Zen of Paraffin

Making Windows Installer XML (WiX) easier to use

Current version: 3.61

Questions or Comments: John Robbins ([john@wintellect.com](mailto:john@wintellect.com))

# Introduction

When building WiX-based installers, you quickly run into the problem of maintaining the list of files you want to install. The HEAT tool that comes with WiX does a great job creating the initial fragments, but doesn’t have any support for what happens during normal development: you’re adding and removing files all the time. For many installs, HEAT works fine, but if you have an installer that’s installing hundreds to hundreds of thousands of files, the thought of maintaining those fragments of XML by hand would fill anyone with dread.

A few years ago, I was running into that same trouble with my WiX installers right around the time .NET 3.0 and LINQ were starting their beta testing. Seeing a need to scratch my itch, I dove in and created Paraffin. The idea was simple, you could maintain your fragments without breaking component rules as files were added and removed from your installer. Paraffin has grown as more people have started using it and suggest features. Thanks to everyone who’s used Paraffin!

This guide is to help you get more out of Paraffin. Obviously, it assumes you are conversant with WiX and Windows Installer. If you’re new to WiX, you should head over to Gábor DEÁK JAHN’s excellent tutorial (<http://www.tramontana.co.hu/wix/>).

# TWO SUPER IMPORTANT NOTES ABOUT PARAFFIN!

The first key to maximizing Paraffin usage is that you always run Paraffin updates from the same relative directory where you first created the initial WiX fragment. For example, if you change to C:\BUILD\IMAGES in PowerShell and create the initial WiX fragment for the IMAGES directory, all updates have to be run from the IMAGES directory. You can hard code starting directory paths in Paraffin, but that will probably break your builds. By running from the IMAGES directory, you are in the same place relative to the rest of the code in your WiX project every time. This allows more flexibility.

The second key is that in Windows Installer, component rules are sacred. Paraffin works very hard to avoid issues, but if you are not careful you can make your life super difficult. This warning is mainly around patching issues. Patching is far more difficult to get right than it should be. From my experience, you would be much better off sticking to making everything a major upgrade for every install. Based on many requests, I’ve made Paraffin much more usable in patching scenarios (minor upgrades), but you really do need to know what you’re doing and have planned your install completely.

# Creating Your First Fragments

Paraffin supports three modes of execution, creating fragments, updating fragments for major upgrades, and updating fragments for minor upgrades. Obviously, you first have to create a fragment in order to upgrade that fragment.

## Required Parameters for Initial File Creation

|  |  |
| --- | --- |
| Parameter | Description |
| -dir <directory> | This is the directory you want Paraffin to recurse looking for files. As I mentioned previously, it’s best to use relative paths here to avoid hard coded drives and directories. |
| -GroupName <value> | Paraffin automatically creates a <ComponentGroup> element in the output file so in your main install .WXS file you can pull in all the components easily with <ComponentRef> element. |
| <file> | The filename for output. |

## Optional Parameters for Initial File Creation

|  |  |
| --- | --- |
| Parameter | Description |
| -alias <alias> | By default, Paraffin uses the hard coded full name to the file in the File elements Source attribute. The –alias option allows you to substitute a text value for the starting directory (where you run Paraffin) so you can make the Source attribute relative to your main installers directory, or you can insert a WiX preprocessor variable. |
| -diskId <number> | The value of the DiskId attribute on component elements in case you have your installer on multiple media. WiX defaults to 1 so you only need to use this switch if you need a value of 2 or higher. The <number> specified must be an integer. |
| -dirref <DirectoryRef> | Paraffin defaults to using INSTALLDIR as the Id attribute to the <DirectoryRef> element. If you are using a different value to put your fragments under, specify that value with this switch. |
| -ext <ext> | File extensions to not include in the Paraffin output. You can specify as many –ext flags as you like. The <ext> specified does not need to include the leading period. |
| -includeFile <file> | Files to be added as WiX includes at the top of the output file. No validation on the contents of the file. You many have as many –includeFile options as you need. |
| -norecurse | Paraffin defaults to recursing the start directory and every directory underneath this, if you only want to do the –dir specified directory, use this switch. |
| -NoRootDirectory | Paraffin defaults to including the –dir specified directory as a <Directory> element under the main <DirectoryRef>. If you want the files in the –dir location to go directly under the <DirectoryRef> use this switch. |
| -regExExclude “regex” | Adds regular expression exclusions to both files and directories. The regular expression specified must be in quotes to account for spaces. Also, all regular expressions are treated as case insensitive. The regular expression is applied to filenames, not the containing directory, and directory names when enumerating directories. Specify as many –regExExclude options as necessary. |
| -verbose | Shows verbose output. |
| -Win64var <var> | If specified adds the Win64=”<var>” attribute to all components. In most cases you should not use this switch but instead specify the architecture with the WiX tools –arch switch. |

## Deprecated Optional Parameters for Initial File Creation

|  |  |
| --- | --- |
| Parameter | Description |
| -direXclude <exdir> | This switch may be deprecated in future versions of Paraffin. Please use the –regExExclude switch going forward.  This switch allows you to specify directory names you want Paraffin to skip and not process. There is no wildcard matching only string contains matching. You can have as many –direXclude switches as necessary. |

# Updating Fragments for Major Upgrades

As I mentioned earlier, major upgrades are really the way to go with Windows Installer. They alleviate you of worrying very much at all about what happens when you install a new version. Before the new bits are installed, Windows Installer uninstalls the old version so there are no problems with component rules, left over goo, or anything else associated with minor upgrades. If you don’t have major prerequisites, such as SQL Server, specific versions of .NET, or anything else complicated, you don’t even need a boot strapper, like the long dreamed about Burn, everything can be done with just the .MSI file.

When using Paraffin to update an existing fragment for a major upgrade, any new files found in the directory are added as appropriate. The Components elements for files that no longer exist are removed. For files that haven’t changed the Component element GUID and Id attributes are properly transported to the output file so the component rules don’t break.

It’s important to realize that Paraffin does not overwrite the existing <input>.WXS file but writes out the changes to <input>.PARAFFIN. Numerous Paraffin users have asked for Paraffin to overwrite the file, but we’re dealing with Windows Installer here. It’s a brittle and scary API in Windows so I’m loath to go down that route.

## Required Parameters for Major Upgrade Processing

|  |  |
| --- | --- |
| Parameter | Description |
| -update | This switch tells Paraffin you want to do updating for a major upgrade. |
| <file> | The original .WXS file to process again. |

## Optional Parameters for Major Upgrade Processing

|  |  |
| --- | --- |
| Parameter | Description |
| -ext <ext> | Additional file extensions to not include in the Paraffin output that will be added to the ones specified when creating the file. You can specify as many –ext flags as you like. The <ext> specified does not need to include the leading period. |
| -regExExclude “regex” | Additional regular expression exclusions to both files and directories on top of the ones specified when the file was created. See the previous discussion of –regExExclude for more information. Specify as many –regExExclude options as necessary. |
| -ReportIfDifferent | If specified, Paraffin’s exit code will be 4 if the input .WXS file and the output .PARAFFIN file are different. If the files are identical, the exit code is 0. The idea behind this switch is that you could run Paraffin as part of your build and know when you have forgotten to add or remove files from the main .WXS files automatically. |
| -verbose | Shows verbose output. |

# Updating Fragments for Minor Upgrades

Adding files is trivial in a minor upgrade. The problem becomes when you need to remove a file during that minor upgrade. According to the Window Installer rules, that’s requires you go directly to a major upgrade. Many people have asked me to implement the scheme described by Vagmi Mudumbai in his blog: <http://geekswithblogs.net/Vagmi.Mudumbai/archive/2006/06/11/81426.aspx>, which will remove files during minor upgrades.

In WiX, the above would look like the following (word wrapped for this document). The trick is to add the Transitive attribute to the Component element, and add a nonsensical condition. The transitive forces the condition to be reevaluated and the condition is false. Thus, the file is uninstalled in a minor upgrade.

<Component Id="comp\_F5B392C05B5249C2AB34810BE4A6163F"   
 Guid="FD9D7BF4-550A-47E2-85A3-634D77F609FB"   
 Transitive="yes">  
   <File Id="file\_81DD236ED61B4BBEB445FAB9E9C2DAB2"   
 Checksum="yes"   
 KeyPath="yes"   
 Source=".\Temp\a.exe" />  
   <Condition>1 = 0</Condition>  
</Component>

There’s one more step that’s involved. The file must be there when building your install and it’s also best if that file is only zero bytes long so it doesn’t take up space in the .MSP. Don’t worry Paraffin takes care of all of this mess for you. All you have to do is to delete the files you used to install before running Paraffin to build the minor upgrade ready file.

Once you’ve deleted a file for a minor upgrade, Paraffin assumes it’s deleted forever. If you try to add a file with the same name and directory back, Paraffin will abort processing because you are going to break the component rules. If you need to add a file with a previously deleted name you need to use a major upgrade. While I’m sure someone will ask that I make Paraffin handle that situation, I won’t because you’re probably screwing up your installation.

With the minor upgrade checking, Paraffin is only looking at files it does not look at directories. If you delete or rename a directory, you need to use major upgrades. The trick of using the transitive attribute only applies to the Component element. There’s no way for a minor upgrade to remove a directory so there’s no need for me to implement support for it.

## Required Parameters for Minor Upgrade Processing

|  |  |
| --- | --- |
| Parameter | Description |
| -PatchUpdate | This switch tells Paraffin you want to do updating for a major upgrade. |
| <file> | The original .WXS file to process again. |

## Optional Parameters for Minor Upgrade Processing

|  |  |
| --- | --- |
| Parameter | Description |
| -ext <ext> | Additional file extensions to not include in the Paraffin output that will be added to the ones specified when creating the file. You can specify as many –ext flags as you like. The <ext> specified does not need to include the leading period. |
| -regExExclude “regex” | Additional regular expression exclusions to both files and directories on top of the ones specified when the file was created. See the previous discussion of –regExExclude for more information. Specify as many –regExExclude options as necessary. |
| -PatchCreateFiles | If you’d like Paraffin to create the zero byte files for your removed files, this switch will take care of that for you. If you rerun Paraffin again with the –PatchUpdate switch as it checks the .WXS file for the transitive attribute on the component so it doesn’t see the zero byte file as a new file. |
| -ReportIfDifferent | If specified, Paraffin’s exit code will be 4 if the input .WXS file and the output .PARAFFIN file are different. If the files are identical, the exit code is 0. The idea behind this switch is that you could run Paraffin as part of your build and know when you have forgotten to add or remove files from the main .WXS files automatically. |
| -verbose | Shows verbose output. |

# Creating the Removed Zero Byte Files

If you’re like me, you probably don’t want to have these odd zero byte files necessary for the minor upgrade file removal littered through your version control system. Paraffin offers a separate command line option to go create all those zero byte files on the fly. The idea with this command line option is you’ll run Paraffin with it as part of your master builds so you don’t have to worry about those long ago deleted files ever again.

## Required Parameters for Zero Byte File Creation

|  |  |
| --- | --- |
| Parameter | Description |
| -PatchCreateFiles | Tells Paraffin to create any zero byte files you want to remove in minor upgrades |
| <file> | The .WXS file that has dead files in it. |

## Optional Parameters for Zero Byte File Creation

|  |  |
| --- | --- |
| Parameter | Description |
| -verbose | Shows verbose output. |

# Paraffin Exit Codes

|  |  |
| --- | --- |
| Exit Code | Description |
| 0 | Paraffin ran without errors.  If –ReportIfDifferent was specified when doing either kind of update, zero also means there is no difference between the .WXS file and the .PARAFFIN file. |
| 1 | An invalid command line option was specified |
| 2 | The input .WXS file is invalid |
| 3 | The file contains multiple files per component. Starting with WiX 3.5, multiple files per components is no longer supported. Please revert to WiX 3.13. |
| 4 | If –ReportIfDifferent was specified, an exit code of 4 indicates the input .WXS file and the output .PARAFFIN file are different. |

# Paraffin .ParrafinMold files

In order to support more customization of the Paraffin output, Paraffin supports the concept of injectable data through files that have the .ParrafinMold extension. For example, since Paraffin does not handle COM and .NET assembly registration, which HEAT does, you may want to put the COM information into a .ParaffinMold file so it can be injected automatically.

As Paraffin is processing a directory and finds any files matching \*.ParaffinMold, it will inject the contents of each .ParaffinMold file into the output file under the current element (normally Directory, or DirectoryRef if -norootdirectory is used).

All .ParaffinMold files are WiX fragment files and must have a DirectoryRef element, but the Id value is ignored. Any elements under the DirectoryRef are placed directly in the output file. There is no checking done on the validity of any of those child nodes. If you manually add a new WiX namespace such as 'xmlns:util="http://schemas.microsoft.com/wix/UtilExtension"' to the WiX element for your .ParaffinMold files to compile correctly, those namespaces will be copied over to the .PARAFFIN file. You should put those additional namespaces before the standard xmlns declarations to make differencing easier.

Here is an example .ParaffinMold file to insert a Registry element.

<?xml version="1.0" encoding="utf-8"?>  
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">  
  <Fragment>  
    <DirectoryRef Id="INSTALLDIR">  
       <Component Id="HappyID"  
                     Guid="PUT-GUID-HERE">  
         <RegistryKey Root="HKLM"  
                            Key="SOFTWARE\My Company\My Product"  
                            Action="createAndRemoveOnUninstall">  
           <RegistryValue Name="InstallRoot" Value="[INSTALDIR]"  
                          Type="string" KeyPath="yes"/>  
        </RegistryKey>  
       </Component>  
     </DirectoryRef>  
   </Fragment>  
 </Wix>

# Notes for Upgraders to Paraffin 3.5 or Higher

Prior to Paraffin 3.5, Paraffin supported multiple files per component. Since one file per component is much preferred for Windows Installer resiliency, and I couldn’t find anyone using multiple files per component, I removed support for it. By doing so it made my development life easier and greatly simplified the new -PatchUpdate feature. If you happen to need multiple files per component, please keep using Paraffin 3.13 you can find here: <http://www.wintellect.com/CS/files/folders/8198/download.aspx>.

Paraffin used to put the KeyPath attribute on the Component. It looks like now it’s much better to have that on the File element itself. If you use Paraffin 3.5 to update a .WXS file created with a previous version, I swap the KeyPath attribute as appropriate. That shouldn’t cause any problems for anyone, but I wanted to give you notice of the change. You’ll also see yellow warning text on the screen when you run Paraffin when the attribute swap occurs.