Basic Tutorial for wix installer

<https://www.codeproject.com/Articles/21442/Creating-an-installer-using-Wix-v3-0-Votive-and-Vi>

Real-World Example: WiX/MSI Application Installer

<https://helgeklein.com/blog/2014/09/real-world-example-wix-msi-application-installer/>

WixUI\_Advanced Dialog Set

<http://wixtoolset.org/documentation/manual/v3/wixui/dialog_reference/wixui_advanced.html>

# WixUI Dialogs

<http://wixtoolset.org/documentation/manual/v3/wixui/dialog_reference/wixui_dialogs.html>

# Customizing Built-in WixUI Dialog Sets

<http://wixtoolset.org/documentation/manual/v3/wixui/wixui_customizations.html>

A Single Dialog

<https://www.firegiant.com/wix/tutorial/user-interface-revisited/a-single-dialog/>

Well Done

<https://www.firegiant.com/wix/tutorial/user-interface-revisited/well-done/>

How to add custom dialogs in WiX installers

<https://www.add-in-express.com/creating-addins-blog/2014/02/07/add-custom-dialogs-wix-installer/>

Source Code

<https://github.com/AnalogJ/Wix3.6Toolset/blob/master/RC0-source/wix36-sources/src/ext/UIExtension/wixlib/AdvancedWelcomeEulaDlg.wxs>

<https://github.com/AnalogJ/Wix3.6Toolset/blob/master/RC0-source/wix36-sources/src/ext/UIExtension/wixlib/ExitDialog.wxs>

Customised UI's for WiX

<http://neilsleightholm.blogspot.com/2008/08/customised-uis-for-wix.html>

# How To: Run the Installed Application After Setup

<http://wixtoolset.org/documentation/manual/v3/howtos/ui_and_localization/run_program_after_install.html>

# Localizing more than strings in WiX v3.6

<http://www.joyofsetup.com/2012/07/14/localizing-more-than-strings-in-wix-v3-6/>

# [WiX: can I change the size or position of controls in the Standard Dialogs?](https://stackoverflow.com/questions/17332531/wix-can-i-change-the-size-or-position-of-controls-in-the-standard-dialogs)

<https://stackoverflow.com/questions/17332531/wix-can-i-change-the-size-or-position-of-controls-in-the-standard-dialogs>

# Adding and Customizing Dialogs in WiX 3

<http://www.dizzymonkeydesign.com/blog/misc/adding-and-customizing-dlgs-in-wix-3/>

# Wix useful Info

<https://stackoverflow.com/questions/52654935/changing-text-color-to-wix-dialogs>

# [WiX tricks and tips](https://stackoverflow.com/questions/471424/wix-tricks-and-tips)

<https://stackoverflow.com/questions/471424/wix-tricks-and-tips>

# [WiX: How to override “C:\Program Files (x86)” on x64 machine in WixUI\_Advanced sequence?](https://stackoverflow.com/questions/5479790/wix-how-to-override-c-program-files-x86-on-x64-machine-in-wixui-advanced-s)

I'm using **WixUI\_Advanced** sequence to allow users pick *per-machine* or *per-user* installation and change destination folder. My WiX project is intended to produce both **x86** and **x64** MSIs (I'm using [***WiX Tips and Tricks***](https://stackoverflow.com/questions/471424/wix-tricks-and-tips#577793) recommendations). I also keep the app **installation folder in the registry** for major upgrades (I use APPLICATIONFOLDER property and directory ID -- instead of INSTALLLOCATION -- per WixUI\_Advanced requirements).

There is a [**bug in WixUI\_Advanced sequence**](https://github.com/wixtoolset/issues/issues/2165) that causes the Destination Folder dialog to display the app folder under *C:\Program Files (x86)* instead of *C:\Program Files* when running on a 64-bit machine, even when the code correctly sets app folder to *ProgramFiles64Folder* property. The bug tracker comment suggests using the *SetDirectory* element to set the value of APPLICATIONFOLDER, but there is **no example** explaining how to do this. When I tried, it did dot make any difference (I also found a number of posts recommending using a custom action to set APPLICATIONFOLDER, but none worked for me). Does anyone know how to make WixUI\_Advanced sequence display correct 'Program Files' folder on a 64-bit system (and also show the originally selected folder during major upgrades)?

If it helps, I'll provide sample WXS snippets, but they pretty much follow the recommendations from the StackOverflow's [***WiX Tips and Tricks***](https://stackoverflow.com/questions/471424/wix-tricks-and-tips#577793) post. Also, my 64-bit MSI package is indeed a 64-bit package (I have the package and components marked as 'x64"; and it does not run on 32-bit platforms). I'm using WiX 3.6 and Visual Studio 2010.

CODE SAMPLE:

<?xml version="1.0" encoding="UTF-8"?>

<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">

<Product

Id="81955f17-31f3-4e51-8294-372f96141c00"

Name="WiX64BitDemo"

Language="1033"

Version="1.0.0.0"

Manufacturer="Test"

UpgradeCode="5bed9777-bea6-4dc3-91d7-5dd93819563a">

<Package

InstallerVersion="300"

Compressed="yes"

InstallScope="perMachine"

Platform="x64" />

<MajorUpgrade

AllowSameVersionUpgrades="no"

DowngradeErrorMessage="Can't downgrade."

Schedule="afterInstallInitialize" />

<Media

Id="1"

Cabinet="media1.cab"

EmbedCab="yes" />

<Property Id="APPLICATIONFOLDER" Secure="yes">

<RegistrySearch Id="FindInstallLocation"

Root="HKLM"

Key="Software\Microsoft\Windows\CurrentVersion\Uninstall\[WIX\_UPGRADE\_DETECTED]"

Name="InstallLocation"

Type="raw"

Win64="yes" />

</Property>

<Property Id="ApplicationFolderName" Value="WiX64BitDemo" />

<Property Id="WixAppFolder" Value="WixPerMachineFolder" />

<SetDirectory

Id="APPLICATIONFOLDER"

Value="[ProgramFiles64Folder][ApplicationFolderName]">APPLICATIONFOLDER=""</SetDirectory>

<Directory Id="TARGETDIR" Name="SourceDir">

<Directory Id="ProgramFiles64Folder">

<Directory Id="APPLICATIONFOLDER" Name="WiX64BitDemo">

<Component

Id="ReadmeComponent"

Guid="\*"

Win64="yes">

<File

Id="ReadmeFile"

Name="readme.txt"

Source="$(var.ProjectDir)readme.txt"

KeyPath="yes"/>

</Component>

</Directory>

</Directory>

</Directory>

<Feature Id="ProductFeature" Title="WiX64BitDemo" Level="1">

<ComponentRef Id="ReadmeComponent" />

</Feature>

<UI Id="UISequence">

<UIRef Id="WixUI\_Advanced"/>

</UI>

</Product>

</Wix>

Many thanks to Sascha Beaumont for solving this problem. Here is the working sample:

<?xml version="1.0" encoding="UTF-8"?>

<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">

<Product

Id="81955f17-31f3-4e51-8294-372f96141c00"

Name="WiX64BitDemo"

Language="1033"

Version="1.0.0.0"

Manufacturer="Test"

UpgradeCode="5bed9777-bea6-4dc3-91d7-5dd93819563a">

<Package

InstallerVersion="300"

Compressed="yes"

InstallScope="perMachine"

Platform="x64" />

<MajorUpgrade

AllowSameVersionUpgrades="no"

DowngradeErrorMessage="Can't downgrade."

Schedule="afterInstallInitialize" />

<Media

Id="1"

Cabinet="media1.cab"

EmbedCab="yes" />

<Property Id="APPLICATIONFOLDER" Secure="yes">

<RegistrySearch Id="FindInstallLocation"

Root="HKLM"

Key="Software\Microsoft\Windows\CurrentVersion\Uninstall\[WIX\_UPGRADE\_DETECTED]"

Name="InstallLocation"

Type="raw"

Win64="yes" />

</Property>

Something like this would probably do the trick:

<MajorUpgrade AllowSameVersionUpgrades="yes"

DowngradeErrorMessage="Can't downgrade."

Schedule="afterInstallInitialize" />

<Property Id="APPLICATIONFOLDER" Secure="yes">

<RegistrySearch Id="FindInstallLocation"

Root="HKLM"

Key="Software\Microsoft\Windows\CurrentVersion\Uninstall\[WIX\_UPGRADE\_DETECTED]"

Name="InstallLocation"

Type="raw"

Win64="yes" />

</Property>

<CustomAction Id="Overwrite\_WixSetDefaultPerMachineFolder" Property="WixPerMachineFolder" Value="[ProgramFiles64Folder][ApplicationFolderName]" Execute="immediate" />

<InstallUISequence>

<Custom Action="Overwrite\_WixSetDefaultPerMachineFolder" After="WixSetDefaultPerMachineFolder" />

</InstallUISequence>

<InstallExecuteSequence>

<Custom Action="Overwrite\_WixSetDefaultPerMachineFolder" After="WixSetDefaultPerMachineFolder" />

</InstallExecuteSequence>

<SetProperty Id="ARPINSTALLLOCATION" Value="[APPLICATIONFOLDER]" After="CostFinalize" />

**UPDATE:** SetDirectory schedules the action prior to WixSetDefaultPerMachineFolder - code updated for manually scheduled elements to schedule between WixSetDefaultPerMachineFolder and WixSetPerMachineFolder. Tested OK with OP sample code under Win7 x64

**UPDATE2:** Added action to set ARPINSTALLOCATION as <http://robmensching.com/blog/posts/2011/1/14/ARPINSTALLLOCATION-and-how-to-set-it-with-the-WiX-toolset>

# The WiX toolset's "Remember Property" pattern.

[#](http://robmensching.com/blog/posts/2010/5/2/the-wix-toolsets-remember-property-pattern/) by [Rob Mensching](http://robmensching.com/about/) on Sunday, May 2, 2010

This question comes up so often, I should have written this blog entry years ago. The root issue is that the Windows Installer does not save Property values for you. That means if the user enters values in the install UI or passes them on the command-line, those values will be not be present during repair, upgrade nor uninstall. That last one, uninstall, catches people all the time. So let's solve the problem simply then solve it completely.

### **The Simple Solution**

As is often the case, the simple solution is not the correct solution but it definitely shows us the way. In this case, what we're going to do is squirrel away the user provided Property values in registry keys and use a RegistrySearch to read them back out for repair, upgrade and uninstall. The code for this simple solution is pretty straight forward.

1: <?xml version='1.0'?>

2: <Wix xmlns='http://schemas.microsoft.com/wix/2006/wi'>

3: <Product Id='\*' Name='Demo Remember Property' Language='1033'

4: Version='0.1.0.0' Manufacturer='RobMensching.com LLC'

5: UpgradeCode='c2e7dfa4-3faf-4c41-9bcc-eabf7657a2b9'>

6: <Package InstallScope='perUser' InstallerVersion='200' />

7:

8: <MajorUpgrade DowngradeErrorMessage="A newer version already installed" />

9:

10: <Property Id='REMEMBERME'>

11: <RegistrySearch Id='RememberProperty' Root='HKCU'

12: Key='SOFTWARE\Wix\DemoRememberProperty'

13: Name='Remembered' Type='raw' />

14: </Property>

15:

16: <Feature Id='MainFeature' Level='1'>

17: <Component Directory='ApplicationFolder'>

18: <RegistryValue Root='HKCU' Key='SOFTWARE\Wix\DemoRememberProperty'

19: Name='Remembered' Value='[REMEMBERME]'

20: Type='string' />

21: <RemoveFolder Id='CleanupApplicationFolder' On='uninstall' />

22: </Component>

23: </Feature>

24: </Product>

25:

26: <Fragment>

27: <Media Id='1' />

28: <Directory Id='TARGETDIR' Name='SourceDir'>

29: <Directory Id="LocalAppDataFolder" Name="AppData">

30: <Directory Id='ApplicationFolder' Name='Demo Remember Property' />

31: </Directory>

32: </Directory>

33: </Fragment>

34: </Wix>

That's a fully working installation, so I'll comment on the interesting parts of the code:

* **line 6**: Note that I used a per-user install for this example because it saved me UAC prompts during testing. If your install is per-machine use HKLM instead of HKCU for the various registry roots we'll discuss next.
* **line 8**: I'm using the new WiX v3.5 language simplifications with the MajorUpgrade element and the absent Component/@Id,@Guid attributes. If you're using WiX v3.0 you'll have to type more stuff.
* **line 10**: This is the Property that we'll remember. In this case, I have no default value but you could add one if your scenario required it. You can have as many remembered Properties as you want, you just add more registry keys.
* **line 11**: This is the registry search that will reload our Property on repair, upgrade and uninstall scenarios. On the initial install the registry key will be missing so the Property will use its default value (which in my case is nothing).
* **line 18**: This is the registry key that remembers our Property. Put the registry key in a place that is logical for you. Again, remember to make your Root='HKLM' when doing a per-machine install.
* **line 21**: I added the RemoveFolder element to make ICE64 happy.
* **line 26**: Notice that I pulled the directory tree into a separate Fragment. I did this just to keep the rest of the code a bit more tidy. I also added the Media element to make ICE71 happy even though there are no files to install.

Cool. Now let's walk through our install, repair, uninstall scenarios and see the simple solution work. I'll show you the command I executed then the important pieces of the log files that demonstrate how it works. First install and it's log file:

msiexec /l\*v i.txt /i demo.msi REMEMBERME=1

MSI (c) (A0:3C) [21:31:02:897]: Command Line: REMEMBERME=1 CURRENTDIRECTORY=...

MSI (c) (A0:3C) [21:31:02:898]: PROPERTY CHANGE: Adding REMEMBERME property. Its value is '1'.

MSI (c) (A0:3C) [21:31:02:918]: Switching to server: REMEMBERME="1" TARGETDIR=...

MSI (s) (90:F0) [21:31:02:945]: Command Line: REMEMBERME=1 TARGETDIR=...

Property(S): REMEMBERME = 1

Property(C): REMEMBERME = 1

Now, repair and it's log file (Note: that I only had to pass REINSTALLMODE="u" because we only need to repair our per-user registry keys) :

msiexec /l\*v r.txt /i demo.msi REINSTALL=ALL REINSTALLMODE=u

Action start 21:35:52: AppSearch.

AppSearch: Property: REMEMBERME, Signature: RememberProperty

MSI (c) (C0:A0) [21:35:52:722]: PROPERTY CHANGE: Adding REMEMBERME property. Its value is '1'.

MSI (c) (C0:A0) [21:35:52:727]: Switching to server: REMEMBERME="1" TARGETDIR=...

MSI (s) (90:90) [21:35:52:742]: Command Line: REMEMBERME=1 TARGETDIR=...

Property(S): REMEMBERME = 1

Property(C): REMEMBERME = 1

Notice how the REMEMBERME Property was set to 1 after the AppSearch. That's where the RegistrySearch element was executed and set the Property. Uninstall works exactly the same so I'll skip that and move on to where our simple solutoin breaks down. What happens if you want to change the value during a repair? It goes something like this:

msiexec /l\*v r.txt /i demo.msi REINSTALL=ALL REINSTALLMODE=u REMEMBERME=2

Action start 22:57:23: AppSearch.

AppSearch: Property: REMEMBERME, Signature: RememberProperty

MSI (c) (E4:A0) [22:57:23:378]: PROPERTY CHANGE: Modifying REMEMBERME property. Its current value is '2'. Its new value: '1'.

MSI (c) (E4:A0) [22:57:23:383]: Switching to server: REMEMBERME="1" TARGETDIR=...

MSI (s) (88:A8) [22:57:23:398]: Command Line: REMEMBERME=1 TARGETDIR=...

Property(S): REMEMBERME = 1

Property(C): REMEMBERME = 1

Do you see how the AppSearch that solved our problem creates a new one? It overwrites our attempt to set REMEMBERME to "2" with the remembered value "1". That brings us to our more complex but complete solution.

### **The Complete Solution**

The complete solution builds on the simple solution. All we need to do is add a couple custom actions to save the Property value if it set from the command line. I had hoped to use the SetProperty element introduced in WiX v3.0 to make the solution pretty clean but that does not support the "firstSequence" bit for custom actions. So I've had to write out the custom actions completely.

1: <Fragment>

2: <CustomAction Id='SaveCmdLineValue' Property='CMDLINE\_REMEMBERME'

3: Value='[REMEMBERME]' Execute='firstSequence' />

4: <CustomAction Id='SetFromCmdLineValue' Property='REMEMBERME'

5: Value='[CMDLINE\_REMEMBERME]' Execute='firstSequence' />

6:

7: <InstallUISequence>

8: <Custom Action='SaveCmdLineValue' Before='AppSearch' />

9: <Custom Action='SetFromCmdLineValue' After='AppSearch'>

10: CMDLINE\_REMEMBERME

11: </Custom>

12: </InstallUISequence>

13: <InstallExecuteSequence>

14: <Custom Action='SaveCmdLineValue' Before='AppSearch' />

15: <Custom Action='SetFromCmdLineValue' After='AppSearch'>

16: CMDLINE\_REMEMBERME

17: </Custom>

18: </InstallExecuteSequence>

19: </Fragment>

There is nothing special here. We squirrel away the REMEMBERME value from the command line in the CMDLINE\_REMEMBERME Property. Later we set the REMEMBERME Property to the CMDLINE\_REMEMBERME value if one was provided. Since I put this in a Fragment the only thing remaining is to add a reference from the Product element. I added the following just above the close Product element.

25: <CustomActionRef Id='SaveCmdLineValue' />

Try it out yourself and follow the changes to the REMEMBERME Property in a verbose log file. In the future (WiX v4.0, perhaps), we might add something to the WiX language to make this much easier to write. In the meantime, tuck this solution in a Fragment for when you need it later.

# ApplicationFolderName Can add different folder

        <!-- Allow per user or per machine -->   
        <UIRef Id="WixUI\_MyAdvanced" />   
        <Property Id="ApplicationFolderName" Value="$(var.MyCompanyNameShort)\$(var.MyProductName) $(var.Version)" />   
        <Property Id="WixAppFolder" Value="WixPerMachineFolder" />   
        <Property Id="WixUIBannerBMP" Value="$(var.InstallBanner)" />   
        <Property Id="WIXUI\_EXITDIALOGOPTIONALTEXT" Value="Thank you for installing this product." />

# Wix 32/64

<https://stackoverflow.com/questions/6231950/using-wix-to-create-32bit-and-64bit-installers-from-one-wxs-file/6232100>

# Wix Advanced Tutorial

<https://www.blogger.com/blogger.g?blogID=89203067857536568#allposts>

<http://www.cnblogs.com/stoneniqiu/p/4931142.html>

# Wix AutoVersion

<http://wixtoolset.org/documentation/manual/v3/overview/preprocessor.html>

**$(fun.AutoVersion(x.y)) – wix preprocessor function**

# How To: Implement a Major Upgrade In Your Installer

<http://wixtoolset.org/documentation/manual/v3/howtos/updates/major_upgrade.html>

<http://www.joyofsetup.com/2010/01/16/major-upgrades-now-easier-than-ever/>

# Install C++ Redistributable with your WiX Installer

<https://www.stevefenton.co.uk/2014/11/install-c-redistributable-with-your-wix-installer/>

# Visual C++ Universal C Runtime (CRT)

# <https://devblogs.microsoft.com/cppblog/introducing-the-universal-crt/>

I put together a WiX installer and included the VC++ redistributable merge module as described [here](http://wixtoolset.org/documentation/manual/v3/howtos/redistributables_and_install_checks/install_vcredist.html). The merge module I included was C:\Program Files (x86)\Common Files\Merge Modules\Microsoft\_VC140\_CRT\_x86.msm.

This installer appears to work fine but on Windows 7 the installed program will not run, complaining about missing api-ms-win-crt-runtime-l1-1-0.dll.

The VC++ runtime redistributables got more complicated in Visual Studio 2015. See [the VC team blog post](http://blogs.msdn.com/b/vcblog/archive/2015/03/03/introducing-the-universal-crt.aspx) about the universal CRT. Basically, the merge module is insufficient:

There will not be a merge module for the Universal CRT. If you currently use the CRT merge modules and still want to deploy the Visual C++ libraries centrally, we recommend that you move to the above mentioned Windows Update package or to the VCRedist. Alternatively, you may choose to link statically to the Universal CRT and the Visual C++ libraries.

It's worth noting that it is now possible to distribute the Universal C Runtime DLLs with your application. The [MSDN blog post](https://blogs.msdn.microsoft.com/vcblog/2015/03/03/introducing-the-universal-crt/) describes the steps:

1. [Install the Windnows 10 SDK](https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk)
2. Go to C:\Program Files (x86)\Windows Kits\10\Redist\ucrt and find the DLLs for the platform you are targeting
3. Copy them into your application's program directory

They are not big. About 2 megabytes in total.

I ended up using this technique because:

* It does not require me to create a separate WiX package and then bootstrap them;
* It does not require the user to run any extra installers

I cannot speak to whether it would work on all computers, but it works on the ones I have tested.

<https://stackoverflow.com/questions/34592378/redistributables-for-deploying-c-exe-developed-with-visual-studio-2015-on-wind/34597936>

<https://stackoverflow.com/questions/31527969/how-to-i-update-my-c-project-in-visual-studio-2015-to-use-the-new-universal-cr>

# [Add resource files in wix installer](https://stackoverflow.com/questions/43852401/add-resource-files-in-wix-installer)

<https://stackoverflow.com/questions/43852401/add-resource-files-in-wix-installer>