

NSIS User Manual

Sample pages transcribed by: pengyou

This documentation is for NSIS version 2.0b4 (CVS)

NSIS User Manual

NSIS is a free scriptable win32 installer/uninstaller system that doesn't suck and isn't huge.

To get all of the latest information about NSIS development: [NSIS Development Page](#).

To talk with others in the NSIS community: [NSIS forum](#).

For more useful functions, NSIS related software, example scripts, plugins and tutorials: [NSIS Archive](#).

Copyright (c) 1999–2003 Nullsoft, Inc.

0. Table of Contents

- 0. Table of Contents
- 1. Introduction to NSIS
- 2. Tutorial: The Basics
 - 2.1 Introduction
 - 2.2 Script Files
 - 2.3 Scripting structure
 - 2.3.1 Installer Attributes
 - 2.3.2 Sections
 - 2.3.3 Functions
 - 2.3.4 Working with Scripts
 - 2.3.5 Compiler Commands
 - 2.4 Compiler
 - 2.5 Enhancing NSIS
- 3. MakeNSIS Usage
- 4. Scripting Reference
 - 4.1 Script File Format
 - 4.2 Instructions
 - General Purpose Instructions
 - 4.2.1 CallInstDLL
 - 4.2.2 CopyFiles
 - 4.2.3 CreateDirectory
 - 4.2.4 CreateShortCut
 - 4.2.5 GetDLLVersion
 - 4.2.6 GetDLLVersionLocal
 - 4.2.7 GetFileTime
 - 4.2.8 GetFileTimeLocal
 - 4.2.9 GetFullPathName
 - 4.2.10 GetTempFileName
 - 4.2.11 SearchPath
 - 4.2.12 SetFileAttributes
 - 4.2.13 RegDLL
 - 4.2.14 UnRegDLL
 - Miscellaneous Instructions
 - 4.2.15 InitPluginsDir
 - 4.2.16 SetShellVarContext
 - 4.2.17 Sleep
 - 4.3 Plugin DLLs
 - 4.3.1 Using Plugin Commands
 - 4.3.2 Disabling Plugin Unloading
- B. Useful Functions
 - 1 Get parent directory
 - 2 Trim newlines
 - 3 Get command-line parameters
 - 4 Search in a string
 - 5 Get Windows version
 - 6 Get Internet Explorer version
 - 7 Is .NET Framework installed?
 - 8 Is Flash Player installed?
 - 9 Add a shared DLL
 - 10 Remove a shared DLL
 - 11 Upgrade a DLL (macro)
 - 12 Connect to the internet
 - 13 More

Index

1. Introduction to NSIS

About NSIS

NSIS is a free scriptable win32 installer/uninstaller system that doesn't suck and isn't huge.

NSIS exists largely because of the need to distribute Winamp.

In the beginning, Winamp was distributed as a simple .ZIP file (see Winamp 1.0). Eventually, Winzip was nice enough to give us a license to the Winzip self extractor (see Winamp 2.0)

The self extracting ZIP file became too restrictive, so we started using a custom developed installer. This installer was generated using a combination of "bin2h", batch files, and Visual C++. To add a new file into the distribution, or to have the installer do something new on install, it required writing quite a bit of code. (see Winamp 2.5)

Another thing we needed was a good way to distribute Winamp plug-ins. The solution: PIMP (Plug-In Mini Packager). PIMP was actually based on the custom installer (though was a lot simpler and removed a lot of code), but stored the data and strings that were unique to that installer in a block at the end of the file. It was small and simple, allowing the simple functions of detecting where Winamp was installed and extracting files. The limitations of PIMP were that it could only extract files, and it was designed for small installers (i.e. an 8 MB installer would take 8 MB of memory).

NSIS, which stands for "Nullsoft SuperPIMP Installation System" or "Nullsoft Scriptable Installation System" or whatever you want, is based on PIMP but is designed to be much more flexible. NSIS creates installers that are capable of installing, uninstalling, setting system settings, extracting files, etc. Pretty much anything. All with the minimum of overhead – NSIS installers don't bother throwing up a big blue gradient, they don't bother decompressing themselves three different times, telling the user to "please wait". They get to the point and get the job done.

In order to make NSIS even more powerful than it already is, we have released it under an open source license (it is actually the zlib/libpng license, which is approved by opensource.org). What does this mean? It means that if you want to add the functionality you need to NSIS, you can. It means if you want to make your own custom version of NSIS (or some product that includes NSIS), and sell it, you can. Or if you just want to distribute your software using NSIS, you sure as hell can.

The result of all of this is an installation system for win32 that lets you compile nice little scripts (which are text files, no wizards to slow you down) into tiny installers. Many features are supported, and the whole thing just works pretty damn well (at least, we think so).

Main Features

Here's a short list of some of NSIS' features.

- SuperPiMP™ technology (so advanced, so amazing, we won't even tell you what it is).

NSIS User Manual

- Generates self contained, win32 executable installer.
- Uninstall support (installer can automatically generate an uninstaller)
- Optional installer self-verification using a CRC32.
- Compression choices of zlib or bzip2 based compression. The installer can compress everything together, or individually.
- Approximately 20–40k overhead over compressed data size (depending on features enabled, compression algorithm, and so on – the default options are ~34k).
- Ability to display a license agreement, text or RTF
- Ability to detect destination directory from the registry, and let the user override (or not let them)
- Customizable appearance (background, icons, text, checkmarks, images)
- Multiple install configurations (usually Minimal, Typical, Full), and custom configuration
- Easy to use plug-in system (includes plug-ins for generic dialog construction and user interaction, as well as HTTP downloading)
- Fully multilingual. 28 translations available including German, French, Spanish, Italian, Dutch, Chinese and more...
- Installers can be as large as 2GB (theoretically -- when building on Win9x the limit seems to be around 500MB, however building on NT then installing on Win9x works with larger sizes)
- Optional Silent mode for automated installations
- A primitive preprocessor
- User interface changeable from head to toe
- An optional modern user interface
- A lovely coding experience with elements of PHP and assembly (includes 20 general purpose variables, a stack, real flow control, etc)
- Installers have their own VMs that let you write code that can support:
 - ◆ File extraction (with configurable overwrite parameters)
 - ◆ File/directory copying, renaming, deletion
 - ◆ DLL loading (ActiveX control registration/deregistration, extension DLL calling, etc)
 - ◆ Executable execution (shell execute and wait options)
 - ◆ Shortcut creation
 - ◆ Registry key reading/setting/enumerating/deleting
 - ◆ INI file reading/writing
 - ◆ Generic text file reading/writing
 - ◆ Directory scanning
 - ◆ Powerful string and integer manipulation
 - ◆ Window finding based on class name or title (for is-application-running detection)
 - ◆ Window message sending.
 - ◆ User interaction with MessageBox.
 - ◆ Branching, comparisons, etc.
 - ◆ Error checking.
 - ◆ Reboot support, including delete or rename on reboot
 - ◆ Installer behaviour commands (such as show/hide/wait/etc)
 - ◆ User functions in script
 - ◆ Callback functions that let you customize the way the installer behaves from script.
 - ◆ Macros in script
- Completely free for any use. See license.
- More

2. Tutorial: The Basics

2.1 Introduction

Most software packages you download or buy come with an installer. The installer copies and/or updates files, writes registry keys, writes configuration, creates shortcuts, etc. All of this is done automatically for the user. All the user needs to do is supply some information and the installer will do the rest. The user goes through a wizard, makes the appropriate choices and waits until the installer finishes. After the installer has finished the user is left only with the simple task of starting the program. The user doesn't have to worry about things he might have forgotten because all of the necessary steps were done by the installer.

NSIS is a tool for developers to create such installers. NSIS allows you to create everything from basic installers that just copy files to very complex installers that handle a lot of advanced tasks such as writing registry keys, settings environment variables, downloading the latest files from the internet, customizing the configuration file and more. NSIS is very flexible and its scripting language is easy to learn.

NSIS compiles all of the files and the installation script into one executable file, so your application will be easy to distribute. NSIS adds only about 34KB of code of its own (for the default configuration) to the data. NSIS has the smallest overhead available and still have a lot of options thanks to its powerful scripting language and support of external plug-ins.

2.2 Script Files

To create a NSIS installer, you first have to write a NSIS script. A NSIS script is just a regular text file with a special syntax. You can edit scripts with every text editor. It's recommended you use a text editor that shows line numbers because NSIS uses line numbers to indicate where errors lay, and to warn you about where errors might lay. An editor that supports syntax highlighting is also recommended. You can download editors made especially for NSIS and files for syntax highlighting at the [NSIS Archive](#).

In a NSIS script every line is treated as a command. If your command is too long for one line you can use a back-slash - '\' - at the end of the line. The compiler will treat the new line as an addition to the previous line and will not expect a new command. For example:

```
MessageBox MB_OK|MB_ICONINFORMATION \  
    "This sample shows how to use line breaks for larger commands in NSIS scripts"
```

If you want to use a double-quote in a string you can either use \" to escape the quote or quote the string with a different type of quote such as ` or '.

For more details about the script format, see section 4.1 [Script File Format](#)

The default extension for a script file is .nsi. Header files have the .nsh extension. Header files can help you arrange your script by dividing it to more than one block of code, you can also put functions or macros in header files and include the header files in multiple installers. This makes updating easier and it also makes your scripts easier to

read. To include a header file in your script use `!include`. Header files that reside in the Include directory under your NSIS directory can be included just by their name. For example:

```
!include Sections.nsh
```

2.3 Scripting structure

A NSIS script can contain Installer Attributes and Sections/Functions. You can also use Compiler Commands for compile-time operations. The minimum is "OutFile", which tells NSIS where to write the installer, and one section.

2.3.1 Installer Attributes

Installer Attributes determine the behavior and the look and feel of your installer. With these attributes you can define what pages are shown in which order, texts that will be shown during the installation, the number of installation types etc. Most of these commands can only be set and are not changeable during runtime.

The most basic attributes are "Name", "InstallDir" and "DirText".

For more information about installer attributes, have a look at Section x "Installer Attributes".

2.3.2 Sections

In a common installer there are several things the user can install. For example in the NSIS distribution installer you can choose to install the source code, additional plug-ins, examples and more. Each of these components has its own piece of code. If the user selects to install this component, then the installer will execute that code. In the script, that code is in sections. Each visible section is a component for the user to choose from. We will not discuss invisible sections in this tutorial. It is possible to build your installer with only one section, but if you want to use the components page and let the user choose what to install you'll have to use more than one section.

The instructions that can be used in sections are very different from the installer attributes instructions, they are executed at runtime on the user's computer. Those instructions can extract files, read from and write to the registry, INI files or normal files, create directories, create shortcuts and a lot more. You can find out about those instructions in Section 4.2 [Instructions](#).

The most basic instructions are "SetOutPath" which tells the installer where to extract files and "File" which extracts files.

Example:

```
Section "My Program"
  SetOutPath $INSTDIR
  File "My Program.exe"
  File "Readme.txt"
SectionEnd
```

For more information about sections see Section x "Sections".

2.3.3 Functions

Functions, just like sections, can contain code. The difference between sections and functions is the way they are called. There are two types of functions, user functions and callback functions.

User functions are called by the user from within sections or other functions using the "Call" instruction. User functions will not execute unless you call them. After the code of the function will be executed the installer will continue executing the instructions that came after the Call instruction, unless you have aborted the installation inside the function. User functions are very useful if you have a set of instructions that need to be executed at several locations in the installers. If you put the code into a function you can save the copying time and you can maintain the code more easily.

Callback functions are called by the installer upon certain defined events such as when the installer starts. Callbacks are optional. If for example you want to welcome the user to your installer you will define a function called .onInit. The NSIS compiler will recognize this function as a callback function by the name and will call it when the installer starts.

```
Function .onInit
    MessageBox MB_YESNO "This will install My Program. Do you wish to continue?" \
        IDYES gogogo
    Abort
gogogo:
FunctionEnd
```

"Abort" has a special meaning in callback functions. Each callback function has its own meaning for it, have a look at "Callback Functions" for more information. In the above example Abort tells the installer to stop initializing the installer and quit immediately.

For more information about sections see Section x "Functions".

2.3.4 Working with Scripts

2.3.4.1 Variables

NSIS offers 20 Variables that can be used in any Section or Function. In addition there is a Stack, which can also be used for temporary storage. To access the stack use the commands "Push" and "Pop". Push adds a value to the stack, Pop removes one and sets the variable. For example, assume the value of \$0 is 123.

Now you are going to call the function bla:

```
Function bla
    Push $0    ;123 added to the stack
    ...the function can use $0...
    Pop $0     ;Remove 123 from the stack and set $0 back to 123
FunctionEnd
```

After calling the function, the variables contain the same value as before. Note the order when using multiple variables (last-in first-out):

```
Function bla
    Push $0
    Push $1

    ...code...

    Pop $1
    Pop $0
FunctionEnd
```

2.3.4.2 Debugging Scripts

The more you work with NSIS the more complex the scripts will become. This will increase the potential of mistakes, especially when dealing with lots of variables. There are a few possibilities to help you debugging the code. To display the contents of variables you should use "MessageBoxes" or "DetailPrint". To get a brief overview about all variables you should use the plugin [Dumpstate](#). By default all actions of the Installer are printed out in the Log Window. You can access the log if you right-click in the Log Window and select "Copy Details To Clipboard". There is also a way to write it directly to a file. See "Dump Content of Log Window". Very useful if you force to close the installer automatically using "SetAutoClose".

2.3.5 Compiler Commands

Compiler commands will be executed on compile time on your computer. They can be used for conditional compilation, to include header files, to execute applications, to change the working directory and more. The most common usage is **defines**. Defines are compile time constants. You can define your product's version number and use it in your script. For example:

```
!define VERSION "1.0.3"
Name "My Program ${VERSION}"
OutFile "My Program Installer - ${VERSION}.exe"
```

For more information about defines see "Conditional Compilation".

Another common use is macros. Macros are used to insert code on compile time, depending on defines and using the values of the defines. An example of a macro is [UpgradeDLL](#), which you can use to upgrade a DLL file. The macro's commands are inserts at compile time. This allows you to write a general code only once and use it a lot of times but with a few changes. For example:

```
!macro MyFunc UN
    Function ${UN}MyFunc
        Call ${UN}DoRegStuff
        ReadRegStr $0 HKLM Software\MyProgram key
        DetailPrint $0
    FunctionEnd
!macroend
```

```
!insertmacro MyFunc ""
!insertmacro MyFunc "un."
```

This macro helps you avoid writing the same code for both the installer and the uninstaller. The two **!insertmacros** insert two functions, one for the installer called **MyFunc** and one for the uninstaller called **un.MyFunc** and both do exactly the same thing.

For more information see Chapter x "Compile Time Commands".

2.4 Compiler

The second thing you need to do in order to create your installer after you have created your script is to compile your script. MakeNSIS.exe is the NSIS compiler. It reads your script, parses it and creates an installer for you.

To compile you have to right-click your .nsi file and select Compile NSI or Compile NSIS (with bz2). This will cause MakeNSISw to launch and call MakeNSIS to compile your script. MakeNSISw will get the output of MakeNSIS and present it to you in a window where you can see it, copy it, test the installer, browse for it and more. If you don't have the Compile NSI entry in the context (right-click) menu you have probably unselected Shell Extensions in the NSIS installer. You can either use MakeNSIS.exe from a command prompt window (DOS) or reinstall NSIS with Shell Extensions selected.

The compiler will check your script and give you warnings or an error. If an errors occurs (e.g. 2 parameters required but only 1 given) the compiler will abort and a short error message including the line number will be displayed. For non-critical error the compiler will give a warning (e.g. two **DirText** commands in one script). If your script has no errors the compiler will output an installer for you to distribute.

NSIS supports different compression methods zlib and bzip2. zlib is fast and is very efficient in resources consumption. bzip2 usually gives better results for large installers, but requires a bit more memory and is a little slower. To set the compressor use "SetCompressor".

2.5 Enhancing NSIS

NSIS scripts support plug-in calls. Plug-ins are DLL files written in C, C++, Delphi or another programming language. Plug-ins provide a more powerful code base to NSIS as they can contain anything from a code that calculates 1 + 1 to a code that communicates with another computer through FireWire.

To call a plug-in function in your script you need to add a line as follows:

```
DLLName::FunctionName "parameter number 1" "parameter number 2" "parameter number 3"
```

Every plug-in's function has its own requirements when it comes to parameters, some will require none, some will accept as many parameters as you want to send. For example:

```
nsExec::ExecToLog "${NSISDIR}\makensis.exe" /CMDHELP'
InstallOptions::dialog "$PLUGINSDIR\test.ini"
NSISdl::download http://download.nullsoft.com/winamp/client/winamp281_lite.exe $2
```

NSIS User Manual

The plug-ins that NSIS knows of are listed at the top of the output of the compiler. NSIS searches for plug-ins in the Plugins directory under your NSIS directory and lists all of their available functions. You can use `"!addplugindir"` to tell NSIS to search in other directories too.

There are several plug-ins that come with the NSIS distribution. InstallOptions is a popular plug-in that allows you to add custom pages to your installer, in combination with the NSIS Page commands (See Section x "Pages"). The StartMenu plug-in provides a page that allows the user to choose a Start Menu folder. There are a lot of plug-ins for different purposes, have a look at the Contrib directory for help files and examples. You can find additional plug-ins at the on-line NSIS Archive.

You can also create a plug-in of your own if you know programming. ExDLL is the basic plug-in example. As all of the plug-ins packed with NSIS and most of the plug-ins in the archive come with source code you can have a look at the Contrib directory and the on-line NSIS Archive for more examples.

You can also customize the dialog resources without modifying or recompiling the source code. Use a resource editor to customize one of the UI files and use the "ChangeUI" command to use the customized resources.

A popular user interface for NSIS is the **Modern User Interface**, with an interface like the wizards of recent Windows versions. The Modern UI is not only a customized resource file, it has a lots of new interface elements. It features a white header to describe the current step, a description area on the component page, a Finish page that allows you to run the application or reboot the system and more. The Modern UI language files make it easy to create a multilingual installer, because they contain translations for every label in the installer.

The **Modern UI** has a macro system that inserts the code to handle all the new UI elements, so you only have to insert a few lines of code to use it. For more information, check the Modern UI Readme and the Modern UI Examples.

3. MakeNSIS Usage

NSIS installers are generated by using the 'MakeNSIS' program to compile a NSIS script (.NSI) into an installer executable. The NSIS development kit installer sets up your computer so that you can compile a .nsi file by simply right-clicking on it in explorer, and selecting 'compile'.

If you want to use MakeNSIS on the command line, the syntax of the makensis command is:

```
Makensis [/Vx] [/Olog] [/LICENSE] [/PAUSE] [/NOCONFIG] [/CMDHELP [command]]
          [/HDRINFO] [/NOCD] [/Ddefine[=value] ...] ["/XCommand parameter" ...]
          [Script.nsi | - [...]]
```

- **/LICENSE** displays a keen license page.
- The **/V** switch followed by a number between 0 and 4 will set the verbosity of output accordingly.
 - ◆ 0 = no output
 - ◆ 1 = errors only
 - ◆ 2 = warnings and errors
 - ◆ 3 = info, warnings, and errors
 - ◆ 4 = all output
- The **/O** switch followed by a filename tells the compiler to print its log to that file (instead of the screen)
- **/PAUSE** makes Makensis pause before quitting, which is useful when executing directly from Windows (the auto-installed shell extensions use it).
- **/NOCONFIG** disables inclusion of [path to makensis.exe]\nsisconf.nsh. Without this parameter, installer defaults are set from nsisconf.nsi. See [NSIS Configuration File](#).
- **/CMDHELP** prints basic usage information for command (if specified), or all commands (if command is not specified).
- **/HDRINFO** prints out information on what options Makensis was compiled with.
- **/NOCD** disabled the current directory change to that of the .nsi file
- Using the **/D** switch one or more times will add to symbols to the globally defined list (See "!define".)
- Using the **/X** switch one or more times will execute the code you specify following it. Example:
"/XAutoCloseWindow false"
- Specifying a dash (-) for the script name will tell Makensis to use the standard input as a source.
- If multiple scripts are specified, they are treated as one concatenated script.

4. Scripting Reference

- 4.1 Script File Format
- 4.2 Instructions
 - General Purpose Instructions
 - 4.2.1 CallInstDLL
 - 4.2.2 CopyFiles
 - 4.2.3 CreateDirectory
 - 4.2.4 CreateShortCut
 - 4.2.5 GetDLLVersion
 - 4.2.6 GetDLLVersionLocal
 - 4.2.7 GetFileTime
 - 4.2.8 GetFileTimeLocal
 - 4.2.9 GetFullPathName
 - 4.2.10 GetTempFileName
 - 4.2.11 SearchPath
 - 4.2.12 SetFileAttributes
 - 4.2.13 RegDLL
 - 4.2.14 UnRegDLL
 - Miscellaneous Instructions
 - 4.2.15 InitPluginsDir
 - 4.2.16 SetShellVarContext
 - 4.2.17 Sleep
- 4.3 Plugin DLLs
 - 4.3.1 Using Plugin Commands
 - 4.3.2 Disabling Plugin Unloading

4.1 Script File Format

A NSIS Script File (.nsi) is just a text file with a series of commands.

- Lines beginning with ; or # are comments.
- Non-comment lines are in the form of '[command parameters]'
- To call a plugin, use 'plugin::command [command parameters]'. For more info see Section 4.3 Plugin DLLs.
- Anything after a ; or # that is not in a parameter (i.e. in quotes or part of another string) is treated as a comment. (e.g. "File myfile ; this is the file" would work)
- For parameters that are treated as numbers, use decimal (the number) or hexadecimal (with 0x prepended to it, e.g. 0x12345AB), or octal (numbers beginning with a 0 and no x, e.g. 0377).
- To represent strings that have spaces, use quotes.
- Quotes only have the property of containing a parameter if they begin the parameter.
- Quotes can be either single quotes, double quotes, or the backward single quote.
- You can escape quotes using \$\.
- Examples:

```
MessageBox MB_OK "I'll be happy"
; this one puts a ' inside a string
```

```
MessageBox MB_OK 'And he said to me "Hi there!'"
; this one puts a " inside a string
```

```
MessageBox MB_OK `And he said to me "I'll be *****!"`  
; this one puts both ' and "s inside a string
```

```
MessageBox MB_OK "$\"A quote from a wise man$\" said the wise man"  
; this one shows escaping of quotes
```

- To extend a command over multiple lines, use a backslash (\) at the end of the line, and the next line will effectively be concatenated the end of it. For example:

```
CreateShortCut "$SMPROGRAMS\NSIS\ZIP2EXE project workspace.lnk" \  
"$INSTDIR\source\zip2exe\zip2exe.dsw"
```

```
MessageBox MB_YESNO|MB_ICONQUESTION \  
"Remove all files in your NSIS directory? (If you have anything \  
you created that you want to keep, click No)" \  
IDNO NoRemoveLabel
```

- If a file named "nsisconf.nsh" in the same directory as makensis.exe exists, it will be included by default before any scripts (unless the /NOCONFIG command line parameter is used).

4.2 Instructions

General Purpose Instructions

4.2.1 CallInstDLL

```
CallInstDLL dllfile [/NOUNLOAD] function_name
```

Calls a function_name inside a NSIS extension DLL. See Contrib\ExDLL for an example of how to make one. Extension DLLs can access the stack and variables. Use /NOUNLOAD to force the installer to leave the DLL loaded.

4.2.2 CopyFiles

```
CopyFiles [/SILENT] [/FILESONLY] filespec_on_destsys destination_path \  
[size_of_files_in_kb]
```

Copies files from the source to the destination on the installing system. Useful with \$EXEDIR if you want to copy from installation media, or to copy from one place to another on the system. Uses SHFileOperation, so the user might see a status window of the copy operation if it is large (to disable this, use /SILENT). The last parameter specifies how big the copy is (in kilobytes), so that the installer can approximate the disk space requirements. On error, or if the user cancels the copy (only possible when /SILENT was omitted), the error flag is set. If /FILESONLY is specified, only files are copied.

4.2.3 CreateDirectory

```
CreateDirectory path_to_create
```

Creates (recursively if necessary) the specified directory. The error flag is set if the directory couldn't be created.

4.2.4 CreateShortcut

```
CreateShortcut link.lnk target.file [parameters \
    [icon.file [icon_index_number \
    [start_options [keyboard_shortcut [description]]]]]]
```

Creates a shortcut 'link.lnk' that links to 'target.file', with optional parameters 'parameters'. The icon used for the shortcut is 'icon.file,icon_index_number'; for default icon settings use empty strings for both icon.file and icon_index_number. start_options should be one of: *SW_SHOWNORMAL*, *SW_SHOWMAXIMIZED*, *SW_SHOWMINIMIZED*, or an empty string. keyboard_shortcut should be in the form of 'flag|c' where flag can be a combination (using |) of: *ALT*, *CONTROL*, *EXT*, or *SHIFT*. c is the character to use (a-z, A-Z, 0-9, F1-F24, etc). Note that no spaces are allowed in this string. A good example is "ALT|CONTROL|F8". \$OUTDIR is used for the working directory. You can change it by using "SetOutPath" before creating the Shortcut. description should be the description of the shortcut, or comment as it is called under XP. The error flag is set if the shortcut cannot be created (i.e. the path does not exist, or some other error).

4.2.5 GetDLLVersion

```
GetDLLVersion filename user_var(high dword output) \
    user_var(low dword output)
```

Gets the version information from the DLL (or any other executable containing version information) in "filename". Sets the user output variables with the high and low dwords of version information on success; on failure the outputs are empty and the error flag is set. The following example reads the DLL version and copies a human readable version of it into \$0:

```
GetDllVersion "$INSTDIR\MyDLL.dll" $R0 $R1
IntOp $R2 $R0 / 0x00010000
IntOp $R3 $R0 & 0x0000FFFF
IntOp $R4 $R1 / 0x00010000
IntOp $R5 $R1 & 0x0000FFFF
StrCpy $0 "$R2.$R3.$R4.$R5"
```

4.2.6 GetDLLVersionLocal

```
GetDLLVersionLocal localfilename user_var(high dword output) \
    user_var(low dword output)
```

This is similar to GetDLLVersion, only it acts on the system building the installer (it actually compiles into two

StrCpy commands). Sets the two output variables with the DLL version information of the DLL on the build system.

4.2.7 GetFileTime

```
GetFileTime filename user_var(high dword output) \
               user_var(low dword output)
```

Gets the last write time of "filename". Sets the user output variables with the high and low dwords of the timestamp on success; on failure the outputs are empty and the error flag is set.

4.2.8 GetFileTimeLocal

```
GetFileTimeLocal localfilename user_var(high dword output) \
                      user_var(low dword output)
```

This is similar to GetFileTime, only it acts on the system building the installer (it actually compiles into two StrCpy commands). Sets the two output variables with the file timestamp of the file on the build system.

4.2.9 GetFullPathName

```
GetFullPathName [/SHORT] user_var(output) path_or_file
```

Assign to the user variable \$x, the full path of the file specified. If the path portion of the parameter is not found, the error flag will be set and \$x will be empty. If /SHORT is specified, the path is converted to the short filename form.

4.2.10 GetTempFileName

```
GetTempFileName user_var(output)
```

Assign to the user variable \$x, the name of a temporary file. The file will have been created, so you can then overwrite it with what you please. The name of the temporary file is guaranteed to be unique. Delete the file when done with it.

4.2.11 SearchPath

```
SearchPath user_var(output) filename
```

Assign to the user variable \$x, the full path of the file named by the second parameter. The error flag will be set and \$x will be empty if the file cannot be found. Uses SearchPath() to search the system paths for the file.

4.2.12 SetFileAttributes

```
SetFileAttributes filename attribute1|attribute2|...
```

Sets the file attributes of 'filename'. Valid attributes can be combined with | and are:

- *NORMAL* or *FILE_ATTRIBUTE_NORMAL* (you can use 0 to abbreviate this)
- *ARCHIVE* or *FILE_ATTRIBUTE_ARCHIVE*
- *HIDDEN* or *FILE_ATTRIBUTE_HIDDEN*
- *OFFLINE* or *FILE_ATTRIBUTE_OFFLINE*
- *READONLY* or *FILE_ATTRIBUTE_READONLY*
- *SYSTEM* or *FILE_ATTRIBUTE_SYSTEM*
- *TEMPORARY* or *FILE_ATTRIBUTE_TEMPORARY*

The error flag will be set if the file's attributes cannot be set (i.e. the file doesn't exist, or you don't have the right permissions). You can only set attributes. It's not possible to unset them. If you want to remove an attribute use *NORMAL*. This way all attributes are erased. This command doesn't support wildcards.

4.2.13 RegDLL

```
RegDLL dllfile [entrypoint_name]
```

Loads the specified DLL and calls DllRegisterServer (or entrypoint_name if specified). The error flag is set if an error occurs (i.e. it can't load the DLL, initialize OLE, find the entry point, or the function returned anything other than *ERROR_SUCCESS* (=0)).

4.2.14 UnRegDLL

```
UnRegDLL dllfile
```

Loads the specified DLL and calls DllUnregisterServer. The error flag is set if an error occurs (i.e. it can't load the DLL, initialize OLE, find the entry point, or the function returned anything other than *ERROR_SUCCESS* (=0)).

Miscellaneous Instructions

4.2.15 InitPluginsDir

```
InitPluginsDir
```

Initializes the plugins dir (\$PLUGINSDIR) if not already initialized.

4.2.16 SetShellVarContext

```
SetShellVarContext current|all
```

Sets the context of \$SMPROGRAMS and other shell folders. If set to 'current' (the default), the current user's shell folders are used. If set to 'all', the 'all users' shell folder is used. The all users folder may not be supported on all OSes. If the all users folder is not found, the current user folder will be used. Please take into consideration that a "normal user" has no rights to write in the all users area. Only admins have full access rights to the all users area. You can check this by using the UserInfo Plugin. See Contrib\UserInfo\UserInfo.nsi for an example.

4.2.17 Sleep

```
Sleep sleeptime_in_ms
```

Pauses execution in the installer for sleeptime_in_ms milliseconds. sleeptime_in_ms can be a variable, e.g. "\$0" or a number, e.g. "666".

4.3 Plugin DLLs

The abilities of the NSIS scripting language can be extended by utilising functionality provided in a DLL file. Probably the best known example of this is the InstallOptions.dll bundled with every NSIS release.

When the NSIS compiler starts it scans the plugins directory for DLLs and makes a list of the plugins found and their exported functions. During compilation if a sequence such as fred::flintstone is encountered where the compiler expected to find a language keyword the compiler will look through this list. If a list entry specifies that fred.dll exports function flintstone NSIS will pack the fred.dll file into the created installer binary.

During execution of the created installer if a plugin command is executed NSIS will unpack the necessary DLL to the \$TEMP directory, push all of the arguments specified (right-to-left order), and then execute the DLL function. If the /NOUNLOAD option is specified the DLL will not be unloaded until the installer exits or the next time you use the DLL without /NOUNLOAD. Please note that the last call to the plugin must not have the /NOUNLOAD flag or the plugin will not be deleted from \$PLUGINSDIR, thus garbage will be left on the user's machine.

4.3.1 Using Plugin Commands

The following two examples both invoke the same plugin command. The first example shows the (still okay) syntax that scripts written for versions of NSIS earlier than 2.0a4 had to use, and the second is how it can be scripted more succinctly now. The newer syntax automatically handles packing & extraction of the DLL file, and stacks up arguments for you too.

```
# Pre 2.0a4 syntax
SetOutPath $TEMP
GetTempFileName $8
File /oname=$8 InstallOptions.dll
Push "ini_file_location.ini"
```

```
CallInstDLL dialog
```

```
#Newer syntax  
InstallOptions::dialog "ini_file_location.ini"
```

InstallOptions needs the name of its ini file as a parameter to the dialog function so it has to be pushed onto the stack before the dialog call is made. Some plugin commands may not need any parameters on the stack, others might require two or three. To use a plugin command you will need to read the documentation for the plugin so that you know what parameters its functions require, if any.

4.3.2 Disabling Plugin Unloading

CallInstDLL has an option not to unload the DLL after usage. To use it with the newer plugin command syntax just specify the first parameter as /NOUNLOAD. For example:

```
InstallOptions::dialog /NOUNLOAD "ini_file_location.ini"
```

You can also use SetPluginUnload alwaysoff to avoid writing /NOUNLOAD each and every time you use the same plugin.

B. Useful Functions

1 Get parent directory

```
; GetParent
; input, top of stack (e.g. C:\Program Files\Poop)
; output, top of stack (replaces, with e.g. C:\Program Files)
; modifies no other variables.
;
; Usage:
;   Push "C:\Program Files\Directory\Whatever"
;   Call GetParent
;   Pop $R0
;   ; at this point $R0 will equal "C:\Program Files\Directory"
```

```
Function GetParent
  Exch $R0 ; old $R0 is on top of stack
  Push $R1
  Push $R2
  StrCpy $R1 -1

loop:
  StrCpy $R2 $R0 1 $R1
  StrCmp $R2 "" exit
  StrCmp $R2 "\" exit
  IntOp $R1 $R1 - 1
  Goto loop

exit:
  StrCpy $R0 $R0 $R1
  Pop $R2
  Pop $R1
  Exch $R0 ; put $R0 on top of stack, restore $R0 to original value
FunctionEnd
```

2 Trim newlines

```
; TrimNewlines
; input, top of stack (e.g. whatever$\r$\n)
; output, top of stack (replaces, with e.g. whatever)
; modifies no other variables.
```

```
Function TrimNewlines
  Exch $R0
  Push $R1
  Push $R2
  StrCpy $R1 0

loop:
  IntOp $R1 $R1 - 1
```

```

StrCpy $R2 $R0 1 $R1
StrCmp $R2 "$\r" loop
StrCmp $R2 "$\n" loop
IntOp $R1 $R1 + 1
IntCmp $R1 0 no_trim_needed
StrCpy $R0 $R0 $R1

no_trim_needed:
    Pop $R2
    Pop $R1
    Exch $R0
FunctionEnd

```

3 Get command-line parameters

```

; GetParameters
; input, none
; output, top of stack (replaces, with e.g. whatever)
; modifies no other variables.

Function GetParameters
    Push $R0
    Push $R1
    Push $R2

    StrCpy $R0 $CMDLINE 1
    StrCpy $R1 ""
    StrCpy $R2 1
    StrCmp $R0 "" loop
    StrCpy $R1 ' ' ; we're scanning for a space instead of a quote

loop:
    StrCpy $R0 $CMDLINE 1 $R2
    StrCmp $R0 $R1 loop2
    StrCmp $R0 "" loop2
    IntOp $R2 $R2 + 1
    Goto loop

loop2:
    IntOp $R2 $R2 + 1
    StrCpy $R0 $CMDLINE 1 $R2
    StrCmp $R0 " " loop2
    StrCpy $R0 $CMDLINE "" $R2

    Pop $R2
    Pop $R1
    Exch $R0
FunctionEnd

```

4 Search in a string

```

; StrStr
; input, top of stack = string to search for
;         top of stack-1 = string to search in
; output, top of stack (replaces with the portion of the string remaining)
; modifies no other variables.

```



```

;
; Usage:
;   Push "this is a long ass string"
;   Push "ass"
;   Call StrStr
;   Pop $R0
;   ($R0 at this point is "ass string")

Function StrStr
  Exch $R1    ; st=haystack,old$R1, $R1=needle
  Exch       ; st=old$R1,haystack
  Exch $R2    ; st=old$R1,old$R2, $R2=haystack
  Push $R3
  Push $R4
  Push $R5

  StrLen $R3 $R1
  StrCpy $R4 0

  ; $R1=needle
  ; $R2=haystack
  ; $R3=len(needle)
  ; $R4=cnt
  ; $R5=tmp

loop:
  StrCpy $R5 $R2 $R3 $R4
  StrCmp $R5 $R1 done
  StrCmp $R5 "" done
  IntOp $R4 $R4 + 1
  Goto loop

done:
  StrCpy $R1 $R2 "" $R4

  Pop $R5
  Pop $R4
  Pop $R3
  Pop $R2
  Exch $R1
FunctionEnd

```

5 Get Windows version

```

; GetWindowsVersion
;
; Based on Yazno's function, http://yazno.tripod.com/powerpimpit/
; Updated by Joost Verburg
;
; Returns on top of stack
;
; Windows Version (95, 98, ME, NT x.x, 2000, XP, 2003)
; or
; '' (Unknown Windows Version)
;
; Usage:
;   Call GetWindowsVersion
;   Pop $R0
;   ; at this point $R0 is "NT 4.0" or whatnot

```

Function GetWindowsVersion

```

Push $R0
Push $R1

ReadRegStr $R0 HKLM \
"SOFTWARE\Microsoft\Windows NT\CurrentVersion" CurrentVersion

IfErrors 0 lbl_winnt

; we are not NT
ReadRegStr $R0 HKLM \
"SOFTWARE\Microsoft\Windows\CurrentVersion" VersionNumber

StrCpy $R1 $R0 1
StrCmp $R1 '4' 0 lbl_error

StrCpy $R1 $R0 3

StrCmp $R1 '4.0' lbl_win32_95
StrCmp $R1 '4.9' lbl_win32_ME lbl_win32_98

lbl_win32_95:
    StrCpy $R0 '95'
    Goto lbl_done

lbl_win32_98:
    StrCpy $R0 '98'
    Goto lbl_done

lbl_win32_ME:
    StrCpy $R0 'ME'
    Goto lbl_done

lbl_winnt:

StrCpy $R1 $R0 1

StrCmp $R1 '3' lbl_winnt_x
StrCmp $R1 '4' lbl_winnt_x

StrCpy $R1 $R0 3

StrCmp $R1 '5.0' lbl_winnt_2000
StrCmp $R1 '5.1' lbl_winnt_XP
StrCmp $R1 '5.2' lbl_winnt_2003 lbl_error

lbl_winnt_x:
    StrCpy $R0 "NT $R0" 6
    Goto lbl_done

lbl_winnt_2000:
    Strcpy $R0 '2000'
    Goto lbl_done

lbl_winnt_XP:
    Strcpy $R0 'XP'
    Goto lbl_done

lbl_winnt_2003:

```

```

    Strcpy $R0 '2003'
    Goto lbl_done

    lbl_error:
        Strcpy $R0 ''
    lbl_done:

    Pop $R1
    Exch $R0

FunctionEnd

```

6 Get Internet Explorer version

```

; GetIEVersion
;
; Based on Yazno's function, http://yazno.tripod.com/powerpimpit/
; Returns on top of stack
; 1-6 (Installed IE Version)
; or
; '' (IE is not installed)
;
; Usage:
;   Call GetIEVersion
;   Pop $R0
;   ; at this point $R0 is "5" or whatnot

Function GetIEVersion
    Push $R0
    ClearErrors
    ReadRegStr $R0 HKLM "Software\Microsoft\Internet Explorer" "Version"
    IfErrors lbl_123 lbl_456

lbl_456:                ; ie 4+
    Strcpy $R0 $R0 1
    Goto lbl_done

lbl_123:                ; older ie version
    ClearErrors
    ReadRegStr $R0 HKLM "Software\Microsoft\Internet Explorer" "IVer"
    IfErrors lbl_error

    StrCpy $R0 $R0 3
    StrCmp $R0 '100' lbl_ie1
    StrCmp $R0 '101' lbl_ie2
    StrCmp $R0 '102' lbl_ie2

    StrCpy $R0 '3'      ; default to ie3 if not 100, 101, or 102.
    Goto lbl_done

lbl_ie1:
    StrCpy $R0 '1'
    Goto lbl_done

lbl_ie2:
    StrCpy $R0 '2'
    Goto lbl_done

lbl_error:

```

```

StrCpy $R0 ''
lbl_done:
  Exch $R0
FunctionEnd

```

7 Is .NET Framework installed?

```

; IsDotNETInstalled
;
; Usage:
;   Call IsDotNETInstalled
;   Pop $0
;   StrCmp $0 1 found.NETFramework no.NETFramework

Function IsDotNETInstalled
  Push $0
  Push $1
  Push $2
  Push $3
  Push $4

  ReadRegStr $4 HKEY_LOCAL_MACHINE \
    "Software\Microsoft\.NETFramework" "InstallRoot"
  # remove trailing back slash
  Push $4
  Exch $EXEDIR
  Exch $EXEDIR
  Pop $4
  # if the root directory doesn't exist .NET is not installed
  IfFileExists $4 0 noDotNET

  StrCpy $0 0

EnumStart:

  EnumRegKey $2 HKEY_LOCAL_MACHINE \
    "Software\Microsoft\.NETFramework\Policy" $0
  IntOp $0 $0 + 1
  StrCmp $2 "" noDotNET

  StrCpy $1 0

EnumPolicy:

  EnumRegValue $3 HKEY_LOCAL_MACHINE \
    "Software\Microsoft\.NETFramework\Policy\"$2" $1
  IntOp $1 $1 + 1
  StrCmp $3 "" EnumStart
  IfFileExists "$4\$2.$3" foundDotNET EnumPolicy

noDotNET:
  StrCpy $0 0
  Goto done

foundDotNET:
  StrCpy $0 1

done:

```

```

Pop $4
Pop $3
Pop $2
Pop $1
Exch $0
FunctionEnd

```

8 Is Flash Player installed?

```

; IsFlashInstalled
;
; By Yazno, http://yazno.tripod.com/powerpimpit/
; Returns on top of stack
; 0 (Flash is not installed)
; or
; 1 (Flash is installed)
;
; Usage:
;   Call IsFlashInstalled
;   Pop $R0
;   ; $R0 at this point is "1" or "0"

Function IsFlashInstalled
  Push $R0
  ClearErrors
  ReadRegStr $R0 HKCR "CLSID\{D27CDB6E-AE6D-11cf-96B8-444553540000}" ""
  IfErrors lbl_na

  StrCpy $R0 1
  Goto lbl_end

lbl_na:
  StrCpy $R0 0

lbl_end:
  Exch $R0
FunctionEnd

```

9 Add a shared DLL

```

; AddSharedDLL
;
; Increments a shared DLLs reference count.
; Use by passing one item on the stack (the full path of the DLL).
;
; Usage:
;   Push $SYSDIR\myDll.dll
;   Call AddSharedDLL
;

Function AddSharedDLL
  Exch $R1
  Push $R0
  ReadRegDword $R0 HKLM Software\Microsoft\Windows\CurrentVersion\SharedDLLs $R1
  IntOp $R0 $R0 + 1
  WriteRegDWORD HKLM Software\Microsoft\Windows\CurrentVersion\SharedDLLs $R1 $R0

```

```

    Pop $R0
    Pop $R1
FunctionEnd

```

10 Remove a shared DLL

```

; un.RemoveSharedDLL
;
; Decrements a shared DLLs reference count, and removes if necessary.
; Use by passing one item on the stack (the full path of the DLL).
; Note: for use in the main installer (not the uninstaller), rename the
; function to RemoveSharedDLL.
;
; Usage:
;   Push $SYSDIR\myDll.dll
;   Call un.RemoveSharedDLL
;
Function un.RemoveSharedDLL
    Exch $R1
    Push $R0
    ReadRegDword $R0 HKLM Software\Microsoft\Windows\CurrentVersion\SharedDLLs $R1
    StrCmp $R0 "" remove
    IntOp $R0 $R0 - 1
    IntCmp $R0 0 rk rk uk

rk:
    DeleteRegValue HKLM Software\Microsoft\Windows\CurrentVersion\SharedDLLs $R1
    goto Remove

uk:
    WriteRegDWORD HKLM Software\Microsoft\Windows\CurrentVersion\SharedDLLs $R1 $R0
    Goto noremove

remove:
    Delete /REBOOTOK $R1

noremove:
    Pop $R0
    Pop $R1
FunctionEnd

```

11 Upgrade a DLL (macro)

```

; Macro - Upgrade DLL File
; Written by Joost Verburg
; -----
;
; Example of usage:
; !insertmacro UpgradeDLL "dllname.dll" "$SYSDIR\dllname.dll"
;
; !define UPGRADEDLL_NOREGISTER if you want to upgrade a DLL which cannot
; be registered
;
; Note that this macro sets overwrite to ON (the default) when it has been
; inserted. If you are using another setting, set it again after inserting

```

; the macro.

```
!macro UpgradeDLL LOCALFILE DESTFILE
```

```
    Push $R0
```

```
    Push $R1
```

```
    Push $R2
```

```
    Push $R3
```

```
    ;-----
```

```
    ;Check file and version
```

```
    IfFileExists "${DESTFILE}" "" "copy_${LOCALFILE}"
```

```
    ClearErrors
```

```
    GetDLLVersionLocal "${LOCALFILE}" $R0 $R1
```

```
    GetDLLVersion "${DESTFILE}" $R2 $R3
```

```
    IfErrors "upgrade_${LOCALFILE}"
```

```
    IntCmpU $R0 $R2 "" "done_${LOCALFILE}" "upgrade_${LOCALFILE}"
```

```
    IntCmpU $R1 $R3 "done_${LOCALFILE}" "done_${LOCALFILE}" \
        "upgrade_${LOCALFILE}"
```

```
    ;-----
```

```
    ;Let's upgrade the DLL!
```

```
    SetOverwrite try
```

```
"upgrade_${LOCALFILE}:"
```

```
    !ifndef UPGRADEDLL_NOREGISTER
```

```
        ;Unregister the DLL
```

```
        UnRegDLL "${DESTFILE}"
```

```
    !endif
```

```
    ;-----
```

```
    ;Try to copy the DLL directly
```

```
    ClearErrors
```

```
    StrCpy $R0 "${DESTFILE}"
```

```
    Call ":file_${LOCALFILE}"
```

```
    IfErrors "" "noreboot_${LOCALFILE}"
```

```
    ;-----
```

```
    ;DLL is in use. Copy it to a temp file and Rename it on reboot.
```

```
    GetTempFileName $R0
```

```
    Call ":file_${LOCALFILE}"
```

```
    Rename /REBOOTOK $R0 "${DESTFILE}"
```

```
    ;-----
```

```
    ;Register the DLL on reboot
```

```
    !ifndef UPGRADEDLL_NOREGISTER
```

```
        WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\RunOnce" \
```

```
            "Register ${DESTFILE}" \
```

```
            "$SYSDIR\rundll32.exe" "${DESTFILE},DllRegisterServer"
```

```
    !endif
```

```
    Goto "done_${LOCALFILE}"
```

```

;-----
;DLL does not exist - just extract

"copy_${LOCALFILE}:"
  StrCpy $R0 "${DESTFILE}"
  Call ":file_${LOCALFILE}"

;-----
;Register the DLL

"noreboot_${LOCALFILE}:"
  !ifndef UPGRADEDLL_NOREGISTER
    RegDLL "${DESTFILE}"
  !endif

;-----
;Done

"done_${LOCALFILE}:"

  Pop $R3
  Pop $R2
  Pop $R1
  Pop $R0

;-----
;End

  Goto "end_${LOCALFILE}"

;-----
;Called to extract the DLL

"file_${LOCALFILE}:"
  File /oname=$R0 "${LOCALFILE}"
  Return

"end_${LOCALFILE}:"

;-----
;Set overwrite to default
;(was set to TRY above)

  SetOverwrite on

!macroend

```

12 Connect to the internet

```

; ConnectInternet (uses Dialer plugin)
; Written by Joost Verburg
;
; This function attempts to make a connection to the internet if there is no
; connection available. If you are not sure that a system using the installer
; has an active internet connection, call this function before downloading
; files with NSISdl.
;
; The function requires Internet Explorer 3, but asks to connect manually if
; IE3 is not installed.

```


Function ConnectInternet

```

    Push $R0

    ClearErrors
    Dialer::AttemptConnect
    IfErrors noie3

    Pop $R0
    StrCmp $R0 "online" connected
    MessageBox MB_OK|MB_ICONSTOP "Cannot connect to the internet."
    Quit ;Remove to make error not fatal

noie3:

    ; IE3 not installed
    MessageBox MB_OK|MB_ICONINFORMATION "Please connect to the internet now."

connected:

    Pop $R0

FunctionEnd

```

13 More

You can find more useful functions at [the NSIS Archive](#), [the NSIS forum](#) and [NSIS development page](#).

Index

[A](#), [C](#), [D](#), [E](#), [F](#), [G](#), [I](#), [M](#), [P](#), [R](#), [S](#), [T](#), [U](#), [V](#), [W](#)

A

[Add a shared DLL](#) (*Appendix B, Useful Functions*)

C

[CallInstDLL](#)

[Compiler Commands, Tutorial](#)

[Compiler, Tutorial](#)

[Connect to the internet](#) (*Appendix B, Useful Functions*)

[CopyFiles](#)

[CreateDirectory](#)

[CreateShortCut](#)

D

[Debugging Scripts](#) (*Working with Scripts, Tutorial*)

[Disabling Plugin Unloading](#)

E

[Enhancing NSIS, Tutorial](#)

F

[Functions, Tutorial](#)

G

[General Purpose Instructions](#)

[Get Internet Explorer version](#) (*Appendix B, Useful Functions*)

[Get Windows version](#) (*Appendix B, Useful Functions*)

[Get command-line parameter](#) (*Appendix B, Useful Functions*)

[Get parent directory](#) (*Appendix B, Useful Functions*)

[GetDLLVersion](#)

[GetDLLVersionLocal](#)

[GetFileTime](#)

[GetFileTimeLocal](#)

[GetFullPathName](#)

[GetTempFileName](#)

I

[InitPluginsDir](#)

[Instructions](#)

[Introduction to NSIS](#)

[Introduction, Tutorial](#)

[Is .NET Framework installed?](#) (*Appendix B, Useful Functions*)

[Is Flash Player installed?](#) (*Appendix B, Useful Functions*)

M

[Main Features](#)

[MakeNSIS Usage](#)

[Miscellaneous Instructions](#)

P

[Plugin DLLs](#)

R

[RegDLL](#)

[Remove a shared DLL](#) (*Appendix B, Useful Functions*)

S

[Script File Format](#)

[Script Files, Tutorial](#)

[Scripting Reference](#)

[Scripting structure, Tutorial](#)

[Search in a string](#) (*Appendix B, Useful Functions*)

[SearchPath](#)

[Sections, Tutorial](#)

[SetFileAttributes](#)

[SetShellVarContext](#)

[Sleep](#)

T

[Trim newlines](#) (*Appendix B, Useful Functions*)

[Tutorial: The Basics](#)

U

[UnRegDLL](#)

[Upgrade a DLL \(macro\)](#) (*Appendix B, Useful Functions*)

[Useful Functions, Appendix B](#)

[Using Plugin Commands](#)

V

[Variables](#) (*Working with Scripts, Tutorial*)

W

Working with Scripts, Tutorial

