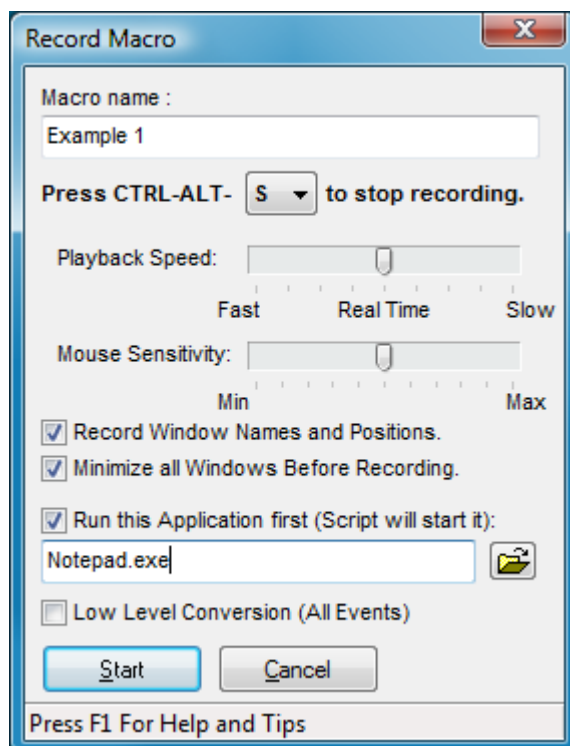
Macros can be organised into groups. Groups can be created using the [Groups menu](#)^[2], [Toolbar](#)^[3] or context sensitive menu in the groups pane of the [Main Window](#)^[2]. Macro groups can be assigned a physical path in [Group Properties](#)^[22]. You can then drag macros from group to group.

You can also set a macro's group in Macro Properties under the Group tab.

If 'Show In System Tray Quick Launch Group' is checked in Macro Properties the macro will appear in the pop-up menu that appears when 'Quick Launch' is selected from Macro Scheduler's system tray icon. By default new macros are added to the Quick Launch group. Remove a macro from the Quick Launch group by unselecting this option.

1.18 Recording Macros

To record a macro select Run/Record, or press the 'Record' [Toolbar](#)^[3] button (looks like a video camera) button on the [Main Window](#)^[2]. You will be prompted for a name for the macro and recording will commence when Start is pressed.



By default CTRL-ALT-S will terminate the recording. You can select an alternative key from the drop down list box if required. SHIFT-ESC (or the shortcut chosen in [Options](#)^[24] to stop running scripts) will also stop the recording in the same way that it stops playback.

Carry out the tasks you want to be captured and finally press CTRL-ALT-S (or chosen key) or click on 'Stop' to end the recording.

The new macro will appear in the macro list and can be executed by clicking the 'Run Now' button.

You can modify the playback speed using the slider. This applies a modification to the Wait times between individual events. In many cases, attempting to speed the macro up by reducing these times will actually make the macro less reliable. It is advisable to leave the speed at real time.

The mouse sensitivity setting is used to ignore small mouse movements. Modify the degree of

sensitivity by adjusting the slider. The further towards Max, the more sensitive the recorder is and the smaller the movements that are ignored. By ignoring small mouse movements the macro recorder can refine the script where small, unnecessary mouse movements appear between other events which can be combined into one script command. If these small movements are not ignored then the individual events need to be handled separately, so lengthening the script. If in doubt leave it in the default, middle setting.

With the 'Record Window Names and Positions' option selected the macro recorder will notice when your actions cause windows to appear and insert code to make sure the macro waits for those windows to appear before continuing. It will also add code to ensure those windows appear in the same positions and with the same dimensions as they appeared when you recorded the macro. This ensures the macro will work correctly each time with your mouse events landing in the right places.

When recording a macro for a specific application, select the application to run. Macro Scheduler will run the application and remember it's name and position and add the necessary commands to the top of the recorded macro to ensure the application is executed, sized and positioned in the same place each time, so ensuring the reliability of the recorded macro.

The last setting - 'Low Level Conversion' - will avoid parsing to result in a script of lower level commands. E.g. an LDown and LUp will be left alone and not merged into an LClick command.

Recommendations:

- To make a reliable macro we recommend using the "Run this application first" option. The macro recorder will then start that application and the generated script will also ensure it is started when the script is run. This is a far more reliable way to start an application than by double clicking on a desktop shortcut icon, for example. Don't record pointless mouse clicks to start an application when the application can be started directly. Let the macro recorder start the application and have it generate a script that runs that application specifically before the rest of the macro proceeds.
- Use the "Record Windows Names and Positions" option to make the generated script wait for target windows to appear; and size and position them the same way each time.
- Minimize all windows before recording your macro so that you are starting against a "clean" desktop and to avoid recording window names that are present now but may not be during playback.
- Learn how to write scripts manually. Written macros can be made more efficient, more reliable and more aware. [Read the scripting tutorial](#)^[28].

1.19 Playing Scripts And Macros

To play a macro or script without scheduling it use the 'Run Now' [Toolbar](#)^[3] button on the [Main Window](#)^[2]. You can also press CTRL-R, or choose Run/Run Now from the [Main Menu](#)^[2]. Another way to start a macro is to right click on it and choose 'Run Now' from the pop-up menu.

After starting a macro the stop button on the tool bar will become enabled. You can press this to stop the macro at any time. CTRL-B or Run/Stop will also stop a macro.

To stop macros no matter what program or window is active, press SHIFT-ESC. You can change this shortcut in [Options](#)^[24].

The menu that pops up from the icon in the task bar also has an option called 'Stop'. This works like the stop button and allows you to cancel the execution of a script. This option is available even when a script is executed automatically by the scheduler.

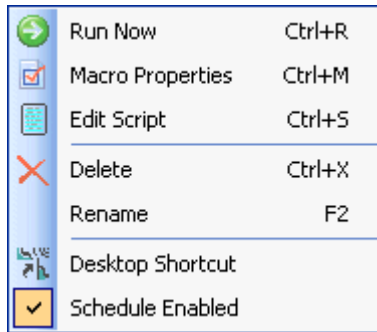
To pause all running scripts during execution press the Pause key. Pressing the Pause key again will resume paused scripts. You can specify an alternative to the Pause key in [Options](#)^[24].

You can run other macros within scripts by using the [Macro](#)^[118] command.

Macros can also be assigned to desktop shortcuts or run from the command line. See [Creating Desktop Shortcuts](#)^[14] and [Command Line Option](#)^[16].

1.20 Creating Desktop Shortcuts

You can tell Macro Scheduler to create a shortcut for a macro by selecting the appropriate macro from the main window and then clicking the right mouse button to display a pop up menu. Alternatively you can select Tools/Desktop Shortcut from the [Main Menu](#)^[2], or press the Desktop Shortcut [Toolbar](#)^[3] button.



Select the second from last option and a shortcut will be placed on your desktop. To run the macro you then only need to double click your desktop icon. Once it is on your desktop you can, if you prefer, move it elsewhere in the usual way using explorer etc.

1.21 Using the Debugger

In the Editor there is a menu called 'Debug' with the following options

Step (F8)

This highlights the currently selected line, and then subsequently executes that line and moves to the next line in execution flow.

Step Over (Shift-F8)

This option is enabled if the current line contains a GoSub command or an If statement which contains a call to a subroutine. Step Over executes the subroutine in whole and moves the debug cursor to the next line.

Stop Debug (CTRL-F2)

Stops debugging and resets everything.

Trace

This displays a small dialog where you can set an interval. Then the debugger auto-steps at the specified frequency.

Run

Runs the script without stepping, from the currently selected line. The script will run to the end or until the next breakpoint, from where the script can be stepped or run.

Insert Breakpoint

Inserts a breakpoint after the selected line. Simply inserts ****BREAKPOINT**** on the next line. If running the script using 'Run' from the Debug menu, execution will pause when this Breakpoint line is reached and the script can then be stepped from this point. As of version 10 multiple breakpoints can be set. At any time pressing Run will run the script to the next breakpoint from where the script can again be stepped or run.

Refocus Windows (check on or off)

Most of the time you want this checked on. Due to the nature of Macro Scheduler, and the fact that it is commonly used to automate other windows, most scripts will focus other apps and chop and change focus a lot. 'Refocus Windows' ensures that if a command causes focus to shift, focus will be set back to that window before executing each subsequent line so that (hopefully) that line will affect the correct app. Focus is returned to the editor after each line so that you can see what you're doing. Sometimes it is useful to turn this off ... if you're not doing any GUI scripting, or even at some point during a GUI script where you might have a loop, say, calculating some value but not actually requiring focus of anything. So you can switch it on and off during debug as required.

Show Watch List

Displays a list to the left hand side of the editor containing all current script variables and their values - updates as they change. This is the most useful debugging device as you can see the values of variables as they are set. By default the watch list shows the most recently modified variable at the top of the list. Or you can sort the watch list by right clicking and selecting "Sort". Right clicking on an item allows you to copy the line or just the variable value to the clipboard.

Notes

Remember that stepping through a script will slow it down, since you will be pausing at each line. Since many scripts that automate windows applications need to be time-sensitive - to make sure things don't happen before apps are ready etc - the process of debugging may give the impression that the script is fine, when it actually needs delays and waits to be built in!

Other Debug Methods

Other things you can do to help debug your script include:

- Use LogFiles - See [Logging](#)^[11] options
- Use [MessageModal](#)^[89] command to display diagnostic information on screen
- Use [DUMP_VARS variable](#)^[144] to dump all variable values to the log file
- Use the [Variable Explorer](#)^[145] (Tools/Variable Explorer or CTRL-ALT-V) to view all user variables and find out where they are created/modified.
- Use the [STEP_DELAY](#)^[144] variable to slow the script down.

1.22 Dialog Designer

This is accessed from the Tools menu in the [Editor](#)^[4]. This is an easy way to create MacroScript dialogs.

If a [Dialog](#)^[52] block is highlighted when the Dialog Designer is selected, that dialog will be displayed in edit mode. Otherwise a new dialog is displayed.

You can also edit a dialog by right clicking on the first line of the Dialog block and select the edit option from the pop-up menu.



Select objects from the Objects bar to add to the dialog. Objects can be moved around and resized.

Toolbar buttons exist to copy, cut, paste, lock, resize, and move the objects. You can also modify the tab order.

Double click on an object, or select the Object Properties button to modify the properties as

appropriate for the selected item.

The bottom left toolbar button saves the dialog to the clipboard so that you can paste it into your script.

When you have closed the Dialog Designer simply paste the new dialog into the Editor by using Edit/Paste, or CTRL-V, as you would paste any other text.

1.23 Command Line Options

It is possible to run macros from the command line using the following syntax:

msched macroname

e.g. to run the Defragment Disk example script you would type:

msched Defragment Disk

This is useful for creating shortcuts and running Macro Scheduler scripts from other programs or from macros created in other applications such as Word or Excel.

However, if you want to create a shortcut, you can get Macro Scheduler do it for you. See [Creating Desktop Shortcuts](#) ¹⁴.

Parameters

When running a Macro from the command line in this way, you can also pass parameter values into the script :

msched Example Script /filename=testfile.txt /path=c:\outpath

The above example runs a script called 'Example Script' and passes two variable values in filename and path. These variables can be used in the usual way within the script, e.g. :

```
Change Directory>path
ifFileExists>filename,ok
Goto>end
Label>ok
Message>File Exists !!
Label>end
```

It is also possible to run any file by passing the full path and filename to Macro Scheduler like this :

msched c:\scripts\my script.scp

In this instance, the script 'my script' does not have to exist in the Macro Scheduler macro list. If you wish you can associate .scp files with Macro Scheduler so that Macro Scheduler scripts can be run by double clicking on them in explorer.

The same method can be used for compiled scripts:

compiledmacro.exe /fullname="Donald Duck" /filename=testfile.txt

Note that quotes can be used to surround values that contain spaces.

Log File

You can specify a log file for the macro by passing the LOGFILE variable:


```
msched.exe c:\scripts\my script.scp /LOGFILE=c:\mylogfile.log
```

As this is passed as a script variable it will also be available to the macro as a regular variable.

It is not necessary to specify a log file if running a macro using only the macro name for a macro that already has logging enabled. In this case the log file settings for that macro will be used. However, if the LOGFILE parameter is passed it will override the macro's existing logging settings. The LOGFILE parameter can also be used with compiled scripts.

Script URLs

The script file can also be an HTTP URL, so that you can run macros from the web:

```
msched.exe http://www.mjtnet.com/scripts/sample.scp
```

If you need to specify a username and password to access the script do so as follows:

```
msched.exe http://username:password@www.mjtnet.com/scripts/sample.scp
```

Opening the Editor

To run Macro Scheduler in macro editor mode, use the -EDITOR switch:

```
msched.exe -EDITOR
```

To open a script file directly in the Macro Scheduler editor, add the filename to the command line:

```
msched.exe -EDITOR c:\myscripts\somescrpt.scp
```

Disabling System Tray Support

To switch off system tray support, so that the Macro Scheduler icon does not show in the System Tray add the parameter -NOSYSTRAY or /NOSYSTRAY at the end of the command line :

```
msched -NOSYSTRAY
```

Hiding Macro Scheduler

You can hide the Macro Scheduler window with the -HIDE parameter:

```
msched -HIDE
```

To hide Macro Scheduler completely use this in conjunction with the -NOSYSTRAY parameter:

```
msched -HIDE -NOSYSTRAY
```

-HIDE and -NOSYSTRAY can also be used with compiled macros.

1.24 Help and Resources

By far the best place to get support is at the Macro Scheduler forum:

<http://www.mjtnet.com/usergroup/>

Also, keep an eye on Marcus' blog:

<http://www.mjtnet.com/blog/>

On the blog you will find articles on how to work with databases, how to use image recognition, working with Microsoft Excel, using the debugger, how to get started with automation and lots

more. The blog is updated regularly, so keep an eye on it or subscribe to the RSS feed.

For more support resources or to contact us please visit:

<http://www.mjtnet.com/support.htm>

Please also send bug reports, comments and suggestions via the above page.

For hints and tips have a look at the Scripts & Tips page at:

<http://www.mjtnet.com/scripts.htm>

Finally, keep an eye on the MJT Net Ltd web site for new product announcements and information:

<http://www.mjtnet.com>


Menu Commands

Part



2 Menu Commands

2.1 New Macro

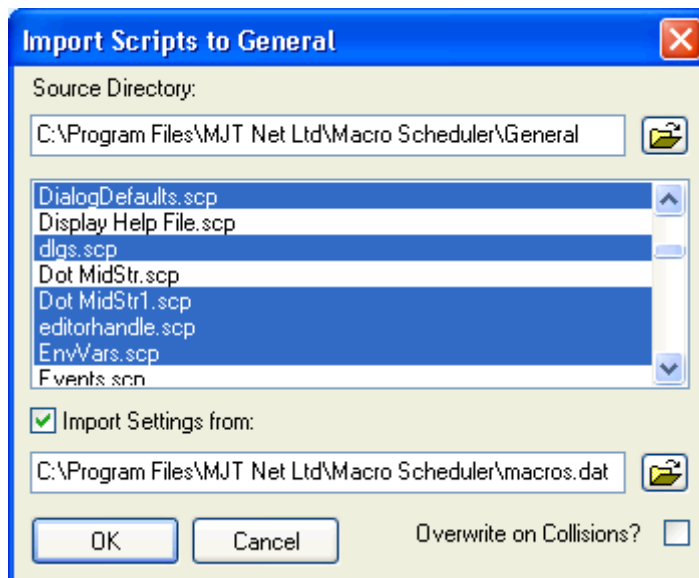
Displays the [Macro Properties](#)  window for creating a new macro.

2.2 Macro Properties

Displays the [Macro Properties](#)  window for the currently selected macro.

2.3 Import Macros

Allows macros from an external directory or another Macro Scheduler installation to be imported into the currently selected group.



Locate the source directory and highlight the files to be imported.

The files will be copied into the current group directory and will appear in the macro list. To import script settings, such as schedules, hot keys and logging details, select 'Import Settings from' and locate the macros.dat file that contains the data.

During the import process if a macro with the same name already exists in Macro Scheduler the new name will be suffixed with a numeric value to make a name that is unique.

'Overwrite on Collision' relates to physical files in the directory, not macro names. If a target file already exists it will be overwritten only if 'Overwrite on Collision' is checked.

Please note: Import Macros can not successfully copy encrypted macros. If you wish to import encrypted macros please first decrypt them.

2.4 Hide

Hides the main form (minimizes to the system tray). Not available in NT3.5x

2.5 Exit

Closes Macro Scheduler.

2.6 Delete

After prompting for confirmation, deletes the selected macro.

2.7 Remove from Group

Removes the selected macro from the group. This differs from [Delete](#) ²¹ in that this option does not delete the macro's script file from the disk. The macro is simply removed from the list of macros in Macro Scheduler. Its script file still exists on the drive.

2.8 Rename

Displays the rename dialog to allow the selected macro to be renamed.

2.9 Select All

Selects all macros in the macro list.

2.10 New Group

Select this to create a new macro group. The new group is created beneath the currently selected group. Therefore, to make a new top level group, select 'All Macros' first.

On selecting this option the [Group Properties](#) ²² dialog is displayed where the new group name and a macro path are selected.

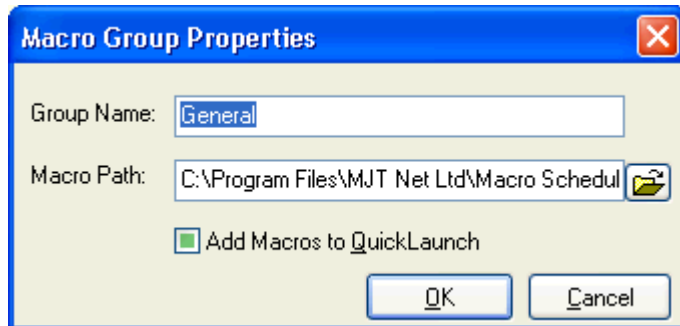
A macro path is simply a folder on a drive or network drive which is used to store the physical files associated with macros that you place in the associated group. By default the main Macro Scheduler directory is used.

2.11 Delete Group

Deletes the selected group. A group containing macros cannot be deleted. If an attempt is made to delete a group that has macros a message box will be displayed to this effect. Delete or move any macros before deleting the group.

2.12 Group Properties

Displays the group properties for the currently selected group. The name of the group and/or the macro path can be changed.



You can add or remove all the group's macros to and from the Quick Launch system tray menu in one go by checking or unchecking 'Add Macros to QuickLaunch'.

2.13 Sort

Presents a further menu containing the names of the macro list columns. Selecting one sorts the macro list by that column.

You can also sort the macro list by clicking on a column header.

2.14 Font

Displays the standard font dialog to allow the macro list font to be set.

2.15 Grid Lines

Toggles the grid lines in the macro list on and off.

2.16 List

Switches the macro list into list view, showing just the icon and macro name.

2.17 Details

Switches the macro list into classic details mode, showing all information in columns.

2.18 Large Buttons

Toggles the toolbar buttons between regular small image only buttons, and larger buttons with text as well as images.

2.19 Toolbar

Shows or Hides the toolbar.

2.20 Search Bar

Shows or hides the Search Bar. Use the Search Bar to search for macros. The Search Bar searches inside the script file as well as the name of the macro.

2.21 Create Exe

If you have the Professional edition installed this option will create a standalone executable version of the highlighted script.

Output file:

The compiler will prompt for an output exe file. By default the exe will have the same name and path as the script being compiled. You can browse for a different location using the browse button.

Icon:

As an option a custom icon can be embedded into the compiled executable by selecting an icon (.ico) file. This is not supported in Windows 95/98. To compile a custom icon into the executable use Windows NT, 2000, XP or above.

Options:

Create Console App	Creates an EXE which runs as a console app and can write to STDOUT
Compile Includes	Scripts referenced by Include> statements will be embedded and compiled into the Exe. Will not work where Include> references scripts via variables other than SCRIPT_DIR.
Disable Logging	Prevents the Exe from being logged (adds /LOGFILE=\dev\null to Include Parameters)
No System Tray Icon	No system tray icon will be created (adds /NOSYSTRAY to Include Parameters)
Run Hidden	Task bar icon will not be shown (adds /HIDE to Include Parameters)
Disable SHIFT+ESC Stop Key	Prevents SHIFT+ESC being used to stop the Exe (adds /NOSTOPKEY to Include Parameters)

Include Parameters:

Optionally you can also specify a command line to include in the compiled executable. This means command line parameters can be "hard-coded" into the executable so that they don't have to be specified on the command line when the executable is run. Any of the command line options that can be passed into scripts can be included, including parameters to be passed into scripts. E.g. /HIDE, /NOSYSTRAY, /LOGFILE= and script parameters can be hard coded into the executable. See [Command Line Options](#)¹⁶.

The title of the compiled executable can be modified by setting the APP_TITLE variable in the script or pass it in Include Parameters. The default is "Macro Scheduler". E.g.:
Let>APP_TITLE=My Program

The executable file created by the compiler can be run on any machine without having to have Macro Scheduler installed, and it will run the script.

Compiling from the command line:

To compile on the command line run msrt.exe as follows:

msrt.exe -COMPILE source.scp target.exe [-QUIET] [-DEL] [-OPTS:options] [-ICON:iconfile]

Optional switches are shown in square brackets and are as follows:

- QUIET will prevent compiler error/success messages being displayed
- DEL deletes the source script file
- OPTS: then a command line to compile into the executable, e.g.: -OPTS:-NOSYSTRAY /parm1=fred
- ICON: then a .ico file to change the default program icon

To compile a console app use msrt_console.exe instead of msrt.exe.

2.22 View Log File

If the selected macro has a log file assigned to it, this option will display that log file in Notepad.

If you wish to use a different viewer application you can add the following key to the registry:

HKEY_CURRENT_USER\Software\MJTNET\MSched - LogViewer

Set the LogViewer value to the path of the log viewer application/editor.

2.23 Desktop Shortcut

Creates a desktop shortcut for the selected macro. See [Desktop Shortcuts](#) .

2.24 View System Windows

Displays a tree of all windows currently open in the system. The tree represents a hierarchy showing the handle, class name and title text for each window and their child windows.

You can search through the tree by entering some text in the Find box and pressing Enter or clicking on 'Find'. Press Find again to locate the next match.

Refresh the list by clicking the 'Refresh List' button.

By right-clicking on an entry you can:

- Copy the object's caption - this is the text of the object
- Copy the entire text shown in the list (handle - class name and object text)
- Show the selected window
- Hide the selected window
- Close the selected window - use this with caution.

2.25 Options

Displays the options dialog. Options are:

Stop Running Scripts With

By default SHIFT-ESCAPE will stop running scripts. You can change the shortcut used to stop running scripts here by entering the key sequence you prefer.

Pause Running Script With

By default the PAUSE key will pause/resume running scripts. You can change this key sequence here.

Start Minimized

To make Macro Scheduler startup minimized, check the box - 'Start Minimized'.

Close Button Minimizes

If you want the close button to minimize Macro Scheduler instead of closing it, check the box marked 'Close Button Minimizes'.

Allow Multiple Instances

By default it is possible to have more than one instance of Macro Scheduler running at the same time. However, if you need to ensure that only one instance can run at a time, uncheck the third option 'Allow Multiple Instances'.

Escape Key Cancels Macro Properties Dialog

If you want to be able to cancel changes when using the Macro Properties dialog just by pressing the Escape key then enable this option. For safety this is disabled by default.

Warn about trailing spaces when saving macros

With this option checked Macro Scheduler will warn you when saving a macro of any lines that have trailing invisible characters. If no lines have trailing invisible characters no warning will be made. In any case you can switch this check off altogether by unselecting this item.

Keep Scheduling on Exit (Starts Scheduler Service)

This option is not available if you are running Windows 95, 98 or Me. Requires Windows NT, 2000, XP or later.

If this option is checked the scheduler service will be started when Macro Scheduler exits so that macros continue to be scheduled. The scheduler service will start automatically when the computer is started and will schedule macros when Macro Scheduler is not running and even when the machine is logged out. Use [AutoLogon](#)¹⁰ for macros that need to be logged in to work, and/or for macros that need to run as a specific user.

If you run multiple instances of Macro Scheduler only one copy can use the scheduler service. If this option is selected in one instance it will be disabled in another. I.e. the scheduler service will only schedule macros for the instance that has this option checked.

Go Straight to Editor on New Macro

With this option checked when you create a new macro the Macro Properties window will be bypassed, taking you straight to the Advanced Editor. Recommended only for experienced scripters.

Show Running Indicator when Hidden

When a macro is started while Macro Scheduler is hidden (i.e. via the scheduler or from a hotkey) a balloon tip will appear by the system tray icon indicating which macro was started.

Attempt to Deactivate Screen Saver on Run

By default when a macro is started Macro Scheduler will try to deactivate any running screen saver, as screen savers can prevent windows from being focused and receiving keystrokes etc. Uncheck this option if you would prefer not to deactivate the screen saver, but bear in mind that any scripts that need to interact with windows may not work correctly if a screen saver is running.

2.26 Run Now

Runs the selected macro. See [Playing Scripts & Macros](#)¹³.

2.27 Record

Displays the record dialog. See [Recording Macros](#) ^[12].

2.28 Stop

Stops running macros. SHIFT-ESC (or the alternative shortcut chosen in [options](#) ^[24]) will also stop macros without Macro Scheduler having to be active. There is also a break option on the system tray menu.

2.29 Contents

Displays the help file contents.

2.30 Check for Update

Use this option to check if an update to Macro Scheduler exists. This option starts your Internet browser and gives you information about updates.

2.31 Support Forums

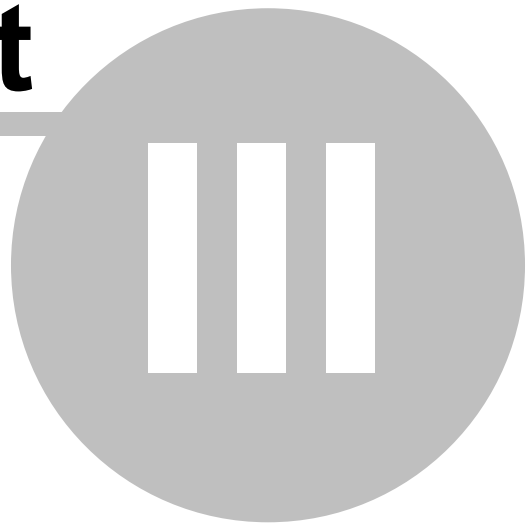
Starts your internet browser and takes you direct to the Macro Scheduler support forums.

2.32 About

Displays the About dialog box. This is useful to determine which version of Macro Scheduler you are using, who it is registered to and for a quick link to our web site.

Scripting Windows For Beginners

Part



3 Scripting Windows For Beginners

3.1 Introduction

Macro Scheduler provides a simple but powerful language for automating Windows applications. This language is similar in many respects to the BASIC programming language. As well as the usual programming constructs it provides specialised commands for automating Windows applications at the user interface level, such as sending keystrokes, mouse events and activating windows.

This guide works through a simple automation example. The scenario is a rather pointless "Hello World" type of situation, but it will work unmodified on any Windows PC and aims to demonstrate key methods used in almost every other automation scenario, and to introduce a number of important script elements.

You can watch a video version of this tutorial here:

<http://www.mjtnet.com/demos/tutorial.html>

3.2 Requirements

This document assumes you are running Macro Scheduler 6.0 or above on a standard PC running Windows 9x, Me, NT, 2000 or XP with a default printer enabled. You should be familiar with the Windows operating system. Some basic programming experience is helpful but not essential.

3.3 The Problem

We need to build a script, which opens Notepad, writes a few lines of text, saves the document with a given name and prints it to the default printer before finally closing Notepad. We will look at some of the issues that could crop up – for example what to do if the file already exists and different methods of dealing with this.

3.4 Using Macro Scheduler

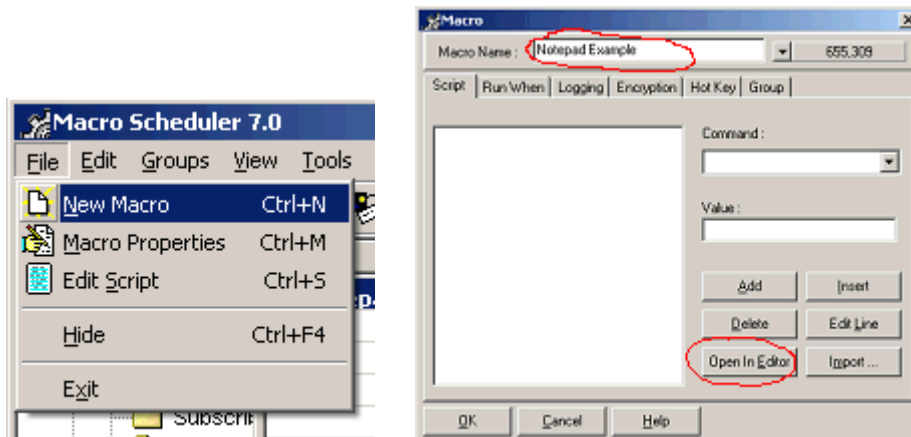
This guide is concerned with introducing script development and does not concern itself too much with using the Macro Scheduler interface. Refer to [Using Macro Scheduler](#) ^[2] for a tour around the Macro Scheduler interface.

However, we will assume that you are using the Macro Scheduler editor to build the example script here. Do the following to create the macro and start editing:

1. Double click on the Macro Scheduler icon in your system tray.
2. Select "New Macro" from the File menu.
3. Enter a name for this new Macro.
4. Click the 'Open in Editor' button to open the script editor.

Double click on the gear icon to view Macro Scheduler's control panel.

Select File/New Macro to create a new macro, enter the name of the macro in the macro properties dialog and then click the 'Open in Editor' button.



To get back to the editor after saving the macro you can highlight the macro in the list and just click the 'Edit Script' tool bar button, or select from the File menu or context sensitive (right click on macro list) menu. In Macro Scheduler 7.0 the editor can also be used as a standalone editor and a shortcut is available in the Macro Scheduler program group or on the start programs menu.

While in the editor pressing F1 will display the help topic for the command on the current line or the highlighted command. There is also access to the useful command locator, which makes finding the function or command you need easy. Help is also available from the command locator. Just press F1 while a command is selected.

3.5 First Steps

Usually, we'll already be familiar with the process that needs to be automated. This is important as Macro Scheduler works by simulating user input, and since we have to tell it what keys to press, and which apps to activate, then we need to know what these are in the first place. It is also advisable to try to avoid mouse events, as these are subject to window placement, which can vary. Although there are ways round this, such as running apps maximised, and using Macro Scheduler's relative mouse move commands, it is usually possible to use the alternative keyboard shortcuts, and in most cases this improves the reliability and efficiency of scripts.

Therefore it is always helpful to run through the process manually, making a note of the key presses needed to perform each action. Note down titles of the windows that appear, how long certain actions take and if there is anything that signifies that an action has completed. The list of key presses and windows is the basis of our script.

It is probably best to break the process up into manageable chunks. In our example the first thing we need to do is to run Notepad and wait for it to be ready. Then send the lines of text. We can safely develop this portion of the script before we begin to consider the next sequence.

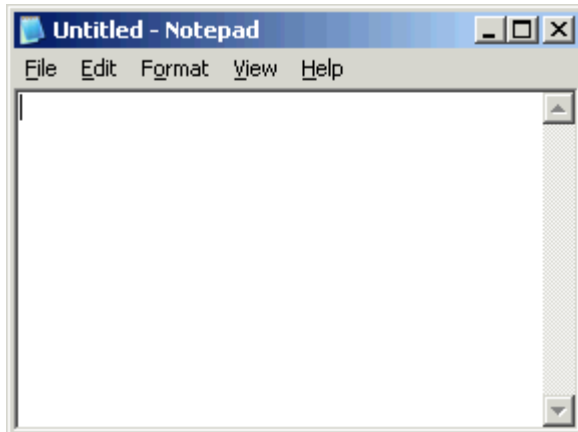
3.6 Building The Script

Let's make a list of what our script needs to do:

- Run Notepad
- Wait for Notepad to be ready for input
- Send the lines of text
- Save the document
- Print the document
- Close Notepad

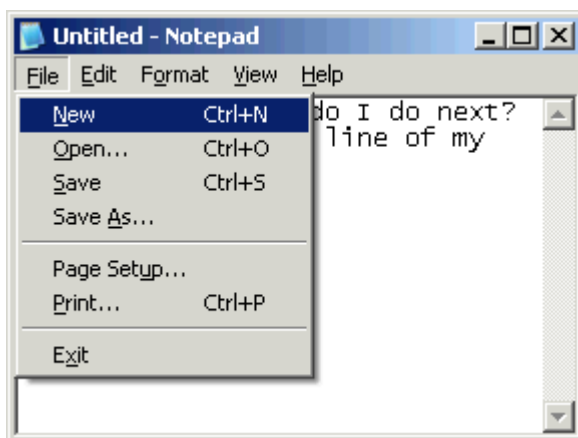
Steps 4 and 5 will probably need to be further broken down – they will involve some key sequences and dialogs. Let's run through the process manually.

This is what Notepad looks like when we run it. The first thing to note is the Window title:



"Untitled – Notepad"

Notice that focus is already in the editor portion (i.e. the editor is already active), so to send our lines of text we can just go ahead and start typing.



So I've written some text and now I want to save the document. Let's see what the key sequence is for this.

ALT-F opens the File menu.
ALT-A will perform the Save As.. action.

Therefore ALT-FA will do this in one go.

Remember: ALT-FA

The ALT keystrokes that issue a command are underlined on Windows. So, when Notepad is opened, the first item is marked File – this tells us that if we press 'ALT' and 'F', the File menu will appear.

So, back to our example. When we press ALT-FA we get the "Save As" dialog box. Note that the window title is "Save As" and the focus is already in the filename box – this will be helpful.

So when we do this manually, we are subconsciously waiting for the Save As dialog box to appear and then we can type the filename. Normally most of us would probably use our mouse to locate the folder we want to save the document in and then provide a filename. However, we can, of course, type the full path and filename directly into the filename box and just press Enter. Assuming we were able to get this right first time it would work. Well, of course, our script will manage this happily because we're going to define the filename up front.

Notice that the Save button on the Save As dialog is the default object – this means that just Pressing Enter will save the file.

So we have determined the actions required for the first part of our process:

- Run Notepad
- Wait for the window "Untitled – Notepad"
- Send our lines of text
- Press ALT-FA
- Wait for the window "Save As"
- Send our filename
- Press Enter

So let's turn this into a Macro Scheduler script. Read through the following sections in turn.

[Run Notepad](#) ³¹
[Wait for the window "Untitled – Notepad"](#) ³¹
[Send our lines of text](#) ³²
[Press ALT-FA](#) ³³
[Wait for the window "Save As"](#) ³³
[Send our filename](#) ³³
[The Script So Far](#) ³⁵
[Printing the Document](#) ³⁶
[Closing Notepad](#) ³⁷

3.6.1 Run Notepad

There are a number of ways to run Notepad. I expect most of us will use our mouse to click on the Start button, and find the shortcut in our Programs list. All this does is execute a shortcut, which in turn runs Notepad (more technically, it runs the Notepad executable – notepad.exe). Since the shortcut could appear anywhere and no doubt appears in different places for different people depending on their preferences, and since all it does is run Notepad, we should bypass the shortcut and run Notepad directly, in the same way that we would if using the Run command in Windows Start menu.

Furthermore, as the Notepad program lives in the System folder, Windows knows where to find it so we don't even have to specify the path to it.

So in Macro Scheduler we just have to write the following line:

```
Run Program>Notepad.exe
```

In many other automation scenarios, however, this won't be the case, and sometimes it is necessary to switch to an application's directory before running the application. In which case we would do something like this:

```
Change Directory>c:\windows\system32  
Run Program>c:\windows\system32\notepad.exe
```

This is totally unnecessary for Notepad, but there's no harm demonstrating how you might have to execute your own applications.

3.6.2 Wait for the window "Untitled – Notepad"

Macro Scheduler gives us two very useful commands: WaitWindowOpen and WaitWindowClosed.

As their names suggest these wait for the specified window to be fully open, or closed respectively before script execution continues. The argument given to these commands can be the exact window title, or a portion of the window title followed by an asterisk. The latter tells Macro Scheduler to search all windows until it finds a window whose title *contains* the specified text regardless of case and stop at the first one it finds.

In our case we know the title is always "Untitled – Notepad" after running Notepad, so we can write the next line of our script:

```
WaitWindowOpen>Untitled – Notepad
```

3.6.3 Send our lines of text

For this we use the Send/Character Text command, which is commonly abbreviated to just Send:

```
Send>some text
```

To send more than one line we would manually type the text and press enter to move on to the next line, so this would look like this:

```
Send>This is the first line of my notepad file
Press Enter
Send>This is line two
Press Enter
Send>This is the third and final line.
```

But let's be a bit cleverer. Let's create three lines of text and assign them to variables using the Let command:

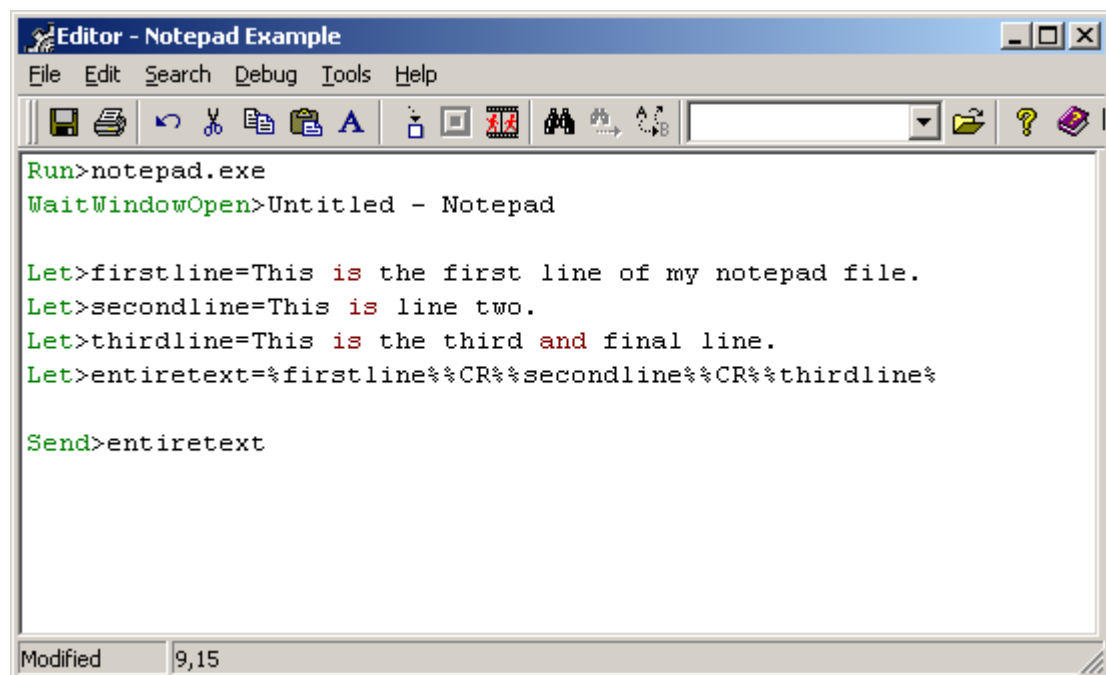
```
Let>firstline=This is the first line of my notepad file.
Let>secondline=This is line two.
Let>thirdline=This is the third and final line.
```

Now let's join them together with a carriage return character to ensure the line breaks. CR is a Macro Scheduler system variable for a carriage return:

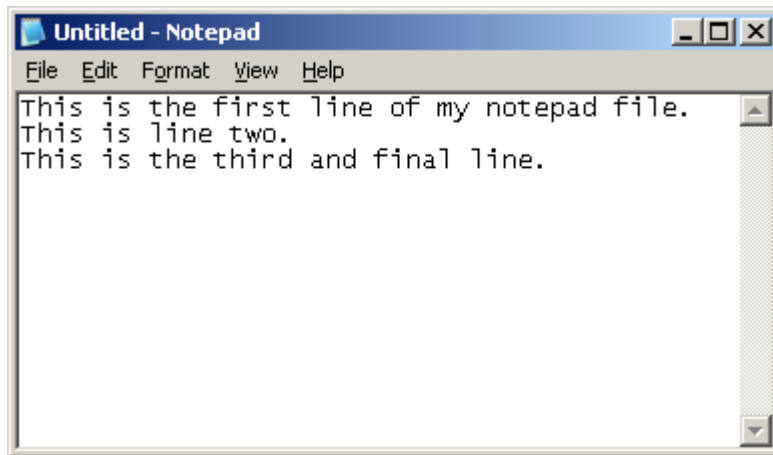
```
Let>entiretext=%firstline%%CR%%secondline%%CR%%thirdline%
Send>entiretext
```

The % symbols tell Macro Scheduler that the value within the % symbols is a variable, so it creates a new variable called 'entiretext' made up of the variables firstline, CR, secondline, CR and thirdline all joined together.

Let's test the script so far. Close any open Notepad windows and run the script:



Hopefully you ended up with a new Notepad window appearing on your desktop looking like this:



3.6.4 Press ALT-FA

When we press the ALT key we will hold it down, send the other keys with it and then release the ALT key again. So we actually do this:

```
Press ALT
Send the F and A characters
Release ALT
```

In Macro Scheduler this is almost exactly as written above:

```
Press ALT
Send>fa
Release ALT
```

I prefer to use lower case characters when I am issuing keystrokes. It sometimes avoids issues where an upper case character can be interpreted as having the Shift key down at the same time – think about it ...

Remember the Release ALT. With the ALT, SHIFT and CTRL keys we should always remember to release them after pressing them. They are used in conjunction with other keys. If we forget to Release the ALT key, subsequent key sends will act as if the ALT key was also pressed because it has yet to be released.

3.6.5 Wait for the window "Save As"

We've dealt with this before, so without further ado:

```
WaitWindowOpen>Save As
```

3.6.6 Send our filename

When the Save As dialog appears, the filename box has the keyboard focus, so all we need to do is send our filename, and it will be filled in the right place. If we needed to move keyboard focus a good tip is to use Press Tab to move from object to object.

What we should do is set a variable at the top of the script with our filename, so that we can modify that easily:

```
Let>filename=c:\my documents\sample.txt
```

Assuming we have done this we could just now send the filename and press enter to save it as follows:

```
Send>filename  
Press Enter
```

However, what if this file already exists? We would get an error and we would have to script the error window and make changes as necessary.

It would be easier to avoid this and check the existence of the file first. Macro Scheduler has a command called `IfFileExists`. But what do we do if it does exist? We could just delete it? This is fine in this instance because I know that this file is just an example and I'm never going to confuse it with a valid alternative file. But if we wanted to play safe we could give it a new filename. We could use the date and time, and/or a random number on the filename and keep performing the existence check before saving the file.

To keep things simple at this stage we'll just delete the file if it already exists. `IfFileExists` is given the filename and a label to branch to if the file exists. If it doesn't exist it continues on to the next line. So we should do this:

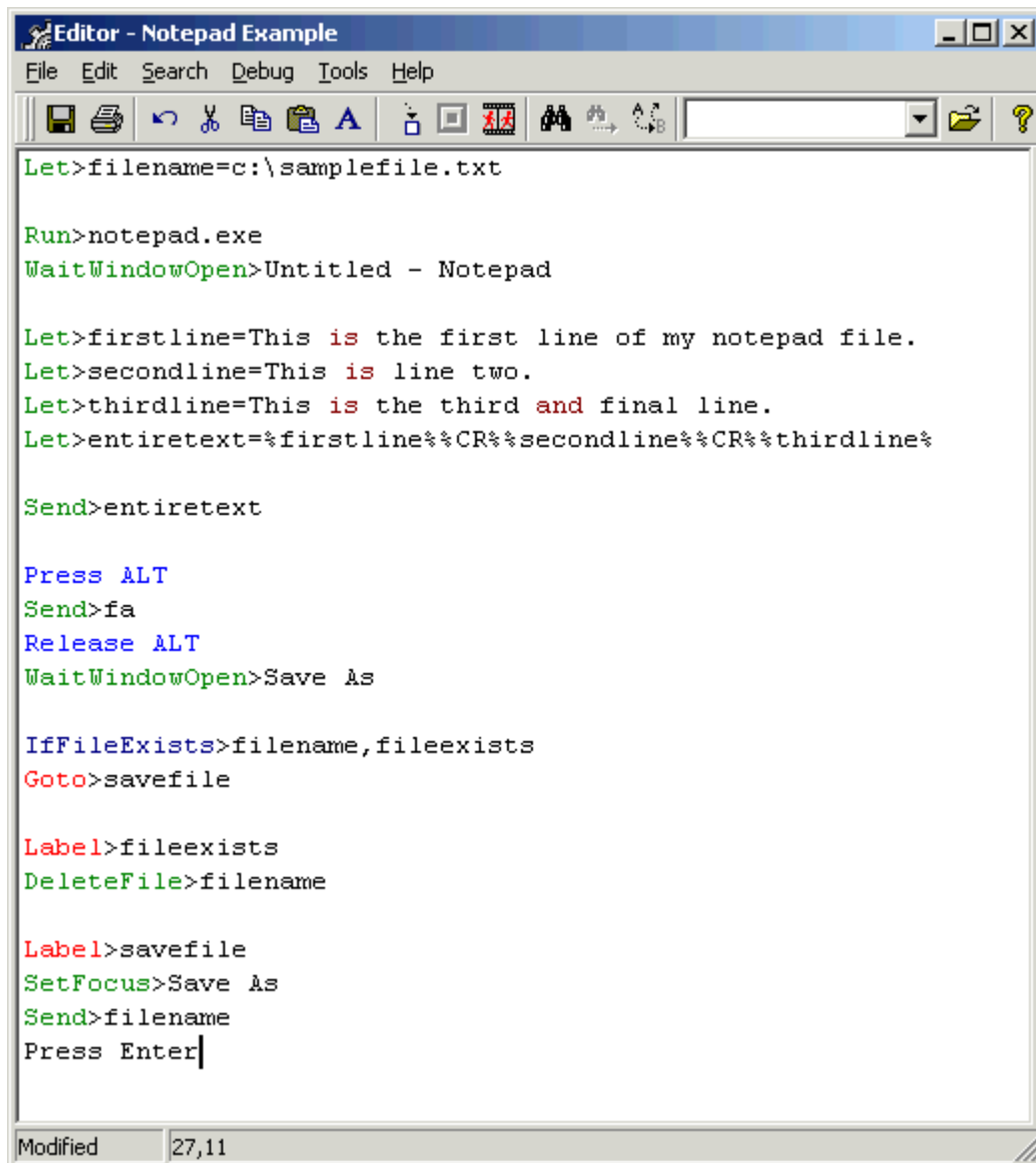
```
IfFileExists>filename,fileexists  
Goto>savefile
```

```
Label>fileexists  
DeleteFile>filename
```

```
Label>savefile  
SetFocus>Save As  
Send>filename  
Press Enter
```

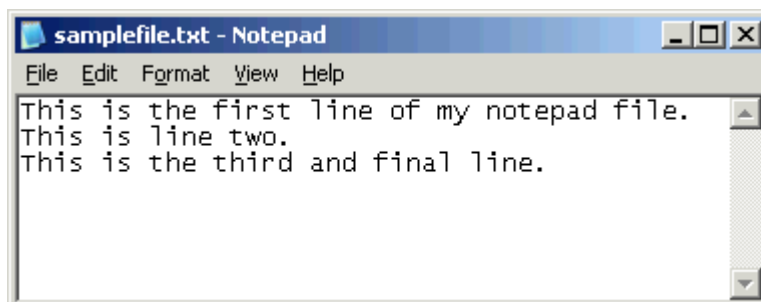
The last two lines here are the two that send the filename and press enter to save it. We've added a `SetFocus` line just for good measure. It shouldn't be needed as the `Save As` dialog should already be active, but for the sake of example it has been added. There is never any harm in ensuring the correct window has the focus before we send any keystrokes. I've put a label above so that we can jump to this bit of the script – it's appropriately been called 'savefile'. Likewise I've created a section called 'fileexists' which performs the `DeleteFile` command, which simply deletes the specified file. `IfFileExists` will jump to 'fileexists' if the file exists, or continue otherwise, in this case on to the `Goto` command which makes it jump straight to the 'savefile' section.

3.6.7 The Script So Far



Note the variable 'filename' declared at the top.

After running the script you should see a few flashes as the windows and dialogs appear and disappear faster than you can type and the end result should look something like:



Note the new window title – "samplefile.txt – Notepad" – proof that the file has been saved.

If you close Notepad and run the script again the same thing will happen. No errors, because the

filename has been deleted and there was no clash.

3.6.8 Printing the Document

Having worked through sending keystrokes to initiate the Save As dialog, printing shouldn't be very much different. If we look at the File menu we can see that 'P' does the print. In later versions of Windows Notepad CTRL-P is also a shortcut for printing, so we can use either ALT-FP or CTRL-P.

Remember it is always sensible to focus the app we want to send keystrokes to. The Notepad window title has changed – it is no longer 'Untitled – Notepad' as we have saved the document.

We could give the new window title, but what if we can't predict easily what the filename is going to be? Assuming no other Notepad windows are open we can use the wildcard as explained previously when discussing WaitWindowOpen. So we can do this:

```
SetFocus>notepad*  
Press ALT  
Send>fp  
Release ALT
```

What happens when we send ALT-FP to Notepad depends on which version of Notepad you have. In earlier versions Notepad just went straight ahead and printed the document. But in later versions it displayed the Print dialog. So please try it manually first to decide which version you have. This reiterates the importance of working through the process manually to understand what needs to be automated.

If the Print dialog does not appear and Notepad just starts printing the document then skip directly to Step 2.

Step 1

If the Print dialog appears, we should wait for it to appear – it's window title is 'Print' so we would do this:

```
WaitWindowOpen>Print
```

Now, we just want to print to the default printer. As it is the default it is already selected and we just have to press enter to activate the default 'Print' button:

```
Press Enter
```

And the Print dialog now disappears.

Step 2

What happens next? Well, if you look closely you will see that a new, smaller dialog, entitled 'Notepad' appears indicating that it is printing. With the small document we are dealing with here it will flash up rather quickly, but we need to be aware of it. Our macro needs to wait for this to disappear before we can finally close Notepad.

So after pressing enter on the print dialog we should wait for this little status window to appear, and then disappear again before we know Notepad is ready again for input. We can do this with a WaitWindowOpen followed by a WaitWindowClosed:

```
WaitWindowOpen>Notepad  
WaitWindowClosed>Notepad
```

Don't be tempted to just use a WaitWindowClosed command on its own, as this pauses until the

specified window is not present, and immediately after pressing Enter, the window may not yet be present – your script's so fast that the little dialog has not had a chance to appear – so the script would continue. We should wait for it to open and then wait for it to disappear. This is the only sure-fire method.

3.6.9 Closing Notepad

So we've printed the document. Now all we need to do is close Notepad. How do we close Notepad with keystrokes? There are a number of ways. Notice that under the File menu is the usual Exit option. If we use ALT-F to open the file menu the Exit option has 'x' underlined, so ALT-FX will close Notepad.

In Windows ALT-F4 will also close a window. Macro Scheduler also offers a CloseWindow command, which will probably do the job (though this doesn't necessarily close all applications as not all programs process "close window" messages in the same way).

Let's use ALT-FX to simulate true user input:

```
SetFocus>Notepad*  
Press ALT  
Send>fx  
Release ALT
```

And Notepad should disappear.

3.7 The Complete Script

So let's review the script. I have added comments to document the script – to make it more readable and easy to follow. You can use almost anything that isn't a recognised script command for comments. There is a reserved word called Remark which can be used:

```
Remark>This is a comment
```

But I've used // to indicate my comments. Some people like to use **. Use what you feel comfortable with – as long as it isn't a recognised script command – or it will try to execute your comments. Which could be interesting!

Remember to modify the section of the script that does the printing depending on whether or not your Notepad displays the Print dialog. There are clever ways that we could make the script generic to cope with both cases, but that's for a later discussion. But if you're interested look at the WW_TIMEOUT variable that you can set for WaitWindowOpen, and the IfWindowOpen command.

If you have any problems running this script you may want to slow it down a bit. We have made it run as fast as it possibly can, but some systems react too slowly for windows to be ready in time for the script. Consider adding Wait statements at key places, such as before a SetFocus, or before sending keystrokes:

```
//This will wait 2 seconds.  
Wait>2
```

```
//Set the name of our file  
Let>filename=c:\samplefile.txt  
  
//Start Notepad  
Run>notepad.exe  
WaitWindowOpen>Untitled - Notepad  
  
//Construct the text
```

```
Let>firstline=This is the first line of my notepad file.
Let>secondline=This is line two.
Let>thirdline=This is the third and final line.
Let>entiretext=%firstline%%CR%%secondline%%CR%%thirdline%

//Send the text to Notepad
Send>entiretext

//Save As
Press ALT
Send>fa
Release ALT
WaitWindowOpen>Save As

//Delete the file if it already exists
IfFileExists>filename,fileexists
Goto>savefile

Label>fileexists
DeleteFile>filename

//Send the filename to the Save As dialog
Label>savefile
SetFocus>Save As
Send>filename
Press Enter

//Print the file
SetFocus>notepad*
Press CTRL
Send>p
Release CTRL

//Remove the next two lines if your version of
//Notepad does not display the Print dialog
WaitWindowOpen>Print
Press Enter

//Wait for the print status dialog to finish
WaitWindowOpen>Notepad
WaitWindowClosed>Notepad

//Now close Notepad
SetFocus>Notepad*
Press ALT
Send>fx
Release ALT
```

3.8 Where To Go From Here

In this guide we have met the following common commands:

- Run Program
- WaitWindowOpen
- WaitWindowClosed
- SetFocus
- Send Character/Text (Send)
- Press ...
- Label and Goto
- Wait

These commands will always be found in Windows automation scripts. They are essential for

simulating user input, and we have seen how just these commands can be used to automate a Windows application. Other useful commands for windows manipulation include the mouse event functions:

- MouseMove
- MouseMoveRel
- LClick
- LDbClick
- RClick
- RDbClick

For further detail on these commands please refer to the Command Reference.

The Command Reference also gives examples for each command.

For further examples and tips you should check out the sample scripts that come with Macro Scheduler and the Scripts & Tips archive at <http://www.mjtnet.com/scripts.htm>

Further scripting articles and tutorials can be found at <http://www.mjtnet.com/blog>

Happy Scripting!

Command Reference

Part



IV

4 Command Reference

4.1 Complex Expressions

Syntax

In Macro Scheduler all complex functions should be contained within curly braces ("{" and "}"). Variables should be enclosed within % symbols and literal strings should be enclosed within double quotes (""). Parameters are separated by commas (,).

Complex expressions are supported in Macro Scheduler's [IF](#) ⁷⁷ statements and [LET](#) ¹¹⁷ statement, and since version 9.0 can now also be included within all other commands and function calls in the place of regular variables.

Operators

[Arithmetic Operators](#) ⁴¹

[Logical Operators](#) ⁴¹

[String Operators](#) ⁴²

[Relational Operators](#) ⁴²

Functions

[Arithmetic Functions](#) ⁴²

[String Functions](#) ⁴³

[Conditional Expressions](#) ⁴³

Examples

[Complex Expression Examples](#) ⁴⁴

4.1.1 Arithmetic Operators

Binary arithmetic operators:

^	exponent
+	addition
-	subtraction
*	multiplication
/	division
div	integer division
mod	remainder

Unary arithmetic operators:

+	sign positive identity
-	sign negation

4.1.2 Logical Operators

Bitwise logical operators

not	bitwise negation
and	bitwise and
or	bitwise or
xor	bitwise xor
shl	shift left
shr	shift right

Boolean logical operators

not	negation
and	logical and
or	logical or
xor	logical xor

4.1.3 String Operators**String operators**

+	concatenation
---	---------------

4.1.4 Relational Operators

=	equal
<>	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

4.1.5 Arithmetic Functions
[Examples](#) 
function Trunc(X: Extended): Integer

The Trunc function truncates a float-type value to an integer-type value. X is a float-type expression. Trunc returns an Integer value that is the value of X rounded toward zero.

function Round(X: Extended): Integer

The Round function rounds a float-type value to an integer-type value. X is a float-type expression. Round returns a Longint value that is the value of X rounded to the nearest whole number. If X is exactly halfway between two whole numbers, the result is the number with the greatest absolute magnitude.

function Abs(X): Float

The Abs function returns the absolute value of the argument. X is an integer-type or float-type expression.

function ArcTan(X: Float): Float

ArcTan calculates the arctangent of the given number. Calculate other trigonometric functions using Sin, Cos, and ArcTan in the following expressions

$$\text{Tan}(x) = \text{Sin}(x) / \text{Cos}(x)$$

$$\text{ArcSin}(x) = \text{ArcTan}(x / \sqrt{1 - \text{sqr}(x)})$$

$$\text{ArcCos}(x) = \text{ArcTan}(\sqrt{1 - \text{sqr}(x)} / x)$$
function Cos(X: Float): Float

The Cos function returns the cosine of the angle X, in radians.

function Exp(X: Float): Float

Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms.

function Frac(X: Float): Float

The Frac function returns the fractional part of the argument X. X is a float-type expression. The result is the fractional part of X; that is: $\text{Frac}(X) = X - \text{Int}(X)$.

function Int(X: Float): Float

X is a float-type expression. The result is the integer part of X; that is, X rounded toward zero.

function Ln(X: Float): Float

The Ln function returns the natural logarithm ($\text{Ln}(e) = 1$) of the float-type expression X.

function Pi: Float

Use Pi in mathematical calculations that require pi, the ratio of a circle's circumference to its diameter. Pi is approximated as 3.1415926535897932385.

function Sin(X: Float): Float

The Sin function returns the sine of the argument. X is a float-type expression. Sin returns the sine of the angle X in radians.

function Sqr(X: Float): Float

The Sqr function returns the square of the argument. X is a floating-point expression. The result, of the same type as X, is the square of X, or $X \times X$.

function Sqrt(X: Float): Float

X is a floating-point expression. The result is the square root of X.

function Power(Base, Exponent: Float): Float

The Power function raises Base to any power. For fractional exponents or exponents greater than MaxInt, Base must be greater than 0.

4.1.6 String Functions

[Examples](#) 

function Upper(S: string): string

The Upper function returns a string containing the same text as S, but with all 7-bit ASCII characters between 'a' and 'z' converted to uppercase.

function Lower(S: string): string

Lower returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'.

function Copy(S: string; Index, Count: Integer): string

The Copy function returns a substring of a string. S is a string-type expression. Index and Count are integer-type expressions. Copy returns a string containing Count characters starting at S[Index]. If Index is larger than the length of S, Copy returns an empty string. If Count specifies more characters than are available, the only the characters from S[Index] to the end of S are returned.

function Pos(Substr: string; S: string): Integer

Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions. Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos ignores case-insensitive matches. If Substr is not found, Pos returns zero.

function Length(S: string): Integer

The Length function returns the number of characters actually used in the string S.

4.1.7 Conditional Functions

[Examples](#) 

function If(Condition: Boolean, TrueResult, FalseResult): ResultType

Condition is a Boolean expression. When the function is evaluated, it returns TrueResult if Condition else FalseResult. TrueResult and FalseResult need not be of the same type and result type of the IF expression may change depending on Condition.

4.1.8 Examples

Numeric Functions:

```
Let>r={round(34.6)}
Let>ans={cos(340)}
```

Using numeric variables in complex expressions we enclose variables in % symbols:

```
Let>r=34.6
Let>r={round(%r%)}
```

String Functions:

In Complex expressions string literals must be enclosed in double quotes ("). Variables are just enclosed with % symbols:

```
Let>email=fred@someserver.com
Let>at={Pos("@",%email%)}
```

Here at equals 5 as it is the 5th character in fred@someserver.com

Let's now use the copy function to extract everything before the @ sign:

```
Let>namepart={copy(%email%,1,%at%-1)}
```

Conditional Functions:

```
Let>email=fred@someserver.com
Let>IsItAnEmail={if(pos("@",%email%)>0,"yes","no")}
```

Of course we could do:

```
Let>email=fred@someserver.com
Let>at={Pos("@",%email%)}
Let>IsItAnEmail={if(%at%>0,"yes","no")}
```

Here IsItAnEmail would be set to "yes".

4.2 Clipboard Commands

4.2.1 GetClipboard

GetClipboard>result_variable

Retrieves the contents of the clipboard as text and places it in the specified variable.

Abbreviation : [GCB](#)

See also : [PutClipboard](#)⁴⁵, [WaitClipboard](#)⁴⁵

Example

```
GetClipboard>WhatsInTheClipboard
```

4.2.2 PutClipboard

PutClipboard>SomeValue

Places the specified text onto the clipboard. A variable may be used, or a literal value.

Abbreviation : [Put](#)

See also : [GetClipboard](#)^[44], [WaitClipboard](#)^[45]

Example

```
PutClipboard>Hello World !!
```

4.2.3 WaitClipboard

WaitClipboard

Waits for the clipboard to be ready to access (not open by another application).

Abbreviation: [WCB](#)

See Also: [GetClipboard](#)^[44], [PutClipboard](#)^[45]

4.3 Console App Funtions

4.3.1 SOWrite

SOWrite>text

Writes text to STDOUT. For compiled console EXEs only. This function does nothing in non-compiled scripts or compiled scripts that have not been compiled as console applications.

Abbreviation: [SOW](#)

See Also: [SOWriteLn](#)^[45]

4.3.2 SOWriteLn

SOWriteLn>text

Writes line of text to STDOUT. For compiled console EXEs only. This function does nothing in non-compiled scripts or compiled scripts that have not been compiled as console applications.

Abbreviation: [SOL](#)

See Also: [SOWrite](#)^[45]

4.4 Database Commands

4.4.1 DBClose

DBClose>database_reference

Closes a database connection established by the DBConnect command.

Abbreviation: [DBX](#)

See also: [DBConnect](#)^[46], [DBExec](#)^[46], [DBQuery](#)^[47]

For more advanced database manipulation use VBScript. See:
<http://www.mjtnet.com/blog/2006/02/20/accessing-databases/>

Example

```
Let>str=Driver={MySQL ODBC 3.51
Driver};Server=someserver.com;Port=3306;Database=example;User=admin;Password=xxx
x;Option=3;
DBConnect>str,dbH

Let>SQL=delete from mytable where ID=1234
DBExec>dbH,SQL,result

DBCclose>dbH
```

4.4.2 DBConnect

DBConnect>connection_string,database_reference

Connects to a database via a system data source or ADO/ODBC connection string.

Accepts an ADO/ODBC connection string or system data source name.

Returns a reference to the database, to be used in DBClose, DBExec and DBQuery commands.

Use DBClose to close the database.

For more advanced database manipulation use VBScript. See:
<http://www.mjtnet.com/blog/2006/02/20/accessing-databases/>

Abbreviation: **DBC**

See also: [DBCclose](#)^[45], [DBExec](#)^[46], [DBQuery](#)^[47]

Example

```
Let>str=Driver={MySQL ODBC 3.51
Driver};Server=someserver.com;Port=3306;Database=example;User=admin;Password=xxx
x;Option=3;
DBConnect>str,dbH

Let>SQL=delete from mytable where ID=1234
DBExec>dbH,SQL,result

DBCclose>dbH
```

4.4.3 DBExec

DBExec>database_reference,SQL_Statement,result

Executes an SQL statement. Use DBExec to execute SQL statements that do not return a recordset. E.g. DELETE, INSERT and UPDATE queries.

For SELECT queries use DBQuery to return a recordset array.

Accepts a database reference created by a call to DBConnect, and an SQL_Statement. Specify a result variable to return the number of rows affected by the SQL statement.

For more advanced database manipulation use VBScript. See:
<http://www.mjtnet.com/blog/2006/02/20/accessing-databases/>

Abbreviation: **DBE**

See also: [DBConnect](#)^[46], [DBCclose](#)^[45], [DBQuery](#)^[47]

Example

```
Let>str=Driver={MySQL ODBC 3.51
Driver};Server=someserver.com;Port=3306;Database=example;User=admin;Password=xxx
x;Option=3;
DBConnect>str,dbH

Let>SQL=delete from mytable where ID=1234
DBExec>dbH,SQL,result

DBCclose>dbH
```

4.4.4 DBQuery

DBQuery>database_reference,SQL_Statement,recordset_array,num_recs,num_fields[,fieldnames]

Use DBQuery to perform a SQL query which returns a recordset, such as a SELECT statement.

database_reference is a reference variable returned by a previous call to DBConnect. Specify the SQL statement in SQL_Statement.

Specify a variable name in recordset_array in which to return the recordset. The array is constructed in the format recordset_rownum_fieldnum. So a recordset with 2 rows and 3 fields per row would create an array as follows:

```
variablename_1_1
variablename_1_2
variablename_1_3
variablename_2_1
variablename_2_2
variablename_2_3
```

Specify variables for num_recs and num_fields to store the number of records and number of fields returned in the recordset.

Optionally set fieldnames to 1 to return the names of the fields rather than the index. This would return an array like:

```
variablename_1_FIELD1
variablename_1_FIELD2
variablename_1_FIELD3
variablename_2_FIELD1
variablename_3_FIELD2
variablename_3_FIELD3
```

Where FIELD1, FIELD2, FIELD3 were the actual field names of the fields returned.

For more advanced database manipulation use VBScript. See:
<http://www.mjtnet.com/blog/2006/02/20/accessing-databases>

Abbreviation: **DBQ**

See also: [DBConnect](#)^[46], [DBCclose](#)^[45], [DBExec](#)^[46]

Example

```
//Connect to Datasource
Let>str=Driver={MySQL ODBC 3.51
```

```

Driver}};Server=someserver.com;Port=3306;Database=example;User=admin;Password=xxxx;Option=3;
DBConnect>str,dbH

//Perform SELECT query
Let>SQL=select * from customers where custID='abc123'
DBQuery>dbh,SQL,CUSTOMERS,numrecs,numfields

//loop through returned recordset
Let>r=0
Repeat>r
  Let>r=r+1
  Let>f=0
  Repeat>f
    Let>f=f+1
    Let>this_field=CUSTOMERS_%r%_%f%
    Message>this_field
    Wait>0.5
  Until>f=numfields
Until>r=numrecs

//Close database connection
DBCclose>dbh

```

4.5 Date/Time Commands

4.5.1 Day

Day>result

Returns the current day number of the month in the specified variable.

See also : [Month](#)^[50], [Year](#)^[51], [GetDate](#)^[49], [GetTime](#)^[49]

Example

```

Day>the_day
Month>the_month
Year>the_year
Message>The date is : %the_day% - %the_month% - %the_year%

```

4.5.2 DayOfWeek

DayOfWeek>result

Returns the current week day number, starting with Sunday as day 1, and ending on Saturday with day 7.

Abbreviation : [DOW](#)

See also : [Day](#)^[48], [Month](#)^[50], [Year](#)^[51], [GetDate](#)^[49], [GetTime](#)^[49]

Example

This example displays the current day as a proper day name. It also shows how to use variables in a Goto command.

```

DayOfWeek>result
Goto>Day%result%

Label>Day1
Let>DayString=Sunday

```



```
Goto>Continue

Label>Day2
Let>DayString=Monday
Goto>Continue

Label>Day3
Let>DayString=Tuesday
Goto>Continue

Label>Day4
Let>DayString=Wednesday
Goto>Continue

Label>Day5
Let>DayString=Thursday
Goto>Continue

Label>Day6
Let>DayString=Friday
Goto>Continue

Label>Day7
Let>DayString=Saturday
Goto>Continue

Label>Continue
MessageModal>DayString
```

4.5.3 GetDate

GetDate>result

Returns the current date in the specified variable. The format of the date depends on the regional settings of the system.

Abbreviation : **GDT**

See also : [GetTime](#)^[49], [Year](#)^[51], [Month](#)^[50], [Day](#)^[48], [FileDate](#)^[60]

Example

```
GetDate>date
Let>msg=The Date Is :
ConCat>msg,date
Message>msg
```

4.5.4 GetTime

GetTime>result

Returns the current time in the specified variable. The format of the time depends on the regional settings of the system.

Abbreviation : **GTM**

See also : [GetDate](#)^[49], [Year](#)^[51], [Month](#)^[50], [Day](#)^[48], [FileDate](#)^[60]

Example

```
GetTime>time
Let>msg=The Time Is :
ConCat>msg,time
Message>msg
```

4.5.5 Hour

Hour>result

Returns the seconds portion of the current time in the specified variable.

See also : [Min](#)^[50], [Sec](#)^[50], [GetDate](#)^[49], [GetTime](#)^[49]

Example

```
Sec>Seconds
Min>Minutes
Hour>Hour
Message>The time is : %Hour% : %Minutes% : %Seconds%
```

4.5.6 Min

Min>result

Returns the minutes portion of the current time in the specified variable.

See also : [Sec](#)^[50], [Hour](#)^[50], [GetDate](#)^[49], [GetTime](#)^[49]

Example

```
Sec>Seconds
Min>Minutes
Hour>Hour
Message>The time is : %Hour% : %Minutes% : %Seconds%
```

4.5.7 Month

Month>result

Returns the current month number in the specified variable.

Abbreviation : [Mon](#)

See also : [Day](#)^[48], [Year](#)^[51], [GetDate](#)^[49], [GetTime](#)^[49]

Example

```
Day>the_day
Month>the_month
Year>the_year
Message>The date is : %the_day% - %the_month% - %the_year%
```

4.5.8 Sec

Sec>result

Returns the seconds portion of the current time in the specified variable.

See also : [Min](#)^[50], [Hour](#)^[50], [GetDate](#)^[49], [GetTime](#)^[49]

Example

```
Sec>Seconds
```

```
Min>Minutes
Hour>Hour
Message>The time is : %Hour% : %Minutes% : %Seconds%
```

4.5.9 Year

Year>result

Returns the current year in the specified variable.

See also : [Month](#)^[50], [Day](#)^[48], [GetDate](#)^[49], [GetTime](#)^[49]

Example

```
Day>the_day
Month>the_month
Year>the_year
Message>The date is : %the_day% - %the_month% - %the_year%
```

4.6 DDE Commands

4.6.1 DDEPoke

DDEPoke>Server,Topic,Item,Data

Pokes data to the given DDE server. The server's DDE topic and item must be specified.

Abbreviation : [DPK](#)

See also : [DDERequest](#)^[51]

Example

```
DDEPoke>MyServer,System,Item1,Testing 123
```

4.6.2 DDERequest

DDERequest>Server,Topic,Item,Result,Timeout

Requests data from a DDE server. The data returned is stored in the Result variable. If the conversation does not complete within the Timeout value specified, Result will contain 'DDE_TIMEDOUT'. The Timeout value is in seconds.

Abbreviation : [DRQ](#)

See also : [DDEPoke](#)^[51]

Example

The following example gets the URL and window title from Netscape.

```
DDERequest>Netscape,WWW_GetWindowInfo,0xFFFFFFFF,ret,10
Message>ret
```

The DDERequest command can also be used to open a web page in Netscape :

```
DDERequest>Netscape,WWW_OpenUrl,www.mjtnet.com,ret,0
```

4.7 Dialogs

4.7.1 CloseDialog

CloseDialog>DialogName

Closes non-modal dialog specified by DialogName

Abbreviation: [CDG](#)

See also: [GetDialogAction](#)^[55], [ResetDialogAction](#)^[56]

See [Dialog](#)^[52] for details on creating dialogs and an example.

4.7.2 Dialog

Dialog>DialogName

Marks the start of a dialog block. Use [EndDialog](#)^[55] to end the dialog block.

A dialog block is used to create a dialog which can be displayed later in the script with the [Show](#)^[57] command. Dialogs can be modal or non-modal.

The [Dialog Designer](#)^[15] can be used to design dialog layouts and edit and create dialog blocks.

Set the properties and create objects within the Dialog block. Properties are declared within the dialog block in the following format:

property_name=values

The following simple properties take one value. All these properties are set by the [Dialog Designer](#)^[15] so manual coding is normally not required.

Caption	Title of the dialog - appears in the title bar.
Top	Y coordinate of dialog - the position of the top edge on the screen. Can be set to CENTER
Left	X coordinate of dialog - the position of the left edge on the screen. Can be set to CENTER
Width	Width of the dialog.
Height	Height of the dialog.
Max	1=Maximize button enabled, 0=No maximize button
Min	1=Minimize button enabled, 0=No minimize button
Close	1=Close button enabled, 0=No close button
Resize	1=Form can be resized, 0=Form not resizable

The following create objects on the dialog and take more than one value.

Most of these objects and their properties can be created and set using the [Dialog Designer](#)^[15] and no coding is required, so do not worry too much about the syntax below as the code is created for you. Only MainMenu and FileBrowse objects must be coded manually.

Object	Values	Comments
Button	<i>Caption,Left,Top,Width,Height,ModalResult[,ImageFile/Data[,hint]]</i>	<i>ModalResult</i> is a numeric value which is set to the result variable of the Show ^[57] or GetDialogAction ^[58] commands if this button was pressed. Optionally <i>ImageFile</i> can be the path to a .bmp bitmap file to display on the button. The lower left pixel of the bitmap is used as the "transparent" colour.
Edit	<i>Name,Left,Top,Width,Preset-Text[,PasswordChar]</i>	Creates variable defined by <i>DialogName.Name</i> set to the text entered by the user. <i>Preset-Text</i> can be used to pre-set the contents of the box.
Memo	<i>Name,Left,Top,Width,Height,Preset-Text</i>	Creates variable defined by <i>DialogName.Name</i> set to the text entered by the user. <i>Preset-Text</i> can be used to pre-set the contents of the box.
Checkbox	<i>Name,Caption,Left,Top,Width,Checked</i>	Creates variable defined by <i>DialogName.Name</i> set to "True" if checked or "False" if unchecked. Set <i>Checked</i> to True to pre-set the box to checked.
ListBox	<i>Name,Left,Top,Width,Height,Items[,multiselect]</i>	<i>Items</i> is a list of items separated by %CRLF%. Creates variable defined by <i>DialogName.Name</i> set to the value of the item(s) selected by the user. Use <i>DialogName.Name.Items.Text</i> to set/retrieve the entire list at runtime. Optionally set <i>multiselect</i> to TRUE. If more than one item is selected in a multiselect list box <i>DialogName.Name</i> is set to the list of selected values separated by %CRLF%.
ComboBox	<i>Name,Left,Top,Width,Items</i>	<i>Items</i> is a list of items separated by %CRLF%. Creates variable defined by <i>DialogName.Name</i> set to the value of the item selected by the user. Use <i>DialogName.Name.Items.Text</i> to set/retrieve the entire list at runtime.
Label	<i>Caption,Left,Top[,transparent]</i>	Optional parameter, <i>transparent</i> , can be set to true to make the label transparent.
RadioGroup	<i>Name,Caption,Left,Top,Width,Height,Items,SelectedItem</i>	<i>Items</i> is a list of radio button items separated by %CRLF%. Creates variable defined by <i>DialogName.Name</i> set to the value of the item selected by the user.
ProgressBar	<i>Name,Left,Top,Width,Height,Position</i>	<i>Position</i> is a percentage that sets how far the progress bar has completed. Creates a variable defined by <i>DialogName.Name</i> which can be used to set the position.
Image	<i>ImageFile/Data,Left,Top,Width,Height[,name[,result]]</i>	<i>ImageFile</i> is the path to a graphics file to display on the image. Optional <i>name</i> means images can be changed at run time, e.g.: Let> <i>DialogName.Name</i> =filename.bmp If a result value is given the image becomes clickable and that value is returned just as if a button was clicked on.
TabBook	<i>Name,Left,Top,Width,Height,FitForm</i>	Creates a Tab control. Set <i>FitForm</i> to 1 to fill to form. Use <i>TabPage</i> to define pages. Use <i>EndTabBook</i> to mark end of <i>TabBook</i> declaration.
TabPage	<i>PageCaption</i>	Use within a <i>TabBook/EndTabBook</i> block. Objects defined after a <i>TabPage</i> declaration will appear on that page.
FileBrowse	<i>ButtonObjectName,TextObjectName[,mask[,open save dir[,initialdir[,filename]]]]</i> e.g.: <i>btn1,edit1,Text Files*.txt All Files*.*,open</i> <i>mask: mask description mask description[etc]</i>	Associates a button with a text field so that when the button is pressed the file browser dialog is displayed and the selected filename is placed in the text box. Must be placed after the button and text object. By default FileBrowse dialogs are "open" dialogs, but can be changed to "save" type to show save buttons, or "dir" to show a folder browse dialog. <i>initialdir</i> and <i>filename</i> allow the browse box to default to a directory and/or filename. To set <i>initialdir</i> and/or <i>filename</i> at run time set <i>DialogName.TextObjectName.fbinit=initialdir,filename</i>
MainMenu	<i>Caption,SubItem(result)[,SubItem(result),...]</i> e.g.: <i>File,Open(10),Save As(15),Exit(20)</i>	Creates a main menu containing the sub items. Each subitem is given a result value which is set to the result variable of the GetDialogAction ^[58] command if this item is selected.
Default	<i>ButtonObjectName</i>	Sets the specified button object as the default button so that when Enter is pressed that button is invoked.
Font	<i>Face,Size</i>	Sets font face and size of dialog and all objects.

Note that objects that return a result create variables of the form *DialogName.ObjectName*. Values can also be set with variables before the Dialog is displayed even though the Dialog block appears before the declaration as in the example below where the Edit box is preset.

To pre-select items in a ListBox, ComboBox or RadioGroup set the object's ItemIndex property. 0 is the first item. E.g. Let>MyDialog.msListBox1.ItemIndex=0

Images and Buttons can reference bitmap files, or bitmap data embedded in the script. To import bitmap data into the script use Tools/Import Bitmap Data in the editor. The bitmap data will appear below a label at the end of the script. Reference the label name in the Image/Button declaration of the dialog block in place of the filename.

The Dialog block creates the dialog. Once a dialog is created it will exist until the script terminates. Therefore a dialog block should not be repeated or duplicated with the same dialog name. Use Show to make the dialog visible.

The result of a dialog (returned by the [Show](#)^[57] and [GetDialogAction](#)^[55] commands) is set to the modal result value of the button that was pressed. If a Dialog is closed without a button being pressed, e.g. the user pressed the close button, the result will be set to 2. Modal result 2 means the dialog was canceled. Bear this in mind when assigning modal result values to buttons. It is advisable to use anything other than 2. 1 is commonly used for OK.

Modal Dialog Example

```
Dialog>MyDialog
Caption=This is My Dialog
Left=300
Top=300
Width=305
Height=120
Label=Type Something Here:,5,5
Edit=MyEdit,5,22,230,EditText
Button=Display,240,22,50,20,3
Button=Exit Macro,5,50,100,20,2
//Use Memos for multiline edits
//Memo=MyEdit2,80,80,200,200,dd
//ListBox=MyListBox,80,80,200,200,Line One%CRLF%Line Two%CRLF%Line Three
//Above returns variable MyListBox_Result containing selected item
CheckBox=chkMyCheckBox,Play Chimes?,130,50,150,False
EndDialog>MyDialog

Let>EditText=Enter Something Here and Press Display
Label>MainLoop
Show>MyDialog,result
If>result=2,End
If>result=3,SayHello
if>MyDialog.chkMyCheckBox=True,PlayWav
Goto>MainLoop

SRT>SayHello
MessageModal>%MyDialog.MyEdit%
END>SayHello

SRT>PlayWav
PlayWav>Chimes.wav
End>PlayWav

Label>End
```

Non-Modal Dialog Example

```
Dialog>Dialog1
Caption=Non-Modal Dialog
Top=113
Width=264
```

```

Left=16
Height=113
Button=Update,16,8,75,25,5
Button=Exit,16,48,75,25,6
Edit=msEdit1,104,8,121,Fred
ComboBox=msComboBox1,104,32,121,1%CRLF%2
Edit=msEdit2,104,56,121,
EndDialog>Dialog1

Show>Dialog1

Label>ActionLoop
  GetDialogAction>Dialog1,r
  if>r=5,Update
  if>r=6,Close
  if>r=2,exit
Goto>ActionLoop
Label>exit

SRT>Update
  Let>Dialog1.msEdit2=%Dialog1.msEdit1% %Dialog1.msComboBox1%
  ResetDialogAction>Dialog1
END>Update

SRT>Close
  CloseDialog>Dialog1
  Let>r=2
END>Close

```

See the Calculator sample script for another non-modal dialog example.

4.7.3 EndDialog

EndDialog>DialogName

Marks the end of a dialog block. See [Dialog](#)^[52] for details.

4.7.4 GetDialogAction

GetDialogAction>DialogName,Result

GetDialogAction retrieves the values of all objects on the non-modal dialog to their respective variables and returns in Result the identifier of the button that is pressed (modal result value). If a menu item was selected GetDialogAction returns the menu item result value in Result.

Abbreviation: **GDA**

See also: [ResetDialogAction](#)^[56], [CloseDialog](#)^[52]

Example:

This displays a non modal dialog and then sets up a loop which constantly uses GetDialogAction to retrieve the button result value. It then determines what to do based on which button has been pressed. After a button is pressed ResetDialogAction is called so that control stays within the loop, the edit field is updated, and we continue to check for a pressed button. When the dialog is closed r is set to 2. In our example we also set r to 2 when the second button with identifier 6 is pressed.

```

Dialog>Dialog1
  Caption=Non-Modal Dialog
  Top=113
  Width=264

```

```

Left=16
Height=113
Button=Update,16,8,75,25,5
Button=Exit,16,48,75,25,6
Edit=msEdit1,104,8,121,Fred
ComboBox=msComboBox1,104,32,121,1%CRLF%2
Edit=msEdit2,104,56,121,
EndDialog>Dialog1

Show>Dialog1

Label>ActionLoop
  GetDialogAction>Dialog1,r
  if>r=5,Update
  if>r=6,Close
  if>r=2,exit
Goto>ActionLoop
Label>exit

SRT>Update
  Let>Dialog1.msEdit2=%Dialog1.msEdit1% %Dialog1.msComboBox1%
  ResetDialogAction>Dialog1
END>Update

SRT>Close
  CloseDialog>Dialog1
  Let>r=2
END>Close

```

See the Calculator sample script for another non-modal dialog example.

4.7.5 ResetDialogAction

ResetDialogAction>DialogName

Sets the result of the dialog back to zero.

Also updates the values of Edits, Memos, Checkboxes, Listboxes, Comboboxes, RadioGroups and Progressbars. Set these values with the variables created by the Show command which follow the following format:

DialogName.Name

See [Dialog](#)^[52] for more details.

Use ResetDialogAction with [GetDialogAction](#)^[55]. See [GetDialogAction](#)^[55] for an example.

Abbreviation: **RDA**

4.7.6 SetDialogObjectFont

SetDialogObjectFont>Dialog_Name,Object_Name,Font_Name,Font_Size,Font_Style,Font_Color

Sets the font characteristics of the specified dialog object.

If you are unsure of the name of the object edit the dialog in the [Dialog Designer](#)^[15] and double click on the object to see it's name.

Font_Style can be one of:

0: Normal
1: Bold
2: Italic
3: Underline
4: Strikeout

Use the RGB function to create a color code. 0 is black.

Abbreviation: **SDF**

Example:

```
SetDialogObjectFont>Dialog1,msMemo2,Arial,8,1,0
```

4.7.7 SetDialogObjectFocus

SetDialogObjectFocus>Dialog_Name,Object_Name

Focuses the specified object of the specified dialog.

If you are unsure of the name of the object edit the dialog in the [Dialog Designer](#) ^[15] and double click on the object to see it's name.

Abbreviation: **DSF**

Example:

```
SetDialogObjectFocus>Dialog1,msMemo2
```

4.7.8 SetDialogObjectVisible

SetDialogObjectVisible>Dialog_Name,Object_Name,Show(0|1)

Shows or hides the specified dialog object. Set Show to 1 to make the object visible. Set Show to 0 to hide the object.

If you are unsure of the name of the object edit the dialog in the [Dialog Designer](#) ^[15] and double click on the object to see it's name.

Abbreviation: **SDV**

Example:

```
SetDialogObjectVisible>Dialog1,msButton1,0
```

4.7.9 Show

Show>DialogName[,Result]

Displays a dialog created with a [Dialog](#) ^[52] block.

If a result variable is provided in Result the dialog is shown modal. Otherwise it is displayed non-modal. For non-modal dialogs see [GetDialogAction](#) ^[55], [ResetDialogAction](#) ^[56] and [CloseDialog](#) ^[52] for determining which button was pressed and to close the dialog.

Result is set to the modal result value of the button that was pressed. If the Dialog is closed

without a button being pressed, e.g. the user pressed the close button, Result will be set to 2. Modal Result 2 means the dialog was canceled. Bear this in mind when assigning modal result values to other buttons. It is advisable to use anything other than 2. 1 is commonly used for OK.

See [Dialog](#) ^[52] for details on creating dialogs and an example.

4.8 File Handling

4.8.1 AppendFile

AppendFile>sourcefile1,sourcefile2,newfile,result

Appends sourcefile1 and sourcefile2 into newfile. result is the number of bytes written.

Abbreviation : [App](#)

See also : [MoveFile](#) ^[63], [DeleteFile](#) ^[59], [IfFileExists](#) ^[62], [CopyFile](#) ^[58]

Example

```
AppendFile>c:\temp\file1.txt,c:\temp\file2.txt,c:\temp\newfile.txt,result
```

4.8.2 ChangeDirectory

ChangeDirectory>path

Changes the current directory to the directory specified in path. path can be a literal string or a variable.

Abbreviation : [Cha](#)

See also : [CreateDir](#) ^[59]

Example

```
Change Directory>c:\program files\my directory\
```

4.8.3 CopyFile

CopyFile>sourcefile,destinationfile

Copies the file (or files), sourcefile, to destinationfile

sourcefile, and destinationfile may be variables. Wildcards can be used.

By default if destinationfile already exists it will not be overwritten and the new file will be renamed appropriately. It is possible to change the behaviour of the CopyFile command to overwrite instead by setting CF_OVERWRITE to 1. The default value of CF_OVERWRITE is 0.

Abbreviation : [Cop](#)

See also : [MoveFile](#) ^[63], [DeleteFile](#) ^[59], [IfFileExists](#) ^[62], [AppendFile](#) ^[58], [RenameFile](#) ^[64]

Example

```
CopyFile>c:\temp\myfile.txt,c:\my documents\myfile.old
```

Or with variables:

```
Let>filename=c:\temp\myfile.txt
Let>newfilename=c:\temp\myfile.new
CopyFile>filename,newfilename
```

4.8.4 CountFiles

CountFiles>file_spec,result,subdir_flag

Returns the number of files matching file_spec in the result variable.

file_spec can include a directory path, and should include a mask. e.g. c:\temp*.*

To recurse sub directories set subdir_flag to 1.

Abbreviation : [Cou](#)

Examples

```
CountFiles>c:\my documents\*.*.doc,DocCount,0
Message>Number of .doc files : %DocCount%
```

The following example will count all the files anywhere on the c: drive.

```
CountFiles>c:\*.*,TotalFiles,1
Message>Total Number Of Files On PC : %TotalFiles%
```

4.8.5 CreateDir

CreateDir>directory

Creates a new directory. The directory may be a full path.

Abbreviation : [Cre](#)

See also : [Change Directory](#) ^[58]

Example

```
CreateDir>c:\my documents\test
```

or

```
Let>dir=c:\my documents\test
CreateDir>dir
```

4.8.6 DeleteFile

DeleteFile>path

Deletes the file/files in filename. Can also be used to delete folders.

filename can be a variable. Wildcards can be used.

Abbreviation : [Del](#)

See also : [CopyFile](#) ^[58], [MoveFile](#) ^[63], [IfFileExists](#) ^[62]

Example

```
DeleteFile>c:\temp\*.*
```

4.8.7 EditIniFile

EditIniFile>infile,section,entry,newvalue

Edits a section entry in an ini file. infile can be a full path.

All parameters can be variables containing strings.

Abbreviation : [Edi](#)

See also : [ReadIniFile](#) 

Example

```
EditIniFile>c:\program files\myini.ini,settings,user,fred
```

4.8.8 FileDate

FileDate>filename,result

Returns the file date of filename in the result variable.

The date returned is in the format YYYYMMDD

filename can include a full path.

Abbreviation : [FDT](#)

See also : [FileTime](#) , [FileSize](#) 

Example

```
FileDate>c:\program files\myfile.exe,MyFileDate
```

4.8.9 FileSize

FileSize>filename,result

Returns the size of filename in the result variable.

The size is returned in bytes.

filename can include a full path.

Abbreviation : [FSZ](#)

See also : [FileDate](#) , [FileTime](#) 

Example

```
FileSize>c:\program files\myfile.exe,MyFileSize
```

4.8.10 FileTime

FileTime>filename,result

Returns the file time of filename in the result variable.

The time returned is in the format HHMMSS

filename can include a full path.

Abbreviation : [FTM](#)

See also : [FileDate](#)^[60], [FileSize](#)^[60]

Example

```
FileTime>c:\program files\myfile.exe,MyFileTime
```

4.8.11 GetFileList

GetFileList>filespec,result[,delimiter]

GetFileList returns a list of the files found matching the specified filespec. For instance, to return the list of files in the temp directory specify c:\temp*. * as the filespec.

If delimiter is omitted each filename is separated by a semicolon (;). Otherwise the filenames are separated by the delimiter specified.

By default GetFileList returns files. To return directories only set GFL_TYPE to 1 (Let>GFL_TYPE=1).

Abbreviation : [GFL](#)

Example

```
GetFileList>c:\temp\*. *,files
Separate>files,;,file_names
MessageModal>Num Files: %file_names_count%

Let>k=0
Repeat>k
    Let>k=k+1
    Message>file_names_%k%
Until>k,file_names_count
```

4.8.12 IfDirExists

IfDirExists>directory_name[,label_name[,false_label_name]]

statements

[[Else

else statements]

Endif]

If directory_name exists the first statements are executed. Otherwise the else statements are executed.

When label names are provided execution jumps to the specified label if the directory specified in directory_name exists. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a

subroutine name execution will jump to that subroutine and return when the subroutine has completed. If label names are specified Else and Endif are ignored and are unnecessary.

Abbreviation : **IFD**

See also : [Label](#)^[116], [Goto](#)^[115], [IfWindowOpen](#)^[79], [IfFileChanged](#)^[62], [If](#)^[77], [IfFileExists](#)^[62], [Subroutines](#)^[113]

Example

```
IfDirExists>c:\windows\temp,end
..
..
Label>end
```

4.8.13 IfFileChanged

IfFileChanged>filename,range[,label_name[,false_label_name]]

statements

[[Else

else statements

Endif]

If the given file's date is in the range of days specified the first statements are executed, otherwise the else statements are executed. Alternatively, instead of Else/Endif specify label names.

When label names are specified execution jumps to the specified label if the given file's date is in the range of days specified. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return to the line after the If statement when the subroutine has completed.

Abbreviation : **IFC**

See also : [Label](#)^[116], [Goto](#)^[115], [IfWindowOpen](#)^[79], [IfFileExists](#)^[62], [If](#)^[77], [Subroutines](#)^[113]

Example

To see if test.txt is less than 30 days old :

```
IfFileChanged>test.txt,<30,end
..
..
Label>end
```

4.8.14 IfFileExists

IfFileExists>filename[,label_name[,false_label_name]]

statements

[[Else

else statements

Endif]

If filename exists the first statements are executed. Otherwise the else statements are executed.

When label names are provided execution jumps to the specified label if the file specified in filename exists. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return when the subroutine has

completed. If label names are specified Else and Endif are ignored and are unnecessary.

Abbreviation : [IFE](#)

See also : [Label](#)^[116], [Goto](#)^[115], [IfWindowOpen](#)^[79], [IfFileChanged](#)^[62], [If](#)^[77], [IfDirExists](#)^[61], [Subroutines](#)^[113]

Example

```
IfFileExists>myfile.txt,end
..
..
Label>end
```

4.8.15 MoveFile

MoveFile>sourcefile,destinationfile

Moves the file (or files), sourcefile, to the file (or files) destinationfile.

sourcefile, and destination_path may be variables. Wildcards can be used.

To make MoveFile perform a simple Rename function, rather than physically moving the entire file, set MF_RENAME to 1.

Abbreviation : [Mov](#)

See also : [CopyFile](#)^[58], [DeleteFile](#)^[59], [IfFileExists](#)^[62], [AppendFile](#)^[58], [RenameFile](#)^[64]

Example

```
MoveFile>c:\temp\myfile.txt,c:\temp\myfile.bak
```

Or with variables:

```
Let>filename=c:\temp\myfile.txt
Let>newfilename=c:\temp\myfile.bak
MoveFile>filename,newfilename
```

4.8.16 ReadFile

ReadFile>file_name,result

Reads the entire file contents into the specified result variable. If the file does not exist, result is set to ##NOFILE##.

Abbreviation : [RFL](#)

See also : [ReadLn](#)^[64], [WriteLn](#)^[65]

Example

```
ReadFile>c:\temp\test.txt,file
MessageModal>file
```

4.8.17 ReadIniFile

ReadIniFile>inifile,section,entry,result

Reads a value from an entry in an ini file and places the result in a variable.

All parameters can be variables containing strings.

Abbreviation : [Rea](#)

See also : [EditIniFile](#) 

Example

This example reads a username from an ini file and displays it in a message :

```
ReadIniFile>c:\program files\myini.ini,settings,user,username
Let>msg=The Username is
ConCat>msg,username
Message>msg
```

4.8.18 ReadLn

ReadLn>file_name,line_number,result

Reads the specified line from the given ANSI text file. If line_number is past the end of the file, result will contain ##EOF##. If the file does not exist, result is set to ##NOFILE##. If an error occurred, result is set to ##ERR## followed by the error code.

Abbreviation : [RLN](#)

See also : [ReadFile](#) , [WriteLn](#) 

Example

This example reads each line from the text file and displays it in a Message window.

```
Let>k=1
Label>start
ReadLn>c:\temp\test.txt,k,line
If>line==EOF##,finish
Message>line
Let>k=k+1
Goto>start
Label>finish
```

4.8.19 RenameFile

RenameFile>file,newfile

Renames file to newfile.

This function does the same as [MoveFile](#)  with MF_RENAME set to 1.

Abbreviation : [Ren](#)

See also : [CopyFile](#) , [DeleteFile](#) , [IfFileExists](#) , [AppendFile](#) , [MoveFile](#) 

Example

```
RenameFile>c:\temp\myfile.txt,c:\temp\myfile.bak
```


4.8.20 WriteLn

WriteLn>file_name,result,line

Writes text to an ANSI text file. If the file does not exist it is created and the given line is written to it. If it does exist, the line is written to the end of the file. If the command was successful, result is set to zero. A non zero result means an error occurred.

To omit line break characters from the end of the line set WLN_NOCRLF to 1 before issuing the WriteLn command. This will prevent subsequent text written with WriteLn appearing on a new line.

Abbreviation : [WLN](#)

See also : [ReadLn](#)^[64], [ReadFile](#)^[63]

Example

```
GetDate>date
WriteLn>c:\temp\test.txt,result,%date% - Starting Macro
```

4.9 FTP/HTTP/Telnet/Email Commands

4.9.1 FTPDelFile

FTPDelFile>Server,Username,Password,port,Host_File_Spec

This command will connect to the specified FTP server and delete the file spec specified.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set FTP_STATUS to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for FTP_STATUS is 1.

The result of the FTP operation is stored in FTP_RESULT.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify username@remote_server.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the FTP_TIMEOUT variable.

Abbreviation : [FDF](#)

See also [FTPGetFile](#)^[67], [FTPPutFile](#)^[69], [FTPRenameFile](#)^[71], [FTPGetDirList](#)^[66]

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set FTP_USETLS to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using SSL_CERT, SSL_ROOT_CERT, SSL_KEY

Example

```
FTPDelFile>ftp.domain.com,anonymous,user@domain.com,21,/pub/readme.txt
```

4.9.2 FTPGetDirList

FTPGetDirList>Server,Username,Password,port,Local_File,Host_Dir,Host_File_Spec,Type

This command will connect to the specified FTP server and download a directory listing for the file spec specified.

The directory listing is output to the local text file given in Local_File.

Type can be either D or L, where D obtains a full directory listing, including directories, subdirectories and their contents. L creates a simple file list, containing just filenames matching the file spec.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set FTP_STATUS to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for FTP_STATUS is 1.

The result of the FTP operation is stored in FTP_RESULT.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify username@remote_server.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the FTP_TIMEOUT variable.

Abbreviation : FGD

See also [FTPGetFile](#)⁶⁷, [FTPPutFile](#)⁶⁹, [FTPDelFile](#)⁶⁵, [FTPRenameFile](#)⁷¹

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set FTP_USETLS to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using `SSL_CERT`, `SSL_ROOT_CERT`, `SSL_KEY`

Example

The following command will retrieve a listing of the entire contents of the server.

```
FTPGetDirList>ftp.domain.com,anonymous,user@domain.com,21,c:\temp\readme.txt,*,D
```

4.9.3 FTPGetFile

FTPGetFile>Server,Username,Password,port,Local_Path,Host_Dir,Host_Filespec,Mode

This command will connect to the specified FTP server and retrieve a file or files.

The file(s) retrieved from the server is specified using `Host_Dir` and `Host_Filespec`. The file(s) is saved on the local machine under the name specified in `Local_Path`. If `Host_Filespec` is a file spec which matches multiple files (includes wildcards), all matching files will be downloaded and stored to `Local_Path`. Note that `Local_Path` can be a path to a directory or file. Specify a directory when retrieving multiple files. A filename is only sensible when retrieving a single file where the local filename should be different. Depending on the server type `Host_Dir` can be left empty if retrieving files from the root folder.

Mode can be either A or I, where A represents ASCII transfer mode and I represents binary.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set `FTP_STATUS` to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for `FTP_STATUS` is 1.

If passive mode is required, set `FTP_PASSIVE` to 1. Set to zero to switch passive mode off.

The result of the FTP operation is stored in `FTP_RESULT`. The number of files transferred is stored in `FTP_NUMFILES`.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify `username@remote_server`.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the `FTP_TIMEOUT` variable.

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set `FTP_USETLS` to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using SSL_CERT, SSL_ROOT_CERT, SSL_KEY

Abbreviation : **FGF**

See also : [FTPPutFile](#)^[69], [FTPGetDirList](#)^[66], [FTPDelFile](#)^[65], [FTPRenameFile](#)^[71]

Example

```
FTPGetFile>
ftp.domain.com,anonymous,user@domain.com,21,c:\temp\myfile.txt,/pub/readme.txt,A
```

Multiple Files:

```
FTPGetFile>ftp.domain.com,anonymous,user@domain.com,21,c:\temp\,/pub/*.txt,A
```

4.9.4 FTPMakeDir

FTPMakeDir>Server,Username,Password,port,Host_Folder_Path

This command will connect to the specified FTP server and create the folder specified.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set FTP_STATUS to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for FTP_STATUS is 1.

The result of the FTP operation is stored in FTP_RESULT.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify username@remote_server.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the FTP_TIMEOUT variable.

Abbreviation : **FDF**

See also [FTPGetFile](#)^[67], [FTPPutFile](#)^[69], [FTPRenameFile](#)^[71], [FTPGetDirList](#)^[66]

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set FTP_USETLS to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using SSL_CERT, SSL_ROOT_CERT, SSL_KEY

Example

```
FTPMakeDir>ftp.domain.com,user1,password,21,/user1/myfiles/
```

4.9.5 FTPPutFile

FTPPutFile>Server,Username,Password,port,Local_Filespec,Host_Dir,Host_File,Mode

This command will connect to the specified FTP server and upload a file or files.

The file to be uploaded is specified in Local_Filespec, which can contain a full path and wildcards. Depending on the server type Host_Dir can be left empty if adding files to the root folder. When uploading multiple files leave Host_File empty, or leave Host_File empty in order to use the same file name as the local file. You would only set Host_File when uploading one file and you want to store it with a different name.

Mode can be either A or I, where A represents ASCII transfer mode and I represents binary.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set FTP_STATUS to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for FTP_STATUS is 1.

If passive mode is required, set FTP_PASSIVE to 1. Set to zero to switch passive mode off.

The result of the FTP operation is stored in FTP_RESULT. The number of files transferred is stored in FTP_NUMFILES.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify username@remote_server.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the FTP_TIMEOUT variable.

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set FTP_USETLS to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using SSL_CERT, SSL_ROOT_CERT, SSL_KEY

Abbreviation : **FPF**

See also [FTPGetFile](#)^[67], [FTPGetDirList](#)^[66], [FTPDelFile](#)^[65], [FTPRenameFile](#)^[71]

Example

```
FTPputFile>
ftp.domain.com,anonymous,user@domain.com,21,c:\temp\readme.txt,/pub/readme.txt,A
```

Multiple Files:

```
FTPputFile>ftp.domain.com,anonymous,user@domain.com,21,c:\temp\*.txt,/pub/,A
```

4.9.6 FTPRemoveDir

FTPRemoveDir>Server,Username,Password,port,Host_Folder_Path

This command will connect to the specified FTP server and delete the folder specified.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set FTP_STATUS to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for FTP_STATUS is 1.

The result of the FTP operation is stored in FTP_RESULT.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify username@remote_server.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the FTP_TIMEOUT variable.

Abbreviation : **FDF**

See also [FTPGetFile](#)^[67], [FTPputFile](#)^[69], [FTPRenameFile](#)^[71], [FTPGetDirList](#)^[66]

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set FTP_USETLS to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using SSL_CERT, SSL_ROOT_CERT, SSL_KEY

Example

```
FTPRemoveDir>ftp.domain.com,user1,password,21,/user1/myfiles/
```

4.9.7 FTPRenameFile

FTPRenameFile>Server,Username,Password,port,Host_File_Spec,New_File_Name

This command will connect to the specified FTP server and rename the host file specified with New_File_Name.

When the FTP commands are active a small floating window is displayed at the top left of the screen showing the status of the FTP session. To stop the status window appearing set FTP_STATUS to 0 before issuing an FTP command. Set it back to 1 to show the window. The default value for FTP_STATUS is 1.

The result of the FTP operation is stored in FTP_RESULT.

If you are connecting via a proxy server, enter the name or IP address of the proxy server in 'Server', and for 'Username' specify username@remote_server.

By default the timeout for the FTP commands is set to 15 seconds. To change this set the FTP_TIMEOUT variable.

Abbreviation : **FRF**

See also [FTPGetFile](#)^[67], [FTPPutFile](#)^[69], [FTPDelFile](#)^[65], [FTPGetDirList](#)^[66]

TLS/SSL Support

To use TLS/SSL you will first need to install the OpenSSL libraries. See: <http://www.openssl.org/related/binaries.html>

For TLS/SSL support set FTP_USETLS to one of the following values:

0	No TLS/SSL Support	Default - no encryption
1	Use Implicit TLS	You wish to use implicit TLS. In implicit TLS FTP, you connect to port 990, start TLS negotiation, and the entire session is encrypted. IETF has deprecated implicit TLS FTP so this setting should only be used with FTP servers that do not yet support explicit TLS.
2	Use Require TLS	You wish your session to use explicit TLS. In explicit TLS FTP, you connect to port 21 like the unencrypted version of FTP and encryption lasts until you disconnect. No attempt is made to continue the FTP session if TLS negotiation fails.
3	Use Explicit TLS	You wish your session to use explicit TLS if the FTP server supports it.

If needed you can also specify the SSL certificate and key files to use using SSL_CERT, SSL_ROOT_CERT, SSL_KEY

Example

```
FTPRenameFile>
ftp.domain.com,anonymous,user@domain.com,21,/pub/readme.txt,/pub/readme.new
```

4.9.8 HTTPRequest

HTTPRequest>URL,[LocalFilename],Method,[POST_Data],Result_Variable[,ProxyServer,ProxyPort,ProxyUsername,ProxyPassword]

Retrieves a web document via the HTTP protocol using either GET or POST methods.

URL: URL of document to retrieve

LocalFileName: Optional (may be left blank) - local file to save response to.

Method: GET or POST

Post_Data: Data to Post to URL if using POST method. Use name=value pairs separated by '&'. See example below.

Result_Variable: Stores result of operation. If LocalFileName is not specified and the operation is successful this will contain the HTML returned. If LocalFileName was specified and the data was successfully written to the file Result_Variable will be blank. Otherwise it will contain an error message.

ProxyServer: Optional - if using a proxy server set this to domain or IP address of proxy server.

ProxyPort: Optional - if using a proxy server set to port number of proxy server.

ProxyUsername: Optional - if using a proxy server that needs a username.

ProxyPassword: Optional - if using a proxy server that needs a password.

To post files use the HTTP_POSTFILES variable. HTTP_POSTFILES takes name=value pairs separated by '&' in the same format as Post_Data but for 'file' parameters. Post_Data can be empty if only files are being posted.

By default there is no timeout for the HTTPRequest command and requests will wait indefinitely if the server fails to respond. To change this set the HTTP_TIMEOUT value to the number of seconds to wait.

By default HTTPRequest will automatically resolve redirects. To disable this behaviour set HTTP_REDIRECTS to 0.

For basic authentication where a username and password is required by the server before the request will complete put the username and password in the URL using the following format:
<http://username:password@www.server.com/etc.etc>

For SSL (https) connections set HTTP_SSL to 1. To use SSL you will first need to install the OpenSSL libraries. See:
<http://www.openssl.org/related/binaries.html>

Abbreviation : [HTT](#)

Example

The following line does a simple GET request and saves the resulting HTML to a variable called HTMLResponse:

```
HTTPRequest>http://www.mjtnet.com,,GET,,HTMLResponse
```

The following line does the same thing but also saves the output to a file:

```
HTTPRequest>http://www.mjtnet.com,d:\HTML\mjtnet.html,GET,,HTMLResponse
```

This demonstrates a POST operation, sending name=value pairs to the page:

```
Let>PostData=email=myemail@home.com&name=Joe Bloggs  
HTTPRequest>http://www.someplace.com/someform.html,,POST,PostData,HTMLResponse
```

Posting files:

```
Let>HTTP_POSTFILES=upload1=c:\stuff\logo.gif  
HTTPRequest>http://www.someplace.com/form.html,,POST,,HTMLResponse
```

4.9.9 RetrievePOP3

RetrievePOP3>server,username,password,output_path

Retrieves mail from a POP3 server and stores new mail messages in the folder specified in output_path.

server: the name/address of the POP3 server

username: the POP3 account username

password: the POP3 account password

output_path: a local directory where message files should be stored

Unencoded messages and headers are stored as MSGn.TXT where n is an incremented suffix number. e.g. MSG1.TXT, MSG2.TXT etc. Text body parts will be stored as MSGn_BODYn.TXT and attachments will be stored as MSGn_attachment_filename

The command returns two variables:

POP3_MSGFILES: A semicolon delimited list of files downloaded. Use Separate to explode and retrieve file count.

POP3_RESULT: Contains the last response, such as errors, returned by the server.

Optional variables are:

POP3_STATUS: Set to 0 to suppress the status window. Default is 1.

POP3_PORT: The POP3 port number. Default is 110.

POP3_TIMEOUT: Timeout in seconds. Default is 5.

POP3_DELETE: Set to 1 to delete messages from the server after downloading. Default is 0.

POP3_MSGSIZELIMIT: Size limit in Kb. Messages over this limit will not be downloaded. Default is 0 (no size limit).

Abbreviation: [RET](#)

See Also: [SMTPSendMail](#) 

Example

```

Let>POP3_STATUS=1
Let>POP3_MSGSIZELIMIT=3
Let>POP3_TIMEOUT=5
RetrievePOP3>mail.server.com,username1,password,d:\emailfiles
MessageModal>POP3_RESULT
Separate>POP3_MSGFILES,;,MsgFiles
MessageModal>Files Downloaded: %MsgFiles_count%%CRLF%File List: %POP3_MSGFILES%

```

4.9.10 SMTPSendMail

SMTPSendMail>recipients,server,from_address,from_name,subject,body,attachments

Sends email via an SMTP server to the recipients entered.

recipients : Include one or more email addresses, separated by semicolons (;).

server : The name or IP address of your SMTP server.

from_address : Your email address, or the email address you want the message to come from.

from_name : Your real name, or a string to appear in the from name field.

subject : The subject line.

body : The body text of the message.

attachments : if sending files include each file separated by semicolons (;). If no attachments end the line with a comma.

By default when this command is in use a small status window will appear at the top left of the screen. You can stop this window being displayed by setting SENDMAIL_STATUS to 0.

To include CC or BCC email addresses set the SMTP_CCLIST and SMTP_BCCLIST variables. Separate multiple email address with semicolons.

If you are using an SMTP server that requires authentication, you can enable authentication by setting SMTP_AUTH to 1, and then setting SMTP_USERID and SMTP_PASSWORD to the username and password that you need to connect to the SMTP server. e.g:

```

Let>SMTP_AUTH=1
Let>SMTP_USERID=myuser
Let>SMTP_PASSWORD=frogslegs

```

To enable return receipt of the email, set SMTP_RECEIPT to 1.

By default the port used by SMTPSendMail is 25. If you need to change the port number set the value of SMTP_PORT.

The result of the sendmail operation is placed into the variable SMTP_RESULT. The format of this response depends on the SMTP server being communicated with. If successful the value will **contain** the result code 250. Otherwise it will contain the appropriate error message.

SMTP_TIMEOUT can be used to set a connection timeout in milliseconds. This timeout relates to the initial connection to the SMTP server.

Abbreviation : **SMT**

See Also: [RetrievePOP3](#)

Example

```

Let>SENDMAIL_STATUS=1
Let>subject=Test Message
Let>me=myname@myplace.com
Let>myname=Mr Smith
Let>recipients=mickey@disney.com;someone@somewhere.com
Input>body,Enter your message:
SMTPSendMail>recipients,post.mail.com,me,myname,subject,body,

```

```
Message>Result of SendMail: %SMTP_RESULT%
```

4.9.11 TelnetClose

TelnetClose>session_id

Terminates a Telnet session. Provide the session_id returned by a call to [TelnetConnect](#) ^[75].

The TELNET_SESSIONLOG variable can be used at any time to see all session activity.

Abbreviation: **TNQ**

See Also: [TelnetConnect](#) ^[75], [TelnetSend](#) ^[76], [TelnetWaitFor](#) ^[76]

Example

```
TelnetConnect>my.domain.com,23,hTN
TelnetWaitFor>hTN,Press any key to continue,5,r
TelnetSend>hTN,a
TelnetWaitFor>hTN,login,5,r
TelnetSend>hTN,administrator%CR%
TelnetWaitFor>hTN,password,5,r
TelnetSend>hTN,monsoon%CR%
TelnetWaitFor>hTN,>,5,r
TelnetSend>hTN,dir%CR%
TelnetWaitFor>hTN,>,5,dirlist
MessageModal>dirlist
TelnetClose>hTN
```

4.9.12 TelnetConnect

TelnetConnect>host,port,session_id

Connects to a Telnet server. Host is the server name or address of the server, port the port number. The standard Telnet port is 23. Returns a Session ID to be used with other Telnet commands to manipulate the session.

The TELNET_SESSIONLOG variable can be used at any time to see all session activity.

Abbreviation: **TNC**

See Also: [TelnetClose](#) ^[75], [TelnetSend](#) ^[76], [TelnetWaitFor](#) ^[76]

Example

```
TelnetConnect>my.domain.com,23,hTN
TelnetWaitFor>hTN,Press any key to continue,5,r
TelnetSend>hTN,a
TelnetWaitFor>hTN,login,5,r
TelnetSend>hTN,administrator%CR%
TelnetWaitFor>hTN,password,5,r
TelnetSend>hTN,monsoon%CR%
TelnetWaitFor>hTN,>,5,r
TelnetSend>hTN,dir%CR%
TelnetWaitFor>hTN,>,5,dirlist
MessageModal>dirlist
TelnetClose>hTN
```

4.9.13 TelnetSend

TelnetSend>session_id,text

Sends text to a Telnet session. session_id is a session ID returned by a call to [TelnetConnect](#)^[75].

To send a carriage return use %CR%.

The TELNET_SESSIONLOG variable can be used at any time to see all session activity.

Abbreviation: **TNS**

See Also: [TelnetClose](#)^[75], [TelnetConnect](#)^[75], [TelnetWaitFor](#)^[76]

Example

```
TelnetConnect>my.domain.com,23,hTN
TelnetWaitFor>hTN,Press any key to continue,5,r
TelnetSend>hTN,a
TelnetWaitFor>hTN,login,5,r
TelnetSend>hTN,administrator%CR%
TelnetWaitFor>hTN,password,5,r
TelnetSend>hTN,monsoon%CR%
TelnetWaitFor>hTN,>,5,r
TelnetSend>hTN,dir%CR%
TelnetWaitFor>hTN,>,5,dirlist
MessageModal>dirlist
MessageModal>TELNET_SESSIONLOG
TelnetClose>hTN
```

4.9.14 TelnetWaitFor

TelnetWaitFor>session_id,text,timeout,response

Waits for text in a Telnet session. Specify a session ID returned by a call to [TelnetConnect](#)^[75].

Enter a timeout in seconds. If the command times out response will be set to TIMEDOUT otherwise response will return session output since the last TelnetSend.

The TELNET_SESSIONLOG variable can be used at any time to see all session activity.

Abbreviation: **TNW**

See Also: [TelnetClose](#)^[75], [TelnetConnect](#)^[75], [TelnetSend](#)^[76]

Example

```
TelnetConnect>my.domain.com,23,hTN
TelnetWaitFor>hTN,Press any key to continue,5,r
TelnetSend>hTN,a
TelnetWaitFor>hTN,login,5,r
TelnetSend>hTN,administrator%CR%
TelnetWaitFor>hTN,password,5,r
TelnetSend>hTN,monsoon%CR%
TelnetWaitFor>hTN,>,5,r
TelnetSend>hTN,dir%CR%
TelnetWaitFor>hTN,>,5,dirlist
MessageModal>dirlist
MessageModal>TELNET_SESSIONLOG
TelnetClose>hTN
```

4.10 Conditional Commands (If...)

4.10.1 If

```

If>expression[,true_label_name[,false_label_name]]
    statements
[ Else
    else statements
Endif ]

```

Evaluates **expression**. If the expression is true the first statements are executed. If the expression is false the statements after Else are executed. Else is optional. IF blocks can be nested. When label names are provided execution of the script jumps to the specified labels or subroutine names. If label names are specified Else and Endif are ignored and are unnecessary.

The **expression** can be *simple (legacy)*, or *complex*.

Complex Expressions

The expression must be contained within curly braces "{" and "}". String literals must be delimited with double quotes ("), e.g.: "string". Variables must be delimited with % symbols, e.g.: %VarA%.

Several types of operators and functions can be used with complex expressions. For more details see [Complex Expressions](#)^[41].

Simple Expressions

Simple expressions can contain only two parts separated by one of the following operators:

```

=      Equals
>      Greater than
<      Less than
<>     Not Equal

```

Values in the expression can be numeric or string values, or variables containing such values. Spaces must not be used as simple expressions do not require string delimiters.

See also : [Label](#)^[116], [Goto](#)^[115], [IfFileChanged](#)^[62], [IfFileExists](#)^[62], [IfWindowOpen](#)^[79], [Subroutines](#)^[113]

Example

```

Let>a=5
//simple expression only
IF>a=5
    //do something
ELSE
    //do something else
ENDIF

//complex expression:
IF>{(%a% = 5) AND (%VarA% = "allen")}
    //do something
ELSE
    //do something else
ENDIF

//using labels
Label>Start
..
..

```

```
..
If>a<b,Start
```

4.10.2 IfDirExists

```
IfDirExists>directory_name[,label_name[,false_label_name]]
    statements
[ [Else
    else statements]
Endif ]
```

If directory_name exists the first statements are executed. Otherwise the else statements are executed.

When label names are provided execution jumps to the specified label if the directory specified in directory_name exists. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return when the subroutine has completed. If label names are specified Else and Endif are ignored and are unnecessary.

Abbreviation : [IFD](#)

See also : [Label](#)^[116], [Goto](#)^[115], [IfWindowOpen](#)^[79], [IfFileChanged](#)^[62], [If](#)^[77], [IfFileExists](#)^[62], [Subroutines](#)^[113]

Example

```
IfDirExists>c:\windows\temp,end
..
..
Label>end
```

4.10.3 IfFileChanged

```
IfFileChanged>filename,range[,label_name[,false_label_name]]
    statements
[ [Else
    else statements]
Endif ]
```

If the given file's date is in the range of days specified the first statements are executed, otherwise the else statements are executed. Alternatively, instead of Else/Endif specify label names.

When label names are specified execution jumps to the specified label if the given file's date is in the range of days specified. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return to the line after the If statement when the subroutine has completed.

Abbreviation : [IFC](#)

See also : [Label](#)^[116], [Goto](#)^[115], [IfWindowOpen](#)^[79], [IfFileExists](#)^[62], [If](#)^[77], [Subroutines](#)^[113]

Example

To see if test.txt is less than 30 days old :

```
IfFileChanged>test.txt,<30,end
..
```

```
..
Label>end
```

4.10.4 IfFileExists

```
IfFileExists>filename[,label_name[,false_label_name]]
    statements
[ [Else
    else statements]
Endif ]
```

If filename exists the first statements are executed. Otherwise the else statements are executed.

When label names are provided execution jumps to the specified label if the file specified in filename exists. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return when the subroutine has completed. If label names are specified Else and Endif are ignored and are unnecessary.

Abbreviation : [IFE](#)

See also : [Label](#)^[116], [Goto](#)^[115], [IfWindowOpen](#)^[79], [IfFileChanged](#)^[62], [If](#)^[77], [IfDirExists](#)^[61], [Subroutines](#)^[113]

Example

```
IfFileExists>myfile.txt,end
..
..
Label>end
```

4.10.5 IfWindowOpen

```
IfWindowOpen>window_title[,label_name[,false_label_name]]
    statements
[ [Else
    else statements]
Endif ]
```

Checks to see if the specified window is open. If so the first statements are executed. Otherwise the else statements are executed. Optionally, instead of Else and Endif a label or subroutine name can be specified.

When label names are specified the command causes the script to continue from the specified label without running any other lines of code in between. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return to the line after the If statement when the subroutine has completed.

The window_title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will first attempt to find the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window

title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

It is possible to limit the type of windows this command effects using the WF_TYPE variable:

Let>WF_TYPE=0 - No Child Windows
 Let>WF_TYPE=1 - ALL Windows (Default)
 Let>WF_TYPE=2 - Visible Windows Only

Abbreviation : IfW

See also : [Label](#)^[116], [Goto](#)^[115], [IfFileExists](#)^[62], [IfFileChanged](#)^[62], [If](#)^[77], [Subroutines](#)^[113]

Example

```
IfWindowOpen>Notepad - [Untitled],donotepad
..
..
Label>donotepad
```

or, with a wildcard :

```
IfWindowOpen>notepad*,donotepad
..
..
Label>donotepad
```

4.11 Image Recognition

4.11.1 CompareBitmaps

CompareBitmaps>bitmap_file_1,bitmap_file_2,result

Compares two bitmap files and returns the percentage match. If the two files are not of the same dimension then result will be -1.

Abbreviation: CPB

See also: [GetScreenRes](#)^[81], [FindImagePos](#)^[80], [ScreenCapture](#)^[82], [WaitScreenImage](#)^[82]

Example

```
GetScreenRes>w,h
ScreenCapture>0,0,w,h,d:\current_screen.bmp
CompareBitmaps>d:\required_screen.bmp,d:\current_screen.bmp,match
If>match=100
  MessageModal>Screens Match
Endif
```

4.11.2 FindImagePos

FindImagePos>bitmap_to_find,bitmap_to_scan|SCREEN,color_tolerance,return_center,X_Array,Y_Array,NumFound

Scans bitmap_to_scan looking for occurrences of bitmap_to_find. bitmap_to_find is usually a small image such as an image of a button or other GUI component captured at design time, and bitmap_to_scan would be a screen image or capture of a window taken at runtime with ScreenCapture.

bitmap_to_scan can be set to SCREEN to scan the screen directly, rather than a bitmap file.

If `color_tolerance` is zero the pixel colors must match exactly. `color_tolerance` can be set to a value between 0 and 255. If larger than zero the red, green and blue values of the pixels in `bitmap_to_scan` are checked to see if they are within the tolerance specified (color value + or - `color_tolerance`). Smaller values match less and larger values match more.

If `return_center` is 0 the returned coordinates are the starting position of the located image within `bitmap_to_scan`. In other words X, Y is the top left corner of `bitmap_to_find` within `bitmap_to_scan`. If `return_center` is 1 X,Y will be set to the center of the located image within the main `bitmap_to_scan` image.

`X_Array` is the name of a variable to store the X coordinates of each match. The first match is stored in `X_Array_0`, the second in `X_Array_1`, etc.

`Y_Array` is the name of a variable to store the Y coordinates of each match. The first match is stored in `Y_Array_0`, the second in `Y_Array_1`, etc.

`NumFound` returns the number of matches found.

Advanced: By default `FindImagePos` scans 100 random pixels in each matching rectangle on `bitmap_to_scan`. Scanning only 100 random pixels, rather than every pixel of the rectangle, dramatically increases performance and yields a reasonable degree of accuracy. However, in some cases you may want to scan more than 100 pixels for increased accuracy. To do this set the variable `FIP_SCANPIXELS` to the number of pixels to scan (e.g. `Let>FIP_SCANPIXELS=200`). The higher this value the slower the scan time will be.

Abbreviation: **FIP**

See also: [GetScreenRes](#)^[81], [CompareBitmaps](#)^[80], [ScreenCapture](#)^[82], [WaitScreenImage](#)^[82]

Example

```
GetScreenRes>X,Y
ScreenCapture>0,0,X,Y,d:\screen.bmp
FindImagePos>d:\today_button.bmp,d:\screen.bmp,0,1,X,Y,NumFound
If>NumFound>0
    MouseMove>X_0,Y_0
Endif
```

In 9.2 (and above) this can be simplified to:

```
FindImagePos>d:\today_button.bmp,SCREEN,0,1,X,Y,NumFound
If>NumFound>0
    MouseMove>X_0,Y_0
Endif
```

4.11.3 GetScreenRes

GetScreenRes>Width,Height

Returns the dimensions of the primary monitor.

Abbreviation: **GSR**

See also: [CompareBitmaps](#)^[80], [FindImagePos](#)^[80], [ScreenCapture](#)^[82], [WaitScreenImage](#)^[82]

Example

```
GetScreenRes>X,Y
```

4.11.4 ScreenCapture

ScreenCapture>X1,Y1,X2,Y2,Filename

Captures a portion of the screen to a .BMP (bitmap) or .JPG (jpeg) file and to the clipboard.

X1 and Y1 represent the upper left corner of the screen area to be copied.

X2 and Y2 represent the lower right corner of the screen area to be copied.

Filename is the name of the file to create and can end with a .BMP or .JPG extension.

ScreenCapture returns a variable called SCREENCAP_RESULT which will contain one of the following values:

0: Successful

1: Invalid file type specified. Valid file types are .BMP or .JPG

2: Unable to save image to specified file

3: No filename specified

Abbreviation: [SCP](#)

See also: [CompareBitmaps](#)^[80], [FindImagePos](#)^[80], [WaitScreenImage](#)^[82], [GetScreenRes](#)^[81]

Example

```
ScreenCapture>10,10,200,200,d:\mypic.bmp
```

4.11.5 WaitScreenImage

WaitScreenImage>BitmapFile[,Color_Tolerance]

Waits until the image in BitmapFile is found on the screen.

Returns the number of occurrences found on the screen in the WSI_RESULT variable.

Optionally, a Color tolerance value can be specified. The default is 0 (no tolerance). If color_tolerance is zero the pixel colors must match exactly. color_tolerance can be set to a value between 0 and 255. If larger than zero the red, green and blue values of the pixels in bitmap_to_scan are checked to see if they are within the tolerance specified (color value + or - color_tolerance). Smaller values match less and larger values match more.

The system variable WSI_TIMEOUT can be used to set the number of seconds after which this command should timeout. If set to zero (the default) the timeout will not occur and the command will continue indefinitely. If WSI_TIMEOUT is used, WSI_TIMEDOUT will indicate whether or not the command ended successfully. If it timed out WSI_TIMEDOUT will be set to TRUE otherwise, if the image was found within the time out period it will be set to FALSE.

Abbreviation: [WSI](#)

See also: [CompareBitmaps](#)^[80], [FindImagePos](#)^[80], [ScreenCapture](#)^[82], [GetScreenRes](#)^[81]

Example


```
WaitScreenImage>d:\today_button.bmp  
MessageModal>FoundIt
```

4.12 Keyboard Commands

4.12.1 Ascii

Ascii>ASCII_Code[,ASCII_Code[,ASCII_Code[...]]]

Inserts the characters specified by the given ASCII codes into the current application. If sending more than one ASCII code at once, separate them with commas.

This function is useful for sending non-printable, low, or high ASCII value characters to an editor, which you might otherwise be unable to send using the [Send](#)  function.

Abbreviation : [ASC](#)

See also : [Send Character/Text](#) 

Example


The following example shows three methods of sending the same text value to Notepad.

```
SetFocus>Notepad*
Ascii>77
Ascii>65
Ascii>82
Ascii>67
Ascii>85
Ascii>83
Press Enter
Ascii>77,65,82,67,85,83
Press Enter
Send>MARCUS
```

4.12.2 CapsOff

Switches caps lock off. If Caps lock is already off, no action is taken. If Caps lock is on, it is switched off.


Abbreviation : [COF](#)

See also : [CapsOn](#) 

4.12.3 CapsOn

Switches caps lock on. If Caps lock is already on, no action is taken. If Caps lock is off, it is switched on.


Abbreviation : [CAP](#)

See also : [CapsOff](#) 

4.12.4 NumOff

Switches Num lock off. If Num lock is already off, no action is taken. If Num lock is on, it is switched off.

Abbreviation : [NOF](#)

See also : [NumOn](#) 

4.12.5 NumOn

Switches Num lock on. If Num lock is already on, no action is taken. If Num lock is off, it is switched on.

Abbreviation : [NON](#)

See also : [NumOff](#) 83

4.12.6 Press ...

All commands starting with Press facilitate the sending of non-character keys. To issue a Press command more than once add the '*' symbol followed by the number of times to issue the command at the end of the line:

Press Key * n

E.g.:

Press Tab * 5

By setting PRESS_ALLOWVARS to 1, Press can accept a variable name in place of the key name.

The following is a complete list of all commands, with explanations where necessary :

Press Backspace	
Press Tab	
Press Enter	
Press Esc	
Press F1	Press F13
Press F2	Press F14
Press F3	Press F15
Press F4	Press F16
Press F5	Press F17
Press F6	Press F18
Press F7	Press F19
Press F8	Press F20
Press F9	Press F21
Press F10	Press F22
Press F11	Press F23
Press F12	Press F24
Press Home	
Press End	
Press Up	
Press Down	
Press Left	
Press Right	
Press Page Up	
Press Page Down	
Press Ins	
Press Del	
Press Shift	
Release Shift	
Press LShift	} Left/Right Shift Key
Release LShift	} ONLY in Windows NT and later
Press RShift	} you can distinguish specifically
Release RShift	} between left and right keys.
Press CTRL	

Release CTRL	
Press LCTRL	} Left/Right CTRL Key
Release LCTRL	} ONLY in Windows NT and later
Press RCTRL	} you can distinguish specifically
Release RCTRL	} between left and right keys.
Press ALT	
Release ALT	
Press LALT	} Left/Right ALT Key
Release LALT	} ONLY in Windows NT and later
Press RALT	} you can distinguish specifically
Release RALT	} between left and right keys.
Press ALTGR	
Release ALTGR	
Press CAPS	has the effect of toggling caps lock.
Press Num Lock	
Press Scroll Lock	
Press NP0	0 on Number Pad
Press NP1	etc
Press NP2	
Press NP3	
Press NP4	
Press NP5	
Press NP6	
Press NP7	
Press NP8	
Press NP9	
Press NP Add	Num pad operator keys
Press NP Subtract	etc
Press NP Multiply	
Press NP Divide	
Press NP Decimal	
Press NP Enter	
Press LWinKey	}
Press RWinKey	} Windows Keys
Press MenuKey	}
Press Print Screen	

There have been reports that Press CTRL, Press SHIFT and Press ALT sometimes fail when running Macro Scheduler under a Citrix environment. This can be fixed by setting SK_LEGACY to 1 prior to issuing these keystroke commands. Some DOS applications also need SK_LEGACY set to 1.

4.12.7 Release ...

Some keys that can be pressed must also be released. This facilitates holding down a key while another is pressed, such as with the ALT key for instance.

For example, to exit a program you would press ALT and F together to activate the File menu, followed by the X key to select the Exit option. To simulate this in a script you would Press ALT, then send the text FX and finally Release ALT. This would appear in the script window as :

```
Press ALT
Send Character/Text>FX
Release ALT
```

By setting PRESS_ALLOWVARS to 1, Release can accept a variable name in place of the key name.

The following release key commands exist :

Release ALT
Release LALT
Release RALT
Release ALTGR


Release CTRL
Release LCTRL
Release RCTRL

Release Shift
Release LShift
Release RShift

Release LWinKey
Release RWinKey


4.12.8 ScrollOff

Switches Scroll lock off. If Scroll lock is already off, no action is taken. If Scroll lock is on, it is switched off.

Abbreviation : [SOF](#)
See also : [ScrollOn](#) 


4.12.9 ScrollOn

Switches Scroll lock on. If Scroll lock is already on, no action is taken. If Scroll lock is off, it is switched on.

Abbreviation : [SON](#)
See also : [ScrollOff](#) 

4.12.10 SendText


SendText>text_to_send

This command sends the specified text to the window that currently has the focus. See [SetFocus](#) .

It is possible to slow down the speed at which each character in the string is sent by using the SK_DELAY variable. Set this to the number of milliseconds to pause between each individual character. e.g.: Let>SK_DELAY=10

Note that this command sends keystrokes and therefore any keys sent will be subject to the current state of modifier keys and the caps lock key. To ignore the caps lock state and send the text as entered set SK_IGNORECAPS to 1. (Let>SK_IGNORECAPS=1).

This command is commonly abbreviated to Send.

Abbreviation : [Sen \(Send\)](#)
See Also : [Ascii](#) 

Examples

[Send Character/Text](#)>Here Is Some Text ...

The following example simulates pressing ALT-FX, a standard key combination for closing a program :

```
Press ALT
Send Character/Text>fx
Release ALT
```

Variables can be used :

```
Let>SomeText=Hello World
Send Character/Text>SomeText
```

4.12.11 WaitKeyDown

WaitKeyDown>key_to_wait_for

WaitKeyDown pauses execution until the specified key is pressed. For an ordinary character key specify the character of the key. For other keys use the virtual key code preceded by VK:

WaitKeyDown>H - waits for the H key to be pressed

WaitKeyDown>VK101 - waits for virtual key code 101 (Numpad 5) to be pressed.

A list of virtual key codes can be found here: <http://www.mjtnet.com/vkcodes.htm>

Abbreviation : **WKD**

See also : [Wait](#)^[121], [WaitWindowOpen](#)^[140], [WaitWindowClosed](#)^[139], [WaitCursorChanged](#)^[101], [WaitPixelColor](#)^[101], [WaitReady](#)^[121]

4.13 Messages

4.13.1 Ask

Ask>prompt,result_variable

Displays a Yes, No dialog box with the specified prompt. If the user presses 'Yes', result_variable is set to 'YES', else result_variable becomes 'NO'.

The result is returned in upper case.

prompt can be a variable containing the prompt to display.

It is possible to set a timeout by setting the value of ASK_TIMEOUT before issuing the Ask command. ASK_TIMEOUT takes a number of milliseconds. If the Ask box is still active after the timeout has elapsed it will be closed and the result set to YES.

See also : [Input](#)^[88], [Message](#)^[88], [MessageModal](#)^[89]

Example

```
Ask>Do you want to continue ?,continue
```

4.13.2 Input

Input>variable,prompt[,default_value]

Displays a dialog box to request information from the user. The dialog box displays the prompt specified in prompt and accepts input into variable. Optionally, a default value can be specified.

The Input box now also has a file browse button, making it useful for accepting filenames from the user. The file browse button can be hidden by setting INPUT_BROWSE to 0 (Let>INPUT_BROWSE=0). Set INPUT_BROWSE to 2 to display a folder browse dialog for locating folders instead of files.

If the Input dialog is cancelled (cancel is pressed) Input returns an empty string.

prompt can be a variable, containing the prompt to display.

It is possible to mask the value entered by the user with asterisks by setting the INPUT_PASSWORD variable to 1. Set to zero for default behaviour.

It is possible to set a timeout by setting the value of INPUT_TIMEOUT before issuing the Input command. INPUT_TIMEOUT takes a number of milliseconds. If the Input box is still active after the timeout has elapsed it will be closed and the result set to YES.

Abbreviation : [Inp](#)

See also : [Ask](#)^[87], [Message](#)^[88], [MessageModal](#)^[89]

Example

```
Input>name,Please enter your name ...
```

4.13.3 Message

Message>message_text

Displays a message box containing the text specified in message_text. In order that execution of the script can continue, these message boxes are not modal. This means they can be used to display information even if the script is not being run interactively. For modal message boxes, use [MessageModal](#)^[89].

To make a message box stay on top, set the variable MSG_STAYONTOP to 1. Likewise, setting MSG_CENTERED to 1 will ensure that the message box is always shown centrally on the screen.

You can set/get the height and width of the message box using the variables MSG_HEIGHT and MSG_WIDTH. Set the position of the message box using the variables MSG_XPOS and MSG_YPOS.

Abbreviation : [MSG](#)

See also : [MessageModal](#)^[89], [Input](#)^[88], [Ask](#)^[87]

Example

```
Message>Hello World!
```

or with variables ..

```
Let>mymsg=Hello World!
Message>mymsg
```


To force a new line in a message box, use the CRLF system variable :

```
Message>Hello World %CRLF% %CRLF%End Message.
```

This would display :

Hello World

End Message.

To make the message box stay on top (above all other windows) :

```
Let>MSG_STAYONTOP=1  
Message>Hello World
```

4.13.4 MessageModal

MessageModal>message_text

Displays a message box containing the text specified in message_text. With this command the Message box is modal. This means that the script will not continue until OK is pressed. For non modal message boxes use [Message](#)^[88].

To make a message box stay on top, set the variable MSG_STAYONTOP to 1. Likewise, setting MSG_CENTERED to 1 will ensure that the message box is always shown centrally on the screen.

You can set/get the height and width of the message box using the variables MSG_HEIGHT and MSG_WIDTH. Set the position of the message box using the variables MSG_XPOS and MSG_YPOS.

Abbreviation : [MDL](#)

See also : [Message](#)^[88], [Input](#)^[88], [Ask](#)^[87]

Example

```
MessageModal>Hello World!
```

or with variables ..

```
Let>mymsg=Hello World!  
MessageModal>mymsg
```

4.14 Miscellaneous

4.14.1 Assigned

Assigned>Variable_Name,result

Used to determine whether or not a variable has been assigned. Assigned sets result to TRUE if Variable_Name exists, FALSE if not.

Abbreviation : [ASS](#)

Example

```
Assigned>strName,strNameExists  
If>strNameExists=TRUE,DoName,SkipName
```

4.14.2 BlockInput

BlockInput>block

Blockinput prevents or enables keyboard and mouse input from the user.

Set block to 1 to block input from the user.

Set block to 0 to enable input.

Be aware that this prevents all user input apart from CTRL-ALT-DEL. If you need to stop a running script that has blocked input press CTRL-ALT-DEL and then stop the script.

NB: This function has no effect under Windows 95. On Windows 98/Me and NT (pre SP6) this function will also prevent Macro Scheduler from sending input. On Windows NT4 with SP6, 2000, XP and Windows Server 2003 it will block the user but will still allow Macro Scheduler to send input.

Abbreviation: [BLO](#)

Example:

```
//Block input only if version is not 98/Me
Pos>98,WIN_VER,1,IsWin98orMe
If>{(%IsWin98orMe%=0)}
    BlockInput>1
Endif

Run>Notepad
WaitWindowOpen>Notepad*
Send>Hello World

BlockInput>0
```

4.14.3 DateStamp

DateStamp>filename,comment

Outputs the date and time in milliseconds and the given comment to the given text file.

Abbreviation : [DAT](#)

See also : [WriteLn](#)^[65], [TimeStamp](#)^[100]

Example

```
DateStamp>c:\temp\mylogfile.txt,Macro Started
```

The file entry would appear :

1999-08-06:13:38:10:352 - Macro Started

4.14.4 ExportData

ExportData>Data_Label,Filename

Exports binary data from the script to the specified file. Use this in conjunction with Tools/Import Binary File in the editor. This allows you to include binary files in your scripts and export them for use at run time.

Abbreviation: EXP

Example

```
ExportData>CALC.DLL_DATA,%SCRIPT_DIR%\calc.dll
```

```
CALC.DLL_DATA:
4D5A50000200000004000F00FFFF0000B8000000000000...
```

4.14.5 GetCheckBox

GetCheckBox>window_title,object_caption,result

GetCheckBox determines whether or not the given check box, or radio button, is checked. result is set to 1 if checked, 0 if not checked, or -1 if the command failed to locate the given check box. Specify the window title of the window containing the check box/radio button, and the object's caption. The caption of a check box or radio button is the text appearing next to it. The caption must be specified accurately with attention paid to case. Where a letter is underlined indicating a shortcut key, enter the '&' character before it.

window_title can end with an asterisk to indicate a substring match. See [SetFocus](#)^[138] for a more detailed explanation of how this works.

Abbreviation : [CBX](#)

See also : [SetCheckBox](#)^[99]

Example

```
GetCheckBox>Internet Explorer Prop*,Never dial a &connection,res
If>res=1,checked,unchecked
```

```
Label>checked
Message>The checkbox is checked!
Goto>end

Label>unchecked
Message>The checkbox is not checked!
Label>end
```

4.14.6 GetControlText

GetControlText>WindowTitle,ClassName,Instance,Result

Returns the text of the control specified by ClassName and Instance on the window specified by WindowTitle.

Result contains the text of the control if found. If the window is not found Result contains ##NOSUCHWINDOW##. If the class and instance was not found ##NOSUCHOBJECT## is returned.

If WIN_USEHANDLE is set to 1 WindowTitle must be a window handle.

To determine class names of objects on windows use [View System Windows](#)^[24]. A window may contain several objects of the same class name. Instance is used to determine which instance of the class to use.

GetControlText retrieves the published text property of the specified object. Not all text that you

see on the screen is retrievable in this way. Some text is painted via lower level routines and some text is graphical. Some objects, such as labels, are not windowed controls, and therefore text associated with them cannot be retrieved with GetControlText. Try the new GetTextAtPoint, GetTextInRect and GetWindowTextEx commands which use lower level hooks to trap more text.

Abbreviation: **GCT**

See Also : [SetControlText](#)^[100], [GetTextAtPoint](#)^[125], [GetTextInRect](#)^[125], [GetWindowTextEx](#)^[126], [WaitScreenText](#)^[126]

Example:

```
GetControlText>Notepad*,Edit,1,npEdit
```

npEdit will now contain the contents of the notepad edit window.

4.14.7 GetEnvVar

GetEnvVar>EnvironmentVariable,LocalVariable

Retrieves the value of the specified environment variable into the local variable given in LocalVariable.

Abbreviation: **GEV**

See Also: [SetEnvVar](#)^[100]

Example:

```
GetEnvVar>TMP,myTMPDir
MessageModal>myTMPDir
```

4.14.8 GetListItem

GetListItem>WindowTitle,ClassName,Instance,Text,Column,Case,Partial,Result,Handle

Returns in Result the row index of the text specified in Text in the specified ListView (List Box) object.

WindowTitle: The title of the window containing the ListView object

ClassName: Usually SysListView32 but may differ. Use View System Windows to determine class name.

Instance: The zero based index of the instance to use (a window may contain several listview objects).

Text: The text to search for in the list box.

Column: The column number to search in. 0 for first column or if only one column.

Case: 1 to perform a case sensitive match, 0 for non-case sensitive.

Partial: 1=partial, 0=full. A partial match will match Text at the **start** of the item text.

Result: Gets set to the zero based index of the item or -1 if not found.

Handle: Gets set to the handle of the ListView object.

The zero based index is returned in Result. Handle will be set to the handle of the ListView object.

Some list boxes allow you to "drill down" to the item you want by typing the value of the item. To select an item in these kinds of boxes all you therefore need to do is send keystrokes to them using the [Send](#)^[86] command. However, some list boxes do not "drill down" and this is where GetListItem is needed. You may know the item text you want to select but can not predict it's index. Use GetListItem to retrieve the index and then you know how many times to "Press Down".

Abbreviation: [GLI](#)
 See Also: [GetTreeNode](#) ^[94]

Example

This example uses `GetListItem` to retrieve the index of an item in the Display Properties backgrounds box.

```
//Open Display Properties
Run>rundll32 shell32.dll,Control_RunDLL Desk.cpl
WaitWindowOpen>Display Properties

//Select Desktop Tab
Press CTRL
Press TAB
Release CTRL
WaitReady>0

//Get listindex of "Azul" in Background box
GetListItem>Display Properties,SysListView32,0,Azul,0,0,0,nDx,lvHwnd

//Select it
Press Home
Press Down * nDx

WaitReady>0

//OK!
Press Enter
```

4.14.9 GetPixelColor

GetPixelColor>X,Y,result

Returns the pixel color of the specified screen coordinate. Enter the coordinates in X and Y, and specify a variable to store the pixel color in.

Abbreviation : [GPC](#)
 See also : [WaitPixelColor](#) ^[101], [GetRectChecksum](#) ^[93], [WaitRectChanged](#) ^[101]

Example

```
GetPixelColor>652,355,PC
Message>Pixel Color is : %PC%
```

4.14.10 GetRectChecksum

GetRectChecksum>TLX,TLY,BRX,BRY,result_value

When passed the top left and bottom right screen coordinates of a rectangle, this function returns the checksum of that rectangle. The checksum is based on the color and position in the rectangle of each individual pixel. The checksum is returned in the variable `result_value`.

This command will allow for waiting for a specific part of the screen to become equal to a known graphic. Use the command when the graphic to be compared against is present to determine it's checksum. The example below shows a loop which will cause the script to wait until the rectangle contains the required graphic.

TLX - Top Left corner X coordinate

TLY - Top Left corner Y coordinate
 BRX - Bottom Right corner X coordinate
 BRY - Bottom Right corner Y coordinate

This function was optimized in version 10 to improve performance and now operates very quickly, taking around 0.1 second to scan a 500x500 area of screen. However, on some systems it may consume a lot of CPU cycles in the process. If you need to force the function to yield to the system more you can set GRC_YIELD to 1, but this will have the side effect of slowing the function down, as it gives over CPU time to other processes.

Abbreviation : **GRC**

See also : [WaitRectChanged](#)^[10], [WaitPixelColor](#)^[10], [GetPixelColor](#)^[93]

Example

```
Label>waitforit
GetRectChecksum>10,10,15,15,cs
If>cs=6.20066481623195E16,Done
Goto>waitforit
Label>Done
Message>There You Go
```

4.14.11 GetTreeNode

GetTreeNode>WindowTitle,ClassName,Instance,Text,Case,Partial,Result,Handle

Returns in Result the index of the visible tree view node whose caption matches Text.

WindowTitle: The title of the window containing the TreeView object

ClassName: Usually SysTreeView32 but may differ. Use View System Windows to determine class name.

Instance: The zero based index of the instance to use (a window may contain several treeview objects).

Text: The text to search for in the list box.

Case: 1 to perform a case sensitive match, 0 for non-case sensitive.

Partial: 1=partial, 0=full. A partial match will match Text at the **start** of the item text.

Result: Gets set to the zero based index of the item or -1 if not found.

Handle: Gets set to the handle of the TreeView object.

The zero based index of the node is returned in Result. Handle will be set to the handle of the ListView object. Only visible nodes are searched.

Abbreviation: **GTN**

See Also: [GetListItem](#)^[92]

Example

This example uses GetTreeNode to retrieve the index of an item in Explorer's Folders treeview.

```
//Assumes Windows Explorer is open at "My Documents" with
//explorer bar showing Folders
GetTreeNode>My Documents,SysTreeView32,0,My Computer,1,1,result,handle

Let>WIN_USEHANDLE=1
SetFocus>handle
Let>WIN_USEHANDLE=0

Press Home
Press Down * result
Press Right
```

4.14.12 LibFree

LibFree>module_handle

Frees a library previously loaded with LibLoad.

Abbreviation : [LFR](#)

See Also: [LibLoad](#)⁹⁷, [LibFunc](#)⁹⁵

Examples

```
LibLoad>user32,hDll
Let>WIN_USEHANDLE=1
GetActiveWindow>win,x,y
LibFunc>hDll,GetWindowTextLengthA,wtlen,win
Let>buffer_SIZE=wtlen
LibFunc>hDll,GetWindowTextA,gwt,win,buffer,wtlen
MessageModal>Window Text: %gwt_2%
LibFree>hDll
```

4.14.13 LibFunc

LibFunc>module,function,resultvar[[ref:]var1][ref:]var2[...][ref:]varn]]

LibFunc can be used to call DLL functions and is therefore an ideal way of extending the language using custom DLLs or for calling Win32 API functions. (NB: Alternatively, for language extensions, create an ActiveX object and access it using VBScript).

LibFunc takes the following parameters:

module: the DLL module filename to load or a module handle created with a previous call to [LibLoad](#)⁹⁷
 function: the name of the function in the DLL.
 resultvar: numeric value returned by the function.
 parameter list: See below.

You can specify as many parameters as needed by the DLL function. To pass numeric parameters by reference precede the value with "ref:" (without the quotes). See examples below.

As string variables are actually pointers to memory locations strings are effectively always passed by reference and can be modified by the DLL. String buffer sizes are set to 255 characters by default. To set the size of a string variable to some other value create a parmname_SIZE variable, where parmname is the name of the string variable being passed, set to the length required. See window title example below.

LibFunc determines whether a variable is a numeric integer or a string automatically by looking at it's contents. However, if you need to send a number as a string you can force it to be a string by preceding the parameter with "str:" (without the quotes).

Note that LibFunc supports long integer and string types only. LibFunc supports ANSI strings only. For Unicode strings use LibFuncW.

As well as returning the function result in resultvar LibFunc creates new variables corresponding to each of the passed parameters with the format resultvar_n where n is the index of the parameter, starting from 1. If the function changed the value of any of the parameters these new variables will contain the new value. Otherwise they will contain the value passed. E.g. if resultvar is set to "answer" and you specified three parameters LibFunc will return four variables called answer, answer_1, answer_2 and answer_3. See examples.

Beware: When you pass data to DLLs you are passing references to memory areas. If you send the wrong type of information or the wrong number of arguments you are likely to cause an access violation error in the DLL which may cause Macro Scheduler to crash. This function is recommended for experienced developers only.

Abbreviation : [Lib](#)

See Also: [LibLoad](#)^[97], [LibFree](#)^[95]

Examples

Get Windows System Directory:

```
LibFunc>Kernel32,GetSystemDirectoryA,dir,buffer,255
MessageModal>System Directory: %dir_1%
```

Use ShellExec to open a file/run an application:

```
LibFunc>shell32,ShellExecuteA,r,0,open,notepad,,1
```

Get free disk space on drive C:

```
LibFunc>kernel32,GetDiskFreeSpaceExA,spacefree,c:\,REF:0,REF:0,REF:0
Let>used=spacefree_3/1024
Let>free=spacefree_4/1024
MessageModal> Disk space used: %used%Kb %CRLF% Free space: %free%Kb
```

Display a system yes/no message box:

```
Let>YesNo=4
LibFunc>user32,MessageBoxA,r,0,Hello World,Message!,YesNo
If>r=6
    MessageModal>You Pressed Yes
else
    MessageModal>You Pressed No
endif
```

Use the system API Beep function to play sounds:

```
Let>k=200
Repeat>k
    LibFunc>kernel32,Beep,r,k,50
    Let>k=k+10
Until>k,1000
```

Run a function in a custom DLL:

```
LibFunc>C:\mydll\testdll.dll,ChangeText,r,hello
MessageModal>%r% %r_1%
```

Get Active Window Title:

```
Let>WIN_USEHANDLE=1
GetActiveWindow>win,x,y
LibFunc>user32,GetWindowTextLengthA,wtlen,win
Let>buffer_SIZE=wtlen
Let>wtlen=wtlen+1
LibFunc>user32,GetWindowTextA,gwt,win,buffer,wtlen
MessageModal>Window Text: %gwt_2%
```


4.14.14 LibFuncW

LibFuncW>module,function,resultvar[,[[ref:]var1][,[ref:]var2][,...][,[ref:]varn]]

LibFuncW is the Unicode version of [LibFunc](#) ⁹⁵.

4.14.15 LibLoad

LibLoad>module_name,module_handle

Loads the DLL library specified in module_name and returns a handle to the library which can be used in subsequent LibFunc calls. Libraries loaded with LibLoad should always be freed with a call to LibFree after being used.

Abbreviation : **LLD**

See Also: [LibFunc](#) ⁹⁵, [LibFree](#) ⁹⁵

Examples

```
LibLoad>user32,hDll
Let>WIN_USEHANDLE=1
GetActiveWindow>win,x,y
LibFunc>hDll,GetWindowTextLengthA,wtlen,win
Let>buffer_SIZE=wtlen
Let>wtlen=wtlen+1
LibFunc>hDll,GetWindowTextA,gwt,win,buffer,wtlen
MessageModal>Window Text: %gwt_2%
LibFree>hDll
```

4.14.16 PlayWav

PlayWav>wav_file

Plays a .WAV sound file.

Abbreviation : **Pla**

Example

```
PlayWav>c:\windows\media\chimes.wav
```

4.14.17 PushButton

PushButton>window_title,button_caption

Attempts to 'click' the specified button of the specified window.

window_title can contain an asterisk (*) as with all other window functions. For buttons that have a hot key associated with them, and represented on the button by an underscored letter, pass a & character before that letter. e.g.: for a button called 'C_l_o_s_e', send &C_l_o_s_e.

This command works by attempting to send the BM_CLICK message to the button when it finds a button in the specified window with the given caption.

This will only work with objects of the standard 'Button' class. It may not work for all buttons on all windows. For instance, it doesn't work for buttons on html documents in Netscape Navigator 4.05 or Internet Explorer 3.02, as they are not real buttons. For these use the [MouseOver](#)^[105] command.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to locate the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and stops at the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation : [PUS](#)

See also : [MouseOver](#)^[105]

Example

```
Run Program>rundll32.exe shell32.dll,Control_RunDLL TimeDate.cpl
WaitWindowOpen>Date/Time Properties
...
PushButton>Date/Time*,OK
```

4.14.18 Random

Random>Range,Result

Returns a random number within the specified range where $0 \leq \text{Result} < \text{Range}$.

The seed is set automatically and is stored in the RND_SEED variable. It is possible to set the seed programmatically by modifying the value of RND_SEED.

Result is a variable in which the result is stored.

Abbreviation : [RAN](#)

Example

```
Random>6,DiceResult
Let>DiceResult=DiceResult+1
Message>You threw a %DiceResult%
```

4.14.19 RGB

RGB>red,green,blue,result

Creates a color code based on the specified mix of red, green and blue intensities.

Red, Green and Blue can be numbers from 1 to 255. The higher the number the greater the intensity of that color.

Example:

```
RGB>20,50,200,aColor
```

4.14.20 SelectMenu

SelectMenu>WindowTitle,MenuIndex[,SubMenuIndex[,SubMenuIndex[,...]]]

Selects the menu item specified by MenuIndex and optional SubMenuIndex parameters of the window specified by WindowTitle.

MenuIndex and SubMenuIndex are integer numeric values. For MenuIndex 0 is the first menu item on the left of the menu bar, 1 the next and so on. For SubMenuIndex, 0 is the first item in the submenu, 1 the next etc. The index includes separating lines, so remember to count them.

For example, to select the Save submenu item in Notepad, Save is the 3rd Submenu item of the File menu. The File menu is the 1st menu item in the menu bar. So MenuIndex is 0 and SubMenuIndex is 2. There are no further submenus. So the command would look like this:

```
SelectMenu>Notepad*,0,2
```

WindowTitle can contain the asterisk wildcard to indicate that it is a substring search (See SetFocus for more explanation). If the WIN_USEHANDLE variable is set to 1 WindowTitle must be a window handle rather than window title.

Abbreviation: **MNU**

Note: SelectMenu only works with standard Windows menus. Many menus are now in fact custom built menus that are other controls made to look like menus. SelectMenu will not therefore work with these objects.

4.14.21 SetCheckBox

SetCheckBox>window_title,object_caption,TRUE|FALSE

SetCheckBox is used to check or uncheck a given checkbox or radio button. Specify the window title of the window containing the check box/radio button, and the object's caption. The caption of a check box or radio button is the text appearing next to it. The caption must be specified accurately with attention paid to case. Where a letter is underlined indicating a shortcut key, enter the '&' character before it.

window_title can end with an asterisk to indicate a substring match. See [SetFocus](#)^[138] for a more detailed explanation of how this works.

Abbreviation : **SBX**

See also : [GetCheckBox](#)^[91]

Example

```
GetCheckBox>Internet Explorer Prop*,Never dial a &connection,res
If>res=1,checked,unchecked

Label>checked
SetCheckBox>Internet Explorer Prop*,Never dial a &connection,FALSE
Goto>end

Label>unchecked
SetCheckBox>Internet Explorer Prop*,Never dial a &connection,TRUE
Label>end
```

4.14.22 SetControlText

SetControlText>WindowTitle,ClassName,Instance,NewText

Sets the text of the control specified by ClassName and Instance on the window specified by WindowTitle.

If WIN_USEHANDLE is set to 1 WindowTitle must be a window handle.

To determine class names of objects on windows use [View System Windows](#)^[24]. A window may contain several objects of the same class name. Instance is used to determine which instance of the class to use.

Abbreviation: [SCT](#)

See Also : [GetControlText](#)^[91]

Example:

```
SetControlText>Notepad*,Edit,1,Hello World
```

This will set the Notepad edit window to the text "Hello World".

4.14.23 SetEnvVar

SetEnvVar>EnvironmentVariable,Value

Sets the value of the specified environment variable with Value.

Abbreviation: [SEV](#)

See Also: [GetEnvVar](#)^[92]

Example:

```
SetEnvVar>name,fred
```

4.14.24 TimeStamp

TimeStamp>filename,comment

Outputs the time in milliseconds and the given comment to the given text file.

Abbreviation : [TIM](#)

See also : [WriteLn](#)^[65], [DateStamp](#)^[90]

Example

```
TimeStamp>c:\temp\mylogfile.txt,Macro Finished
```

The file entry would appear :

15:43:03:066 - Macro Finished

4.14.25 WaitCursorChanged

WaitCursorChanged>Timeout

This command causes Macro Scheduler to wait until the cursor of the foreground window changes. If it doesn't change within the number of seconds specified in Timeout, the command stops waiting and the variable WCC_RESULT is set to FALSE. WCC_RESULT is TRUE if the command terminated because the foreground window cursor changed within the specified time. If Timeout is set to 0, the command will wait indefinitely.

This command is useful for waiting for applications to become idle. For example, it can be used after initiating some operation in an application that invokes the hourglass cursor, so that you can wait for the application to become idle again.

Abbreviation : [WCC](#)

See also : [Wait](#)^[12†], [WaitWindowOpen](#)^[140†], [WaitWindowClosed](#)^[139†], [WaitPixelColor](#)^[10†]

Example

```
Change Directory>c:\program files\agent\data
Run Program>"c:\program files\agent\agent.exe"
WaitCursorChanged>500
SetFocus>Agent*
```

4.14.26 WaitPixelColor

WaitPixelColor>ColorCode,X,Y,Timeout

This command causes Macro Scheduler to wait until the pixel colour at the specified pixel coordinates changes to the colour specified in ColorCode. If it doesn't change to that colour within the number of seconds specified in Timeout, the command stops waiting and the variable WPC_RESULT is set to FALSE. WPC_RESULT is TRUE if the command terminated because the colour changed to the specified colour within the specified time. If Timeout is set to 0, the command will wait indefinitely.

To determine the correct colour code to use, click the drop down menu button next to the cursor position monitor on the macro properties form. From the drop down menu select 'Pixel Color' and the pixel colour of the current mouse cursor position will be added to the display along with the X and Y coordinates. Now you can determine the correct colour code of any pixel on the screen.

Abbreviation : [WPC](#)

See also : [Wait](#)^[12†], [WaitWindowOpen](#)^[140†], [WaitWindowClosed](#)^[139†], [WaitCursorChanged](#)^[10†], [GetPixelColor](#)^[93†], [GetRectChecksum](#)^[93†], [WaitRectChanged](#)^[10†]

Example

```
WaitPixelColor>16777215,652,355,10
Message>WPC_RESULT
```

4.14.27 WaitRectChanged

WaitRectChanged>TLX,TLY,BRX,BRY,Timeout

This command causes Macro Scheduler to wait until the image bound by the specified pixel coordinates changes. If it doesn't change within the number of seconds specified in Timeout, the command stops waiting and the variable WRC_RESULT is set to FALSE. WRC_RESULT is TRUE if the command terminated because the image changed within the specified time. If

Timeout is set to 0, the command will wait indefinitely.

TLX - Top Left corner X coordinate
 TLY - Top Left corner Y coordinate
 BRX - Bottom Right corner X coordinate
 BRY - Bottom Right corner Y coordinate

This function was optimized in version 10 to improve performance and now operates very quickly, taking around 0.1 second to scan a 500x500 area of screen. However, on some systems it may consume a lot of CPU cycles in the process. If you need to force the function to yield to the system more you can set GRC_YIELD to 1, but this will have the side effect of slowing the function down, as it gives over CPU time to other processes.

Abbreviation : [WRC](#)

See also : [Wait](#)^[121], [WaitWindowOpen](#)^[140], [WaitWindowClosed](#)^[139], [WaitCursorChanged](#)^[101], [WaitPixelColor](#)^[101], [GetRectChecksum](#)^[93]

Example

```
WaitRectChanged>10,10,100,100,10
Message>WRC_RESULT
```

4.15 Mouse Commands

4.15.1 GetCaretPos

GetCaretPos>X,Y,Relative

GetCaretPos retrieves the X, Y coordinates of the text caret (text cursor) of the foreground window. The coordinates are returned in the variables provided. Set Relative to 0 to return absolute screen coordinates, or 1 to return coordinates relative to the window.

Abbreviation: [GTP](#)

See Also: [GetCursorPos](#)^[102]

Example

```
SetFocus>notepad*
WaitReady>0
GetCaretPos>X,Y,0
MouseMove>X,Y
```

4.15.2 GetCursorPos

GetCursorPos>X,Y

Returns the X and Y coordinates of the current mouse cursor position. X and Y are variables in which to store the coordinates.

Abbreviation : [GCP](#)

See also : [GetWindowPos](#)^[134], [GetCaretPos](#)^[102]

Example

```
GetCursorPos>X,Y
Message>Current Position : %X%,%Y%
```

4.15.3 LClick

Simulates a left button mouse click at the current point on the screen.

The following commands will produce the same result :

LDown
LUp

Abbreviation : **LCI**

See also : [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.4 LDbClick

Simulates a left mouse button double click at the current point on the screen.

The following will achieve the same result :

LClick
LClick

or

LDown
LUp
LDown
LUp

Abbreviation : **LDb**

See also : [LDown](#)^[103], [LUp](#)^[103], [LClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.5 LDown

Simulates a press of the left mouse button. This is like pressing the mouse button down but not releasing it. It is half of a click.

Issuing this command and then using MouseMove would implement dragging. Use LUp to complete the operation.

Abbreviation : **LDo**

See also : [LClick](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.6 LUp

Releases the left mouse button. It is the latter half of a click.

See LDown.

See also : [LDown](#)^[103], [LClick](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.7 MClick

Simulates a middle button mouse click at the current point on the screen.

The following commands will produce the same result :

MDown
MUp

Abbreviation : **MCI**

See also : [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [LClick](#)^[103], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.8 MDbClick

Simulates a middle mouse button double click at the current point on the screen.

The following will achieve the same result :

MClick
MClick

or

MDown
MUp
MDown
MUp

Abbreviation : **MDb**

See also : [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [MDown](#)^[104], [MUp](#)^[106], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.9 MDown

Simulates a press of the middle mouse button. This is like pressing the mouse button down but not releasing it. It is half of a click.

Abbreviation : **MDo**

See also : [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDbClick](#)^[106], [RUp](#)^[107], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.10 MouseMove

MouseMove>X,Y

Moves the mouse cursor to screen position X,Y. 0,0 is the upper left hand corner of the screen. The maximum limits are determined by your screen resolution settings. Variables containing the coordinates can be used in the command.

To help determine a particular point on the screen, the macro window has a cursor monitor which updates as you move the cursor. See [Creating Scripts](#)^[4].

Abbreviation : **Mou**

See also : [MouseMoveRel](#)^[105], [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107]

, [RUp](#)^[107], [RDbClick](#)^[106], [MouseOver](#)^[105]

Example

If position 504,252 is within the area taken up by a button, the following script would cause that button to be clicked :

```
MouseMove>504,252
LClick
```

4.15.11 MouseMoveRel

MouseMoveRel>X,Y

Moves the mouse cursor to the position X,Y relative to the upper left corner of the window currently in focus. 0,0 will be the upper left hand corner of the active window. Variables containing the coordinates can be used in the command.

The advantage of this command over the MouseMove command, is that this will not fail when the window changes its position or resizes, or if the screen resolution is changed.

To help determine a particular point on the screen, the macro window has a cursor monitor which updates as you move the cursor. See [Creating Scripts](#)^[4]. To determine a point relative to a specific window, try moving that window so that its upper left corner is in the upper left corner of the screen (position 0,0). Maximising the app would achieve the same result. Then you can use the values displayed in the cursor position monitor of Macro Scheduler.

Abbreviation : **MMR**

See also : [MouseMove](#)^[104], [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MouseOver](#)^[105]

Example

If position 40,50 is a point relative to the current window, on a button, the following script would cause that button to be clicked :

```
MouseMoveRel>40,50
LClick
```

4.15.12 MouseOver

MouseOver>window_title,button/object_caption

Attempts to position the mouse cursor over the specified button (or object) of the specified window.

window_title can contain an asterisk (*) as with all other window functions. For buttons that have a hot key associated with them, and represented on the button by an underscored letter, pass a & character before that letter. e.g.: for a button called 'Close', send &Close.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to locate the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and stops at the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation : [MVR](#)

Example

```
Run Program>rundll32.exe shell32.dll,Control_RunDLL TimeDate.cpl
WaitWindowOpen>Date and Time Properties
...
MouseOver>Date/Time*,OK
LClick
```

4.15.13 MUp

Releases the middle mouse button. It is the latter half of a click.

See MDown.

See also : [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDbClick](#)^[106], [RDown](#)^[107], [MDown](#)^[104], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.14 RClick

Simulates a right button mouse click at the current point on the screen.

The following commands will produce the same result :

RDown
RUp

Abbreviation : [RCI](#)

See also : [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [LClick](#)^[103], [RDown](#)^[107], [RUp](#)^[107], [RDbClick](#)^[106], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.15 RDbClick

Simulates a right mouse button double click at the current point on the screen.

The following will achieve the same result :

RClick
RClick

or

RDown
RLUp
RDown
RUp

Abbreviation : [RDb](#)

See also : [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDown](#)^[107], [RUp](#)^[107], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.16 RDown

Simulates a press of the right mouse button. This is like pressing the mouse button down but not releasing it. It is half of a click.

Abbreviation : **RDo**

See also : [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDbClick](#)^[106], [RUp](#)^[107], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.17 RUp

Releases the right mouse button. It is the latter half of a click.

See RDown.

See also : [LClick](#)^[103], [LDown](#)^[103], [LUp](#)^[103], [LDbClick](#)^[103], [RClick](#)^[106], [RDbClick](#)^[106], [RDown](#)^[107], [MDown](#)^[104], [MUp](#)^[106], [MDbClick](#)^[104], [MClick](#)^[104], [MouseMove](#)^[104], [MouseMoveRel](#)^[105], [MouseOver](#)^[105]

4.15.18 Toolbar

Toolbar>window_title,toolbar_index,button_index

NB: This function does not work in Windows 95

Positions the mouse cursor over the specified toolbar button of the specified window. Works only with toolbar objects of class `ToolbarWindow32`.

`toolbar_index` and `button_index` are integer values and begin at zero, where a `toolbar_index` of 0 indicates the first toolbar created on the window, and 1 the next and so on. Use Tools/View System Windows to find objects of class `ToolbarWindow32` belonging to a window, and/or experiment to determine which index should be used.

Many modern menu systems are actually `ToolbarWindow32` class objects rather than menu objects. Internet Explorer uses a `ToolbarWindow32` for it's main menu.

`window_title` can contain an asterisk (*) as with all other window functions.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to locate the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and stops at the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation : **TBR**

Example

```
Toolbar>Internet Explorer*,0,2
LClick
```

4.16 Numeric Functions

4.16.1 Add

Add>Value,Number

Adds Number to Value.

Interpreted as $\text{Value} = \text{Value} + \text{Number}$

Value must be a variable containing either a numeric or date value. Number can be either a literal number or a variable containing a numeric value.

For date values this function will add the number of days, represented in Number to the given date value.

See also : [Sub](#)¹⁰⁹

Example

```
Let>Counter=5
Add>Counter,2
```

i.e. $\text{Counter} = \text{Counter} + 2$

In this example the numeric variable, Counter, is given a new value of 7.

4.16.2 Let

Let>variable_name=value

Let is used to assign a value to a variable.

Where parameters are passed to commands, variables can also be passed. Variables can also be embedded within a parameter by enclosing the variable name within % symbols.

Let can also be used to perform basic calculations, and to concatenate strings. NB. If you want to assign to a variable a string that already contains a + sign, add an extra + sign to avoid the two strings being concatenated.

The **value** can contain a complex expression:

Complex Expressions

The expression must be contained within curly braces "{" and "}". String literals must be delimited with double quotes ("), e.g.:

"string". Variables must be delimited with % symbols, e.g.: %VarA%.

Several types of operators and functions can be used with complex expressions. For more details see [Complex Expressions](#)⁴¹.

Examples

```
Let>name=freddy
Let>a=5

Let>path=c:\Program Files\
Run Program>%path%myapp.exe
```

```
Let>k=k+1
Let>A=60-4
Let>A=5*3
Let>F=75/32
Let>Name=John+ Smith
Name would now equal John Smith
Let>Name=John++Smith
Name would equal 'John+Smith'
Let>MyVal={Upper( "WoRLd" ) }
Let>MyVal={5 * 10 + 25}
Let>d={%a%+%b%*%c%-23}
```

4.16.3 Sub

Sub>Value,Number

Subtracts a number from a value.

Interpreted as Value = Value - Number

Value must be a variable containing a numeric or date value. Number can be either a literal number or a variable containing a numeric value.

For date values this function will subtract the number of days, represented in Number from the given date value.

See also : [Add](#)^[108], [Let](#)^[108]

Example

```
Let>Counter=5
Sub>Counter,2
```

i.e. Counter=Counter-2

In this example the numeric variable, Counter, is given a new value of 3.

```
Let>Counter=Counter-1
```

Will now do the same thing.

4.17 Registry Functions

4.17.1 RegistryDelKey

RegistryDelKey>root_key,key

Removes a key from the registry.

Use all registry functions with caution. If you are unfamiliar with the Windows Registry we recommend that you do not use these functions. Removing or modifying a registry entry that you did not create could cause your system to become unstable.

Abbreviation : **RDK**

See also : [RegistryDelVal](#)^[110], [RegistryReadKey](#)^[110], [RegistryWriteKey](#)^[110]

Example

```
RegistryDelKey>HKEY_CURRENT_USER,Software\MJTNET\Temp
```

4.17.2 RegistryDelVal

RegistryDelKey>root_key,key,Value

Removes a value from the registry.

Use all registry functions with caution. If you are unfamiliar with the Windows Registry we recommend that you do not use these functions. Removing or modifying a registry entry that you did not create could cause your system to become unstable.

Abbreviation : **RDV**

See also : [RegistryDelKey](#)^[109], [RegistryReadKey](#)^[110], [RegistryWriteKey](#)^[110]

Example

```
RegistryDelVal>HKEY_CURRENT_USER,Software\MJTNET\Temp,TestVal1
```

4.17.3 RegistryReadKey

RegistryReadKey>root_key,key,entry,result_variable

Reads the value of an entry from the registry. The value is stored in the given variable.

Use all registry functions with caution. If you are unfamiliar with the Windows Registry we recommend that you do not use these functions. Removing or modifying a registry entry that you did not create could cause your system to become unstable.

Abbreviation : **RRK**

See also : [RegistryDelKey](#)^[109], [RegistryDelVal](#)^[110], [RegistryWriteKey](#)^[110]

Example

```
RegistryReadKey>HKEY_CURRENT_USER,Control Panel\Colors,ActiveTitle,VActTitle  
Message>VActTitle
```

4.17.4 RegistryWriteKey

RegistryWriteKey>root_key,key,entry,value

Creates or modifies a registry entry. If the key and entry do not exist they are created and the new value assigned to the entry. If the key and entry already exist, the value is changed to the value provided. The function will create integer entries if the value specified is an integer, or else the new value will be a string. To force an integer value to be written as a string set REG_INTASSTR to 1 before calling RegistryWriteKey.

Use all registry functions with caution. If you are unfamiliar with the Windows Registry we recommend that you do not use these functions. Removing or modifying a registry entry that you did not create could cause your system to become unstable.

Abbreviation : [RWK](#)

See also : [RegistryDelKey](#)^[109], [RegistryDelVal](#)^[110], [RegistryReadKey](#)^[110]

Example

```
RegistryWriteKey>HKEY_CURRENT_USER,MyStuff,MyName,Fred Bloggs
```

4.18 Running Programs/Files

4.18.1 ExecuteFile

ExecuteFile>file_to_execute[,parameters]

Executes a file using the application associated with the given file's filetype.

file_to_execute can include a full path.

Abbreviation : [Exe](#)

See also : [RunProgram](#)^[111]

Example

```
ExecuteFile>report.doc
```

or

```
Let>filename=c:\my documents\accounts.xls  
ExecuteFile>filename
```

4.18.2 RunProgram

RunProgram>path

Executes a specified file. Files than can be executed are .exe, .bat, and .com files.

By setting the RP_WAIT variable to 1 prior to issuing the Run Program command the script will wait until the program launched by Run Program has terminated before continuing. The default value of RP_WAIT is 0.

Set RP_WAIT to 2 if you want the script to wait until it thinks the program is idle and ready for input. This only works for GUI applications and may not be appropriate for all applications. Some applications may appear ready for input before they really are. See [WaitWindowOpen](#)^[103] and [WaitReady](#)^[121] for other ways to wait until an application is ready.

By setting the RP_WINDOWMODE variable programs can be executed minimized, maximised, hidden or normal. RP_WINDOWMODE can be one of the following :

- 0: Hidden
- 1: Normal (default)
- 2: Minimized
- 3: Maximized

If you set RP_WINDOWMODE it is used by all subsequent Run Program commands, so remember to set it back if you don't want the same window mode to be used each time.

By default a message is displayed if this command encounters an error when running the specified program. Error messages from this command can be suppressed by setting RP_DISPLAYERROR to 0.

The result of the Run Program command is stored in the variable RP_RESULT.

The value of RP_RESULT differs depending on whether RP_WAIT is set to 0 or 1.

If RP_WAIT is 1 RP_RESULT will be set to the process exit code of the called program if successful. Therefore it will be zero if successful and the program does not return an exit code. -1 indicates that an error occurred.

If RP_WAIT is 0 (default) a value greater than 31 indicates success and the following values represent errors :

- 0 - The system was out of memory, or the executable file was corrupt, or relocations were invalid. (RP_WAIT=0 only)
- 2 - The file was not found.
- 3 - The path was not found.
- 5 - An attempt was made to dynamically link to a task, or there was a sharing or network protection error.
- 6 - The library required separate data segments for each task.
- 10 - The Windows version was incorrect.
- 11 - The executable file was invalid. It was either not a Windows-based application or there was an error in the .EXE image.
- 12 - The application was designed for OS/2.
- 13 - The application was designed for MS-DOS 4.0.
- 14 - The type of executable file was unknown.
- 15 - An attempt was made to load a real-mode application (developed for an earlier version of Windows).
- 16 - An attempt was made to load a second instance of an executable file containing multiple data segments that were not marked "read-only."
- 17 - Attempt in large-frame EMS mode to load a second instance of an application that links to certain non-shareable DLLs already in use.
- 18 - Attempt in real mode to load an application marked for protected mode only.

To run as admin, set RP_ADMIN to 1.

Abbreviation : [Run](#)

See also : [ExecuteFile](#) 

Example

To open Notepad :

```
Run Program>notepad.exe
```

A path may be specified if necessary :

```
Run Program>c:\my programs\eudora\eudora.exe
```

To start notepad minimized :

```
Let>RP_WINDOWMODE=2
Run Program>notepad.exe
```

4.19 Script Control

4.19.1 Sub Routines

4.19.1.1 Gosub

Gosub>Subroutine_Name[,value1[,value2[,...]]]

Branches to the specified subroutine (SRT). When that the subroutine returns, processing continues at the next line after the Gosub command.

Subroutines can be nested. Subroutines can be located anywhere within the script.

If values are passed on the Gosub line variables will be created to contain these values. The variables are named with the following format: SubroutineName_Var_1, Subroutine_Var_2, ... Subroutine_Var_n

See also [SRT](#)^[113], [END](#)^[114]

Example

```
Gosub>EnterName
Gosub>CloseApp

///// Subroutines Below /////

SRT>FocusApp
  SetFocus>Data Entry Screen
End>FocusApp

SRT>EnterName
  Gosub>FocusApp
  Send>Firstname
  Press Tab
  Send>Surname
  Press Enter
End>EnterName

SRT>CloseApp
  Gosub>FocusApp
  Press ALT
  Press F4
  Release ALT
End>CloseApp
```

4.19.1.2 SRT

SRT>Subroutine_Name

Identifies the start of a Subroutine block. Use [End](#)^[114] to end the subroutine.

Subroutines can be nested. Subroutines can be located anywhere within the script.

See also [Gosub](#)^[112], [END](#)^[114]

Example

```
Gosub>EnterName
Gosub>CloseApp

///// Subroutines Below /////

SRT>FocusApp
  SetFocus>Data Entry Screen
End>FocusApp

SRT>EnterName
  Gosub>FocusApp
  Send>Firstname
  Press Tab
  Send>Surname
  Press Enter
End>EnterName

SRT>CloseApp
```

```
Gosub>FocusApp
Press ALT
Press F4
Release ALT
End>CloseApp
```

4.19.1.3 END

END>Subroutine_Name

Identifies the end of a Subroutine block. Use [SRT](#)^[113] to start the subroutine.

Subroutines can be nested. Subroutines can be located anywhere within the script.

See also [Gosub](#)^[112], [SRT](#)^[113]

Example

```
Gosub>EnterName
Gosub>CloseApp

//// Subroutines Below ////

SRT>FocusApp
SetFocus>Data Entry Screen
End>FocusApp

SRT>EnterName
Gosub>FocusApp
Send>Firstname
Press Tab
Send>Surname
Press Enter
End>EnterName

SRT>CloseApp
Gosub>FocusApp
Press ALT
Press F4
Release ALT
End>CloseApp
```

4.19.2 Exit

Exit>return_code

Immediately terminates the current script. When used in a compiled macro you can set the executable's return code with return_code.

Example

```
Exit>0
```

4.19.3 Goto

Goto>Label_Name

Causes execution to continue at the specified label, missing any commands in between. If the label does not exist an error message will be displayed.

Label_Name can be or include a variable name.

Goto in conjunction with Label, can be used to create infinite loops. Use the If.. commands to cause conditional branching. To break out of infinite loops press Stop, or choose the Break option from the taskbar pop up menu.

See also : [Label](#) ¹¹⁶

Example

```
Label>Start
..
..
Goto>SecondBit
..
..
Label>SecondBit
..
```

4.19.4 If

```
If>expression[,true_label_name[,false_label_name]]
    statements
[ [Else
    else statements
Endif ]
```

Evaluates **expression**. If the expression is true the first statements are executed. If the expression is false the statements after Else are executed. Else is optional. IF blocks can be nested. When label names are provided execution of the script jumps to the specified labels or subroutine names. If label names are specified Else and Endif are ignored and are unnecessary.

The **expression** can be *simple (legacy)*, or *complex*.

Complex Expressions

The expression must be contained within curly braces "{" and "}". String literals must be delimited with double quotes ("), e.g.: "string". Variables must be delimited with % symbols, e.g.: %VarA%.

Several types of operators and functions can be used with complex expressions. For more details see [Complex Expressions](#) ⁴¹.

Simple Expressions

Simple expressions can contain only two parts separated by one of the following operators:

=	Equals
>	Greater than
<	Less than
<>	Not Equal

Values in the expression can be numeric or string values, or variables containing such values. Spaces must not be used as simple expressions do not require string delimiters.

See also : [Label](#)^[116], [Goto](#)^[115], [IfFileChanged](#)^[62], [IfFileExists](#)^[62], [IfWindowOpen](#)^[79], [Subroutines](#)^[113]

Example

```
Let>a=5
//simple expression only
IF>a=5
    //do something
ELSE
    //do something else
ENDIF

//complex expression:
IF>{(%a% = 5) AND (%VarA% = "allen")}
    //do something
ELSE
    //do something else
ENDIF

//using labels
Label>Start
..
..
..
If>a<b,Start
```

4.19.5 Include

Include>scriptfile

Includes code from an external script file. The code is executed and made available to the calling script. Therefore variables and subroutines in the external script file are made available to the calling script.

4.19.6 Label

Label>Label_Name

Marks a point in the script to allow execution to be passed to that point by the Goto, and If.. commands.

Goto in conjunction with Label, can be used to create infinite loops. Use the If.. commands to cause conditional branching. To break out of infinite loops press Stop, or choose the Break option from the taskbar pop up menu.

See also : [Goto](#)^[115], [If](#)^[77], [IfWindowOpen](#)^[79], [IfFileExists](#)^[62], [IfFileChanged](#)^[62]

Example

```
Label>Start
..
..
Goto>SecondBit
..
..
```

```
Label>SecondBit  
..
```

4.19.7 Let

Let>variable_name=value

Let is used to assign a value to a variable.

Where parameters are passed to commands, variables can also be passed. Variables can also be embedded within a parameter by enclosing the variable name within % symbols.

Let can also be used to perform basic calculations, and to concatenate strings. NB. If you want to assign to a variable a string that already contains a + sign, add an extra + sign to avoid the two strings being concatenated.

The **value** can contain a complex expression:

Complex Expressions

The expression must be contained within curly braces "{" and "}". String literals must be delimited with double quotes ("), e.g.: "string". Variables must be delimited with % symbols, e.g.: %VarA%.

Several types of operators and functions can be used with complex expressions. For more details see [Complex Expressions](#)^[41].

Examples

```
Let>name=freddy  
Let>a=5  
  
Let>path=c:\Program Files\  
Run Program>%path%myapp.exe  
  
Let>k=k+1  
  
Let>A=60-4  
  
Let>A=5*3  
  
Let>F=75/32  
  
Let>Name=John+ Smith
```

Name would now equal John Smith

```
Let>Name=John++Smith
```

Name would equal 'John+Smith'

```
Let>MyVal={Upper( "WoRLd" )}  
  
Let>MyVal={5 * 10 + 25}  
  
Let>d={%a%+%b%*%c%-23}
```

4.19.8 Macro

Macro>file_name [/variable=value|variable [/variable=value|variable] ...]

Executes another script file. file_name must be a filename of a macro file. It is advisable to specify the full path should the path of the script file differ from the current path or change during the execution of the calling macro.

By default the extension of macro files is .scp, so for macros that are listed in Macro Scheduler add .scp to the macroname and prefix with the correct path. The path of a macro listed in Macro Scheduler is defined under Group Properties for the group the macro is listed in.

To pass values to the macro specify each one after a / character. The variable name given should exist in the script to be run. The value to assign to that variable is specified after the = character.

Data can be returned to the calling macro by setting the MACRO_RESULT variable in the called script. After the script has been run MACRO_RESULT is available to the calling script.

If the calling macro has a log file the script being called with the Macro command will log to the same log file. You can override this using the /LOGFILE parameter to set a log file for the sub macro.

Abbreviation : [Mac](#)

Examples

```
Macro>Defragment Disk.scp
```

```
Macro>%SCRIPT_DIR%\MyMoveFile.scp /source=c:\temp\myfile.bat
/destination=c:\temp\myfile.bak
```

4.19.9 OnEvent

OnEvent>EventType,EventParm,ExtraParm,Subroutine

Establishes an event handler. When the event occurs the script branches to the specified Subroutine.

Event Types:

EventType	EventParm	ExtraParm	Description
WINDOW_OPEN	Window_Title	WF_TYPE	Triggers when specified window is open. See IfWindowOpen ^[136] for window title & WF_TYPE syntax.
WINDOW_NOTOPEN	Window_Title	WF_TYPE	Triggers when specified window is closed. See IfWindowOpen ^[136] for window title & WF_TYPE syntax.
WINDOW_NEWACTIVE		0	Triggers when the foreground window changes.
FILE_EXISTS	Filename	0	Triggers when the specified file exists.
FILE_NOTEXISTS	Filename	0	Triggers when the specified file does not exist.
KEY_DOWN	Key/KeyCode	Modifier	Triggers when the key is down. See WaitKeyDown ^[87]
DIALOG_EVENT	Dialog Name	Modal Result	Triggers when dialog's modal result matches e.g. when a button is pressed.
DIALOG_CHANGE	Dialog Name	Object Name	Triggers when object value is changed.

Explanations:

WINDOW_OPEN and WINDOW_NOTOPEN

EventParm takes a window title (or handle if WIN_USEHANDLE has already been set to 1).

ExtraParm takes a WF_TYPE value. See [SetFocus](#)^[138] and [IfWindowOpen](#)^[136] for explanation.

WINDOW_NEWACTIVE

EventParm and ExtraParm are not used. Set to zero.

FILE_EXISTS and FILE_NOTEXISTS

EventParm takes a filename.

ExtraParm is not used. Set to zero.

KEY_DOWN

EventParm takes a character for an ordinary key. For other keys use the virtual key code preceded by VK. See [WaitKeyDown](#)^[87] for more information.

ExtraParm is used to specify a modifier key:

- 0: No modifier key
- 1: SHIFT key must also be pressed
- 2: CONTROL key must also be pressed
- 3: ALT key must also be pressed
- 4: SHIFT + ALT keys must be pressed
- 5: CONTROL + ALT keys must be pressed
- 6: SHIFT + CONTROL keys must be pressed
- 7: CONTROL + ALT + SHIFT keys must also be pressed
- 8: Windows key must also be pressed

DIALOG_EVENT

EventParm is the dialog name. ExtraParm is a modal result value to detect. This could be the modal result of a button or menu option. So when that button is pressed the subroutine is triggered.

DIALOG_CHANGE

EventParm is the dialog name. ExtraParm is the object name. When the value of the object is changed by the user the subroutine is triggered. Valid object types include Edit, Memo, ComboBox, ListBox, RadioGroup and CheckBox.

Abbreviation: [ONE](#)

Example

```
OnEvent>WINDOW_OPEN,Notepad*,2,DoNotepad
OnEvent>WINDOW_NOTOPEN,Notepad*,2,DoNotepadNotOpen
OnEvent>WINDOW_NEWACTIVE,0,0,DoNewWindow
OnEvent>KEY_DOWN,VK32,3,KeyPress
```

```
Label>start
  Wait>1
  If>NotepadOpen=1
    Message>Notepad is open
  Else
    Message>Notepad is not open
  Endif
Goto>start

SRT>DoNotepad
  Let>NotepadOpen=1
END>DoNotepad

SRT>DoNewWindow
  GetActiveWindow>title,x,y
  MessageModal>New window: %title%
```

```

END>DoNewWindow

SRT>DoNotepadNotOpen
  Let>NotepadOpen=0
END>DoNotepadNotOpen

SRT>KeyPress
  MessageModal>ALT+Space was pressed
END>KeyPress

```

4.19.10 Remark

Remark>Some Comment

The remark statement is ignored by the interpreter. It exists simply to allow comments to be placed in the code. In fact, any text that is not a recognised command can be used for this purpose.

Several lines of code can also be commented out in one go by using `/*` before and `*/` after the lines to be commented. E.g.:

```

/*
Let>r=5
Repeat>r
  DDERequest>Excel,efile.xls,R%r%C2,Name,60
  Let>r=r+1
Until>r=5
*/

```

4.19.11 Repeat

Repeat>variable

Use in conjunction with Until. Iterates the code from the Repeat statement to the Until statement until the specified expression in the Until statement becomes true. Until can take one of the following simple expressions:

variable,value	for backward compatibility. Same as variable=value
variable=value	Until variable equals the value specified
variable>value	Until variable is greater than the value specified
variable<value	Until variable is less than the value specified
variable<>value	Until variable does not equal the value specified

Until will loop back to the last Repeat that has the same counter variable.

See also : [Until](#)^[12]

Example

```

GetFileList>c:\temp\*.*,files
Separate>files,;,file_names
MessageModal>Num Files: %file_names_count%

Let>k=0
Repeat>k
  Let>k=k+1
  Message>file_names_%k%
Until>k,file_names_count

```


4.19.12 Until

Until>variable,finalvalue

Use in conjunction with Repeat. Iterates the code from the Repeat statement to the Until statement until the specified expression in the Until statement becomes true. Until can take one of the following simple expressions:

variable,value	for backward compatibility. Same as variable=value
variable=value	Until variable equals the value specified
variable>value	Until variable is greater than the value specified
variable<value	Until variable is less than the value specified
variable<>value	Until variable does not equal the value specified

Until will loop back to the last Repeat that has the same counter variable.

See also : [Repeat](#)^[120]

Example

```
GetFileList>c:\temp\*.*,files
Separate>files,;,file_names
MessageModal>Num Files: %file_names_count%

Let>k=0
Repeat>k
    Let>k=k+1
    Message>file_names_%k%
Until>k,file_names_count
```

4.19.13 Wait

Wait>seconds_to_wait

This command makes Macro Scheduler pause for the specified number of seconds.

See also : [WaitWindowOpen](#)^[140], [WaitWindowClosed](#)^[139], [WaitPixelColor](#)^[101], [WaitCursorChanged](#)^[101]

Example

To wait 5 seconds :

```
Wait>5
```

This will work too :

```
Let>WaitTime=5
Wait>WaitTime
```

4.19.14 WaitReady

WaitReady>paint_events

WaitReady suspends script execution until the foreground window has finished processing mouse, keyboard, show window, and optionally, paint events. Issue 1 to include paint events, and 0 to exclude paint events. This command can therefore be used to wait until the active application is ready to receive keyboard and mouse events in most situations.

Abbreviation : [WRD](#)

See also : [Wait](#)^[121], [WaitWindowOpen](#)^[140], [WaitWindowClosed](#)^[139], [WaitCursorChanged](#)^[101], [WaitPixelColor](#)^[101]

Example

```
Run Program>"C:\Program Files\Microsoft Office\Office\WINWORD.EXE"
WaitWindowOpen>Microsoft Word*
WaitReady>0
Message>Word is now ready for input
```

4.20 String Handling

4.20.1 Base64

Base64>source,ENCODE|DECODE,target

Encodes or decodes source to target using Base64. Base64 encodes data using pure ASCII characters. It is therefore ideal for storing binary data in text files etc and is commonly used for sending binary attachments in email. Base64 works with ANSI strings only.

This function is most useful when used with the [Crypt](#)^[123] function. Crypt results in binary data, so Base64 can be used to encode that binary data to a plain text string which can then safely be stored in a text file or database.

Abbreviation: [Bas](#)

See also: [Crypt](#)^[123]

Example

```
Crypt>abc,hello,cryptval
Base64>cryptval,ENCODE,ascii_cryptval
WriteLn>d:\test.txt,r,ascii_cryptval

..

ReadLn>d:\test.txt,1,ascii_cryptval
Base64>ascii_cryptval,DECODE,bin_cryptval
Crypt>abc,bin_cryptval,clean
```

4.20.2 ConCat

ConCat>string1,string2

Concatenates string1 with string2. String1 must be a variable containing a string. String2 can be a literal string or a variable. The result is that string1 has string2 appended to it.

Abbreviation : [Con](#)

Example

```
Let>path=c:\temp\
ConCat>path,myfile.txt
```

In this example path becomes 'c:\temp\myfile.txt'

4.20.3 Crypt

Crypt>key,source,target

Encrypts or decrypts source to target using specified key. Encrypts a plain ANSI string, decrypts an encrypted string if the same key as used for the encryption is specified.

By default the encryption strength is set to 32 bit. This can be changed to medium level 64 bit or high level 96 bit using the CRYPT_LEVEL variable:

1 = Default 32 bit
2 = Medium 64 bit
3 = High 96 bit

Abbreviation: [Cry](#)

See also: [Base64](#)^[122]

Example

```
Crypt>abc,hello,cryptval
```

```
..
```

```
Crypt>abc,cryptval,clean
```

Crypt often produces binary output. Therefore, if writing encrypted strings to text files first encode the string with Base64. Decode the string when reading from the file before decrypting. E.g.:

```
//Encrypt the text
Crypt>abc,texttoencrypt,encdata

//Encode with Base64
Base64>encdata,ENCODE,encdata

//Write to the file
WriteLn>file,r,encdata

//To read the data back and decrypt:
ReadLn>file,1,encdata
//UnEscape the encoded data
Base64>encdata,DECODE,encdata
//Decrypt
Crypt>abc,encdata,decryptedtext
```

4.20.4 Length

Length>string,result

Returns the length of the given string

Abbreviation : [Len](#)

See also : [MidStr](#)^[124], [Position](#)^[124], [ConCat](#)^[122]

Example

```
Length>Hello World,Len
```

4.20.5 MidStr

MidStr>string,start,length,result

Returns a substring of specified length from a given position in a string. result is a variable in which to store the returned string. Any parameter can be a variable containing the appropriate values.

Abbreviation : [Mid](#)

See also : [Position](#)^[124], [ConCat](#)^[122], [Length](#)^[123]

Example

In the following example, the variable somevalue becomes equal to 'Happy' :

```
MidStr>Happy Birthday,1,5,somevalue
Message>somevalue
```

4.20.6 Position

Position>substring,string,start,result[,relative]

Returns the starting position of a substring in a string. The search commences at the position specified in start. If found the starting position of the substring is returned in the result variable. If no match is found this value will be zero.

If string is an empty string (length zero) or start is greater than the length of the string then the result will be zero.

The optional parameter 'relative' can be used to determine whether the position returned is relative to the start position or an absolute position in the string being searched. The default is TRUE (compatible with previous versions). Set to FALSE to return the absolute position.

Abbreviation : [Pos](#)

See also : [Length](#)^[123], [MidStr](#)^[124], [ConCat](#)^[122]

Example

In this example, StartPos will contain the value 4 :

```
Position>Smith,Mr Smith,1,StartPos
```

4.20.7 Separate

Separate>list,delimiter,returnvar

Separate takes a list, a delimiter and returns the elements of the list. The command returns a number of variables, one for each list element, each with the suffix "_n" where n is the index of the element in the list. The variable names are determined by the name given in returnvar. e.g. returnvar_1, returnvar_2, etc. Also returned is the number of elements in the list, in returnvar_count.

Abbreviation : [SEP](#)

Example

```
GetFileList>c:\temp\*.*,files
```

```

Separate>files,;,file_names
MessageModal>Num Files: %file_names_count%
If>file_names_count=0,end
Let>k=0
Repeat>k
    Let>k=k+1
    Message>file_names_%k%
Until>k,file_names_count
Label>end

```

4.20.8 StringReplace

StringReplace>sourcestring,find,replace,newstring

Creates a new string, newstring, by searching sourcestring for all occurrences of find, replacing them with replace.

Abbreviation: [RPL](#)

Examples

```

Let>string=Your name is "Fred"
StringReplace>string," ",",",vbEscapedString

StringReplace>Good evening,evening,morning,newgreeting

```

4.21 Text Capture

4.21.1 GetTextAtPoint

GetTextAtPoint>x,y,result_variable,char_pos

Retrieves the text at the specified screen position. Any text retrieved is returned in result_variable. char_pos returns the zero-based index in result_variable of the character at position x,y.

NB: This function requires Windows 2000, XP, Server 2003, or Vista. It has no effect in Windows 95/98/NT.

Abbreviation: [GTA](#)

See Also: [GetTextInRect](#)^[125], [GetWindowTextEx](#)^[126], [WaitScreenText](#)^[126]

Example:

```

GetTextAtPoint>200,400,strText,nChar
if>nChar>-1
    MidStr>strText,nChar,1,sChar
Endif
MessageModal>%strText% (%sChar%)

```

4.21.2 GetTextInRect

GetTextInRect>left,top,right,bottom,result_variable

Retrieves text found in the screen rectangle bound by coordinates left,top and right,bottom.

NB: This function requires Windows 2000, XP, Server 2003, or Vista. It has no effect in Windows 95/98/NT.

Abbreviation: **GTI**

See Also: [GetTextAtPoint](#)^[125], [GetWindowTextEx](#)^[126], [WaitScreenText](#)^[126]

Example:

```
GetTextInRect>0,0,200,400,strText
MessageModal>strText
```

4.21.3 GetWindowTextEx

GetWindowTextEx>window_title,text

Retrieves visible text in the specified window. Unlike GetWindowText which works by accessing each object's published text property GetWindowTextEx uses lower level hooks to detect all text written to the screen and therefore is capable of capturing far more text.

The specified window is momentarily focused when the function is called.

NB: This function requires Windows 2000, XP, Server 2003, or Vista. It has no effect in Windows 95/98/NT.

Abbreviation: **GWE**

See Also: [GetTextAtPoint](#)^[125], [GetTextInRect](#)^[125], [WaitScreenText](#)^[126]

```
GetWindowText>Document - WordPad,WordPadText
```

4.21.4 WaitScreenText

WaitScreenText>text

Waits for the specified text to appear on the screen.

The system variable WST_TIMEOUT can be used to set the number of seconds after which this command should timeout. If set to zero (the default) the timeout will not occur and the command will continue indefinitely. If the command completes within the timeout period the value WST_RESULT will be set to TRUE. Otherwise WST_RESULT will be false.

NB: This function requires Windows 2000, XP, Server 2003, or Vista. It has no effect in Windows 95/98/NT.

Abbreviation: **WST**

See Also: [GetTextAtPoint](#)^[125], [GetTextInRect](#)^[125], [GetWindowTextEx](#)^[126]

Example:

```
WaitScreenText>Done
MessageModal>Complete
```

4.21.5 GetControlText

GetControlText>WindowTitle,ClassName,Instance,Result

Returns the text of the control specified by ClassName and Instance on the window specified by WindowTitle.

Result contains the text of the control if found. If the window is not found Result contains ##NOSUCHWINDOW##. If the class and instance was not found ##NOSUCHOBJECT## is returned.

If WIN_USEHANDLE is set to 1 WindowTitle must be a window handle.

To determine class names of objects on windows use [View System Windows](#)^[24]. A window may contain several objects of the same class name. Instance is used to determine which instance of the class to use.

GetControlText retrieves the published text property of the specified object. Not all text that you see on the screen is retrievable in this way. Some text is painted via lower level routines and some text is graphical. Some objects, such as labels, are not windowed controls, and therefore text associated with them cannot be retrieved with GetControlText. Try the new GetTextAtPoint, GetTextInRect and GetWindowTextEx commands which use lower level hooks to trap more text.

Abbreviation: [GCT](#)

See Also : [SetControlText](#)^[100], [GetTextAtPoint](#)^[125], [GetTextInRect](#)^[125], [GetWindowTextEx](#)^[126], [WaitScreenText](#)^[126]

Example:

```
GetControlText>Notepad*,Edit,1,npEdit
```

npEdit will now contain the contents of the notepad edit window.

4.21.6 GetWindowText

GetWindowText>window_title,text

GetWindowText retrieves all the detectable text contained within the specified window. Specify the window using the exact window title. The text is retrieved as a list with each object's text on a new line.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

GetWindowText retrieves the published text property of the specified window/object. Not all text that you see on the screen is retrievable in this way. Some text is painted via lower level routines and some text is graphical. Some objects, such as labels, are not windowed controls, and therefore text associated with them cannot be retrieved with GetWindowText. Try the new GetTextAtPoint, GetTextInRect and [GetWindowTextEx](#)^[126] commands which use lower level hooks to trap more text.

Abbreviation : [GWT](#)

See Also: [GetWindowTextEx](#)^[126]

Example

```
GetWindowText>Document - WordPad,WordPadText
```

4.22 VBScript Commands

4.22.1 VBEND

VBEND

Marks the end of a block of VBScript code. Macro Scheduler reads from VBSTART to VBEND and stores the VBScript code between the two markers for later execution by the VBRun, or VBEval commands.

If you wish to evaluate VBScript expressions in your scripts you will need a VBSTART/VBEND block at the start of your script even if it contains no code.

See also : [VBEval](#)^[128], [VBRun](#)^[130], [VBSTART](#)^[129]

Links to VBScript Documentation: <http://www.mjt.net.com/resources.htm>

Example

VBSTART

```
Function MultiplyNums (d,a)
    MultiplyNums = d * a
End Function

Sub DisplayMessage (msg)
    MsgBox msg
End Sub

Function GetName
    GetName = InputBox("Enter Your Name : ")
End Function
```

VBEND

```
Let>a=5
VBEval>MultiplyNums(%a%,2),answer
MessageModal>answer

VBRun>DisplayMessage,Hello World

VBEval>GetName,name
MessageModal>Your Name is : %name%
```

4.22.2 VBEval

VBEval>Function([parms]),result

Evaluates a VBScript expression, or function in the proceeding VBScript code block. Parameters can be passed to the function. The result of the function or expression is stored in the result variable which is a regular Macro Scheduler variable.

This command sometimes causes confusion. As it is evaluating a VBScript expression the syntax used in the first part of the command - the VBScript expression - should be valid VBScript syntax. To pass Macro Scheduler variables, embed them with the % symbol. If passing a Macro

Scheduler variable as a string, remember that VBScript expects strings with quote marks around them. See examples below.

It is possible to set the VBScript code timeout using the VBS_TIMEOUT variable. By default VBScript code will never timeout. To set a timeout, set VBS_TIMEOUT to a value in milliseconds.

Abbreviation : [VBE](#)

See also : [VBRUN](#)^[130]

Links to VBScript Documentation: <http://www.mjtnet.com/resources.htm>

Example

VBSTART

```
Function MultiplyNums (d,a)

    MultiplyNums = d * a

End Function

Sub DisplayMessage (msg)

    MsgBox msg

End Sub

Function GetName

    GetName = InputBox("Enter Your Name : ")

End Function

Function PointlessStringExample(somestring)

    DisplayMessage(somestring)
    StringExample = somestring

End Function
```

VBEND

```
Let>a=5
VBEval>MultiplyNums(%a%,2),answer
MessageModal>answer

VBRUN>DisplayMessage,Hello World

VBEval>GetName,name
MessageModal>Your Name is : %name%

Let>text=Hello World
VBEval>PointlessStringExample("%text%"),sametext

//Evaluate built in VBScript code,
//no code in VBSTART/VBEND block required
VBEval>Timer,elapsed
VBEval>MsgBox(%elapsed%),nul
```

4.22.3 VBSTART

VBSTART

Marks the start of a block of VBScript code. Macro Scheduler reads from VBSTART to VBEND and stores the VBScript code between the two markers for later execution by the VBRUN, or VBEval commands.

If you wish to evaluate VBScript expressions in your scripts you will need a VBSTART/VBEND block at the start of your script even if it contains no code.

See also : [VB Eval](#)^[128], [VBRUN](#)^[130], [VBEND](#)^[128]

Links to VBScript Documentation: <http://www.mjt.net.com/resources.htm>

Example

VBSTART

```
Function MultiplyNums (d,a)

    MultiplyNums = d * a

End Function

Sub DisplayMessage (msg)

    MsgBox msg

End Sub

Function GetName

    GetName = InputBox("Enter Your Name : ")

End Function
```

VBEND

```
Let>a=5
VBEval>MultiplyNums(%a%,2),answer
MessageModal>answer

VBRUN>DisplayMessage,Hello World

VBEval>GetName,name
MessageModal>Your Name is : %name%
```

4.22.4 VBRUN

VBRUN>Subroutine[,Parm1,Parm2,...]

Executes the specified VBScript subroutine in the proceeding VBScript code block. Parameters can be passed to the subroutine. Macro Scheduler allows up to 25 parameters to be passed. The number of parameters passed must equal the number expected by the subroutine, otherwise a run time error will occur.

Macro Scheduler variables may be used in any part of the command.

It is possible to set the VBScript code timeout using the VBS_TIMEOUT variable. By default VBScript code will never timeout. To set a timeout, set VBS_TIMEOUT to a value in milliseconds.

Abbreviation : [VBR](#)

See also : [VB Eval](#)^[128]

Links to VBScript Documentation: <http://www.mjt.net.com/resources.htm>

Example

VBSTART

```
Function MultiplyNums (d,a)
```

```
        MultiplyNums = d * a

End Function

Sub DisplayMessage (msg)

    MsgBox msg

End Sub

Function GetName

    GetName = InputBox("Enter Your Name : ")

End Function

VBEND

Let>a=5
VBEval>MultiplyNums(%a%,2),answer
MessageModal>answer

VBRun>DisplayMessage,Hello World

VBEval>GetName,name
MessageModal>Your Name is : %name%
```

4.23 Window Functions

4.23.1 CloseWindow

CloseWindow>window_title

Attempts to close the specified window. The window_title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

Please note: this command simply sends the internal Windows WM_CLOSE message to the target window. This is the internal instruction to tell the window to close. However, applications all interpret this instruction differently and there is no guarantee that it will cause the window to close. If CloseWindow fails to close the window we recommend simulating user input to close it instead (e.g. by sending ALT-F4 or equivalent).

If the specified window is the main window of an application, then that application will begin to close down. Any processing that the application does on exit will be carried out as usual.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to close the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

```
Let>WF_TYPE=0 - No Child Windows
Let>WF_TYPE=1 - ALL Windows (Default)
Let>WF_TYPE=2 - Visible Windows Only
Let>WF_TYPE=3 - Child Windows Only
```

Abbreviation : [Clo](#)

See also : [GetWindowPos](#)^[134], [GetActiveWindow](#)^[132]

Example

```
CloseWindow>notepad*
```

4.23.2 FindWindowWithText

FindWindowWithText>text_to_find,setfocus_flag,result_variable

This function attempts to locate a window containing somewhere within it the text specified in text_to_find. setfocus_flag can be set to 1 or 0. If set to 1, the function will activate the window it finds containing the specified text. If a window is found, result_variable will contain the found window's title text. If no window is found it will contain "NOT FOUND".

If the WIN_USEHANDLE variable is set to 1 result_variable will contain the window's handle.

NB. Not all text can be detected successfully. Some applications, such as Word Processors, in order to provide their more complex functionality have to display text in the form of graphics, so do not manifest textual data to other applications. FindWindowWithText uses the same routines as the [View System Windows](#)^[24] tool. So to see what text can be detected at any particular time use the [View System Windows](#)^[24] tool.

This function is useful when an application has two windows with the same name, and allows the correct one to be located and focused.

Abbreviation : [Fin](#)

Example

```
FindWindowWithText>Continue,1,windowname
```

4.23.3 GetActiveWindow

GetActiveWindow>window_title,X,Y[,Width,Height]

Retrieves information about the current active window. The window title, and top left coordinates of the active window are stored in window_title, X and Y.

If the WIN_USEHANDLE variable is set to 1 window_title will contain the window's handle.

The optional parameters Width and Height will return the width and height of the window respectively.

If GAW_TYPE is set to 1 (default is 0) this command will retrieve information about the active **child** window of the active foreground window. If there are no child windows window_title will be set to an empty string. For default behaviour set GAW_TYPE back to 0.

Abbreviation : [GAW](#)

See also : [GetWindowPos](#)^[134], [CloseWindow](#)^[131], [GetWindowHandle](#)^[133]

4.23.4 GetWindowHandle

GetWindowHandle>window_title,handle

Returns the window handle of the window specified by its window title.

Window handles are assigned when a window is created by the system so they change from session to session. Using window handles is more reliable than using window titles which are not unique and may require substring matching. You can use GetWindowHandle to retrieve the window handle and then refer to it subsequently in other window commands when you need to access that window by setting WIN_USEHANDLE to 1. See also [GetActiveWindow](#)^[132] which can return the handle of the currently active window.

The window_title may contain the * symbol at the end to indicate a wildcard.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to locate the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and stops at the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation: **GWH**

See Also: [GetActiveWindow](#)^[132]

Example

```
GetWindowHandle>Notepad*,notepad_hwnd
Let>WIN_USEHANDLE=1
SetFocus>notepad_hwnd
```

4.23.5 GetWindowList

GetWindowList>window_list

Returns a list of open top-level windows. The list is delimited by carriage-return-line-feed pairs (CRLF).

Abbreviation: **GWL**

Examples:

```
//Loop through a list of windows
GetWindowList>winlist
Separate>winlist,CRLF,windows
Let>k=1
Repeat>k
  Let>this=windows_%k%
  MessageModal>this
  Let>k=k+1
Until>k>windows_count

//Populate a dialog combo box with a window list
GetWindowList>Dialog1.msComboBox1.Items.Text
ResetDialogAction>Dialog1
```

4.23.6 GetWindowPos

GetWindowPos>window_title,X,Y

Locates the window specified in window_title and retrieves its upper left screen coordinates. X and Y are variables in which to store the coordinates. window_title may be a variable or literal.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

The window_title may contain the * symbol at the end to indicate a wildcard.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to setfocus to the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and sets focus to the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation : **GWP**

See also : [GetActiveWindow](#)^[132], [CloseWindow](#)^[131], [GetWindowSize](#)^[135]

Example

```
GetWindowPos>My Computer,X,Y
```

The following example achieves the same result as the MouseMoveRel command, moving to the point 10,10 relative to Notepad :

```
GetWindowPos>notepad*,npX,npY
Add>npX,10
Add>npY,10
MouseMove>npX,npY
```

4.23.7 GetWindowProcess

GetWindowProcess>window_title,process_id,process_name

Retrieves the process ID and process name (filename) of the process that the specified window belongs to.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

The window_title may contain the * symbol at the end to indicate a wildcard.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to setfocus to the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and sets focus to the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation: **GPW**

Example

```
GPW>Notepad*,pid,name  
MessageModal>%pid% %name%
```

4.23.8 GetWindowSize

GetWindowSize>window_title,Width,Height

Locates the window specified in window_title and retrieves its width and height dimensions.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

The window_title may contain the * symbol at the end to indicate a wildcard.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to setfocus to the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and sets focus to the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

Abbreviation : [GWS](#)

See also : [GetActiveWindow](#)^[132], [GetWindowPos](#)^[134]

Example

```
GetWindowSize>My Computer,nWidth,nHeight  
MessageModal>Dimensions: %nWidth%,%nHeight%
```

4.23.9 GetWindowText

GetWindowText>window_title,text

GetWindowText retrieves all the detectable text contained within the specified window. Specify the window using the exact window title. The text is retrieved as a list with each object's text on a new line.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

GetWindowText retrieves the published text property of the specified window/object. Not all text that you see on the screen is retrievable in this way. Some text is painted via lower level routines and some text is graphical. Some objects, such as labels, are not windowed controls, and therefore text associated with them cannot be retrieved with GetWindowText. Try the new GetTextAtPoint, GetTextInRect and GetWindowTextEx commands which use lower level hooks to trap more text.

Abbreviation : [GWT](#)

See Also: [GetWindowTextEx](#)^[126]

Example

```
GetWindowText>Document - WordPad,WordPadText
```

4.23.10 IfWindowOpen

```

IfWindowOpen>window_title[,label_name[,false_label_name]]
    statements
[ [Else
    else statements]
Endif ]

```

Checks to see if the specified window is open. If so the first statements are executed. Otherwise the else statements are executed. Optionally, instead of Else and Endif a label or subroutine name can be specified.

When label names are specified the command causes the script to continue from the specified label without running any other lines of code in between. If a second false label is specified, execution will jump to that label if the expression resolves false. Subroutine names can also be specified. When specifying a subroutine name execution will jump to that subroutine and return to the line after the If statement when the subroutine has completed.

The window_title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will first attempt to find the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

It is possible to limit the type of windows this command effects using the WF_TYPE variable:

```

Let>WF_TYPE=0 - No Child Windows
Let>WF_TYPE=1 - ALL Windows (Default)
Let>WF_TYPE=2 - Visible Windows Only

```

Abbreviation : **IfW**

See also : [Label](#)^[116], [Goto](#)^[115], [IfFileExists](#)^[62], [IfFileChanged](#)^[62], [If](#)^[77], [Subroutines](#)^[113]

Example

```

IfWindowOpen>Notepad - [Untitled],donotepad
..
..
Label>donotepad

or, with a wildcard :

IfWindowOpen>notepad*,donotepad
..
..
Label>donotepad

```

4.23.11 MoveWindow

```

MoveWindow>window_title,new_X,new_Y

```

Moves the window with the specified window title to the new coordinates. new_X,new_Y denotes the new upper left corner position. The window_title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to move the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

Let>WF_TYPE=0 - No Child Windows
Let>WF_TYPE=1 - ALL Windows (Default)
Let>WF_TYPE=2 - Visible Windows Only

Abbreviation : [MVW](#)

See also : [ResizeWindow](#) ¹³⁷

Example

```
MoveWindow>notepad*,5,5
```

4.23.12 ResizeWindow

ResizeWindow>window_title,new_width,new_height

Resizes the window with the specified window with the new width and height dimensions. The window_title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to resize the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

Let>WF_TYPE=0 - No Child Windows
Let>WF_TYPE=1 - ALL Windows (Default)
Let>WF_TYPE=2 - Visible Windows Only

Abbreviation : [RSW](#)

See also : [MoveWindow](#) ¹³⁶

Example

```
ResizeWindow>notepad*,500,300
```

4.23.13 SetFocus

SetFocus>window_title

Sets focus to the specified window. The window_title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to setfocus to the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and sets focus to the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

To set focus to child windows, such as MDI children, first use SetFocus to focus the application, and then issue a second SetFocus for the child window. In most cases it is necessary to add the * symbol for child windows.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

Let>WF_TYPE=0 - No Child Windows
Let>WF_TYPE=1 - ALL Windows (Default)
Let>WF_TYPE=2 - Visible Windows Only
Let>WF_TYPE=3 - Child Windows Only

Abbreviation : [Set](#)

Example

```
SetFocus>notepad*
```

Child Window Example:

```
SetFocus>Opera*  
SetFocus>google*
```

4.23.14 ShutDownWindows

ShutDownWindows>shutdown_type

Use this command to shutdown or reboot the machine. Set shutdown_type appropriately as follows:

0: Shutdown
1: Reboot
2: Logoff
3: Forced Shutdown
4: Forced Reboot
5: Forced Logoff

The last three options should be used with care as they will force running processes to terminate and this can cause applications to lose data as they are not being allowed to close down gracefully.

Abbreviation : [SDW](#)

Example

```
//Reboot Server  
ShutDownWindows>1
```

4.23.15 WaitWindowChanged

WaitWindowChanged>timeout

This command causes Macro Scheduler to wait until the foreground window changes. I.e. the foreground window's title (caption) changes, or a different window becomes the foreground window. If it doesn't change within the number of seconds specified in Timeout, the command stops waiting and the variable WWC_RESULT is set to FALSE. WWC_RESULT is TRUE if the command terminated because the foreground window changed within the specified time. If Timeout is set to 0, the command will wait indefinitely.

Abbreviation : **WWX**

See also : [Wait](#)^[121], [WaitWindowOpen](#)^[140], [WaitWindowClosed](#)^[139]

4.23.16 WaitWindowClosed

WaitWindowClosed>window_title

Waits for a specified window to close. Execution of the script will not continue until the window with the specified title text is no longer present or a specified timeout value is exceeded. The window title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will first attempt to find the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

The system variable WW_TIMEOUT can be used to set the number of seconds after which this command should timeout. If set to zero (the default) the timeout will not occur and the command will continue indefinitely. If WW_TIMEOUT is used, WW_RESULT will indicate whether or not the command ended successfully. If it timed out WW_RESULT will be set to FALSE. If the window it was waiting for appeared within the timeout setting, the WW_RESULT value will be set to TRUE.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

```
Let>WF_TYPE=0 - No Child Windows  
Let>WF_TYPE=1 - ALL Windows (Default)  
Let>WF_TYPE=2 - Visible Windows Only  
Let>WF_TYPE=3 - Child Windows Only
```

Abbreviation : **WWC**

See also : [Wait](#)^[121], [WaitWindowOpen](#)^[140], [WaitWindowChanged](#)^[139]

Examples

```
WaitWindowClosed>Progress

Let>WW_TIMEOUT=10
WaitWindowClosed>Progress
If>WW_RESULT=FALSE,Endit
..
..
Label>Endit
```

4.23.17 WaitWindowOpen

WaitWindowOpen>window_title

Waits for a specified window to open/appear. Execution of the script will not continue until a window with the specified title text appears or a specified timeout value is exceeded. The window title may contain the * symbol at the end to indicate a wildcard.

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will first attempt to find the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

The system variable WW_TIMEOUT can be used to set the number of seconds after which this command should timeout. If set to zero (the default) the timeout will not occur and the command will continue indefinitely. If WW_TIMEOUT is used, WW_RESULT will indicate whether or not the command ended successfully. If it timed out WW_RESULT will be set to FALSE. If the window it was waiting for appeared within the timeout setting, the WW_RESULT value will be set to TRUE.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

```
Let>WF_TYPE=0 - No Child Windows
Let>WF_TYPE=1 - ALL Windows (Default)
Let>WF_TYPE=2 - Visible Windows Only
Let>WF_TYPE=3 - Child Windows Only
```

Abbreviation : [WWO](#)

See also : [Wait](#)^[121], [WaitWindowClosed](#)^[139], [WaitWindowChanged](#)^[139]

Examples

```
Run Program>c:\program files\msoffice\winword.exe
WaitWindowOpen>microsoft word*

Let>WW_TIMEOUT=30
WaitWindowOpen>microsoft word*
If>WW_RESULT=FALSE,Endit
..
..
Label>Endit
Message>Error starting Word!
```

4.23.18 WindowAction

WindowAction>Action,window_title

Use this function to restore, minimize, maximize or close a window.

Action can be any one of the following:

- 0: Restore the window
- 1: Maximize the window
- 2: Minimize the window
- 3: Close the window

If the WIN_USEHANDLE variable is set to 1 window_title must be a window handle.

Specify the window name in window_title. The window_title may contain the * symbol at the end to indicate a wildcard.

If the last character of the window title specified is an asterisk (*), Macro Scheduler will attempt to select the first window whose title matches the text entered exactly. If it cannot make an exact match it then looks at all windows and selects the first one it finds whose title contains the entered text. This solves the problem with applications such as Word or Netscape which change their titles depending on the document loaded. It is best to try to provide an exact (including case) window title to ensure the correct window is found, as many applications have multiple invisible windows with similar names. Specifying text without a trailing asterisk will force Macro Scheduler to only look for an exact match.

It is possible to limit the type of windows this command affects using the WF_TYPE variable:

- Let>WF_TYPE=0 - No Child Windows
- Let>WF_TYPE=1 - ALL Windows (Default)
- Let>WF_TYPE=2 - Visible Windows Only
- Let>WF_TYPE=3 - Child Windows Only

Abbreviation : WIN

See also : [MoveWindow](#)^[136], [ResizeWindow](#)^[137], [CloseWindow](#)^[131]

Example

```
WindowAction>2,notepad*
```

4.24 Using Variables

Contents:

- [User Defined Variables](#)^[141]
- [Arrays](#)^[142]
- [Explicit Variable Resolution - VAREXPPLICIT](#)^[143]
- [Ignoring Spaces - IGNORESPACES](#)^[143]
- [System Variables](#)^[144]
- [Variable Explorer](#)^[145]
- [Debugger - Watch List](#)^[14]

4.24.1 User Defined Variables

As well as system variables, which are pre-set, variables can be assigned by script commands which return variables, and by the Let command which allows you to create variables at any time. All variables have global scope (are available to the entire script once created).

The Let command is used to assign a value to a variable like so :

```
Let>Name=Freddy Mercury
```

The first time a variable is assigned a value, that variable is created. Subsequently, it's value is modified.

Variables can then be passed into other functions as one of the parameters. For example, assuming the above Let command has taken place, the following command will display a message box containing the words 'Freddy Mercury' :

```
Message>Name
```

If Name did not exist as a variable, the message box would simply display the word 'Name'.

Commands always check to see if a parameter passed to it is a variable. If a variable with that name exists, the value assigned to that variable is used, otherwise just that literal value is used.

All commands which accept a value can also accept a variable in place of that value.

For example, the following command would send the text 'Freddy Mercury' to the current window :

```
Send Character/Text>Name
```

In some cases a variable must be embedded within a string. This can be achieved using the % operator. In the following example a message is displayed saying 'Hello Freddy Mercury' :

```
Message>Hello %Name%
```

The system variable CRLF can be used to force a new line in a message box. Therefore, a list can be built up like so :

```
Message>1 : first line %CRLF%2 : second line %CRLF%3 : third line
```

It is possible to ask the user for a value, using the Input box :

```
Input>path,Please enter directory to create :
```

This would prompt the user to enter a directory name to create, which would then be assigned to the variable path. We could then use it in the CreateDir command as follows :

```
CreateDir>path
```

The If command can be used to check the value of a variable :

```
If>Age>15,adult  
..  
..  
Label>adult
```

4.24.2 Arrays

Array examples:

```
Let>Names[1]=Fred  
Let>Names[2]=Sally  
Let>Names[3]=Geoff  
  
Let>k=2  
Message>Names[%k%]  
Let>Names[%k%]=Louise
```

Using VBScript Arrays:

Use the VBEval command to get the value of a VBScript array variable into a MacroScript variable:

```
VBEval>MyArray(2,3),theValue
```

4.24.3 Explicit Variable Resolution - VAREXPPLICIT

You can tell Macro Scheduler only to resolve variables that are contained within % symbols and to leave anything else as literals by setting the VAREXPPLICIT variable to 1:

```
Let>VAREXPPLICIT=1
```

The default is 0 and will cause default behaviour as described above - i.e. Macro Scheduler will look at any parameter to see if it is a variable. If a variable with that name exists, the value assigned to that variable is used, otherwise just that literal value is used. Setting VAREXPPLICIT to 1 stops this behaviour and tells Macro Scheduler only to resolve a variable if it is contained within % symbols. E.g.

```
Let>VAREXPPLICIT=1
Let>Name=Fred
Send>Name
Send>%Name%
```

In this case the first Send> command will send the literal 'Name' and the second would send the value 'Fred'. In the following example, which is the default behaviour, both Send> commands would send the value 'Fred':

```
Let>VAREXPPLICIT=0
Let>Name=Fred
Send>Name
Send>%Name%
```

4.24.4 Ignoring Spaces - IGNORESPACES

By default, spaces are seen as regular characters and are included in variable assignments. E.g. The following line..

```
Let>a = 5
```

.. would create a variable called "a " with the value " 5".

Normally, therefore, you should use:

```
Let>a=5
```

But experienced programmers are used to the language ignoring spaces. This can be achieved by setting IGNORESPACES to 1:

```
Let>IGNORESPACES=1
Let>a = 5
```

This would set variable "a" to the value 5.

If spaces are ignored then strings that need preceding spaces will need to be delimited. To do this use {"string"}

Try stepping through the example below with the Debugger and Watch List open.

Example

```

Let>VAREXPPLICIT=1
Let>IGNORESPACES=1

Let>a = 5
Let>b = 22

Input>c, Enter a number:, 0
If> %c% > 0
    Let> d = {%a% + %b% * %c%}
Else
    Let> d = {%a% * %b%}
Endif
MessageModal> Answer: %d%

//With IGNORESPACES=1 do the following if you want leading/trailing spaces in a
string
Let>string_with_spaces = {" I want preceding spaces "}
MessageModal> %string_with_spaces%

//Or just switch IGNORESPACES off and on
Let>IGNORESPACES=0
MessageModal> Need these leading spaces
Let>IGNORESPACES=1

//Inline Expressions:
MessageModal>{" I want preceding spaces"}

//Sum:
MessageModal>{5 + 4}

//Embed a complex expression within the string as you would a variable:
MessageModal>W Pos: {%Pos("W","Hello World")}%

```

4.24.5 System Variables

OS_VER	Operating System
WIN_DIR	Windows Directory Path
SYS_DIR	Windows System Directory Path
TEMP_DIR	Windows Temp Directory Path
DESKTOP_DIR	User's Desktop Path
USERDOCUMENTS_DIR	User's Documents Path

The above variables store their values in upper case.

USER_NAME	Current Username
COMPUTER_NAME	Computer Name
MSCHED_VER	Macro Scheduler version number
SCRIPT_DIR	Directory of running script
SCRIPT_FILE	Filename of running script
COMMAND_LINE	Full command line string (path of running executable and parameters passed on the command line)
MACRO_PARMS	Parameters passed by Macro> command
CWD	Current Directory
_LINE_NUM	Stores the current line number being executed
WW_TIMEOUT	Timeout value from WaitWindowOpen ^[140] / Closed ^[139]
WW_RESULT	Result from WaitWindowOpen ^[140] / Closed ^[139]
WCC_RESULT	Result from WaitCursorChanged ^[101]
WSI_TIMEOUT	Used to set a timeout value for WaitScreenImage ^[82]
WST_TIMEOUT	Used to set a timeout value for WaitScreenText ^[126]
DDE_TIMEOUT	Timeout value from DDERequest ^[51]
RP_WAIT	Switches waiting for program termination on and off in RunProgram ^[111]
RP_WINDOWMODE	Used to set window mode for RunProgram ^[111]
RP_RESULT	Used to store return code from RunProgram ^[111]
RP_ADMIN	Set to 1 to make RunProgram ^[111] run as admin
RP_DISPLAYERROR	Allows RunProgram ^[111] error messages to be turned off/on
MSG_STAYONTOP	Used to set message to stay on top in Message ^[88] / MessageModal ^[89]
MSG_CENTERED	Used to set message box to center

MSG_WIDTH	Used to set the width of the message box
MSG_HEIGHT	Used to set the height of the message box
MSG_XPOS	Used to set the X position of the message box
MSG_YPOS	Used to set the Y position of the message box
SK_DELAY	Millisecond delay to pause between sending characters in SendText ^[86]
SK_IGNORECAPS	Set to 1 for Send ^[86] to ignore caps lock and send characters as entered
SK_LEGACY	If Press/Release commands fail in Citrix/DOS, set this to 1 first.
PRESS_ALLOWVARS	Set to 1 to make Press/Release commands accept a variable for the key name
RND_SEED	Allows a seed to be set for the Random ^[98] command
CF_OVERWRITE	Allows changing CopyFile ^[58] to overwrite or rename on collision
FTP_STATUS	Used to switch off/on the FTP status window
FTP_PASSIVE	Used to toggle FTP passive mode
FTP_TIMEOUT	Timeout value for FTPGetFile ^[67] / FTPPutFile ^[69] / FTPGetDirList ^[66]
FTP_RESULT	Result of FTP commands
FTP_USETLS	Set type of TLS/SSL connection to use
SSL_CERT	Specify SSL certificate file
SSL_ROOT_CERT	Specify SSL root certificate file
SSL_KEY	Specify SSL key file
SENDMAIL_STATUS	Used to switch off/on the SMTPSendMail status window
SMTP_RESULT	Result of SMTPSendMail command.
SMTP_AUTH	Used to set SMTP authentication on or off
SMTP_USERID	Used for SMTP authentication
SMTP_PASSWORD	Used for SMTP authentication
SMTP_RECEIPT	Used to enable SMTP return receipt
SMTP_PORT	Used to optionally set the port of the SMTP server
SMTP_CCLIST	Set CC recipients for SMTPSendMail
SMTP_BCCLIST	Set BCC recipients for SMTPSendMail
SMTP_TIMEOUT	Timeout for SMTP connection (milliseconds)
POP3_STATUS	Used to switch off/on the RetrievePOP3 status window
POP3_PORT	Used to set the POP3 port (defaults to 110)
POP3_TIMEOUT	Used to set a POP3 timeout
POP3_DELETE	Used to tell RetrievePOP3 to delete messages from the server
POP3_MSGSIZELIMIT	POP3 message size limit - RetrievePOP3 ignores messages larger
POP3_RESULT	Stores the result of the RetrievePOP3 command
POP3_MSGFILES	A semicolon delimited list of files downloaded by RetrievePOP3
REG_INTASSTR	Used to set RegistryWriteKey to write integers as strings/integers
WF_TYPE	Used to set the type of window the windowing functions should affect
VBS_TIMEOUT	Used to set the timeout for VBScript code
GAW_TYPE	Used for GetActiveWindow to set for active, or active child window.
INPUT_PASSWORD	Used to mask data entry in Input ^[88] .
MF_RENAME	Set MoveFile ^[63] to rename instead of physically move files.
APP_TITLE	Sets the application title of compiled macros.
MACRO_RESULT	Used to set/retrieve the result of a called macro.
WIN_USEHANDLE	Set to 1 to set all window functions to use/return window handles instead of window titles. Set to 0 to set back to window titles.
HTTP_TIMEOUT	Optional timeout value for HTTPRequest ^[72]
HTTP_SSL	Specify whether to use SSL or not for HTTPRequest
STEP_DELAY	Sets a millisecond delay between each script line
CRYPT_LEVEL	Sets the encryption strength for Crypt ^[123] command
GFL_TYPE	Set to 1 to make GetFileList return directories only
FIP_SCANPIXELS	Used to alter the number of random pixels FindImagePos ^[80] scans
CR	Carriage Return
LF	Line Feed
CRLF	Carriage Return, Line Feed Combination (to force a new line)
TAB	Tab character
SPACE	Space character
_DUMP_VARS	Set to 1 to dump all variable names and values to log file.
_WRITE_LOG_FILE	Set to 0 to temporarily disable logging. Set to 1 to enable.
VAREXPPLICIT	Sets variable resolution method - see Using Variables ^[5] .
IGNORESPACES	Set script to ignore leading and trailing spaces. See Using Variables ^[5] .
IGNOREERRORS	Set to 1 to ignore script error messages. Does not apply to VBScript errors (use on error next etc for VBScript errors). Set back to 0 for default behaviour.

4.24.6 Variable Explorer

The variable explorer is activated from the Tools menu of the Script Editor, or by typing CTRL-ALT-V.

The variable explorer lists all the user defined variables created by the script and shows you where

they are created and modified. Double click on a variable name header to insert it into the script. Double click on a line item to locate the line in the editor.

Index

- % -

% 5

- _ -

_DUMP_VARS 144

- A -

About 26
Abs 42
Add 108
add strings 122
App 58
APP_TITLE 144
Append Files 58
AppendFile 58
application 131
ArcTan 42
Arithmetic Functions 42
Arithmetic Operators 41
Arrays 5, 124, 142
ASC 83
Ascii 83
Ask 87
ASS 89
Assigned 89

- B -

Base64 122
Binary Data 90
Binary File 90
BLO 90
Block Keyboard 90
Block Mouse 90
BlockInput 90
Bookmarks 6
Boxer 24
branch 115, 116
button 97

- C -

CAP 83
Caps Lock 83
CapsOff 83
CapsOn 83
Carriage Return 144
CBX 91
CDG 52
CF_OVERWRITE 144
CHA 58
changed 139
ChangeDirectory 58
checkbox 91, 99
click 103, 104, 106, 107
clipboard 44, 45
CLO 131
close 131
closed 139
CloseDialog 52
CloseWindow 131
Code Folding 6
Code Snippets 6
COF 83
Color 98, 101
Command Line 16
Command Locator 5
COMMAND_LINE 144

Commands 5
comment 120
CompareBitmaps 80
compile 23
Complex Expressions 41, 44
COMPUTER_NAME 144
CON 122
ConCat 122
concatinate 122
Conditional Functions 43
Connect Database 46
Contents 26
Control 91, 100, 127
COP 58
copy 43, 58
CopyFile 58
Cos 42
COU 59
count 59
CountFiles 59
CR 144
CRE 59
create 4
create directory 59
Create Exe 23
CreateDir 59
creating 4
CRLF 144
Cry 123
Crypt 123
CRYPT_LEVEL 144
cursor 101, 102, 104, 105
CWD 144

- D -

DAT 90
Database 45, 46, 47
date 48, 49, 50, 51, 60
DateStamp 90
Day 48
DayOfWeek 48
DBCclose 45
DBConnect 46
DBExec 46
DBQuery 47
DDE 51
DDE_TIMEDOUT 144
DDEPoke 51
DDERequest 51
Debugger 14
Decrypt 123
DEL 59
delay 121
delete 21, 59
Delete Folder 70
DeleteFile 59
Desktop 14
Desktop Shortcut 24
Details 22
Dialog 52, 56, 57
Dialogs 15, 52, 55, 56, 57, 87, 88
Directory 61, 65, 66, 71, 78
DLL 95, 97
DOW 48
DPK 51
DRQ 51

- E -

EDI 60
 edit 4
 EditIniFile 60
 Editor 24
 Email 73, 74
 Encode 122
 Encrypt 123
 Encryption 11
 End 112, 113, 114
 EndDialog 55
 Environment Variables 92, 100
 Escape 122
 Event Handler 118
 EXE 111
 execute 111
 ExecuteFile 111
 Executing 13
 exists 78, 79
 Exit 21, 114
 Exp 42
 explicit 5
 Explicit Variable Resolution 143
 ExportData 90

- F -

Favorites 5
 FDF 65
 FDT 60
 FGD 66
 FGF 67
 file 58, 59, 60, 61, 63, 78, 79
 File Event 10
 File I/O 63, 64, 65
 file transfer protocol 65, 66, 67, 69, 71
 FileDate 60
 files 59, 61
 FileSize 60
 FileTime 61
 FIN 132
 find 132
 FindImagePos 80
 FindWindowWithText 132
 FIP_SCANPIXELS 144
 focus 57, 138
 Folder Event 10
 Font 22, 56
 Forms 52, 55, 57
 FPF 69
 Frac 42
 FRF 71
 FSZ 60
 ftp 65, 66, 67, 69, 70, 71
 FTP_PASSIVE 144
 FTP_RESULT 144
 FTP_STATUS 144
 FTP_TIMEOUT 144
 FTPDelFile 65
 FTPGetDirList 66
 FTPGetFile 67
 FTPPutFile 69
 FTPRemoveDir 70
 FTPRenameFile 71
 Functions 112, 113, 114

- G -

GAW 132

GAW_TYPE 144
 GCP 102
 GCT 91, 127
 GDA 55
 GDT 49
 Get Text 125, 126
 get title 132
 GetActiveWindow 132
 GetCaretPos 102
 GetCheckBox 91
 GetClipboard 44
 GetControlText 91, 127
 GetCursorPos 102
 GetDate 49
 GetDialogAction 55
 GetEnvVar 92
 GetFileList 61
 GetListItem 92
 GetPixelColor 93
 GetRectChecksum 93
 GetScreenRes 81
 GetTextAtPoint 125
 GetTextInRect 125
 GetTime 49
 GetTreeNode 94
 GetWindowHandle 133
 GetWindowList 133
 GetWindowPos 134
 GetWindowProcess 134
 GetWindowSize 135
 GetWindowText 127, 135
 GetWindowTextEx 126
 GEV 92
 GFL 61
 GFL_TYPE 144
 Gosub 112, 113, 114
 Goto 115
 GPC 93
 GPW 134
 Graphics 80, 81, 82
 GRC 93
 Grid Lines 22
 group 12
 groups 12
 GTM 49
 GTP 102
 GWP 134
 GWT 127, 135

- H -

Handle 133
 Help 17
 Hide 21
 Hot Keys 12
 hour 50
 HTT 72
 HTTP 72
 HTTP_TIMEOUT 144
 HTTPRequest 72

- I -

Idle 121
 IF 77
 IFC 78
 IFD 78
 IfDirExists 78
 IFE 79
 IfFileChanged 78

IfFileExists 79
 IFW 79
 IfWindowOpen 79
 IGNOREERRORS 144
 IGNORESPACES 144
 Ignoring Spaces 143
 Image Recognition 80, 81, 82
 Include 116
 ini files 60, 63
 INP 88
 Input 88
 INPUT_PASSWORD 144
 Int 42

- K -

keystroke 84, 85, 86

- L -

Label 116
 Large Buttons 22
 LCL 103
 LClick 103
 LDB 103
 LDbClick 103
 LDO 103
 LDown 103
 LEN 123
 Length 43, 123
 Let 108
 LF 144
 LFR 95
 LIB 95, 97
 LibFree 95
 LibFunc 95
 LibFuncW 97
 LibLoad 97
 Line Feed 144
 List 22
 List Box 92
 Listing 65, 66, 71
 ListView 92
 LLD 97
 Ln 42
 lock 10
 Log on 10
 logging 11, 100
 Logical Operators 41
 loop 115, 116, 120, 121
 loops 115, 116, 120, 121
 Lower 43
 LUP 103

- M -

MAC 118
 Macro 118
 Macro Properties 20
 MACRO_PARMS 144
 MACRO_RESULT 144
 MCL 104
 MClick 104
 MDB 104
 MDbClick 104
 MDL 89
 MDO 104
 MDown 104
 Menu 2
 Menu Commands 2
 Menu Options 2

Menus 99
 Message 88
 message box 88, 89
 MessageModal 89
 MF_RENAME 144
 MID 124
 MidStr 124
 Min 50
 minutes 50
 MMR 105
 MNU 99
 Mon 50
 Month 50
 MOU 104
 mouse 97, 102, 103, 104, 105, 106, 107
 MouseMove 104
 MouseMoveRel 105
 MouseOver 105
 MOV 63
 MoveFile 63
 MoveWindow 136
 MSCHED_VER 144
 MSG 88
 MSG_CENTERED 144
 MSG_HEIGHT 144
 MSG_STAYONTOP 144
 MSG_WIDTH 144
 MSG_XPOS 144
 MSG_YPOS 144
 MUP 106
 MVR 105
 MVW 136

- N -

new 4
 New Macro 20
 NOF 83
 NON 84
 Num Lock 83, 84
 NumOff 83
 NumOn 84

- O -

OnEvent 118
 open 79, 111, 140
 Options 24
 Organising 5
 OS_VER 144

- P -

parameters 5
 pause 121
 Pi 42
 Pixel 101
 PLA 97
 Playing 13
 PlayWav 97
 POP3_DELETE 144
 POP3_MSGFILES 144
 POP3_MSGSIZELIMIT 144
 POP3_PORT 144
 POP3_RESULT 144
 POP3_STATUS 144
 POP3_TIMEOUT 144
 POS 43, 124
 Position 124
 Power 42
 Press 84

- Process ID 134
- Process Name 134
- PUS 97
- push 97
- PushButton 97
- PutClipboard 45
- Q -**
- quick launch 12
- R -**
- RAN 98
- Random 98
- RCL 106
- RClick 106
- RDA 56
- RDB 106
- RDbClick 106
- RDK 109
- RDO 107
- RDown 107
- RDV 110
- REA 63
- ReadFile 63
- ReadIniFile 63
- ReadLn 64
- reboot 138
- Record 26
- Recorder 12
- Recording 12
- REG_INTASSTR 144
- RegistryDelKey 109
- RegistryDelVal 110
- RegistryReadKey 110
- RegistryWriteKey 110
- Relational Operators 42
- relative 105
- Release 85
- Remark 120
- Rename 21, 64
- RenameFile 64
- Repeat 120, 121
- ResetDialogAction 56
- ResizeWindow 137
- RET 73
- RetrievePOP3 73
- RFL 63
- RGB 98
- RLN 64
- RND_SEED 144
- Round 42
- RP_ADMIN 144
- RP_DISPLAYERROR 144
- RP_RESULT 144
- RP_WAIT 144
- RP_WINDOWMODE 144
- RPL 125
- RRK 110
- RSW 137
- run 111
- Run Now 25
- Running 13
- RunProgram 111
- RUp 107
- RWK 110
- S -**
- SBX 99
- Scheduler 7, 8
- Scheduling 7, 8, 9
- SCP 82
- Screen Capture 82
- Screen Image 82
- Screen Resolution 81
- ScreenCapture 82
- screensaver 10
- script 4
- SCRIPT_DIR 144
- SCRIPT_FILE 144
- Scroll Lock 86
- ScrollOff 86
- ScrollOn 86
- SCT 100
- SDW 138
- Sec 50
- seconds 50
- SelectMenu 99
- SEN 86
- SENDMAIL_STATUS 144
- SendText 86
- SEP 124
- Separate 124
- Service 8
- SET 138
- SetCheckBox 99
- SetControlText 100
- SetDialogObjectFocus 57
- SetDialogObjectFont 56
- SetDialogObjectVisible 57
- SetEnvVar 100
- SetFocus 138
- SEV 100
- Shortcut 14
- Shortcuts 12
- Show 57
- shutdown 138
- ShutDownWindows 138
- Sin 42
- size 60
- SK_DELAY 144
- SK_IGNORECAPS 144
- SK_LEGACY 144
- SMT 74
- SMTP_AUTH 144
- SMTP_BCCLIST 144
- SMTP_CCLIST 144
- SMTP_PASSWORD 144
- SMTP_PORT 144
- SMTP_RECEIPT 144
- SMTP_RESULT 144
- SMTP_TIMEOUT 144
- SMTP_USERID 144
- SMTPSendMail 74
- SOF 86
- SON 86
- Sort 22
- sound 97
- SOWrite 45
- SOWriteLn 45
- SPACE 144
- SQL 46, 47
- Sqr 42
- SRT 112, 113, 114
- Startup 9

- STDOUT 45
- STEP_DELAY 144
- Stop 26
- string 123, 124
- String Functions 43
- String Operators 42
- StringReplace 125
- Strings 124
- Sub 109
- Subroutines 112, 113, 114
- substring 124
- subtract 109
- Support 17
- Switches 16
- SYS_DIR 144
- System Variables 144
- T -**
- TAB 144
- TBR 107
- Telnet 75, 76
- TelnetClose 75
- TelnetConnect 75
- TelnetSend 76
- TelnetWaitFor 76
- TEMP_DIR 144
- Terminate 114
- text 132
- Text Capture 125, 126
- Text Cursor 102
- Text Editor 24
- TIM 100
- time 49, 50, 61
- TimeStamp 100
- Toolbar 3, 23, 107
- TreeView 94
- Triggers 10
- Trunc 42
- U -**
- unlock 10
- Until 120, 121
- Upper 43
- User Defined Variables 141
- USER_NAME 144
- V -**
- VAREXPlicit 5, 144
- variable 108
- Variable Explorer 145
- Variables 5, 145
- VBE 128
- VBEND 128
- VBEval 128
- VBR 130
- VBRun 130
- VBS_TIMEOUT 144
- VBScript 128, 129, 130
- VBSTART 129
- View Log File 24
- View System Windows 24
- Visible 57
- W -**
- Wait 121
- wait for key press 87
- WaitClipboard 45
- WaitCursorChanged 101
- WaitKeyDown 87
- WaitPixelColor 101
- WaitReady 121
- WaitRectChanged 101
- WaitScreenImage 82
- WaitScreenText 126
- WaitWindowChanged 139
- WaitWindowClosed 139
- WaitWindowOpen 140
- wav 97
- WCC 101
- WCC_RESULT 144
- WEB 72
- WF_TYPE 144
- WIN 141
- WIN_DIR 144
- WIN_USEHANDLE 144
- window 79, 131, 132, 138, 139, 140
- Window Event 10
- Window List 133
- window position 134
- Window Size 135
- Window Text 127, 135
- WindowAction 141
- WKD 87
- WLN 65
- WPC 101
- WRC 101
- WRD 121
- WriteLn 65
- WSI_TIMEOUT 144
- WST_TIMEOUT 144
- WW_RESULT 144
- WW_TIMEOUT 144
- WWC 139
- WWO 140
- WWX 139
- Y -**
- Year 51