



Hawaii Green IT

The NightWatchman[®] Scripting Reference

A unique power management solution. Save energy, the environment and money

The NightWatchman® Scripting Reference

Version 5.6 document revision 1

© 1E Ltd 2009

All rights reserved. No part of this document or of the software ("the software") to which it relates shall be reproduced, adapted, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without permission from 1E Ltd. It is the responsibility of the user to use the software in accordance with this document and 1E Ltd shall not be responsible if the user fails to do so. Although every precaution has been taken in the preparation of this document, 1E Ltd and the authors assume no responsibility for errors or omissions, nor shall they be liable for damages resulting from any information in it.

Trade marks

1E is a trade mark of 1E Ltd registered in the UK[, EU and US], with registration applied for in Australia. The 1E device is a trade mark of 1E Ltd registered in the UK and EU, with registration applied for in the US and Australia. NIGHTWATCHMAN is a trade mark of 1E Ltd registered in the US, with registration applied for in the EU and Australia. MICROSOFT, WINDOWS NT, WINDOWS 2000, WINDOWS XP are all trademarks of Microsoft Corporation in the United States and other countries.

SHA-2 Algorithm

Copyright (c) 2002, Dr Brian Gladman, Worcester, UK. All rights reserved.

LICENSE TERMS

The free distribution and use of this software in both source and binary form is allowed (with or without changes) provided that:

1. distributions of this source code include the above copyright notice, this list of conditions and the following disclaimer;
2. distributions in binary form include the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other associated materials;
3. the copyright holder's name is not used to endorse products built using this software without specific written permission.

ALTERNATIVELY, provided that this notice is retained in full, this product may be distributed under the terms of the GNU General Public License (GPL), in which case the provisions of the GPL apply INSTEAD OF those given above.

DISCLAIMER

This software is provided 'as is' with no explicit or implied warranties in respect of its properties, including, but not limited to, correctness and/or fitness for purpose.

Contents

Section 1	Introduction	1
1.1	What will NightWatchman do for me?	1
1.2	What is new in NightWatchman 5.5?	1
1.3	Conventions used in this guide	1
	Cross references	2
	Notes	2
	Code fragments	2
	Command lines	2
1.4	Who is this guide for?	2
Section 2	Getting started on scripting	3
2.1	What do the NightWatchman shutdown scripts do?.....	3
2.2	What happens when the NightWatchman shutdown scripts run?.....	3
2.3	The NightWatchman shutdown script development process.....	3
	Setup the script	3
	Gather information for the Application	4
	A general script sequence	4
2.4	Things to keep in mind.....	4
	NightWatchman Script timeouts	4
	When multiple applications of the same type are running	4
	Be aware of application states.....	5
2.5	Running and testing a new NightWatchman shutdown script	5
	Test the script from the command line.....	5
	If the script does not work.....	5
	Re-run the script on a locked workstation	5
	Test the script in situ with a NightWatchman shutdown.....	5
Section 3	Writing NightWatchman shutdown scripts	6
3.1	Setting the script running order.....	6
3.2	Writing NightWatchman scripts.....	6
3.3	How the script phase works	6
3.4	A script helper	6
3.5	Testing scripts	7
3.6	An annotated script	7
	Error checking, common to all scripts.....	7
	The main script body	7
	Initialising variables.....	7
	Command line arguments.....	8
	Initialising the script helper object	8
	Changing the default NightWatchman backup directory	8
	Code to help with testing.....	8
	Getting the application process ID	8

	Closing active application dialogs	9
	Getting the currently open filename	9
	Attempting to close the application	9
	Does the open file contain unsaved data	9
	Handling the Save As dialog	9
	Creating a new file name	9
	Closing the application	9
	Tidying up the environment	10
Section 4 The NightWatchman Script Helper Reference		11
4.1	Method and property categories	11
	Application methods	11
	Window methods	11
	Save file environment methods	12
	File methods	12
	NightWatchman methods	12
	Script properties and methods	12
4.2	Method alphabetic reference	13
	ActivateApp	13
	BackupDirectory	13
	ClickButton	13
	ClickMouse	14
	CloseActiveAppDialogs	14
	CloseActiveAppDialogsLike	14
	CopyFile	15
	DeactivateApp	15
	ExpectCancel	15
	ExpectWindow	16
	FindInvisibleWindowLike	17
	FindWindowLike	17
	GetAccItem	18
	GetActiveObject	18
	GetCmdLine	19
	GetCurrDir	19
	GetEnvVar	20
	GetFullPath	20
	GetWindowText	20
	Log	21
	MakeNWMFilename	21
	MoveFile	22
	SelectAccItem	22
	SelectSubItem	22

SendKeys.....	23
SendKeysToWindow.....	23
SendKeysToDialog.....	24
SetActiveWindow	24
WaitForWindow	24
WaitWindowGone	25
WriteToBackupList.....	25
LogFileName.....	25
TimeoutSecs.....	25
debug.....	26
Section 5 Troubleshooting	27
5.1 Checking the problem	27
5.2 Contact 1E technical support.....	27
Creating a technical report.....	27
Section 6 Further Information	28
6.1 Contact details.....	28
Website	28
Telephone and fax.....	28
Post	28
Sales.....	28
Technical support.....	28

Section 1 Introduction

A unique power management solution. Save energy, the environment and money

NightWatchman is one of a range of management products available from 1E. It saves money and effort by coordinating power management into a company wide strategy.

The main purpose of NightWatchman is to shut down machines.

This guide will help you to extend the capabilities of NightWatchman to save information in applications that are running when a shutdown proceeds.

1.1 What will NightWatchman do for me?

NightWatchman saves you money by automatically shutting down machines when they are not in use. Shutdowns can proceed painlessly, ensuring that current work and data are kept safe.

NightWatchman also enhances the security of your network, reducing the danger of uncontrolled network wide access and the possibility of electrical failure and fire.

NightWatchman makes the lives of your IT staff easier, enabling the easy management of users and corresponding internal policies on PC power consumption from a central location.

NightWatchman in conjunction with SMSWakeUp, a machine power-up product from 1E, coordinates these capabilities by enabling the power state for machines to be fully controlled and scheduled. So on the one hand, financial directors and facilities management can control general PC power consumption with minimal disruption to staff. On the other hand systems administrators can ensure that PCs are on, and in a fresh "re-started" state, for scheduled out-of-office-hours software deployments.

1.2 What is new in NightWatchman 5.5?

NightWatchman has been enhanced to provide: sleepless client detection to help machines go into a sleep state, alarm clocks and maintenance windows to allow computers to be brought out of a sleep state and a new "keep active" end-user feature that allows the user to prevent shutdown for a selected time. These are described in more detail below:

- **Sleepless Client Detection** - NightWatchman can now detect why a particular computer is not going into a sleep state and enable you to bypass the cause to allow even more power savings. Certain application interactions with the Operating System have the unnecessary side effect of keeping computers from going into a sleep state, even though the user has not been active on the computer. NightWatchman can detect the processes that keep a computer awake and lets you bypass them to allow the computer to go into a sleep state.

For more information on this feature see *Section 5 - Sleepless Client Detection* in the *NightWatchman Administrator's Guide*.

- **Alarm Clocks and Maintenance Windows** - NightWatchman now lets you define maintenance windows whereby machines will be brought out of a sleep state for a configurable maintenance period and then returned to a sleep state afterwards. When configured from the NightWatchman Console and combined with 1E WakeUp you can even power up computers from an off state.

For more information on this feature see *Section 6 - Alarm Clocks and Maintenance Windows* in the *NightWatchman Administrator's Guide*.

- **Keep Active** - NightWatchman now lets end users inhibit passive or active NightWatchman shutdowns for a selectable period of time. The functionality empowers the end user to choose to opt out of scheduled events so that they can run a batch processing job overnight for example.

For more information on this feature see heading *8.4 - Keep Active* in the *NightWatchman Administrator's Guide*.

1.3 Conventions used in this guide

This section shows how to interpret the different styles used in this document to denote various types of information.

Cross references

Cross references are shown in italics. Cross references may be to diagrams or tables in the current document or to other documents. For example, the following paragraph references another document in the NightWatchman set:

Before trying out the NightWatchman scripting helper documented in this scripting guide, you will need to have installed NightWatchman. For details on installation please refer to *The NightWatchman Installation Guide*.

Notes

Notes are shown in white on an orange background. For example, the following note provides some useful information:

NOTE: always pay attention to notes.

Code fragments

This manual uses courier plus a shaded background to denote code fragments. For example, the following shows part of the standard header included in NightWatchman VB script files:

```
hwnd = 0                '** window handle **  
hwndHelper = 0  
debugx = 0  
Set objArgs = WScript.Arguments
```

Command lines

This manual uses courier plus a shaded background to denote command line entries – text to be entered is highlighted in bold. For example, the following shows the command to be executed to display the NightWatchman help text:

```
C:> nightwatchman -?
```

1.4 Who is this guide for?

This scripting guide is aimed at administrators who intend to extend the functionality of NightWatchman to close down applications other than the default supported set of Microsoft applications.

Section 2 Getting started on scripting

NightWatchman is designed to ensure that systems are shut down or logged off in a consistent manner enabling administrators to take over this responsibility from the user. This assists in desktop management, software delivery and energy saving.

Being able to shut down systems is only half the solution the other half requires a means of saving open, edited files on the machine being shut down. In NightWatchman this task is performed by the shutdown scripts.

2.1 What do the NightWatchman shutdown scripts do?

NightWatchman shutdown scripts are run by NightWatchman before initiating a shutdown. The purpose of the scripts is to protect unsaved data before shutdown can commence by interacting with the open application windows.

- A particular script is designed to shutdown a particular process
- The script for a process will run at least once for each running process
- A particular script may be run multiple times if the process has multiple open documents
- The scripts are designed to fail with an error if anything unexpected happens

An important point to remember is that NightWatchman scripts are designed to protect data. In the case of any unknown situations being encountered by the script you want the shutdown to abort in order to stop unsaved data being lost.

2.2 What happens when the NightWatchman shutdown scripts run?

When NightWatchman enters into a shutdown or log-off operation, in each Windows session it checks each top-most application window and the NightWatchman client performs the following sequence:

1. Find the process the window is running in
2. Find a NightWatchman script in the NightWatchman scripts directory with the same name as the process
3. Run the located script with *CScript.exe*. The *HWND* of each application window will be passed on the command line to the script.

Any given process may have multiple windows open representing different documents. The corresponding scripts will be run in turn for each window. For example, you may be editing a number of documents in Microsoft Word. In this case there is one process that represents Microsoft Word and one corresponding NightWatchman shutdown script. The script will be run multiple times, once for each open Microsoft Word document.

2.3 The NightWatchman shutdown script development process

There are many ways of developing NightWatchman shutdown scripts. Here we recommend one particular way that makes the process of developing a script for a particular application easier. A NightWatchman shutdown script can be written in either JScript or VBScript to handle the intelligent shutdown of applications, so that no data is lost when performing a user log-off.

Setup the script

1. Find the application's process name from Task Manager. For example the process name for Microsoft Word is *WINWORD*.
2. Create a new script in the NightWatchman scripts directory with the same root name as the process. By default the NightWatchman script directory is:

```
C:\Program Files\1E\Nightwatchman50\Scripts
```

So the script for a process called *ACMEAPP* would be saved in this directory with one of the following names:

```
ACMEAPP.vbs
```

or

```
ACMEAPP.js
```

Depending on what type of scripting language is being used.

3. If you are creating a `.vbs` script, copy the contents of the `ScriptTemplate.txt` file - located in the NightWatchman installation directory under the `Scripts` folder - and paste into the new file created in *step 2*. Fill in the placeholder items, for example do a global search and replace of `<Process Name>` with `ACMEAPP`.

Gather information for the Application

1. Use a Windows spy utility, such as Spy++ etc..., to gather more information on the application windows. These will be used in the script to access the interface for the application.
 - Determine the main application class type
 - Find the child dialog classes
 - Retrieve information about any message boxes that may require interaction when closing the application
2. Determine if there are any automation objects for the application
3. Check for embedded applications and other processes
 - Find any Windows that are in different processes
 - Determine any problems related to automation objects

A general script sequence

The following sequence is used by all the provided NightWatchman shutdown scripts, as such it is a tried and tested method for safely saving open files:

1. Find the application window
2. Close all child dialogs
3. Run expectors to help close dialogs

Then, either:

4. Save and close documents from automation objects

or:

5. Initiate a close
6. Use expectors to handle save messages
7. Save and close documents from the UI using `SendKeys`

Finally:

8. Wait and check that the application window closes

2.4 Things to keep in mind

The following points should be kept in mind when developing NightWatchman shutdown scripts:

NightWatchman Script timeouts

- Will the script you are writing be fast enough to complete before the NightWatchman script timeout expires?
- Should you extend the NightWatchman script timeout value?

When multiple applications of the same type are running

- All the child dialogs of the applications can be closed generically.
- Be careful that applications are not switched over during closure.

Be aware of application states

You should know that applications may be in different states when the attempt at closure occurs. You will need to cater for these in your scripts if you want to close successfully.

- Which windows can be open in the application?
- What messages are displayed in different circumstances during closure?
- Is there anything that could interrupt the sequence you are assuming in the shutdown script?

2.5 Running and testing a new NightWatchman shutdown script

The following points give indicators of how the newly developed NightWatchman script can be tested before being placed in a production environment.

Test the script from the command line

Before attempting to launch the script via a NightWatchman prompted shutdown you should test it in a stand-alone mode from a command line. To do this you will need to start up the target application with an open test document (nothing critical just in case the closure fails) and then run the shutdown script.

The following shows an example command line for the *acmeapp.vbs* script.

```
CScript acmeapp.vbs -DEBUG=7
```

Note the use of the *-DEBUG=7* switch. This prompts any script based on the *scripttemplat.txt* file to generate a debug log file, which can be useful if the script fails in some way to shut down the application.

If the script does not work

Check the log file for the script. By default this should have been created in the *C:\temp* directory with the name:

```
NWM<scriptname>.log file
```

where *<scriptname>* is the name of the script. So the log file for our example script would be called *NWMACMEAPP.log*.

Re-run the script on a locked workstation

After testing the script successfully from the command-line in a basic situation, you should then re-run the script on a locked workstation. This simplest way of doing this is to add the line:

```
wscript.sleep( 5000 )
```

at the start of your shutdown script. Then when running the script from the command line you will have a reasonable amount of time to lock the workstation by hand before the main body of the script commences.

Test the script in situ with a NightWatchman shutdown

The final test is to have the new shutdown script running alongside all the other NightWatchman shutdown scripts. To do this you should open and edit test documents (nothing critical, just in case the interaction fails between the scripts) in all the office applications, Microsoft Word, Excel etc..., and also have your target application open with an edited document.

You then need to trigger an immediate shutdown via NightWatchman. This will run all the NightWatchman shutdown scripts enabling you to monitor their interaction and check for any immediate errors. If the shutdown is successful, on restarting the machine you should then check that all the test documents have been saved correctly.

Section 3 Writing NightWatchman shutdown scripts

In the previous section we discussed the process you should go through to develop a NightWatchman shutdown script. This section describes the mechanics of NightWatchman scripts and how these can be put together to implement a new shutdown script.

The functionality of the NightWatchman service can be extended by creating additional shutdown scripts. Administrators can write windows scripts in either JScript or VBScript to handle the intelligent shutdown of applications, so that no data is lost when performing a user log-off.

During shutdown the scripts will attempt to save current versions of all open and edited files. Any open dialogs in the currently running applications are also handled in order to execute the closure of the application in a safe way. At next log-on, the user will be notified that backups have been made and given the option of accepting or rejecting these.

To run correctly, scripts must be saved in the NightWatchman scripts directory, immediately below the NightWatchman installation directory. By default this directory is:

```
C:\Program Files\1E\Nightwatchman50\Scripts
```

3.1 Setting the script running order

If all the scripts are present in the root of the NightWatchman script directory they will be run in an arbitrary order. You can control the order in which the scripts can be run, by creating specifically named subfolders in this directory and placing scripts in the subdirectories.

NightWatchman supports ten subdirectories named *Sub0* up to *Sub9*. The scripts in *Sub0* are run first then the ones in *Sub1* and so on up to *Sub9*. After the scripts in the subdirectories have been run, NightWatchman will then run any scripts in the main scripts directory.

3.2 Writing NightWatchman scripts

NightWatchman is provided with pre-defined scripts that handle the following products: Microsoft Outlook, Microsoft Word, Microsoft PowerPoint, Microsoft Excel and Notepad. To handle other applications you can either contact 1E or write your own additional scripts using the reference details provided in *Section 4 The NightWatchman Script Helper Reference on page 11*.

3.3 How the script phase works

The shutdown script phase of the NightWatchman system contains the following steps:

- NightWatchman compiles a list of all the applications running.
- For each application found, it searches for a script with the same base name as the application executable.
- NightWatchman then executes this script to handle the closing of the application.

For example, a user is logged on when NightWatchman initiates a shutdown, and that user has been editing a Microsoft Word document. NightWatchman first detects that Microsoft Word is running, the executable being *winword.exe*. It then searches the NightWatchman *scripts* directory for *winword.js*. If it fails to find this it will then search for *winword.vbs*. Assuming it finds this script it will then execute it to handle the closing of Microsoft Word.

NOTE: It is also possible to write scripts that are version specific. For example, *winword.900.53.9.0.vbs* .

3.4 A script helper

In order to perform certain operations and to simplify the creation of application shutdown scripts, the NightWatchman installation includes a script helper object called *NWMScriptHelper.ocx*. This is an ActiveX scripting object intended to be instantiated from within a Windows Scripting Host file. For detailed reference information on NightWatchman scripting see *Section 4 The NightWatchman Script Helper Reference on page 11*.

The helper object provides methods that can be called from within a script. These methods cover the following general areas:

- To remove the delays that are sometimes explicitly added to scripts when waiting to ensure a particular window appears. Scripts can now use the helper to wait for a window to appear or disappear, see *WaitForWindow* and *WaitWindowGone*.
- Scripts can start working on an application in an indeterminate state – the helper can "Cancel", "Close" or "OK" dialogs as appropriate, see *CloseActiveAppDialogs*.
- The scripts can be very much more window aware when they use the helper. The helper ensures that the process being controlled contains the active window; see *ActivateApp*, *GetAccItem*, *SelectAccItem* and *SelectSubItem*.
- The helper enables regular expressions to be used in shutdown scripts for matching window names - to allow more generalised matching with in the fields, see *FindWindowLike*.
- The helper allows actions to be performed in response to expected events without a lot of difficult scripting, see *ExpectWindow*.
- The helper supports the automatic creation of backup names, see *MakeNWMFilename*.
- Other functions have been added which may be part of the "WScript.Shell" object or the "FileScriptingObject". This will allow calls to be logged consistently.

3.5 Testing scripts

To make testing easier, it is possible to run the scripts from the command line. For example, the following command will run the Notepad script against the first instance of Notepad.exe found.

```
C:> cscript Notepad.vbs -debug
```

When the script is run by NightWatchman during shutdown, all instances of notepad.exe will be handled.

3.6 An annotated script

This section provides an annotated guide for a simple script to handle Notepad.exe shutdowns. The problems dealt with are general ones, though obviously the shutdown procedures for some applications may be far more complex than the ones used for notepad.

Error checking, common to all scripts.

All scripts should include an error checking section to catch errors and exit the script gracefully. As a convention this is usually placed at the start of the shutdown scripts.

```
Sub ThrowError (Number, Description)
  ObjApp = ""
  WScript.Echo WScript.ScriptName + " " + CStr(number) + " " + description
  NwmScript.Log WScript.ScriptName + " " + CStr(number) + " " + description
  WScript.Quit number
End Sub
```

The main script body

The main body of the script deals with manipulating the application in whichever state it may be encountered during a shutdown.

Initialising variables

The first task is to initialise the variables used in the remainder of the script.

```
hwnd = 0           '** window handle **
debugx = 0
```

Command line arguments

Two arguments are passed in by the service when the script is started – the handle to the window and a debug level parameter – this allows debugging to be enabled globally without changing the scripts.

```
Set objArgs = WScript.Arguments
For i = 0 To objArgs.Count - 1
  If objArgs(i) = "-DEBUG" Then
    debugx = 1
  ElseIf UCase(Left(objArgs(i), 6)) = "-HWND=" Then
    hwnd = CInt("&H" + right(objArgs(i), 8))
  Else
    WScript.Echo "Usage: " + WScript.ScriptName + " [-hwnd=0xnnnnn] [-DEBUG]"
    WScript.Quit -2
  End If
Next
```

Initialising the script helper object

Get the script helper object and initialise a few properties. Debug logging is normally off but can be turned on to help with debugging new scripts, each call to the helper will be logged with its input parameters and the returned value. The backup directory is set to the user's profile application data area.

```
'** get the Nightwatchman script helper object **
Set NwmScript = WScript.CreateObject("NWMSCRIPT.NWMScriptHelperCtrl.2")

NwmScript.logFileName = "c:\temp\nwmNotepad.log"
NwmScript.debug = debugx          '** can increase this to 3 for max
logging **
NwmScript.TimeoutSecs = 2
dumpdir = NwmScript.BackupDirectory
```

Changing the default NightWatchman backup directory

The `NwmScript.BackupDirectory` variable returns the default location where the NightWatchman backups are to be saved. This directory is set by NightWatchman as: `C:\documents and settings\<username>\application data\1e\NightWatchman\`, where *<username>* refers to the current user's logon name.

If you want to change the save location for the NightWatchman backups you should change the code that sets the `dumpdir` variable, as shown in the last line of the above code fragment, to one of your own choosing. For example, the following code sets the `dumpdir` variable to point to another temporary location:

```
dumpdir = "u:\temp"
```

Note: this example specifies the use of the user mapped drive *u:*. You can of course set this to any mapped drive the user may have set up on their PC. The advantage of using a user mapped drive is that users of the PC will be able to keep their NightWatchman backup files separate from all other users.

Code to help with testing

The following line is only used when writing / testing the scripts using "cscript.exe". Normally `hwnd` is passed in by the service so there is no possibility of interacting with the wrong window.

```
if hwnd = 0 Then hwnd = NwmScript.FindwindowLike ( 0, 0, "Notepad", "" )
if hwnd = 0 Then ThrowError -1, "No window"
```

Getting the application process ID

The NightWatchman `ActivateApp` stores the process ID for the currently active application. This may be used subsequently to ensure that keystrokes are only sent to the application being scripted.

```
NwmScript.ActivateApp ( hwnd )
```

Closing active application dialogs

Child and popup windows make it difficult for the script to continue – this function closes dialog boxes by clicking “ESC”, “Cancel”, “Close” or “OK” if that is the only option.

```
NwmScript.CloseActiveAppDialogs ( "Notepad" )
```

Getting the currently open filename

For Notepad the filename is extracted from the window caption.

```

** Get filename from window title **
wndwName=NwmScript.GetWindowText ( hwnd )
WScript.Echo ( wndwName )
FileName = Replace( wndwName, " - Notepad", "" )

```

Attempting to close the application

Try and close the application by typing ‘Alt-F4’. If the application contains unsaved data, the popup window “Do you want to save changes” will appear. We cancel this for now by sending ESC.

```

** find out if it contains unsaved data **
ExpectId = NwmScript.ExpectWindow("Notepad","", "{ESC}","Do you want to save the
changes")
NwmScript.SendKeys ( "{F4}" )                ** Alt-F, 4 (Exit) **
WScript.Sleep ( 1000 )

```

Does the open file contain unsaved data

ExpectCancel returns the number of times the expect window has been seen – this is used to decide if the file contains unsaved data.

```

UnSaved = NwmScript.ExpectCancel ( ExpectId )
if UnSaved > 0 Then

```

Handling the Save As dialog

The Save As dialog box is brought up.

```

NwmScript.SendKeys ( "%FA" )                ** Alt-F, A (Save As) **
h = NwmScript.WaitForWindow ( 0, 0, "", "Save As" )
if h > 0 Then

```

Creating a new file name

The Script Helper creates a new filename using the input name as a base. The helper also makes sure that this file does not already exist as this could cause failures later. The extension and path parameters are optional but the file existence check cannot be performed without these.

```
NewName = NwmScript.MakeNWMFilename ( FileName, ".txt", dumpdir )
```

This new filename is sent to the Save As dialog.

```

NwmScript.SendKeys ( "(" + dumpdir + NewName + "){WAIT_300}%S" )
WScript.Echo " Saving to " + dumpdir + NewName
end if
end if

```

Closing the application

Notepad is closed.

```

NwmScript.SendKeys ( "{F4}" )                ** Alt-F, 4 (Exit) **
WScript.Echo " closing"

```

Tidying up the environment

Clean up. This section should be common to all scripts.

```
wScript.Echo "Script Complete"  
wScript.Quit 0
```

Section 4 The NightWatchman Script Helper Reference

This section details the methods that are available in the NightWatchman *NWMScriptHelper.ocx*. These methods become available for use within a shutdown script once an instance of the NWMScriptHelper has been created. To do this you should call the following code from within your script:

```
WScript.CreateObject("NWMScript.NWMScriptHelperCtrl.2")
```

4.1 Method and property categories

The following tables describe the general uses of the helper methods. These have been divided into general categories for convenience. The section following these tables is an alphabetic reference to each method.

Application methods

These methods provide access to currently running applications.

Method	Description
ActivateApp	Activate the specified application window.
CloseActiveAppDialogs	Close any active dialogs in the specified application.
CloseActiveAppDialogsLike	Closes any active dialogs, like the specified class, the specified application may have open, by selecting the Cancel and Close buttons where appropriate.
GetActiveObject	Create an application object for an application with an application scripting class.

Window methods

The following methods allow you to retrieve and set properties for windows open in the currently running application.

Method	Description
ExpectCancel	Cancel the expect window call.
ExpectWindow	Wait for a window matching the specified search string or title.
FindInvisibleWindowLike	Allows you specify a search string to find an application window that is hidden. This is similar to the Windows API FindWindow, except that it can use regular expressions for the search strings.
FindWindowLike	Find a window using a specified search string.
GetAccItem	Retrieve the value of a specified item in the active window.
GetWindowText	Get the title of the specified window.
SelectAccItem	Make the specified item active.
SelectSubItem	Select the specified sub item in the active window.
SetActiveWindow	Make the specified window active.
WaitForWindow	Wait for the specified window to be created.
WaitWindowGone	Wait for the specified window to close.
ClickButton	Perform the action of the mouse clicking the specified dialog button.
ClickMouse	Perform the action of clicking the specified mouse button.
SendKeys	Send keystrokes to the active application.
SendKeysToWindow	Send keystrokes to a specified window
SendKeysToDialog	Send Keystrokes to the active dialog

Save file environment methods

The following methods enable aspects of the applications open file environment to be utilised within the shutdown script.

Method	Description
GetCmdLine	Get the command line path for the running application.
GetCurrDir	Return the current directory for the running application.
GetEnvVar	Return the value for the specified environment variable.

File methods

These methods provide file specific functionality.

Method	Description
CopyFile	Copy a file to a specified location.
GetFullPath	Get the full path for the specified file.
MakeNWMFilename	Create a filename for a NightWatchman saved file.
MoveFile	Move a file to a specified location.

NightWatchman methods

These methods define an interface for interacting with the NightWatchman application.

Method	Description
WriteToBackupList	Output the name of a backed up file to the logfile so it can be viewed by the user from the NightWatchman Backup Dialog.

Script properties and methods

The following properties and methods define settings for controlling and debugging the NightWatchman scripts.

Method	Description
debug	Property specifies the level of debugging output to the logfile.
Log	Method outputs a string to the scripts logfile.
LogFileName	Property sets the location of the scripts logfile.
TimeoutSecs	Property sets the timeout for wait operations.
BackupDirectory	Method returns the user specific backup directory determined by NightWatchman: NOTE: this directory is fixed so that files backed up are saved to the logged on user's profile.

4.2 Method alphabetic reference

This section provides a specific reference for each method available in the NightWatchman helper. Each reference contains a prototype with inputs and returns. The arguments are shown in italics; the types of the arguments are shown in bold; optional arguments are shown in square brackets.

ActivateApp

Description	Activate the Specified Application Window
Prototype	long ActivateApp(long <i>hWnd</i>)
Inputs	<i>hWnd</i> Window Handle
Returns	The activated application window handle
Remarks	This method is similar to WScript.ActivateApp though this accepts the Windows handle as the input. It also stores the applications process ID for later comparisons.
Example	The following example activates the application specified by <i>hWnd</i> . The handle of the activated window is returned in <i>h1</i> : <pre>h1 = NwmScript.ActivateApp (hWnd);</pre>

BackupDirectory

Description	This returns the user specific directory used to contain the NightWatchman backups.
Prototype	LPCTSTR BackupDirectory()
Inputs	none
Returns	The backup directory specific to the currently logged on user.
Remarks	This directory is set by NightWatchman as: <i>C:\documents and settings\<username>\application data\1e\NightWatchman\</i> , where <username> refers to the current user's logon name. Note: This directory is fixed by NightWatchman according to the user logged on at the time of shutdown.
Example	The following example returns the backup directory for the current logged on user: <pre>Dir = BackupDirectory();</pre>

ClickButton

Description	Performs the operation of the mouse clicking the specified dialog button, by sending a BM_CLICK message to the button window.
Prototype	BOOL ClickButton(long <i>hWnd</i> , LPCTSTR <i>szbuttonTitle</i>)
Inputs	<i>hWnd</i> Dialog window Handle <i>buttonTitle</i> Title of Button
Returns	Indicates success or failure of the button click message
Example	The following example simulates the mouse clicking the OK button of the window <i>h1</i> : <pre>NwmScript.ClickButton (h1, "OK");</pre>

ClickMouse

Description	Performs the operation of clicking the specified mouse button.
Prototype	BOOL ClickMouse (LPCTSTR <i>LeftOrRight</i>)
Inputs	<i>LeftOrRight</i> Mouse Button "L" or "R"
Returns	Indicates success or failure of the mouse click message
Remarks	This method includes code to compensate for switched mouse buttons so scripting can always assume righthanded operation. It is assumed that the cursor has already been positioned before this method is called. See <i>SelectAccItem</i> , <i>SelectSubItem</i> .
Example	The following example positions the mouse cursor on the Cancel button on the window h1, then clicks the left mouse button: <pre>NwmScript.SelectSubItem (h1, "Cancel"); NwmScript.ClickMouse ("L");</pre>

CloseActiveAppDialogs

Description	Closes any active dialogs the specified application may have open, by selecting the Cancel and Close buttons where appropriate.
Prototype	BOOL CloseActiveAppDialogs([const VARIANT FAR& <i>AppName</i>])
Inputs	<i>AppName</i> Optional Application Name.
Returns	Indicates success or failure of the close operation
Remarks	This guarantees the Application is in a state ready for operations. This should be executed at the start of a script after the helper object is instantiated. The input field uses the regular expression facility for text mapping. To find the whole string "^the string\$" - special characters may need to be escaped.
Example	The following example closes any active dialogs open in "Microsoft Word": <pre>NwmScript.CloseActiveAppDialogs ("Microsoft word");</pre>

CloseActiveAppDialogsLike

Description	Closes any active dialogs, like the specified class, the specified application may have open, by selecting the Cancel and Close buttons where appropriate.
Prototype	BOOL CloseActiveAppDialogs([const VARIANT FAR& <i>AppName</i>], [const VARIANT FAR& <i>WindowClass</i>])
Inputs	<i>AppName</i> Optional Application Name. <i>WindowClass</i> The Window Class.
Returns	Indicates success or failure of the close operation
Remarks	This guarantees the Application is in a state ready for operations. This should be executed at the start of a script after the helper object is instantiated. The input field uses the regular expression facility for text mapping. To find the whole string "^the string\$" - special characters may need to be escaped. This function differs from <i>CloseActiveAppDialogs</i> by allowing a dialog class to be specified; applications often use their own custom dialog classes rather than the windows standard, this function is designed to cater for these exceptions
Example	The following example closes any active dialogs open in "Microsoft Word": <pre>CloseActiveAppDialogsLike ("Microsoft word", "#32770");</pre>

CopyFile

Description	Copies a file from one location to another. The extension parameter is used as the extension might not be included in the filename due to user's Explorer settings.
Prototype	BOOL CopyFile (const VARIANT FAR& <i>ExistFileName</i> , const VARIANT FAR& <i>NewFileName</i> , const VARIANT FAR& <i>Extension</i>)
Inputs	<i>ExistFileName</i> Source filename <i>NewFileName</i> Destination filename <i>Extension</i> File extension
Returns	Indicates success or failure of the copy file operation
Remarks	The method first attempts to copy the file specified. If this fails, a second attempt is made using the extension specified. If a file to be copied has zero length, CopyFile automatically deletes the copy, so empty backup files are not created.
Example	The following example copies "guide.doc" to the file "c:\temp\guide.doc": <pre>NwmScript.CopyFile("guide.doc", "c:\temp\guide.doc", ".doc")</pre>

DeactivateApp

Description	Deactivates the currently selected application so that another can be reselected
Prototype	long DeactivateApp()
Inputs	<i>None</i>
Returns	The previously activated application window handle
Remarks	This is primarily required when accessing complicated applications that host embedded application windows running in another process. Deactivating the current application process and reselecting another will allow you to find and manipulate windows in that process
Example	The following example deactivates the application <pre>h1 = NwmScript.DeactivateApp ();</pre>

ExpectCancel

Description	Cancels a call to ExpectWindow. The parameter is the ID returned from the call to ExpectWindow.
Prototype	long ExpectCancel(long <i>ExpectorId</i>)
Inputs	<i>ExpectorId</i> The wait job handle to be cancelled
Returns	The usage count for the "expector"
Remarks	The return value may be used to find out how many times the "Expector" action sequence was used.
Example	The following example sets an <i>ExpectWindow</i> on a "Save As" dialog. The <i>ExpectCancel</i> call cancels the wait: <pre>ExpectId = NwmScript.ExpectWindow("Save As", "", NewName + "{WAIT_500}%S"); NwmScript.ExpectCancel(ExpectId);</pre>

ExpectWindow

Description	Specifies an operation to be performed upon the appearance of a window specified by a search string or its title.								
Prototype	long ExpectWindow(const VARIANT FAR& <i>WindowTitle</i> , const VARIANT FAR& <i>WindowClass</i> , const VARIANT FAR& <i>Action</i> , [const VARIANT FAR& <i>ChildTextName</i>])								
Inputs	<table> <tr> <td><i>WindowTitle</i></td> <td>The Window Title.</td> </tr> <tr> <td><i>WindowClass</i></td> <td>The Window Class.</td> </tr> <tr> <td><i>Action</i></td> <td>Action to be performed as a sequence of keys sent by SendKeys</td> </tr> <tr> <td><i>ChildTextName</i></td> <td>Optional text search string to further identify the Child.</td> </tr> </table>	<i>WindowTitle</i>	The Window Title.	<i>WindowClass</i>	The Window Class.	<i>Action</i>	Action to be performed as a sequence of keys sent by SendKeys	<i>ChildTextName</i>	Optional text search string to further identify the Child.
<i>WindowTitle</i>	The Window Title.								
<i>WindowClass</i>	The Window Class.								
<i>Action</i>	Action to be performed as a sequence of keys sent by SendKeys								
<i>ChildTextName</i>	Optional text search string to further identify the Child.								
Returns	A Job ID that can be used to cancel the expect.								
Remarks	<p>This method may cause an exit from the <i>WaitWindowGone</i> method if the action specified is NULL.</p> <p>The operation to be performed can be cancelled using the <i>ExpectCancel</i> call.</p> <p>The Title and ChildText fields use the regular expression facility for text mapping. To find the whole string "<i>^the string\$</i>", see <i>FindWindowLike</i> for more information.</p> <p>As this call runs asynchronously, it may be necessary to put a delay into the script to give it time to detect the window.</p>								
Example	<p>The following example will wait for a "Save As" window and type the string "Doc1.Doc" into the Active "Filename" text box:</p> <pre>ExpectId = NwmScript.ExpectWindow("Save As", "", "Doc1.Doc")</pre>								

FindInvisibleWindowLike

Description	Allows you specify a search string to find an application window that is hidden. This is similar to the Windows API FindWindow, except that it can use regular expressions for the search strings.
Prototype	long FindInvisibleWindowLike(long hWnd, long hWndChild, LPCTSTR szClassLike, LPCTSTR szTitleLike)
Inputs	<i>hWnd</i> Application Window Handle <i>hWndChild</i> Child Window Handle <i>ClassLike</i> Class Search String <i>TitleLike</i> Title Search String
Returns	The application window handle if found, else 0
Remarks	The input fields <i>szClassLike</i> and <i>szTitleLike</i> use the regular expression facility for text mapping. To find the whole string "^the string\$" - special characters may need to be escaped. This is most useful for applications that run multiple document views in a single process and share dialogs between them such as MSWord. These applications often use a hidden window to host shared child dialogs.
Example	The following example will search for any windows open with a title that starts with the text "Visio": <pre>hWnd = NwmScript.FindInvisiblewindowLike(0, 0, "", "^Visio.*");</pre> This next example searches for the highest level dependent window of the application <i>hWinMain</i> and makes it active. This could be used to close dependent windows before starting to close the main application. <pre>h1 = hwinMain while((h2 = (HWND) FindInvisiblewindowLike((long) h1, 0, "", "")!=0) { h3 = h4; SetForegroundWindow ((long) h3); }</pre>

FindWindowLike

Description	Allows you specify a search string to find an application window. This is similar to the Win API FindWindow, except it can use regular expressions for the search strings.
Prototype	long FindWindowLike(long hWnd, long hWndChild, LPCTSTR szClassLike, LPCTSTR szTitleLike)
Inputs	<i>hWnd</i> Application Window Handle <i>hWndChild</i> Child Window Handle <i>ClassLike</i> Class Search String <i>TitleLike</i> Title Search String
Returns	The application window handle if found, else 0
Remarks	The input fields <i>szClassLike</i> and <i>szTitleLike</i> use the regular expression facility for text mapping. To find the whole string "^the string\$" - special characters may need to be escaped.
Example	The following example will search for any windows open with a title that starts with the text "Visio": <pre>hWnd = NwmScript.FindwindowLike(0, 0, "", "^Visio.*");</pre> The next example will search for the highest level dependent window of application <i>hWinMain</i> and make it active. This could be used in closing

	<p>dependent windows before starting to close the main application.</p> <pre> h1 = hwinMain while((h2 = (HWND) FindWindowLike((long) h1, 0, "", "")!=0) { h3 = h4; SetForegroundwindow ((long) h3); } </pre>
--	---

GetAccItem

Description	Retrieves the value of a specified item in an active window. The Name and Role fields are retrieved from the Windows Accessibility parameters.	
Prototype	BSTR GetAccItem (long <i>hWnd</i> , long <i>depth</i> , [const VARIANT FAR& <i>vName</i>], [const VARIANT FAR& <i>vRole</i>], [const VARIANT FAR& <i>vName2</i>], [const VARIANT FAR& <i>vRole2</i>]
Inputs	<i>hWnd</i> <i>depth</i> <i>vName</i> <i>vRole</i> <i>vName2</i> <i>vRole2</i>	Application Window Handle Depth Optional Item Search String 1 Optional Location of String 1 Optional Item Search String 2 Optional Location of String 2
Returns	The selected item value	
Remarks	<p>The depth parameter reduces the recursion depth. This should be kept as low as possible to increase the efficiency of this method.</p> <p>To assist with script writing, the ListAccessible(long hWnd,short depth) method, can be used to list all the Accessibility parameters for any window.</p>	
Example	<p>This example will search the window section of the window, h1, for the Item label "File name:" and return the value in it's edit box:</p> <pre> FileName = NwmScript.GetAccItem (h1, 3, "File name:", "window", "", "title bar"); </pre>	

GetActiveObject

Description	Creates an Application Object. Similar to the WScript GetObject method.	
Prototype	LPDISPATCH GetActiveObject(const VARIANT FAR& <i>vClassName</i>)
Inputs	<i>vClassName</i>	Application Class
Returns	The application object pointer	
Remarks	NOTE: this method can only be used for applications which expose an application scripting class.	
Example	<p>The following example will return a pointer to the Word object application:</p> <pre> ObjApp = NwmScript.GetActiveObject("word.Application") </pre> <p>Using the returned pointer will allow access to Word settings and functions directly. For example, ObjApp.Documents.Count will tell us the number of Documents that need closing.</p>	

GetCmdLine

Description	Gets the full command line path of the executing program.
Prototype	BSTR GetCmdLine()
Inputs	None
Returns	The full path of the command line
Example	Suppose the following command were executed within a notepad script, CmdLine could possibly return "C:\WINNT\System32\notepad.exe": <pre>CmdLine = NwmScript.GetCmdLine();</pre>

GetCurrDir

Description	Returns the result of a GetCurrentDirectory in the context of the target application process.
Prototype	BSTR GetCurrDir()
Inputs	None
Returns	The current directory of the application process.
Remarks	This call may return different results depending on the state of the target application. In particular, the "OpenFileName" and "SaveAsFileName" dialog boxes will set the current directory to the users default Save path.
Example	The following call returns the current directory: <pre>CurrentDir = GetCurrDir()</pre>

GetEnvVar

Description	Gets the value of the Environment variable specified.	
Prototype	BSTR GetEnvVar(const VARIANT FAR& <i>VarName</i> , const VARIANT FAR& <i>IsPath</i>)	
Inputs	<i>VarName</i>	The variable to find
	<i>IsPath</i>	The application path
Returns	The value of the environment variable	
Example	<p>In this example, the GetEnvVar call makes sure that the <i>SavePath</i> is the full path name, including the drive letter.</p> <pre> if (DlgFileName!=FileName) { SavePath = DlgFileName.replace(FileName,""); if(debug & 1) WScript.Echo("Bad SavePath : " + SavePath); SavePath = NwmScript.GetEnvVar("=" + SavePath.toUpperCase(), 1); } WScript.Echo ("SavePath : " + SavePath); NewName = NwmScript.MakeNWMFileName(FileName, ".txt", SavePath); </pre>	

GetFullPath

Description	Gets the full path of a specified file.	
Prototype	BSTR GetFullPath(const VARIANT FAR& <i>vFileName</i> , const VARIANT FAR& <i>vDlgFileName</i>)	
Inputs	<i>vFileName</i>	Currently unspecified
	<i>vDlgFileName</i>	The filename retrieved from a "Save As" file box
Returns	The absolute path of the dialog file name.	
Remarks	This function removes the complexity of extracting the absolute pathname from the SaveAs File dialog box. The <i>DlgFileName</i> would normally be extracted from the dialog using <i>GetAccItem</i> . This is then used in special cases where the filename was not fully resolved on the command line when the application was started.	
Example	<p>The following example should resolve the following scenarios:</p> <pre> notepad c:junk.txt (has to find the current directory on C:) notepad \junk.txt (has to find the current drive) notepad junk.txt (uses the current directory) notepad (saves the file to the Users default Save As folder) </pre> <pre> DlgFileName = GetAccItem(h, 7, "File name:", "editable text") Fullpath = GetFullPath(FileName, DlgFileName) </pre>	

GetWindowText

Description	Gets the specified window title	
Prototype	BSTR GetWindowText (long <i>hWnd</i>)	
Inputs	<i>hWnd</i>	Window Handle
Returns	The title of the specified window	
Example	<p>The following example will return the window hWnd's title:</p> <pre> wndwName=NwmScript.GetWindowText (hWnd); </pre>	

Log

Description	This method may be used to add data to the script specific log files as defined in the property NWMScript.LogFileName.
Prototype	long Log (LPCTSTR <i>format</i>)
Inputs	<i>format</i> The string to send to the logfile
Returns	The number of bytes written to the log file, if this is 0 the write failed.
Remarks	The script specific log files are used when debugging scripts. Each NWM method called will log input parameters and results to this logfile. The detail in this log may be made more verbose by increasing the value of the debug property. Remember to disable script specific logging before releasing scripts to production.
Example	The following example will write "Error" to the NightWatchman error log. <pre>NwmScript.Log ("Error");</pre>

MakeNWMFilename

Description	Adds the NightWatchman tag and current date to the passed file name or replaces it if it already exists. It then returns this new filename.
Prototype	BSTR MakeNWMFilename(LPCTSTR <i>szFilename</i> , [const VARIANT & <i>vExtension</i>], [const VARIANT & <i>vPath</i>])
Inputs	<i>szFilename</i> The root filename <i>vExtension</i> Optional filename extension. To help ensure the extension is removed correctly. <i>vPath</i> Optional path. so the new filename can be checked to make sure it does not already exist, if it does an extra character is added to the filename
Returns	The generated file name
Remarks	The filename created is of the format "filename.NWM.YYYYMMDDhhmm.ext". If this name already exists in the directory, it is written in the format "filename.NWM.YYYYMMDDhhmm.A.ext" with the character "A" being incremented until we find a non-existent filename.
Example	The NewName variable will hold the returned formatted filename.NWM.YYYYMMDDhhmm.ext: <pre>NewName = NwmScript.MakeNWMFilename(thisDoc.Name, ".doc", DocPath);</pre>

MoveFile

Description	Moves a file from one location to another. The extension parameter is the extension of the new file name and is used when trying to ensure the new file name does not exist.
Prototype	BOOL MoveFile(const VARIANT FAR& <i>vExistFileName</i> , const VARIANT FAR& <i>vNewFileName</i> , const VARIANT FAR& <i>vExtension</i>)
Inputs	<i>vExistFileName</i> Source Filename <i>vNewFileName</i> Destination Filename <i>vExtension</i> The file extension
Returns	Indicates the success or failure of the move operation
Remarks	If a file to be moved has zero length MoveFile deletes it. So empty backup files are not created.
Example	The following examples moves a text file from its current location to the temp directory: <pre>NwmScript.MoveFile("guide.doc", "c:\temp\guide.doc", ".txt");</pre>

SelectAccItem

Description	Makes the specified item active. The Name and Role fields are compared against the Windows Accessibility parameters.
Prototype	BOOL SelectAccItem(<i>long hWnd</i> , <i>long depth</i> , const VARIANT FAR& <i>vName</i> , const VARIANT FAR& <i>vRole</i>)
Inputs	<i>hWnd</i> Application Window Handle <i>depth</i> Depth to search <i>vName</i> Item to activate title <i>vRole</i> Location of Item
Returns	Indicates the success or failure of the selection
Remarks	To assist with script writing, the ListAccesible(long hWnd,short depth) method, can be used to get the Accessibility parameters for any window.
Example	The following example searches the window section of the window, h1, for the Item label "File name:" and make it active. This is useful when inserting filenames in "Save As" boxes. <pre>FileName = NwmScript.SelectAccItem (h1, 3, "File name:", "window", ", "title bar");</pre>

SelectSubItem

Description	Makes an item in a window active so that subsequent mouse clicks will work
Prototype	long SelectSubItem(long <i>hWnd</i> , LPCTSTR <i>ControlName</i>)
Inputs	<i>hWnd</i> Window Handle <i>ControlName</i> Title of Item to activate
Returns	Handle to the sub item. If this call fails the return value is 0.
Example	The following example selects the "Cancel" button on the activate window: <pre>NwmScript.SelectSubItem (h1, "Cancel");</pre>

SendKeys

Description	Sends Keystrokes to the active application.	
Prototype	BOOL SendKeys(LPCTSTR <i>Keystrokes</i> , [const VARIANT FAR& <i>delay</i>])
Inputs	<i>Keystrokes</i> <i>delay</i>	String of Keystrokes to send Send delay (Optional - defaults to zero)
Returns	Always returns TRUE.	
Remarks	<p>The syntax for the <i>Keystrokes</i> string is similar to Jscript/VB SendKeys with the addition of the {WAIT_nn}. This allows you to add a delay during the sending of a sequence of keys.</p> <p>The SendKeys method does a check to see if control is in the activated process before sending keys and exits if not.</p> <p>This method clears Caps Lock and Shift Key states at the start of each sequence.</p> <p>The plus sign (+), caret (^), percent sign (%), tilde (~), parentheses () and braces ({}) characters have special meanings to SendKeys. To ensure that characters are not mapped, enclose the whole string within parenthesis.</p> <p>For example, "(filea+b.txt)"</p>	
Example	<p>This will send the keys ALT, F, A</p> <pre>NwmScript.SendKeys ("%FA");</pre>	

SendKeysToWindow

Description	Send keystrokes to a specified window.	
Prototype	BOOL SendKeysToWindow(long <i>hWnd</i> , LPCTSTR <i>Keystrokes</i> , [const VARIANT FAR& <i>delay</i>])
Inputs	<i>hWnd</i> <i>Keystrokes</i> <i>delay</i>	Window Handle String of Keystrokes to send Send delay (Optional - defaults to zero)
Returns	Always returns TRUE.	
Remarks	The syntax for the <i>Keystrokes</i> string is the same as described in the <i>SendKeys</i> function.	
Example	<p>The following example will send the keys ALT, F, A to the active dialog:</p> <pre>NwmScript.SendKeysToWindow (hwnd, "%FA");</pre>	

SendKeysToDialog

Description	Send Keystrokes to the active dialog
Prototype	BOOL SendKeysToDialog(LPCTSTR <i>Keystrokes</i> , [const VARIANT FAR& <i>delay</i>])
Inputs	<i>hWnd</i> Window Handle <i>Keystrokes</i> String of Keystrokes to send <i>delay</i> Send delay (Optional - defaults to zero)
Returns	Always returns TRUE.
Remarks	Sometimes the child control within a dialog may not properly pass key presses to their parent. This problem can be avoided by sending the keys to the dialog containing the focused control instead. The syntax for the <i>Keystrokes</i> string is the same as described in the <i>SendKeys</i> function.
Example	The following example sends keys to the active dialog: <pre>NwmScript.SendKeysToDialog ("%FA");</pre>

SetActiveWindow

Description	Makes the window specified active
Prototype	BOOL SetActiveWindow(long <i>hWnd</i>)
Inputs	<i>hWnd</i> Window Handle
Returns	Indicates the success or failure of the activate operation
Example	The following example sets the window specified by window handle h1, to active: <pre>NwmScript.SetActiveWindow (h1);</pre>

WaitForWindow

Description	Waits for the window specified to be created.
Prototype	long WaitForWindow (long <i>hWndMain</i> , long <i>hWndUnused</i> , LPCTSTR <i>szClassLike</i> , LPCTSTR <i>szTitleLike</i>)
Inputs	<i>hWndMain</i> Window Handle <i>hWndUnused</i> Unused Window Handle <i>szClassLike</i> Class Search String <i>szTitleLike</i> Title Search String
Returns	The window handle of the created window
Remarks	<i>szClassLike</i> and <i>szTitleLike</i> can use regular expressions to match text. The script <i>TimeoutSecs</i> property sets the timeout length for the wait.
Example	The following example waits for an application hWnd to display a "Save As" window: <pre>h1 = NwmScript.WaitForWindow (hWnd, 0, "", "Save As");</pre>

WaitWindowGone

Description	This method waits for the specified window to close.
Prototype	long WaitWindowGone (long <i>hWnd</i> , [const VARIANT FAR& <i>vTimeout</i>])
Inputs	<i>hWnd</i> Window handle of window wanted closed <i>vTimeout</i> Optional time out
Returns	Returns 0 if the window has gone within the time specified, otherwise returns the window handle given unless an expected window caused the return.
Remarks	You can also use this method to check for "expected" windows with NULL actions. These are windows in the "expected" windows list created by the ExpectWindow method that have no action to perform. Hence acts as an Exit for this function.
Example	The following example waits for the window specified by window handle h1 to be gone: <pre>NwmScript.WaitWindowGone (h1);</pre>

WriteToBackupList

Description	This method will log the name of a backed up file to NWMFiles.txt.
Prototype	BOOL WriteToBackupList(const VARIANT FAR& <i>vFileName</i> , const VARIANT FAR& <i>vOrigName</i>)
Inputs	<i>vFileName</i> Newly NightWatchman created backup file <i>vOrigName</i> Original file name
Returns	Indicates the success or failure of the write action
Remarks	The contents of this file are used for managing NightWatchman backup files through the NightWatchman System Tray icon. Note: Use of this backup file is a requirement if you want to allow end-users to view the files backed up by NightWatchman.
Example	The following example adds the name of the backed up file, "DocPath" + "NewNm" (where those variables have been set previously in the script), to the backup log along with its original name: <pre>X = NwmScript.WriteToBackupFile(DocPath + NewNm, OrigNm);</pre>

LogFileName

Description	This property sets the destination of the log file.
Property	CString NwmScript.LogFileName
Remarks	Any backslash characters in the path will need to be escaped.
Example	The following example sets the location of the NightWatchman logfile for the current script to be: C:\TEMP\NWM.LOG <pre>NwmScript.LogFileName = "C:\\TEMP\\nwm.log";</pre>

TimeoutSecs

Description	Property that sets the wait for window timeout.
Property	long NwmScript.TimeoutSecs
Remarks	This property is also used as the default timeout for WaitWindowGone.
Example	The following sets the timeout value to 10: <pre>NwmScript.TimeoutSecs = 10;</pre>

debug

Description	Property that sets the debugging level
Property	Short <i>NwmScript.debug</i>
Remarks	The debugging level may be between 0 and 3. With 0 turning debug information off and 3 being the maximum verbose level.
Example	<p>The following example sets no debug:</p> <pre>NwmScript.debug = 0;</pre> <p>The following sets just parameter and results debuggin information:</p> <pre>NwmScript.debug = 1;</pre> <p>This example sets the maximum amount of debugging:</p> <pre>NwmScript.debug = 3;</pre>

Section 5 Troubleshooting

When troubleshooting problems with NightWatchman you should first check that the system meets the requirements set out in the *Requirements* section of *The NightWatchman Installation Guide*. If the requirements are met you should then follow through the process for creating a problem report to send to the 1E technical support team shown below.

5.1 Checking the problem

Before contacting 1E technical support with a problem you should check the following:

- Make sure that the minimum requirements have been met.
- Make sure that the problem is reproducible after a machine reboot (if possible) or after stopping and restarting the service.
- Read the readme.txt file included with the NightWatchman installation. This includes late breaking news, and details on known issues.

5.2 Contact 1E technical support

Should you encounter problems with a particular NightWatchman script, the best action is to contact the 1E technical support team who should be able to guide you through the process of determining the cause of the problem.

To help 1E determine the solution quickly, you should create a technical report of the problem.

Creating a technical report

The technical report should contain the following information:

- The NightWatchman logfile.
- Version number of NightWatchman.
- The OS, version number and patch level the machine is running.
- The NightWatchman shutdown script logfile (if applicable).
- The NightWatchman shutdown script (if this has been edited).
- Which type of power management is available on the client machine?

To help determine the cause of technical problems NightWatchman keeps a logfile of all its settings and interactions. This is the main source of information in the technical report. By default this logfile is called *NightWatchman.log* and is located in one of the following directories, depending on which OS you are running.

In Windows Vista this file is located in:

```
C:\ProgramData\1E\Nightwatchman50
```

In Windows XP and Windows 2000 this file is located in:

```
C:\Documents and Settings\All Users\Application Data\1E\Nightwatchman50
```

The version number for NightWatchman can be found using the following NightWatchman command line option:

```
C:> nightwatchman -version
```

You should then email the details of the problem encountered along with the above information to support@1e.com. A technical consultant will then contact you to help find a suitable solution.

Section 6 Further Information

For more detailed questions about specific situations that may be relevant to your network you are always welcome to contact 1E directly. Our contact details are provided below.

6.1 Contact details

1E provide a number of SMS enhancement tools as well as consultancy services. This section provides information on how to get information about 1E and contact details for the various departments within 1E.

Website

The essential resource for information about 1E and its products is the website.

www.1e.com

Telephone and fax

The following UK numbers are available for telephone and fax enquiries:

Tel: +44 (0)20 8326 3880

Fax: +44 (0)20 8840 9578

The following number is available in the United States of America:

Tel: 1-800 516 6938

Post

The postal address for 1E is:

Head Office:
1E Ltd,
97-107 Uxbridge Rd,
London W5 5TL, UK

Sales

To contact the sales department at 1E you can use the following email address:

sales@1e.com

Technical support

To contact the technical support department at 1E you can use the following address:

support@1e.com