Setup & Install by Heath Stewart

About Windows Installer, the .NET Framework, and Visual Studio.

64-bit Managed Custom Actions with Visual Studio

★★★★★

February 1, 2006 by Heath Stewart (MSFT) // 25 Comments

Share 0 0 0

A reader who happened across my post on
Windows
Installer on 64-bit Platforms

mentioned a problem with running 64-bit managed custom actions using the Visual Studio 2005 Windows Installer project. This also recently cropped up in an internal discussion alias.

The issue is that if you build a managed class library project targeting a 64-bit platform
using `/platform:x64` or `/platform:Itanium` and install a Windows Installer package built in Visual Studio 2005 on a 64-bit machine a `System.BadImageFormatException` is thrown. The reason is because the native shim packaged with the *.msi* file is a 32-bit executable.

Let's step back a minute, though, to how to build a Windows Installer setup project with managed custom actions. I won't go into details, but basically you create a new Class Library project that contains one or more derivatives from the

`System.Configuration.Install.Installer` class. In the Custom Actions editor for your Windows Installer project you can right-click on a specific phase (Install, Commit, Rollback, or Uninstall) or, preferably, the root node (which adds the custom action to all phases with the appropriate custom action types) and add whatever you want from

Search MSDN with
Search this blog
Search all blogs

Recent Posts

Updated documentation for Visual Studio Build Tools container April 18, 2018

vswhere now supports -requiresAny to find instances with one or more components installed January 25, 2018

Set both x86 and x64 registry views for custom setup policy November 8, 2017

How to install Builds Tools in a Docker contair October 24, 2017

Live Now on Developer Tools Blogs

Announcing first-class support for CloudEvents on Azure

Announcing low-priority

project output to a specific file in your file system. If your class library is in the same solution I recommend clicking "Add Output" in the custom action editor dialog.

You should also click on the Windows Installer project and change the TargetPlatform property to either x64 or Itanium, depending on what you're targeting. This makes sure that 64-bit components are installed to the 64-bit folders like [ProgramFiles64Folder]. If you don't set this according to what binaries you're installing (which can be a mix of both 32- and 64-bit) 64-bit files will be installed into [ProgramFilesFolder] which, on 64-bit platforms, is, for example, *C:Program Files (x86)*.

Back to the problem. When you build the Windows Installer project in Visual Studio 2005 it embeds the 32-bit version of *InstallUtilLib.dll* into the

Binary table as InstallUtil. When Windows Installer executes your managed custom action it actually is calling the `ManagedInstall` entry point function from *InstallUtilLib.dll* as a
type 1 deferred custom action (1025) which creates
an instance of the CCW
`System.Configuration.Install.IManagedInstaller` interface
and runs your `Installer` classes. Since the native
InstallUtilLib.dll is 32-bit it loads the 32-bit Framework which will throw
the `BadImageFormatException` since your managed class library is 64-bit.

To workaround this issue you either need to import the appropriate bitness of *InstallUtilLib.dll* into the Binary table for the InstallUtil record or –
if you do have or will have 32-bit managed custom actions add it as a new record in the Binary table and adjust the

CustomAction table to use the 64-bit Binary table record for 64-bit managed custom actions.

To replace the 32-bit *InstallUtilLib.dll* with the 64-bit bitness,

1. Open the resulting *.msi* in Orca from the Windows Installer SDK
2. Select the Binary table
3. Double click the cell [Binary Data] for the record InstallUtil
4. Make sure "Read binary from filename" is selected and click the Browse button
5. Browse to *%WINDIR%Microsoft.NETFramework64v2.0.50727*
6. Select *InstallUtilLib.dll*

VMs on scale sets now in public preview

Azure expands certification scope of Health Information Trust Alliance Common Security Framework

Tags

.NET 64-bit ARPSYSTEMCOMPONENT Custom Actions Customizations Detection Development Diagnosing Essentials Installation KB Localization Logging Mailbag MSI4.5 News Pages Personal PowerShell psmsi Script Security Serviceability Shared Components Tip Troubleshootin UAC Uninstall Virtualization Vista Visual Studio VS VS11 VS15 VS2005SP1 VS2008SP1 VS2010 VS2010SP1 VS2012 VS2013 VS2015

7. Click the Open button
8. Click the OK button

Note that the Framework64 directory is only installed on 64-bit platforms and that it corresponds to the 64-bit processor type. That is, you won't find the x64 flavor of InstallUtilLib.dll on an IA64 machine.

If you already have or anticipate having 32-bit custom actions in future patches – and I recommend this approach because the future is difficult to predict – you should add a new record.

1. Open the resulting *.msi* in Orca from the Windows Installer SDK
2. Select the Binary table
3. Click the Tables menu and then Add Row
4. Enter, for example, InstallUtil64 for the Name
5. Select the Data row and click the Browse button
6. Browse to *%WINDIR%Microsoft.NETFramework64v2.0.50727*
7. Select *InstallUtilLib.dll*
8. Click the Open button
9. Click the OK button
10. Select the CustomAction table
11. For each custom action where the Source column is InstallUtil and only those
    custom actions that are 64-bit managed custom actions (or that were built with `/platform:anycpu`,
    the default, where you want to run as 64-bit custom actions), change the value
    to, for example, InstallUtil64

This only affects DLLs build with `/target:library`. Managed EXEs will run correctly according to what platform they target.

Tags 64-bit Development Installation

Join the conversation

Add Comment

12 years ago

VS2017 VSUpdate
Windows 7 WiX

Related Resources

Visual Studio Product Website

Visual Studio Developer Cente

Log Collection Utility

Visual Studio Setup and Installation

Archives

April 2018 (1)
January 2018 (1)
November 2017 (1)
October 2017 (2)
September 2017 (2)
All of 2018 (2)
All of 2017 (16)
All of 2016 (2)
All of 2015 (7)
All of 2014 (10)
All of 2013 (3)
All of 2012 (21)
All of 2011 (9)
All of 2010 (16)
All of 2009 (21)
All of 2008 (32)
All of 2007 (54)
All of 2006 (80)
All of 2005 (82)
All of 2004 (9)

Raghu

I am facing the same probelm when I use InstallShield 9. Is there any work around?.. or do i need to upgrade the InstallShield?.

Heath Stewart (MSFT) *12 years ago*

Raghu, you'll have to speak with InstallShield about that. You could first try using their forums at http://community.macrovision.com/forumdisplay.php?f=133.

"Sharepoint User Group" on LinkedIn *10 years ago*

Building an MSI in Visual Studio 2005/2008 to work on a SharePoint 64 bit installation with a Custom Action!

Microsoft Deployment Technology *10 years ago*

If you create custom action through Visual studio 2005 which targets x64, You would see the Error During

---

CarlosAg Blog *9 years ago*

Introduction IIS 7 provides a rich extensibility model, whether extending the server or the user interface,

---

Chris Crowe's Blog *9 years ago*

64-bit Managed Custom DLL Actions with Visual Studio do not work properly

---

Matthew White *7 years ago*

please help, Upon trying the above I am receving 'Error 1001, InstalUtilLib.dll Unknown error' VS 2010 using managed custom actions

Heath Stewart (MSFT) *7 years ago*

@Matthew, make sure you're using the correct bitness of InstallUtilLib.dll. .NET installs both 32- and 64-bit flavors on a 64-bit machine, and the right bitness must be used for the managed class to be installed correctly.

Sanjay Singh *7 years ago*

@Matthew, may be you need to use correct InstallUtilLib.dll. I was getting the same error, then i selected the InstallUtilLib.dll on "%WINDIR%Microsoft.NETFramework64v4.0.30319" in above mentioned step because my application was built with .net framework 4.0 . Now my problem is resolved and its working fine.

.Net *7 years ago*

Where/How to find this "1.Open the resulting .msi in Orca from the Windows Installer SDK" in VS2010

Heath Stewart (MSFT) *7 years ago*

@.Net, You need to install the Windows SDK from the Download Center (newest Windows SDK is best). Then in the "bin" folder of the installation directory (ex: %ProgramFiles%Microosft SDKsWindowsv7.0Abin) there is Orca.msi. Double-click to install that. After installing, you can right-click on your MSI and select Edit with Orca.

Jon *7 years ago*

Thank you for writing this – it has been very helpful to me. Is there any way that this could be automated in visual studio? Perhaps with a "PostBuildEvent" or something? If so, could you provide any guidance on how to approach it?

Heath Stewart (MSFT) *7 years ago*

@Jon, take a look at blogs.msdn.com/.../696833.aspx. The basic pricinpal is there but you'll need to write your own script.

But really this isn't the right way. Managed CAs are best to avoid, but if you choose to use them use DTF in WiX @ http://wix.source-forge.net. This creates an isolated remoting service that avoids the pitfalls of managed code.

Santhosh K.L

7 years ago

Hi,

I am creating a setup file or .msi to register SOAP DLL.But i want register 32 bit and 64bit dll in ine setup.

Please help me fast.

Heath Stewart (MSFT)

7 years ago

@Santhosh K.L, sorry, but that is not supported. 64-bit content must be registered using a 64-bit MSI. That's not to say you couldn't have a custom action run to register 64-bit content, but 1) it must itself be 64-bit (AnyCPU for managed code runs native to the OS), 2) needs to be properly conditions to only run on 64-bit machines (VersionNT64), and 3) should really avoid managed code (i.e., harvest/reauthor the registration into MSI; but this would re-quire unsupported authoring of 64-bit components in a 32-bit MSI).

It's best to have separate installers for 32- and 64-bit code to avoid all the troubles that can occur. See blogs.msdn.com/.../different--packages-are-required-for-different-processor-architectures.aspx for more information.

Jack

7 years ago

Why are managed custom actions "best to avoid"? How would I install Windows services, WMI schemas, etc?

@Jack, please read blogs.msdn.com/.../custom-action--guidelines.aspx and links therein. You can install services natively using the ServiceInstall and ServiceConfig tables whether they are managed or not, though if you install managed service assemblies (or its dependent assemblies) to the GAC this will not work. It is not recommended that you install service assemblies to the GAC.

For WMI, you can write a native DLL custom action that cocreates CLSID_MofCompiler (implement IMofCompiler: msdn.microsoft.com/.../aa390865(VS.85).aspx) and compiles either your file you installed already (so schedule the CA after InstallFiles as a deferred CA) or compiles a binary blob you could store in the Binary table (to be able for admins to rerun it, the former is recommended.

Khayralla *5 years ago*

what if I have .net 4 x64 dll and a .net 2 x64 dll ?

Heath Stewart (MSFT) *5 years ago*

@Khayralla, please see http://wixtoolset.org. This is a far better way to do managed CAs – though you should avoid using CAs anyway – because it remotes the managed CA to a separate process so the .NET Framework version doesn't really matter. That site also has support links for additional help.

Anup *4 years ago*

But this adds an overhead of modifying the msi after its creation. What changes are needed in installer project to achieve the same without using ORCA

Nicolas *6 months ago*

I see the same issue with Visual Studio 2017 and the Visual Studio 2017 Project installer.

Building the MSI works fine. After running the MSI to install the

64bit service the System.BadImageFormatException is thrown. When manually installing the service via the 64bit version of installutil.exe works fine – the 32bit version of it throws the same error like installing via the MSI.

Heath Stewart (MSFT)

6 months ago

If you're going to use managed custom actions, you need to make sure you're using the right bitness of the InstallUtil.exe – your FileSearch needs to use the Framework64 directory. The very basic Visual Studio installer projects don't really support that level of authoring. I suggest you look at http://wixtoolset.org.

Nicolas

6 months ago

Hi Heath,

thanks for your reply. I know what the issue is but I see this as something that should be fixed. If building a proper installer with custom actions for 64bit services, the proper flavor of installutil.exe should be used, right? Is there a way to file a bug to get this fixed?

Heath Stewart (MSFT)

6 months ago

You can file it at https://developercommunity.visualstudio.com and it will get redirected to the appropriate team.

Nicolas

6 months ago

Thanks Heath. Appreciate your support. Lookin into the wixtoolset in the meantime.