Creating Application Installers for MAXScripts

Jeff Hanna
Technical Artist, Mythic Entertainment
http://www.jeff-hanna.com

The Current State of MAXScript Installation

What's Wrong With .MZP Files?

Having managed tool rollouts at both small (four artists) and large (50+ artists) companies I've become increasingly frustrated with MAXScript installation options. MZP files do ake the process less painful, but they have some inherent problems and limitations:

- **They don't provide any feedback to the user.**
  The mzp.run script that controls an .MZP file's installation can't trigger MAXScript dialogs natively. Unless the script author creates a post-installation script and codes a run line into the mzp.run file there won't be any feedback to the user. I tried such a method for a while but eventually gave up on it. It's cumbersome and can leave unused MAXScripts on the user's hard drive.

- **They require Max to be running.**
  This may not seem like a problem, but it goes against the way most users interact with their Windows based computers. Unless a program auto-updates itself a user isn't conditioned to start a program before they update it. The Windows user experience is usually exactly the opposite of that. A user starts an installer to update a program. If the program being updated is currently running then the installer will ask the user to close it before the update process can continue. The majority of support requests I've received regarding script installations stem from the fact that the user is treating the .MZP files as if it were an installer.

- **.MZP files are .zip files, just renamed.**
  This can be seen as a benefit and a problem. The problem happens when a user doesn't know how to properly install an .MZP file but figures out that they can be opened in a zip compression program. Invariably the user puts all of the extracted files someplace incorrect, like the root of c:\3dsmax, or worse, and then either can't get the script to work in Max or, in the case of scripts with extensions, gets errors when Max starts.

- **Due to a limitation in Max installed icons for toolbars will not show up until Max is restarted.**
  This issue has been around for about as long as MacroScripts have been in Max. It's not possible to force Max to refresh its icon cache. This means that any toolbar bitmap that is added after Max is started will not be visible until the user exits Max and restarts it. This negates whatever benefit Discreet might have seen in having users install scripts from within a running instance of Max. Since an mzp.run file can not display a dialog (as mentioned above) there isn't an easy way to tell the user that they're going to have to close and re-launch Max to truly finish the installation.

- **MZP files are poorly documented in the user's manual.**
  The index for the User Reference help file doesn't contain any entries for ".mzp". Doing a search in the help file for "installing a MAXScript" lists eleven topics. The first one is "Particle Flow FAQ". The first entry that is specifically MAXScript related is #5, "Running Scripts from the Command Line". There is no entry for installing MAXScripts. The help file entry for the MAXScript "Run Script" menu entry doesn't even mention that you should use that menu

command to install .MZP files. If a user goes to the help file to find information on how to install .MZP files they won't gain any useful knowledge.

- **MZP files aren't universally accepted.**
  Since the use of MZP files isn't enforced you can find many scripts that don't use them. ZIP files with (assumed) proper directory structures are used, ZIP files with no directory structure and a file on how to do a manual installation are used, self extracting ZIP files are used, etc… While using true application installers may be seen as just adding to the problem, they provide a lot of benefits and an easier and more powerful way of doing script installations.

Why Use An Installer?

Because of those problems with MZP files I have moved away from using .MZP files for script delivery of our in-house tools. I now build application installers. True application installers get around all of the problems I listed above:

- A good installer system will allow for the display of dialogs so that the user can receive information and/or make choices pertaining to the installation process.

- They follow the accepted Windows actions for installing or updating applications.

- The files can't easily be extracted from the archive and manually placed on the hard drive.

- Max doesn't have to be running so its icon cache will be updated the first time a user runs the program after installing a new tool. If Max is running during the installation the user can be alerted to the fact that they need to close and re-launch the application.

- Installer usage is something most people understand so there doesn't need to be support documentation on how to perform the installation.

There are other benefits to using an application installer:

- **The process can easily add or remove data from the registry and .INI files.**
  It has become standard practice for me to add entries into Max's plugin.ini file for every in-house script I create. Specifically I add new key values into the [help] section that point to documentation for the script. This way the user only has to go to Max's Help\Additional Help menu to find information on any script they have installed.

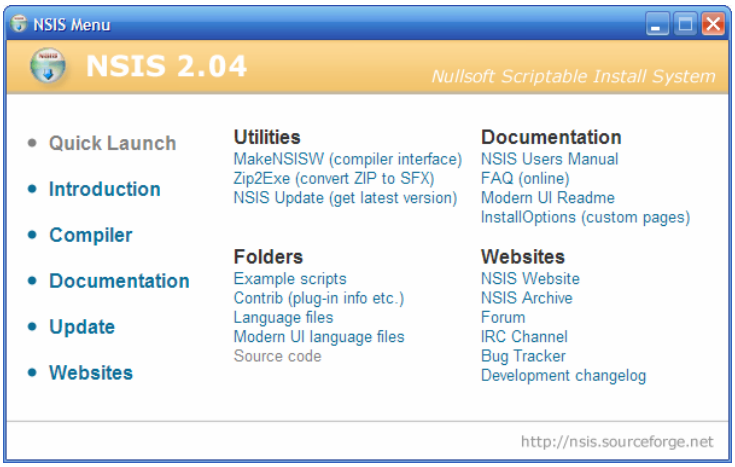
- **The installer can be branded with corporate logos.**
  This might not seem like much, but if you are releasing your scripts either professionally or for free any branding will help people remember you and the work you've done.

- **The installer logic allows for user input.**
  You can put many different scripts into one installer package and then present the user with an options dialog that lets them choose which scripts they

would like to install. Every quarter I take all of the scripts I've released internally over the previous three months and add those to our base art tool install package. The process is very quick and the resulting code is easy to maintain. By doing this new artists and IT personnel outfitting new machines don't have to hunt around for all of the necessary art tools. One installer contains all of the scripts, tools, plug-ins, etc… they will need to accomplish their job.

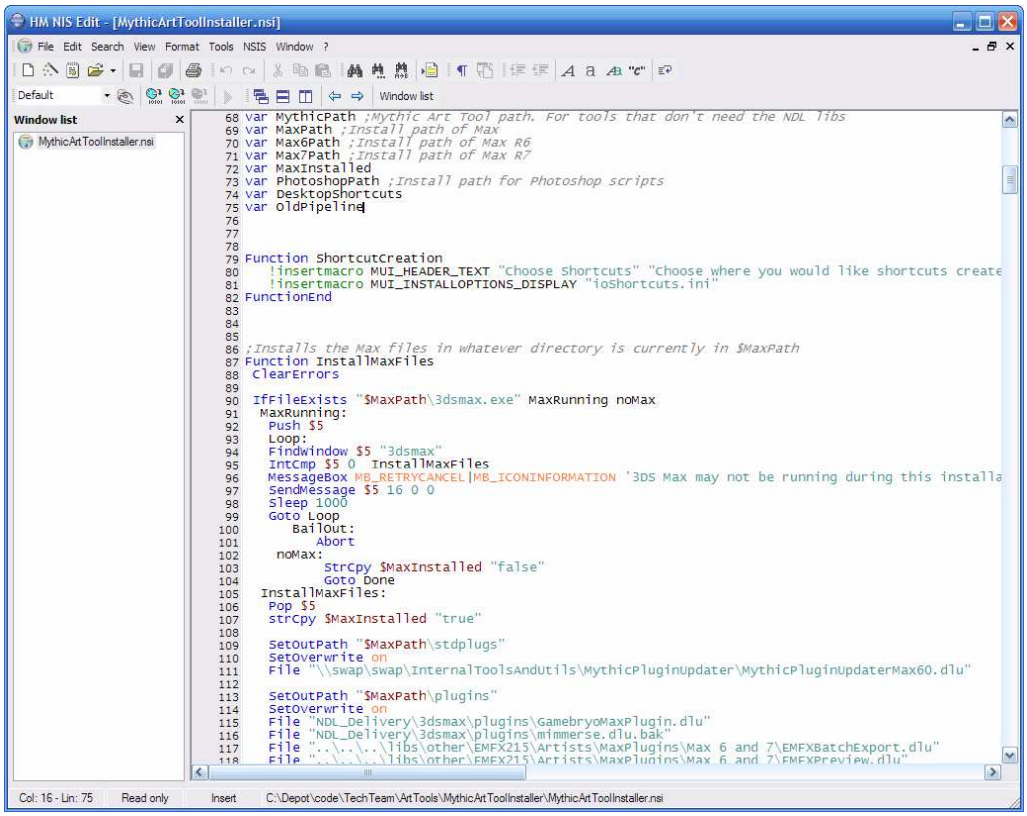
NSIS and HM NIS Edit

Application installer development packages used to be very complex and expensive. That changed when NullSoft (the makers of WinAMP) released the NullSoft Installer System (NSIS) a few years ago. NSIS, now at version 2.04, is free to use and modify for any purpose. The license insures that the source code is available and can be changed by anyone. While this code freedom might not be important to most MAXScripters it does guarantee that NSIS will always be around, even if NullSoft goes away or decides to stop development on it.



The documentation to NSIS is very robust and the user community is very active, much like MAXScript.

NSIS installers are compiled from source code. The language is of a lower-level than MAXScript. There are no loops and code branching is handled by comparing values stored in variables on a stack. To ease NSIS code creation there are dedicated NSIS editors that provide easy access to NSIS functions and wizards to help you create installers. One of the most widely used NSIS editors is HM NIS Edit, now at version 2.0.1. Like NSIS, HM NIS Edit is free to use and modify.



About the only thing that HM NIS Edit doesn't do is integrate with an existing version control system. If you are going to keep your installer files under source control, as you should, you'll have to use a dedicated front end to your source control system for your changes and commits.

Tutorial

Creating an installer for a MAXScript using the HM NIS Edit wizard is easy. This tutorial will create an installer for a MAXScript blackjack game, MaxJack.ms, which I'm currently developing in my spare time.

The first thing you'll need to do is to find the location of the script and any supporting files (.mcr files, icon files, configuration files, etc…) on your hard drive. NSIS will pull these files from those locations when it is compiling the final executable.

Load HM NIS Edit and pick the "File\New Script from Wizard…" command (Ctrl+W). This will start the new installer wizard.

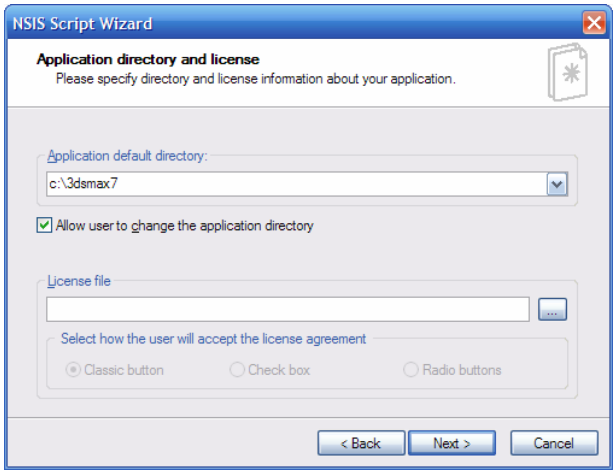
Click the "Next" button and on the Application Information page enter the name of your application, its version number, the author/publisher, and a support website. This information will be displayed on the opening dialog of the compiled installer.

None of the fields on this page are required. You can leave them blank if you don't have entries for them. When you have the dialog filled out click the "Next" button to move on.

On the Setup Options page you can specify an icon file to use for the final executable, the name of the executable, and what language(s) should be available to the end user.

All of the fields on this dialog are required. If you don't have an icon file leave the default entry in that field. The GUI and Compress options control how the executable's dialogs will appear and what compression algorithm NSIS will use to compact the data. These are advanced options and generally shouldn't be changed from their defaults of "Modern" and "zlib".

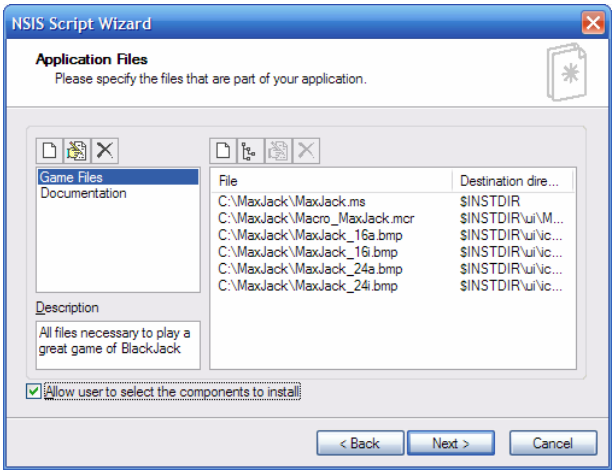
The Application Directory and License page controls where the default installation directory will be, whether or not the user can change this, and what license (EULA) should be displayed during installation.



If you don't have a license file then leave that field blank. The default Max installation folder location (c:\3dsmax<5, 6, 7>) will serve as a good default directory. Later in this document I will show you a technique for putting a variable name in this field and then scanning the user's registry to find the exact Max

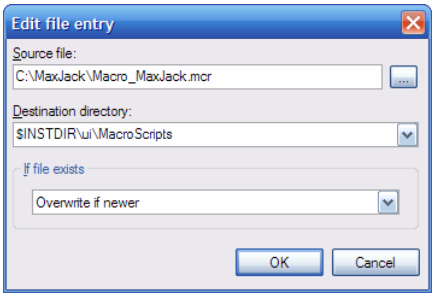
installation folder and having the installer executable use that location. The latter method makes the installer more fault tolerant as many users won't have Max installed in the same location on their machines.

Now on to the Application Files dialog where you can set up components, pick the files to include in your installer, and pick where they will be installed.



Components allow you to group your files into different categories. In this example I have two components, "Game Files" and "Documentation". I've also enabled the option to let users pick which components to install. This way, during installation, the user can decide if they want the blackjack rules installed on their machine or not. If you disable the options checkbox then the user can not make a choice and all files will be installed.

When you click the "add new" or "edit" buttons located above the file list on the right you are presented with a file entry properties dialog.



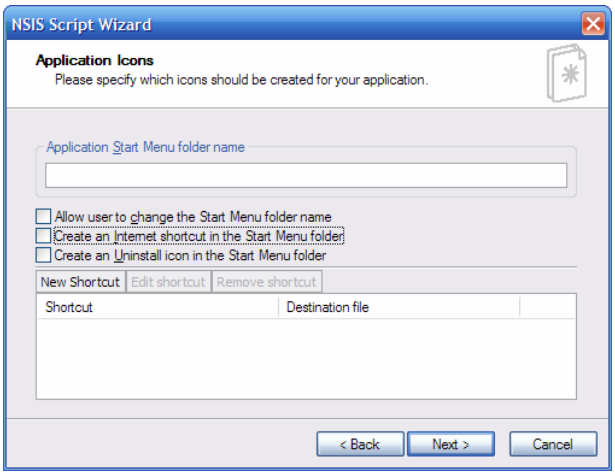
The first field in this dialog contains the path to the file on your machine. The second field contains the destination directory where the file will be put during installation. INSTDIR is a NSIS constant. The value of INSTDIR is the application default directory that was entered on the Application Directory and License dialog. Macro_MaxJack.mcr is a MacroScript so it must go into 3dsmax\7\ui\macroscripts. Since INSTDIR points to the root Max directory I've added the two sub directories onto the destination path for this file.

Making sure destination file paths are correct is crucial. At the core all an installer does is unpack files and move them to the correct locations. If you have those locations wrong then files will not be put where they need to be.

The last field controls what the installer should do if it finds an existing file of the same name in the destination directory. In this case it will do a date comparison of the two files and only make a change if the file in the installer package is newer than the one on the disk. Other choices include: always overwrite, never overwrite, and try to overwrite.

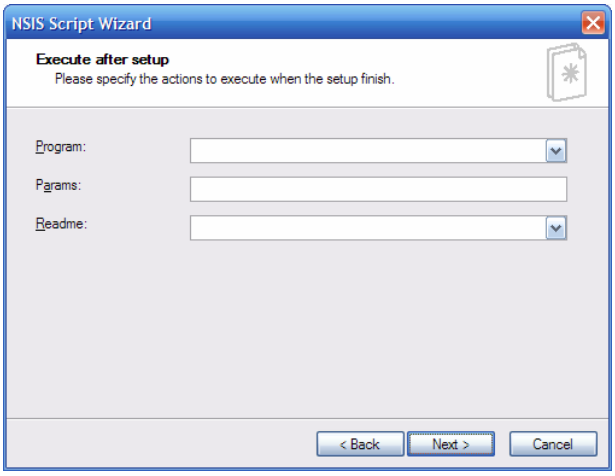
Once you have your files added and their destination paths correct click "Next".

Application shortcuts aren't pertinent to MAXScripts since scripts can't be launched from the desktop or the Start Menu. If you were making an installer for stand alone tools this is where you'd name the Start Menu folder and add the shortcut targets.



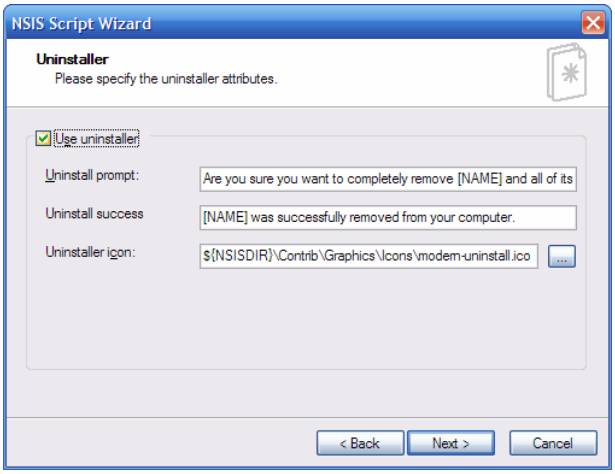
For now clear out all of the fields on this dialog and move on.

Likewise the "Execute after setup" dialog doesn't contain any options that would be used for most MAXScript installations.



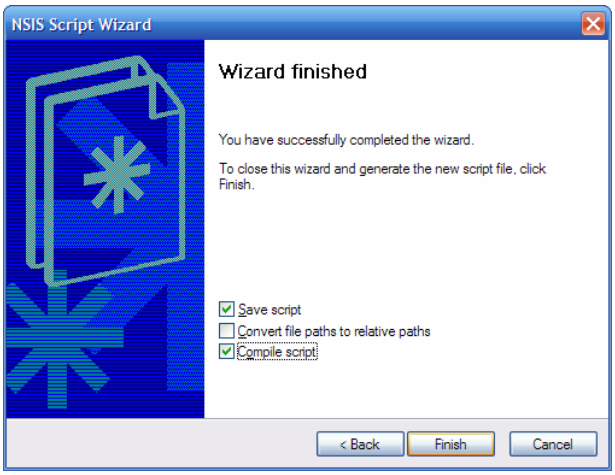
Unless you have a readme file or a program you need to run after the script is installed leave these fields blank and click "Next".

Adding an uninstaller for a script is a personal choice. I don't normally include one. They're simple to enough implement with the wizard, though.



If you have the "Use uninstaller" checkbox enabled NSIS will create an uninstaller application and add it to the user's Add\Remove Programs control panel. The uninstaller will remove any installed files, delete any folders made during installation, remove all of the installed shortcuts, and remove any registry keys created during installation. The variable [NAME] on this dialog references the value entered into the "Application Name" field back on the 2nd wizard dialog, "Application information".

You're just about done. The final dialog has options for saving the script, changing any absolute file paths to relative paths, and performing an automatic compile of the code.



It should go without saying but save your script. You'll save yourself a lot of data entry time if something unforeseen happens immediately after you exit the wizard.

Path conversion can be both good and bad. If you are going to store your installer script (*.nsi) in the same location as your development MAXScript files then I'd say yes, convert the paths to relative names. That way you can move the folder around on your hard drive and not have to worry about updating the code. If your development scripts are always in the same location (e.g. in your 3dsmax\scripts folder) and your *.nsi file will be located elsewhere then leave the paths as absolute.

If you elect to compile the script HM NIS Edit will trigger the NSIS compiler after it's saved the installer script file.

When you click "Finish" HM NIS Edit will ask you for a location and name for the script file, assuming you checked "save script", and will then optionally compile the script and put the executable in the same location as the saved .nsi file.

To install the MAXScript double click the icon of the newly compiled installer.

Advanced Tips - Editing NSIS Code Directly

While the installer wizard is a great starting point it is limited. The most you are going to do with it is create an installer that puts files in a certain destination location, include an uninstaller, and possibly display a EULA or a readme file. For most MAXScript installations that is probably enough. But, if you want to add some intelligence to your installer to, for example, find any and all versions of 3ds max installed on the user's hard drive and configure the UI accordingly, or to add entries to the system registry or an existing INI file, you'll need to get into the actual source code of your installer.

NSIS code is ASCII text so it can be edited in any text editor. I like to edit my NSIS files in HM NIS Edit since it has a nice IDE with a forms designer and can compile the code directly. If you use a text editor to edit your files you will have to run the command line NSIS editor to make your executables.

Covering all of the features of the NSIS language is beyond the scope of this document. Please refer to the NSIS help file or the website and community forums for information about the structure of the NSIS language. The following sections will give you some useful tips and tricks that relate directly to MAXScript installations so you can make your installers more powerful and useful.

Locating the Max installation directory

I would like to thank the engineers at Numeric Data Labs for helping me with this tip. The figured out how to locate Max even though it doesn't have a path key in the system registry.

Locating the installation directory of 3ds Max is trickier than it would seem. To find it you need to get the registry key that tells you the location of the default icon for the version of Max you are using. The default icon is the icon that will be applied to any new .max files, so it has to be registered with the system. The location in the registry for this is HKEY_CLASSES_ROOT\3dsmax#\DefaultIcon, where # is the revision of Max in which you are interested. The value in the key is a string that is a full path to the 3dsmax.exe executable and the icon file stored in the executable that should be used. For example, "c:\program files\3dsmax7\3dsmax.exe,1"

NSIS has a command to pull string information out of the registry. The command is:
ReadRegStr $var <root_key> <sub_key> <name>

Var is a user variable defined at the top of the NSIS code to hold the retrieved string. Remember that INSTDIR is a constant that refers to the destination directory for the installation. While it would be nice to put the registry value directly into INSTDIR there would be too much information. The registry string has to be read into a variable so that it can be massaged into a properly formed path before it is copied into INSTDIR

The NSIS command to get the path to Max 7 from the registry is:
```
ReadRegStr $TempPath HKCR "3dsmax6\DefaultIcon\" ""
```

The Max icon entry in the registry doesn't contain a name value so the last function parameter is entered as a null string.

TempPath now contains the value "C:\3dsmax7\3dsmax.exe,1". That's close to what we want, but the application name and icon offset aren't part of the path. They need to be removed. NSIS doesn't have a trim function but it does have a string copy function, StrCpy(). That command can copy either entire strings or just part of a string. We can use it to copy the part of the string we want into the INSTDIR constant and have our destination path point to the root of the 3ds max folder.

```
StrCpy $INSTDIR $TempDir -13
```

The "-13" parameter tells StrCpy to not copy the last thirteen characters of the source string (\3dsmax.exe,1) to the destination string. INSTDIR now contains the string "C:\3dsmax7" which is the full path to the root folder.

As was mentioned in the tutorial section you can add on to the $INSTDIR path as needed:

"$INSTDIR\scripts" would point to c:\3dsmax6\scripts
"$INSTDIR\ui\icons" would point to c:\3dsmax6\ui\icons
etc…

With this information you can code your installer to always populate the "Destination Folder" field on the options dialog with the exact location of 3ds max. This reduces the possibility of user error in picking the right installation location.

Many of the artists where I work have the latest revision of Max plus at least one (sometimes more) prior versions installed. Since the paths in the registry only differ by the revision number it's possible to put each path into a different variable and then pass those variables into a function that handles the script/tool installation. That way one function can install the script into all versions of Max on the user's hard drive. Some of the MAXScript extensions we use are specific to certain versions of Max. The installer function takes a parameter to instruct it to only install certain plug-ins in certain versions of Max

To support multiple versions of Max with MZP files the artist would have to load each version of Max and execute the installer script, possibly having to use different installers for different versions of Max.

Adding an Entry to Max's Plugin.ini File

With NSIS it is just as easy to manipulate INI files as it is to deal with the registry. Max has a menu under the Help menu called "Additional Help" that is a great, but under used, resource. It is there to allow script and plug-in developers the ability to add the documentation for their tools to the Max user interface. Easy access to documentation for users will lessen support issues, so this menu is a worthwhile feature.

Entries in the "Additional Help" menu are stored in the 3dsmax\plugin.ini file under the [Help] section. Each entry consists of the name to display in the menu and a path to the documentation. In NSIS it is one line of code to add a new string to an INI file.

WriteINIStr ini_file section key value

WriteINIStr "$INSTDIR\plugin.ini" "Help" "MaxJack Rules" "$INSTDIR\Help\MaxJack.chm" will add an entry to the "Additional Help" menu named "MaxJack Rules" and will load "c:\3dsmax7\help\MaxJack.chm" when the user selects it.

Notice that I used $INSTDIR for both the path to the plugin.ini file and as part of the string that will be written. NSIS knows that an entry of "$INSTDIR\..." in an INI file would be useless so it expands any variable or constant used in a string to the value it holds.

Determine if Max is Running

Scripts can be installed while Max is running. But MacroScript icons and any plug-ins that get installed will not be available until Max is restarted. Because of those issues you might want to detect if Max is running at the start of the installation process and inform the user that it needs to be closed.

This tip is more complex than the previous ones. It involves using the NSIS stack, confirmation dialogs, and subroutines. The process can be outlined as:

- Push a supplied variable to the stack. You can't push user named variables to the stack so you must use one of the stack capable variables that NSIS supplies.
- Start a loop
- In the loop try to find a window named "3dsmax". If it exists put its handle into the variable on the stack. If not, put a 0 into the variable.
- If the variable holds a 0 then assume Max isn't running. Exit the loop and go to the subroutine that contains the file installation code.
- If the variable holds a handle then Max is running. Put up a dialog asking the user if they would like to close Max. If the user answers yes then exit the loop and jump to the subroutine that closes the application. If the user answers no then exit the loop and jump to the subroutine that skips any Max file installation.

The commands that will get used are Push(), FindWindow(), IntCmp(), MessageBox(), Sleep(), Goto(), and SendMessage(). The code snippet below shows the logic and contains comments that explain each step.

Putting the value on the stack, comparing against that, and jumping to subroutines acts like an If/Then/Else block. NSIS doesn't have If/Then branching so you have to build the logic yourself. While this format isn't as high-level as MAXScript constructs it still offers more flexibility than mzp.run files.

There aren't Do/While loops in NSIS, either. There are subroutines you can jump to. Keep in mind, though, that the code is executed sequentially. When execution gets to the end of one subroutine it will immediately fall through to the code beneath it unless it is told to go elsewhere. In the example below the "Goto loop" command will always keep execution within the "loop" subroutine until the user clicks either "Yes" or "No". If the Goto statement wasn't there then the installer would put up the message box and immediately try to close Max.

```
;Put a new variable on the top of the stack.
Push $9

;A subroutine that will be used as a loop
loop:

    ;Check for a window named "3dsmax" Put the window handle on the stack if
found
    FindWindow $9 "3dsmax"

    ;Compare the value on the top of the stack to 0
    ;If true then Max isn't running. Jump to the subroutine
    ;that will install the files.
    IntCmp $9 0 installMaxFiles

    ;If the compare isn't true then Max is running.
    ;Put up a Yes/No message box asking if the user would like to close it.
    ;MessageBox() parameters differ depending on what the first
    ;parameter is. In this case it is a Yes/No messagebox so it has to
    ;have subroutine names to jump to for either choice.
    ;If the user picks "Yes" then jump to the "closeMax" subroutine.
    ;If they pick "No" then we're done. Jump to the "done" subroutine.
    MessageBox MB_YESNO 'Max is running. Close it?' IDYES closeMax IDNO done

    ;wait one second before moving on.
    ;If there is no sleep statement the program will actas if it is locked up
    ;since it is redrawing the messagebox faster than the user can click on
    ;a button.
    Sleep 1000

    ;Jump back to the top of this subroutine so the installer doesn't
    ;do anything until the user has made a choice
    Goto loop

closeMax:
    ;SendMessage sends a command to a HWND handle. In this instance the message
    ;is "close". The final two parameters can optionally hold a return value and
    ;a timeout value. Pass 0's into those parameters if not using the options.
    SendMessage $9 ${WM_CLOSE} 0 0

    ;It is recommended to sleep a bit after a SendMessage() to let Windows
    ;clean everything up.
    Sleep 1000

    ;If something went wrong NSIS will have thrown an exception.
    ;IfErrors() checks for an exception and branches the code execution base
    ;on the result. In this case if there was an error then take the
```

```
    ;window handle variable off of the stack and be done.
    ;If there no errors the files will be installed.
    IfErrors done installMaxFiles

installMaxFiles:
    <File installation lines go here>

;This subroutine is put last so that, no matter what, the stack will be
;cleaned up before execution moves on to the rest of the installer.
Done:
    Pop $9
```

In this example the installation is skipped if Max isn't closed. Alternatively you could just inform the user that Max will have to be re-started after installation and proceed to install the files. Another variation would be to have the Yes/No dialog be a "Yes/No/I'll close it myself" dialog and let the user have the option of an automatic closure or a manual one, so they can save their work.

Conclusion

The transition from mzp.run files to a NSIS installer executable isn't impossibly hard but it is a different way of doing things. With HM NIS Edit and the excellent NSIS documentation you should be able to pick up NSIS in a matter of hours. The NSIS documentation, website, and code archive all provide excellent information and instruction on how to properly use NSIS.

While I am not advocating or pushing for the wholesale rejection of .MZP files I am aware of the limitations of them. In talking with other members of the MAXScript community I know that I am not the only one who has wanted a more robust installer system. NSIS, while not perfect, provides much better installation tools for MAXScripters.