

Deploying a Visual Studio 2010 Tools for Office Solution Using

08/17/2010 25 minutes to read

Windows Installer

Summary: Learn how to deploy a Microsoft Visual Studio Tools for the Office system 2010 add-in or document-level solution using a Visual Studio 2010 setup project to create a Windows Installer package that targets the 2007 Microsoft Office system or Microsoft Office 2010.

Wouter van Vugt, Code Counsel

Ted Pattison, Ted Pattison Group

This article was updated by Microsoft with permission from the original authors.

Applies to: Visual Studio 2010 Tools for Office, 2007 Microsoft Office system, Microsoft Office 2010, Visual Studio 2010.

Download: <http://code.msdn.microsoft.com/VSTO2010MSI>

Contents

- Overview
- Deployment Methods
- Deploying Office solutions that target the Visual Studio 2010 Tools for Office runtime
- Download Samples Provided with this Article
- Creating a Basic Installer
- Conclusion
- Additional Resources
- About the Authors

Overview

You can develop a Visual Studio 2010 Tools for Office solution for the 2007 Microsoft Office system or Microsoft Office 2010, and deploy the solution by using a Visual Studio 2010 Setup project to create a Windows Installer package. The discussion includes steps for deploying a simple Office add-in.

For a discussion about how to deploy a solution built with Visual Studio Tools for Office 3.0, see [Deploying a Visual Studio Tools for the Office System 3.0 Solution for the 2007 Microsoft Office System Using Windows Installer \(Part 1 of 2\)](#) and [Deploying a Visual Studio Tools for the Office System 3.0 Solution for the 2007 Microsoft Office System Using Windows Installer \(Part 2 of 2\)](#).

Deployment Methods

You can use ClickOnce to create and install self-updating applications with minimal user interaction. This has an automated mechanism for easily distributing updates to your application. However, ClickOnce is not capable of deploying components that require administrative privilege such as machine level add-ins.

For solutions that require administrative privilege you can use Windows Installer to deploy a Visual Studio Tools for Office customization. Windows Installer allows you to configure the deployment in great detail, but this does require more effort to configure correctly. You lose the simple ease of deploying using ClickOnce technology and advantages such as automated updates, but you receive benefits of configurability.

For an overview of how to deploy a Visual Studio Tools for the Office 2010 solution using ClickOnce, see [Deploying Solutions for the 2007 Office System with ClickOnce Using Visual Studio 2008 Professional](#).

Deploying Office solutions that target the Visual Studio 2010 Tools for Office runtime

Both ClickOnce and Windows Installer packages need to perform the same rudimentary tasks when installing an Office solution.

1. Install prerequisite components on the user computer.
2. Deploy the solution specific components.
3. For add-ins, create registry entries.
4. Trust the solution to allow it to execute.

Required Prerequisite Components on the Target Computer

To run Visual Studio 2010 Tools for Office solutions, the following software must be available on the user computer.

- The 2007 Microsoft Office system or Microsoft Office 2010.
- The Microsoft .NET Framework
Visual Studio Tools for the Office system 2010 can work with the Microsoft .NET Framework 3.5 or Microsoft .NET Framework 4.
- The Microsoft Visual Studio 2010 Tools for Office Runtime.
Visual Studio 2010 Tools for Office Runtime provides a runtime environment that manages add-ins and document-level solutions.
- The primary interop assemblies for the 2007 Microsoft Office system or Microsoft Office 2010.
- Any utilities assemblies referenced by projects that target the .NET Framework 4.

Solution Specific Components

The installer package must install the following components to the user computer.

- The Microsoft Office document, if you create a document-level solution.
- The customization assembly and any assemblies it requires.
- Additional components such as configuration files.
- The application manifest (.manifest).
- The deployment manifest (.vsto).

Registry Entries for Add-ins

Add-ins require a set of registry entries for the Microsoft Office application to locate the add-in. You should create

the registry entries as part of the deployment process. For more information about registry entries pertaining to a

Visual Studio Tools for Office add-in, see [Registry Entries for Application-Level Add-Ins](#). When developing an Outlook add-in that displays custom form regions you should create additional registry keys as part of the installation procedure to allow the form regions to be identified. For more information about registry entries pertaining to Outlook form regions, see [Specifying Form Regions in the Windows Registry](#).

Document-level solutions do not require these registry entries because the customization is located by using

information inside the Microsoft Office document. For more information about the document properties used for

locating document-level customizations, see [Custom Document Properties Overview](#).

Trusting the Visual Studio Tools for Office Solution

Before you allow the customization to execute, a solution must be trusted. This trust relationship is based on the

signed application and deployment manifests. You can either sign the manifests with a certificate that identifies a

known and trusted publisher, or create a trust-relationship at installation time with an inclusion list entry or installing to a trusted location. For more information about how to obtain a certificate for signing, see

[ClickOnce](#)

[Deployment and Authenticode](#). For more information about using the user inclusion list, see [Trusting Office Solutions by Using Inclusion Lists](#).

If neither option is used, a trust prompt is displayed to the users to let them decide whether to trust the solution.

You can add an inclusion list entry with a custom action in your Windows Installer file. For more information about

enabling the inclusion list, see [How to: Configure Inclusion List Security](#).

Document-level solutions have an extra requirement where the document location must also be trusted before the

Office application allows the customization to load. For more information about security related to document-level

solutions, see [Granting Trust to Documents](#).

Download Samples Provided with this Article

The walkthroughs provided in this article use various sample files contained in the download. The download contains the source code for all sample projects created in this article.

By default, these files are installed into the following folder:

%ProgramFiles%\Microsoft Visual Studio Tools for the Office system 2010
Resources\Windows Installer Sample\

This article refers to this installation folder as {SamplesDir}.

Download:<http://code.msdn.microsoft.com/VSTO2010MSI>

Creating a Basic Installer

The Setup and Deployment project template in Visual Studio 2010 can be configured to create a Windows Installer

package that deploys a Visual Studio Tools for Office solution.

To create an installer for a Office solution, the following high level tasks must be accomplished.

- Add the components of the Office Solution that will be deployed.
- For application-level add-ins, configure registry keys.

Configure prerequisite components so that any missing pre-requisites can be installed on the end users computers. Configure launch conditions to verify that the required prerequisite components are available.

These

would be used to block the install in case all required pre-requisites are not installed. Start by creating the setup

project. This project contains the configuration for the Windows Installer package. Part of the configuration is

performed using built-in features. Additional tasks are performed using custom actions.

To create the AddInSetup project

1. Open the Office AddIn Project you want to deploy using Windows Installer.
2. With the Office Project Open, on the **File** menu, expand **Add** and click **New Project** to add a new project.
3. In the **Add New Project** dialog box, in the **Project types** pane, expand **Other Project Types** and then expand **Setup and Deployment** and then select **Visual Studio Installer**.
4. In the **Templates** pane, select **Setup Project** from the **Visual Studio installed** templates group.
5. In the **Name** box, type **OfficeAddInSetup**.
6. Click **Open** to create the new setup project. Visual Studio opens the File Explorer for the new setup project. This explorer allows you to add files to the setup project from locations such as the projects inside your solution or loose files from your file-system.

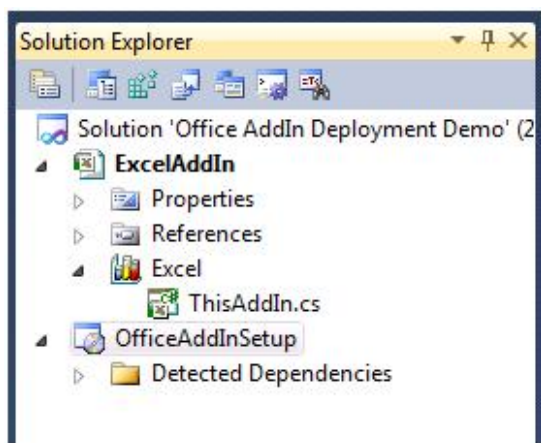


Figure 1: Solution Explorer for the setup project

The setup project needs to deploy the ExcelAddIn DLL. You can configure the setup project for this task by adding

the ExcelAddIn project output to the setup project.

To add the ExcelAddIn project output

1. In the **Solution Explorer**, right-click **OfficeAddInSetup**, click **Add** and then **Project Output**.
2. In the **Add Project Output Group** dialog box, confirm that the **ExcelAddIn** project is selected, and the **Primary Output** option is selected.

3. Click **OK** to add the project output to the setup project.

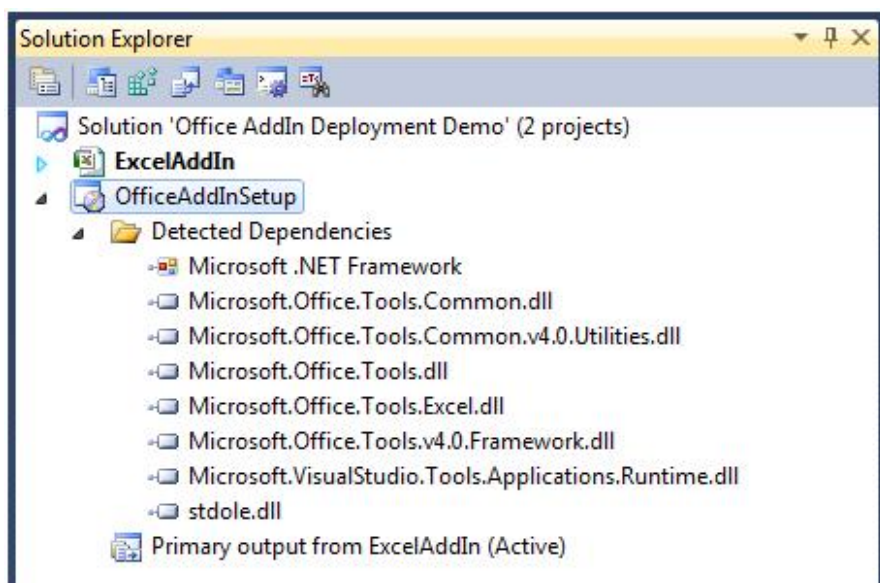


Figure 2: Dependencies for the setup project

The setup project needs to deploy the deployment manifest and application manifest. You can add these two files

to the setup project as stand-alone files from the output folder of the ExcelAddIn project.

To add the deployment and application manifests

1. In the **Solution Explorer**, right-click **ExcelAddInSetup**, click **Add**, and click **File**.
2. In the **Add Files** dialog box, navigate to the **ExcelAddIn** output directory. Usually the output directory is the **bin\release** subfolder of the project root directory, depending on the selected build configuration.
3. Select the **ExcelAddIn.vsto** and **ExcelAddIn.dll.manifest** files and click **Open** to add these two files to the setup project.

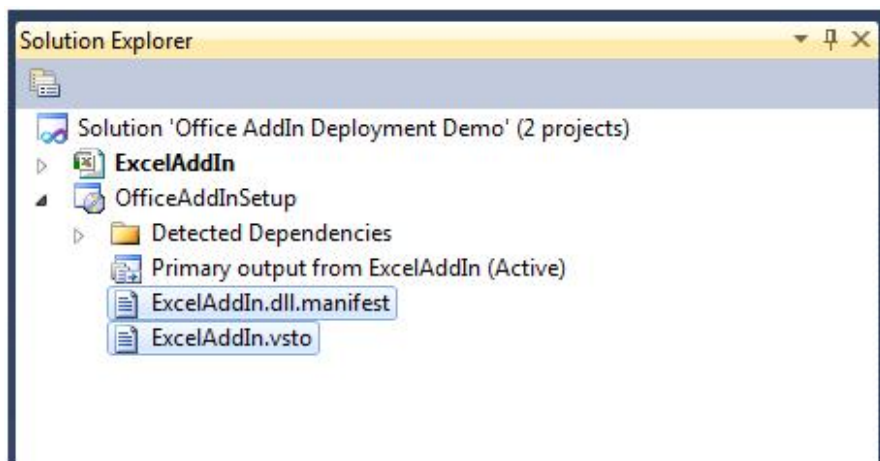


Figure 3: Application and deployment manifests for the add-in

The downside of adding loose files to a setup project is that you reference the default output folder of the Release build configuration explicitly. When you change the build configuration back to Debug, the setup project still references the files from this folder. Because you should only distribute release builds, this is acceptable. Alternatively, you can configure the Debug and Release build configuration to output their files to the same directory.

Referencing the ExcelAddIn includes all the components that ExcelAddIn requires. These components must be excluded and deployed using prerequisite packages to allow them to be registered correctly. Also, the Software License Terms must be displayed and accepted before the installation begins.

To exclude the ExcelAddIn project dependencies

1. In the **Solution Explorer**, in the **OfficeAddInSetup** node, select all dependency items beneath the **Detected Dependencies** item except for **Microsoft .NET Framework** or any assembly that ends with ***.Utilities.dll**. The Utilities assembly is only present when your Office solution targets .NET 4 and the assembly is meant to be deployed along with your application.
2. Right-click the group and select **Properties**.
3. In the **Properties** window, change the **Exclude** property to **True** to exclude the dependent assemblies from the setup project. Make sure to not exclude any Utilities assemblies.

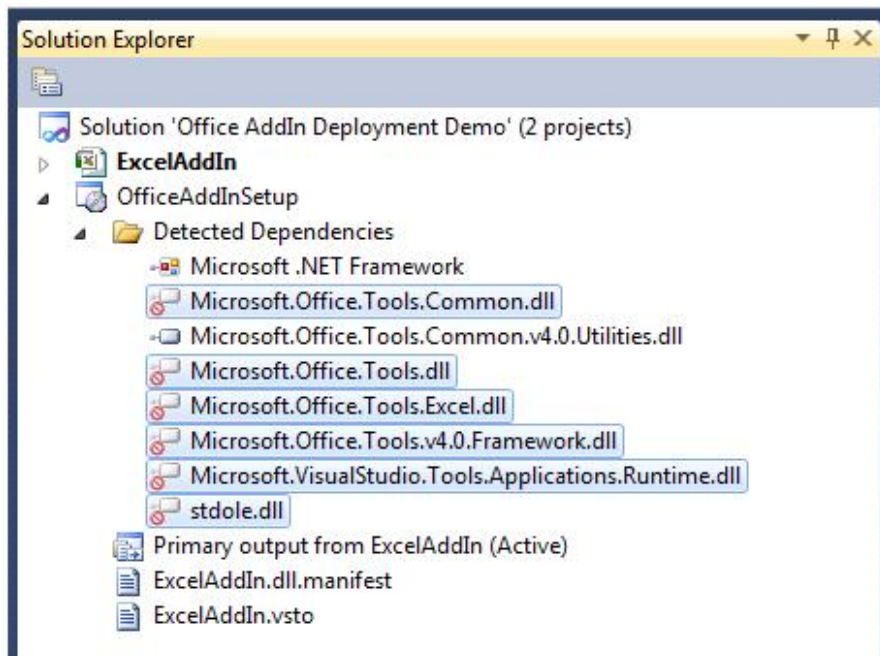


Figure 4: Excluding dependencies

You can configure your Windows Installer package to install the prerequisite components by adding a Setup program, also known as a bootstrapper. The Setup program can include and install the redistributables provided by

the component vendors, a process called bootstrapping. For the ExcelAddIn, the following prerequisites must be

installed before the add-in can run correctly.

- The Microsoft .NET Framework version that the Office Solution targets - .NET Framework 3.5 SP1, .NET Framework 4 Client Profile, or .NET Framework 4.
- Microsoft Visual Studio 2010 Tools for Office Runtime.
- Windows Installer 3.1.
- If your solution targets .NET Framework 3.5, you will also have to redistributable Primary Interop Assemblies for either the 2007 Microsoft Office system or Microsoft Office 2010.

To configure dependent components as prerequisites

1. In the **Solution Explorer**, right-click the **OfficeAddInSetup** project and select **Properties**.
2. The **OfficeAddInSetup Property Pages** dialog box appears.
3. Click **Prerequisites**.
4. In the **Prerequisites** dialog box, perform the following tasks.

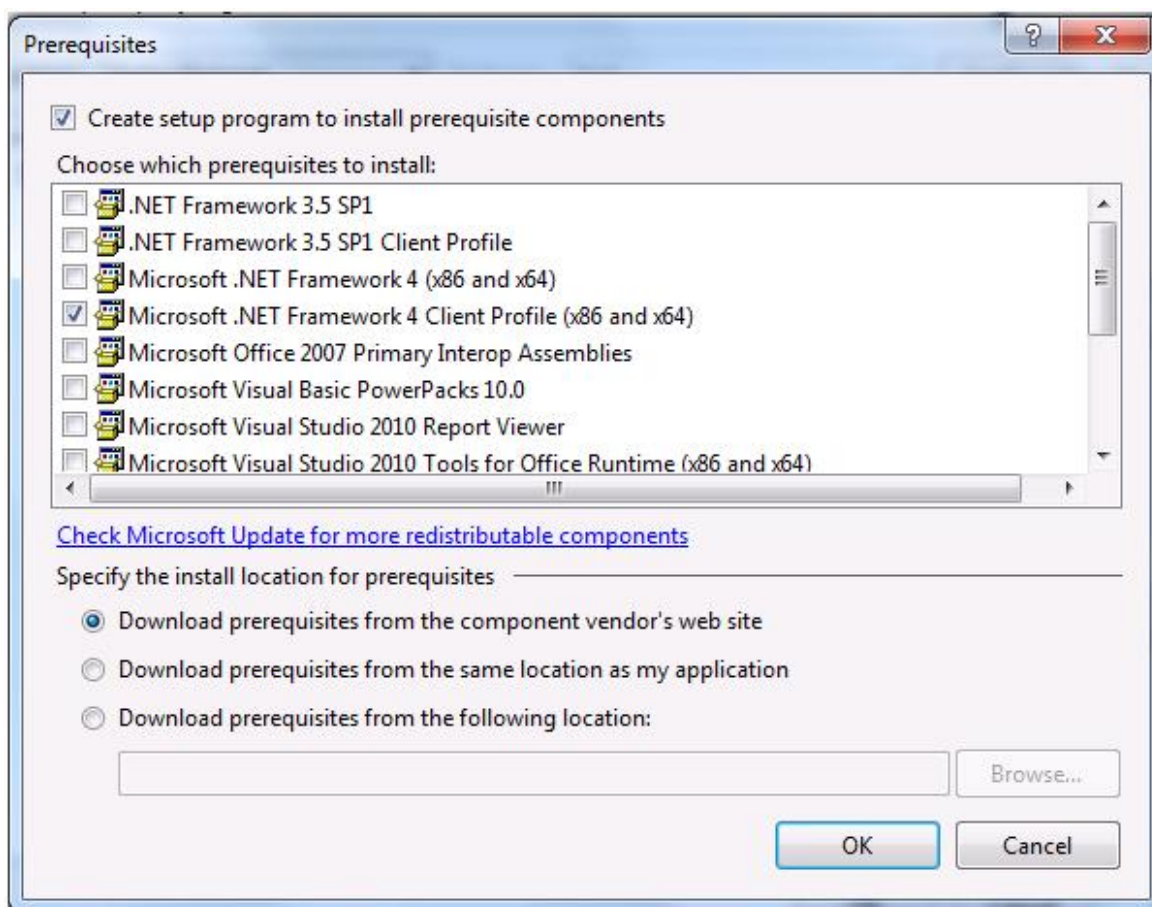


Figure 5: Prerequisites Dialog Box

Note:

The configured prerequisite packages in your Visual Studio 2010 Setup Project are dependent on the selected build configuration. You must select the right prerequisite components for each build configuration that you use. The samples and procedures only use the default **Release** build configuration.

Microsoft Office locates the application-level add-in by using registry keys specific to each Office Application. Keys in the HKEY_CURRENT_USER hive will register the add-in for each individual user of the machine whereas keys under the HKEY_LOCAL_MACHINE hive will register the add-in for all users of the machine.

For more information about registry keys, see [Registry Entries for Application-Level Add-Ins](#).

To configure the registry

1. In the **Solution Explorer**, right-click **OfficeAddInSetup**.
2. Expand **View**.
3. Click **Registry** to open the registry editor window.
4. In the **Registry(ExcelAddInSetup)** editor, expand **HKEY_LOCAL_MACHINE** and then **Software**.
5. Delete the **[Manufacturer]** key found under **HKEY_LOCAL_MACHINE\Software**. This key is automatically added when the setup project is created and is not used by the add-in.
6. Expand **HKEY_CURRENT_USER** and then **Software**.
7. Delete the **[Manufacturer]** key found under **HKEY_CURRENT_USER\Software**. This key is also added automatically and isn't used by the add-in.
8. To add registry keys for the add-in installation right-click the **User/Machine Hive** key, select **New** and then **Key**. Use the text **Software** for the name of the new key. Right-click on the newly created **Software** key and create a new key with the text **Microsoft**.
9. Use a similar process to create the entire key hierarchy required for the add-in registration. The following key hierarchy is used for the ExcelAddIn add-in.

	 Copy
User/Machine	
Hive\Software\Microsoft\Office\Excel\Addins\SampleCompany.ExcelAddIn	

Note:

The name of your company is used as a prefix for the name of the add-in. This provide uniqueness for this part of the registry key and allows add-ins with similar names originating from different suppliers to work without accidentally

overwriting each other's registration keys. This does not provide full uniqueness but should suffice to prevent collisions.

-
10. Right-click the **SampleCompany.ExcelAddIn** key, select **New** and click **String value**. Use the text **Description** for the new value.
-
11. Use this step to add three more values. Use the following names and data type.
-
- a. **FriendlyName** of type **String**
-
- b. **LoadBehavior** of type **DWORD**
-
- c. **Manifest** of type **String**
-
12. Right-click the **Description** value in the registry editor and click **Properties Window**. In the **Properties Window** enter **Excel Demo AddIn** for the **Value** property.
-
13. Select the **FriendlyName** key in the registry editor. In the **Properties Window**, change the **Value** property to **Excel Demo AddIn**.
-
14. Select the **LoadBehavior** key in the registry editor. In the **Properties Window**, change the **Value** property to **3**.
-
- The value 3 for the **LoadBehavior** value indicates that the add-in should be loaded at startup of the host application. For more information about load behavior, see [Registry Entries for Application-Level Add-Ins](#).
-
15. Select the **Manifest** key in the registry editor. In the **Properties Window**, change the **Value** property to **[TARGETDIR]ExcelAddIn.vsto|vstolocal**

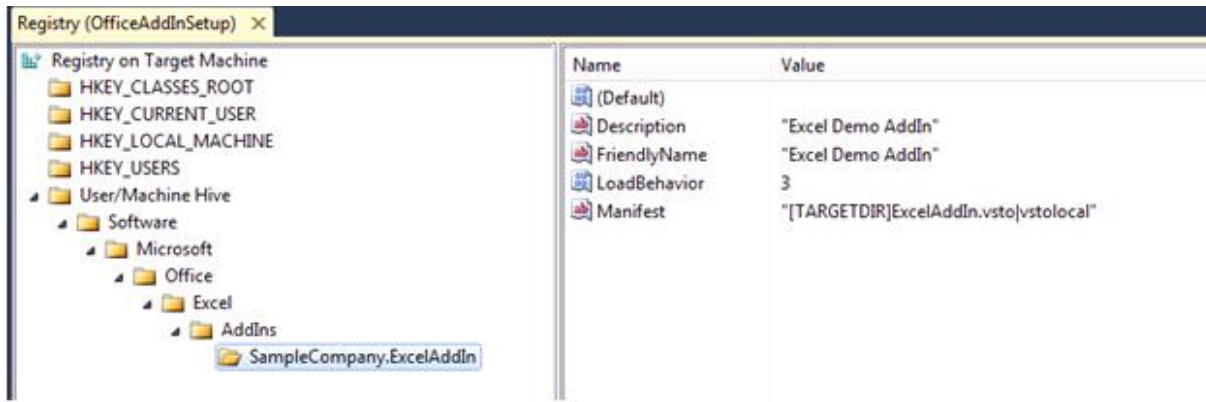


Figure 6: Setting up registry keys

Note:

There are a few interesting notes on how the Visual Studio Tools for Office runtime locates the add-in. Visual Studio Tools for Office runtime searches for the deployment manifest indicated by the Manifest value. The [TARGETDIR] part is a macro that the Windows Installer expands to the actual folder where the add-in is installed. This macro expands to include the trailing \ character, so the filename of the deployment manifest is appended as ExcelAddIn.vsto without the \ character. Finally there is the use of the **vstolocal** postfix. This tells the Visual Studio Tools for Office runtime that the add-in should load from the location indicated by the Manifest value, and not loaded into the ClickOnce cache. Removing this postfix will cause the runtime to copy the customization into the ClickOnce cache.

Warning:

You should be very careful with the Registry Editor in Visual Studio. For example, if you accidentally set DeleteAtUninstall for the wrong key, you might delete an active part of the registry, leaving the user computer in an inconsistent, or even worse, broken state.

Special considerations for Registering All User Add-ins for Office 2010 64-bit: Office 2010 64 bit looks for the add-in in registry keys under the 64-bit hive. In order to register add-ins under the native 64-bit registry hive, the setup project's target platform must be set to 64-bit only.

1. Select the **OfficeAddInSetup** project in solution explorer.
2. Go to **Properties** window and set **TargetPlatform** property to **x64**.

You cannot have a single MSI that will install for 32-bit as well as 64-bit Office versions. You must create two separate MSI packages that target the 32-bit and 64-bit versions of Office separately.

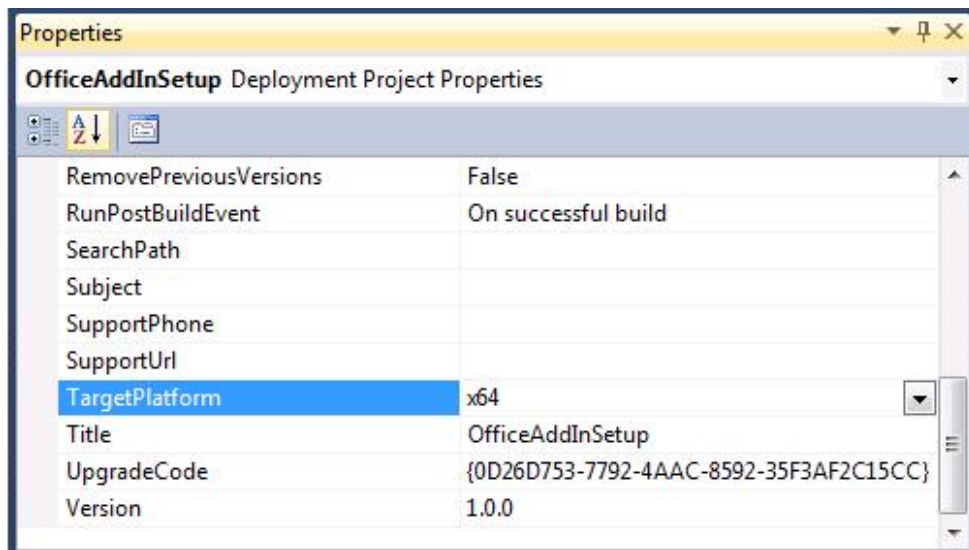


Figure 7: Target Platform for registering add-ins with 64-bit Office 2010

When the user does not execute the setup bootstrapper (setup.exe), it is possible that the MSI package will install

on a computer without the required prerequisites. Launch conditions are used to prevent the ExcelAddIn from installing on computers without installed prerequisites.

To see if the Visual Studio 2010 Tools for Office runtime is installed, check the following registry keys on the user computer.

To configure launch conditions for Visual Studio 2010 Tools for Office Runtime

1. In **Solution Explorer**, right-click **OfficeAddInSetup**.
2. Expand **View**.
3. Click **LaunchConditions**.
4. In the **Launch Conditions(OfficeAddInSetup)** editor, right-click **Requirements on Target Machine**, and then click **Add Registry Launch Condition**. This search condition searches the registry for a key the Visual Studio Tools for Office runtime installs. The value of the key is then available to the various pieces of the installer through a named property. The launch condition uses the property defined by the search condition to check for a certain value.
5. In the **Launch Conditions(OfficeAddInSetup)** editor, select the **Search for RegistryEntry1** search condition, right-click the condition and select **Properties Window**.
6. In the **Properties** window, perform the following tasks.

7. In the **Launch Conditions(OfficeAddInSetup)** editor, select the **Condition1** launch condition, right-click the condition and select **Properties Window**.
8. In the Properties window, perform the following tasks.
 - a. Set **(Name)** to **Verify VSTO 2010 Runtime availability**.
 - b. Change the value of the **Condition** property to the following **VSTORUNTIMEREDIST >= "10.0.30319"**
 - c. Leave the **InstallURL** property blank.
 - d. Change the value of the **Message** property to **The Visual Studio 2010 Tools for Office Runtime is not installed. Please run Setup.exe**.

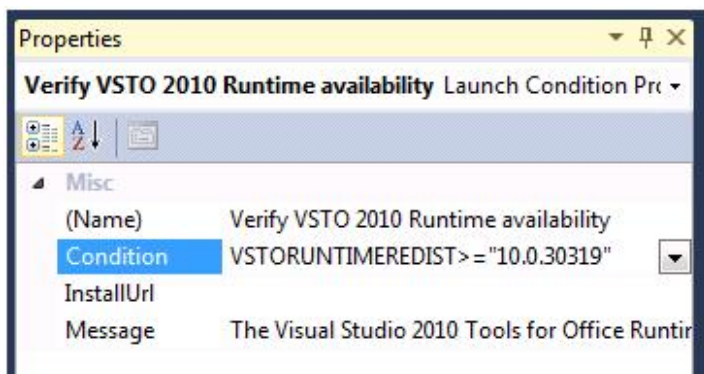


Figure 9: Properties Window for the Verify Runtime Availability launch condition

The Visual Studio 2010 Tools for Office runtime also ships with Microsoft Office 2010. However at the time of Office

2010 RTM, the runtime with Office only supports Office solutions that target the .NET Framework 3.5. If your solution targets the .NET Framework 3.5, it can run either if Office 2010 is installed or if the Visual Studio 2010

Tools for Office Runtime redistributable is installed. If your Office solutions target the .NET Framework 4, you must

redistribute the Visual Studio 2010 Tools for Office runtime. The launch condition above explicitly checks for the

presence of the redistributable runtime. The following procedure demonstrates how you can modify the launch

condition so that it will also check for the presence for Microsoft Office 2010.

To configure launch conditions for Visual Studio 2010 Tools for Office Runtime

1. In the **Launch Conditions(OfficeAddInSetup)** editor, right-click **Search Target Machine**, and then click **Add Registry Search**. Configure this additional search condition to search for the VSTO Runtime installed through Office 2010. The launch condition can use the property defined by the search

condition in

addition to the property for the RUNTIMEREDIST to check whether an appropriate VSTO Runtime is present on the end users system.

2. Select the **Search for RegistryEntry1** search condition, right-click the condition and select **Properties Window**.
3. In the **Properties** window, perform the following tasks.
 - a. Set the value of **(Name)** to Search for Office 2010 VSTO Runtime.
 - b. Change the value of **Property** to **OfficeRuntime**.
 - c. Set the value of **RegKey** to **SOFTWARE\Microsoft\VSTO Runtime Setup\v4**
 - d. Leave the **Root** property set to **vsdrrHKLM**.
 - e. Change the **Value** property to **Version**.
4. In the **Launch Conditions(OfficeAddInSetup)** editor, select the **Verify VSTO 2010 Runtime availability** launch condition defined earlier, right-click the condition and select **Properties Window**.
5. Change the value of the **Condition** property to the following **VSTORUNTIMEREDIST>="10.0.30319" OR OFFICERUNTIME>="10.0.21022"**

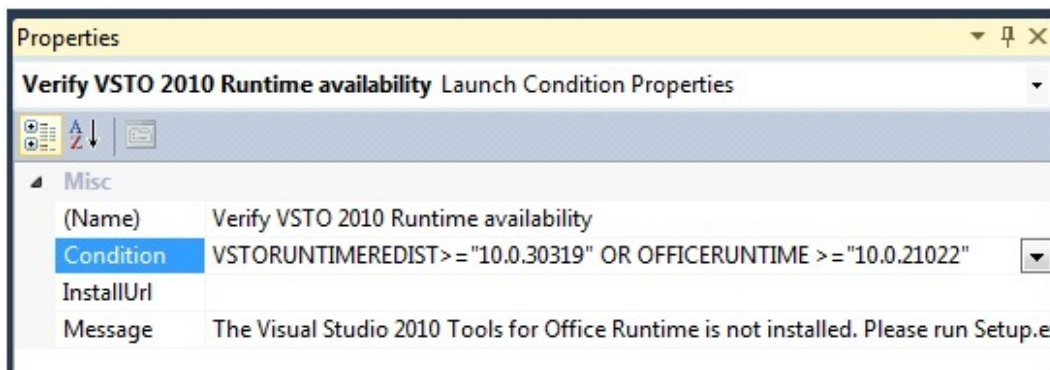


Figure 11: Properties Windows for the Verify Runtime Availability through Redist or Office 2010 launch condition

To configure launch conditions for Office 2010 Shared PIA

1. In the **Launch Conditions(ExcelAddInSetup)** editor, right-click **Requirements on Target Machine**, and then click **Add Windows Installer Launch Condition**. This launch condition searches for the Microsoft Office 2010 Shared Primary Interop Assembly by searching for the specific component ID.
2. Right-click **Search for Component1** and click **PropertiesWindow** to show the properties of the launch condition.

3. In the **PropertiesWindow**, change the values of the following properties:
- a. Set the value of the **(Name)** property to **Search for Office 2010 Shared PIA**

b. Set the value of the **ComponentID** property to {64E2917E-AA13-4CA4-BFFE-EA6EDA3AFCB4}.

c. Set the value of the **Property** property to **HASSHAREDPIA**.
4. In the **Launch Conditions(ExcelAddInSetup)** editor, right-click **Condition1** and click **Properties Window** to show the properties of the launch condition.
5. Set the properties of **Condition1** using the following information:
- a. Set **(Name)** to **Verify Office 2010 Shared PIA availability**.

b. Set **Condition** to **HASSHAREDPIA**.

c. Leave **InstallUrl** blank.

d. Set **Message** to **A required component for interacting with Excel is not available. Please run setup.exe**.

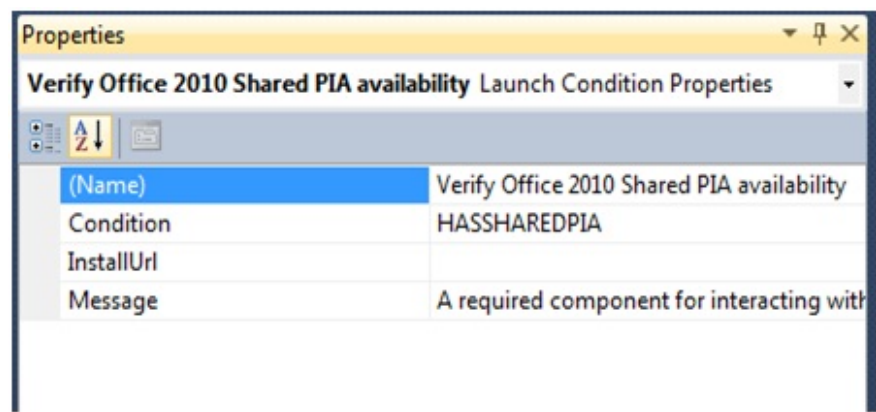


Figure 13: Properties Window for the Verify Office Shared PIA launch condition

You can further refine the launch conditions for the ExcelAddIn installation. For instance, it is useful to verify if the actual target Office application is installed because each individual component can be disabled by the user.

Component IDs of the Redistributable Primary Interop Assemblies for Microsoft Office 2010

Primary interop assembly	Office 2010 component ID
Excel	{EA7564AC-C67D-4868-BE5C-26E4FC2223FF}
InfoPath	{4153F732-D670-4E44-8AB7-500F2B576BDA}

Outlook	{1D844339-3DAE-413E-BC13-62D6A52816B2}
PowerPoint	{EECBA6B8-3A62-44AD-99EB-8666265466F9}
Visio	{3EA123B5-6316-452E-9D51-A489E06E2347}
Word	{8B74A499-37F8-4DEA-B5A0-D72FC501CEFA}
Microsoft Forms 2.0	{B2279272-3FD2-434D-B94E-E4E0F8561AC4}
Microsoft Graph	{011B9112-EBB1-4A6C-86CB-C2FDC9EA7B0E}
Smart Tag	{7102C98C-EF47-4F04-A227-FE33650BF954}
Office Shared	{64E2917E-AA13-4CA4-BFFE-EA6EDA3AFCB4}
Project	{957A4EC0-E67B-4E86-A383-6AF7270B216A}

For information about the Microsoft Office 2007 component IDs, see [Deploying a Visual Studio Tools for the Office System 3.0 Solution for the 2007 Microsoft Office System Using Windows Installer \(Part 1 of 2\)](#).

The final set of launch conditions should resemble the following image.

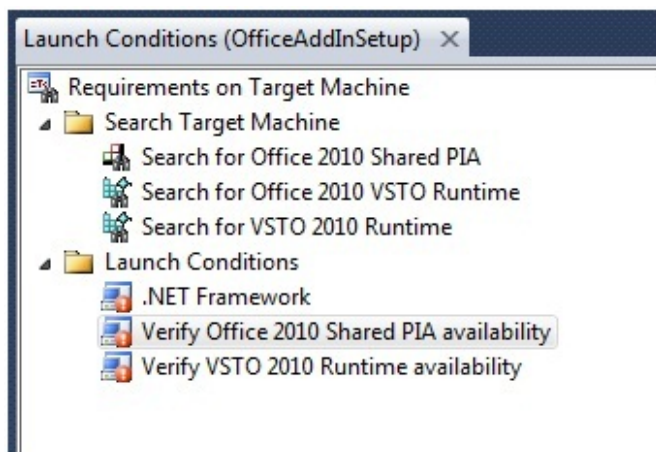


Figure 14: Final launch conditions

The setup project is not compiled as part of a general solution compilation because the build can take a long time on larger projects. You need to manually compile the setup project and copy the resulting files to an accessible location.

To build the setup project

1. In the **Solution Explorer**, right-click the **OfficeAddInSetup** project and click **Build**.
2. Using **Windows Explorer**, navigate to the output directory of the **OfficeAddInSetup** project and go to the Release folder. Copy all files from the Release folder to a location that users can access. Sample locations include a network share, CD or USB stick.

It is a good idea to test the basic setup before you add more advanced features to the setup project. You should perform the following steps on a test computer where the add-in is not installed.

To test the ExcelAddIn setup

1. Navigate to the location where you copied **OfficeAddInSetup** in the previous procedure.
2. Right-click the setup.exe file and click **Open** to install the **OfficeAddInSetup** add-in. Accept any Software License Terms that appear, and complete the setup wizard to install the add-in on the user computer.

The Excel Office solution should install and run from the location specified during setup.

Additional Requirements for Document-level Solutions

Deploying document-level solutions require a few different configuration steps in the Windows Installer setup project.

Document-level solutions do not make use of the registry. Instead, the location of the deployment manifest is defined using custom document properties in the Microsoft Office file. By default, the location uses a relative path to the deployment manifest, meaning that if the document is installed into the same folder as the customization assembly you do not need to alter the manifest location contained within the document. However, if you intend to deploy the document to a different folder as the customization assembly, you need to update the document properties to a fully qualified path.

For more information about document properties used by Visual Studio Tools for Office, see [Custom Properties Overview](#).

The following list outlines the basic steps required for a document-level solution.

- Create the Visual Studio 2010 Setup Project.

- Add the primary output of your document-level solution. The Microsoft Office document itself is also part of the primary output so you do not add it separately to the setup project.
- Add the deployment and application manifests as loose files.
- Exclude the dependent components from the installer package. (except for any utilities assemblies).
- Configure prerequisite packages.
- Configure launch conditions.
- Build the setup project and copy the results to the deployment location.
- Deploy the document-level solution on the user computer by executing the setup.

Changing the Location of the Deployed Document

The download accompanying this article contains a sample custom action you use to move the document to the **My Documents** folder and re-attach the customization. The custom action uses the **ServerDocument** class to re-attach the customization.

The following steps assume you have created a similar setup project for a document-level solution called **ExcelWorkbookProject** with a setup project called **ExcelWorkbookSetup**. The document-level solution sample provided in the download accompanying this article is based on the Excel 2010 Workbook. The setup project has been configured according to the steps outlined for creating an add-in installer, except for the registry keys that are not used by document-level solutions.

To add the custom action project to your Visual Studio solution

1. In **Windows Explorer**, under the **{SamplesDir}** folder and find the **Document Deployment** folder for the appropriate .NET framework target.
2. Copy the language-specific **AddCustomizationCustomAction** folder to the root folder of your document-level solution.
3. In the **Solution Explorer**, right-click the **Office Document Deployment Project** solution
4. Expand **Add** and click **Existing Project**.
5. Navigate to the folder inside the solution root directory that contains the **AddCustomizationCustomAction** project.
6. Select **AddCustomizationCustomAction.vbproj** or **AddCustomizationCustomAction.csproj**.
7. Click **Open** to add the project to the solution.

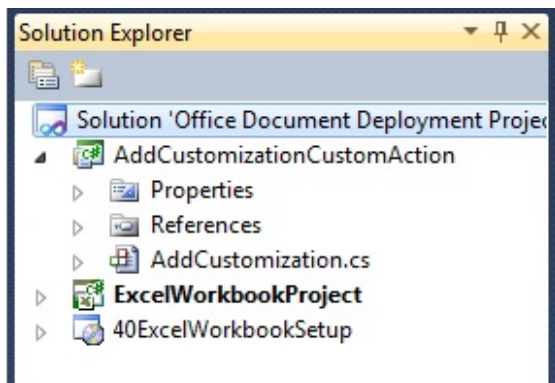


Figure 15: Solution Explorer - AddCustomizationCustomAction

The custom action requires the solution ID of your Visual Studio Tools for Office document-level solution. This value is retrieved from the Visual Studio project file. To begin, compile the document-level solution to add the solution ID to this project file.

To retrieve the solution ID

1. On the **Build** menu, click **Build Solution** to build the document-level solution and add the solution ID property to the project file.
2. In the **Solution Explorer**, right-click the document-level project **ExcelWorkbookProject**
3. Click **UnloadProject** to access the project file from inside Visual Studio.

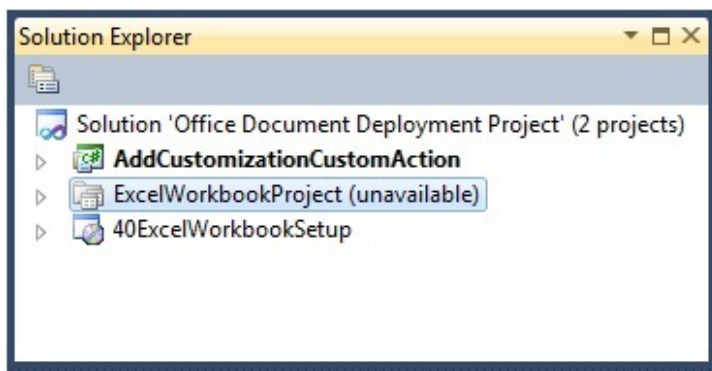
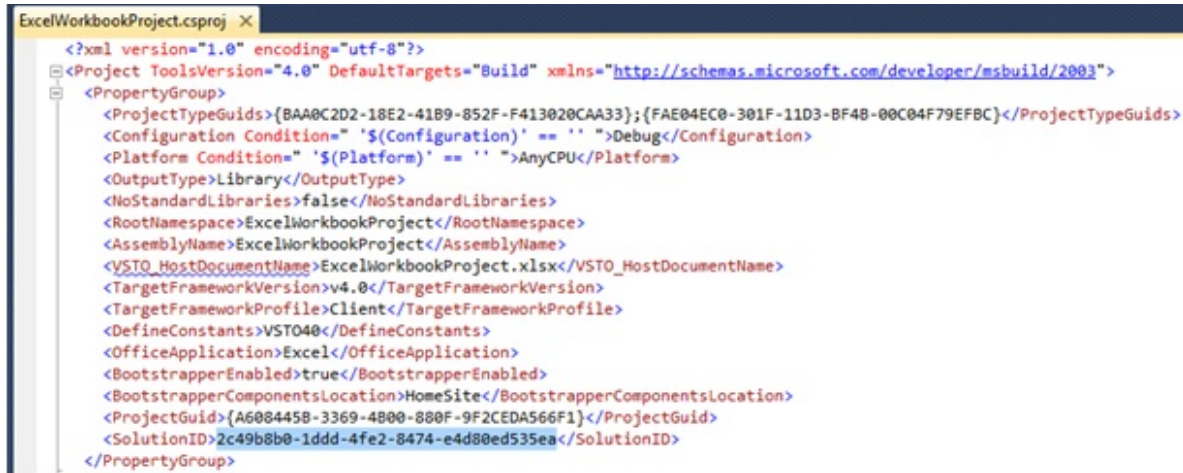


Figure 16: Unloading Excel Document Solution

4. In the **Solution Explorer**, right-click **ExcelWorkbookProject** and click **EditExcelWorkbookProject.vbproj** or **Edit ExcelWorkbookProject.csproj**.
5. In the **ExcelWorkbookProject** editor, locate the **SolutionID** element inside the **PropertyGroup** element.
6. Copy the GUID value of this element to a temporary location such as the clipboard or notepad.



```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <ProjectTypeGuids>{BAA0C2D2-18E2-41B9-852F-F413020CAA33};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <OutputType>Library</OutputType>
    <NoStandardLibraries>false</NoStandardLibraries>
    <RootNamespace>ExcelWorkbookProject</RootNamespace>
    <AssemblyName>ExcelWorkbookProject</AssemblyName>
    <VSTO_HostDocumentName>ExcelWorkbookProject.xlsx</VSTO_HostDocumentName>
    <TargetFrameworkVersion>v4.0</TargetFrameworkVersion>
    <TargetFrameworkProfile>Client</TargetFrameworkProfile>
    <DefineConstants>VSTO40</DefineConstants>
    <OfficeApplication>Excel</OfficeApplication>
    <BootstrapperEnabled>true</BootstrapperEnabled>
    <BootstrapperComponentsLocation>HomeSite</BootstrapperComponentsLocation>
    <ProjectGuid>{A6084458-3369-4800-880F-9F2CEDA566F1}</ProjectGuid>
    <SolutionID>2c49b8b0-1ddd-4fe2-8474-e4d80ed535ea</SolutionID>
  </PropertyGroup>
```


Figure 17: Retrieving the SolutionID

7. In the **Solution Explorer**, right-click **ExcelWorkbookProject** and click **Reload Project**.
8. Click **Yes** in the dialog box that appears to close the **ExcelWorkbookProject** editor.

To configure the custom action with the solution ID

1. In the **Solution Explorer**, expand **AddCustomizationCustomAction**.
2. Right-click **AddCustomization.vb** or **AddCustomization.cs**.
3. Click **View** code to edit the code for the custom action.
4. In the **AddCustomization** code editor, locate **SolutionID** field inside the **AddCustomization** class.
5. Change the field to use the solution ID of the ExcelWorkbookProject.

Visual Basic

	 Copy
<pre><RunInstaller(True)> _ Public Class AddCustomization Inherits Installer Shared ReadOnly SolutionID As New Guid("2c49b8b0-1ddd-4fe2-8474-e4d80ed535ea") End Class</pre>	

C#


```
[RunInstaller(true)]

public class AddCustomization

    : Installer

{

    static readonly Guid SolutionID =

        new Guid("2c49b8b0-1ddd-4fe2-8474-e4d80ed535ea");

}
```

6. On the **File** menu, click **Save AddCustomization.vb** or **SaveAddCustomization.cs** to save your changes to the custom action class

The last step is to configure the custom action for the **Install** and **Uninstall** steps of the Windows Installer package. The Install step copies the document to the **My Documents** folder and re-attaches the customization. To

perform this task, you need to configure the custom action with additional data. The Uninstall step removes the

document from the My Documents folder. This task does not require additional configuration data.

To configure the setup project

1. In the **Solution Explorer**, right-click **ExcelWorkbookSetup**, expand **Add** and click **Project Output**.
2. In the **Add Project Output Group** dialog box, in the **Project** list, click **AddCustomizationCustomAction**.
3. Select **Primary Output** and click **OK** to close the dialog box and add the assembly containing the custom action to the setup project. This copies the assembly to the installation folder chosen by the user and allows it to be called as part of the setup process.

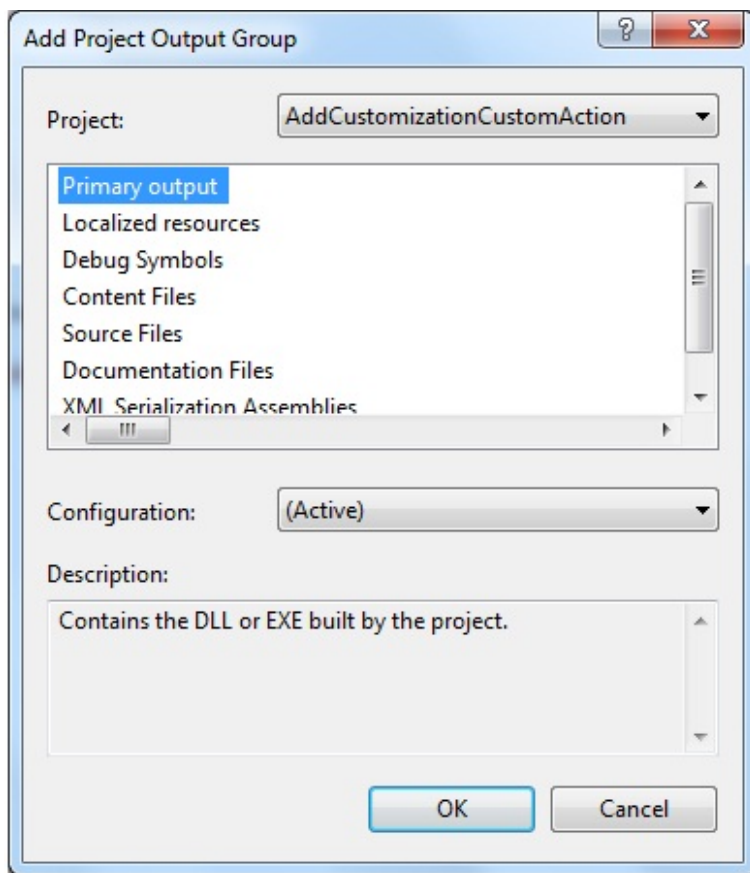


Figure 18: Document Manifest Custom Action - Add Project Output Group

4. In the **Solution Explorer**, right-click **ExcelWorkbookSetup**.
5. Expand **View** and click **Custom Actions**.
6. In the **Custom Actions(ExcelWorkbookSetup)** editor, right-click **Custom Actions** and click **Add Custom Action**.
7. In the **Select Item in Project** dialog box, in the **Look In** list, click **Application Folder**. Select **Primary Output** from **AddCustomizationCustomAction(active)** and click **OK** to add the custom action to the **Install** step.
8. Under the **Install** node, right-click **Primary output from AddCustomizationCustomAction(Active)** and click **Rename**. Name the custom action **Copy document to My Documents** and attach **customization**.
9. Under the **Uninstall** node, right-click **Primary output from AddCustomizationCustomAction(Active)** and click **Rename**. Name the custom action **Remove document from My Documents folder**.

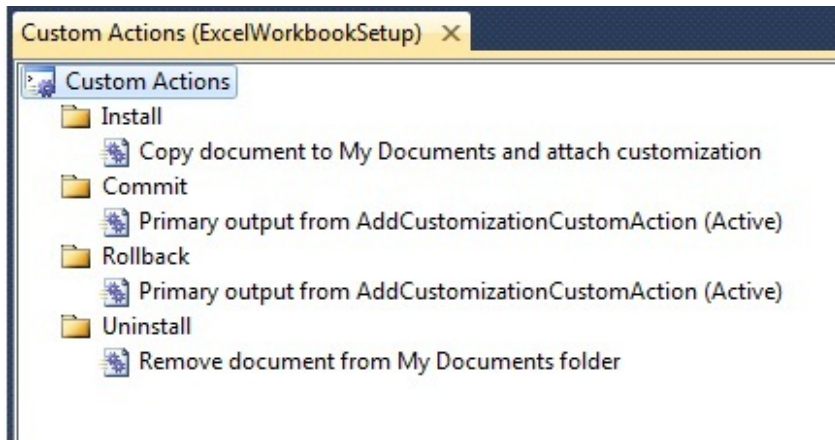


Figure 19: Document Manifest Custom Actions

10. In the **Custom Actions(ExcelWorkbookSetup)** editor, right-click **Copy document to My Documents and attach customization** and click **Properties Window**.
11. In the **Properties** window, for the **CustomActionData** property, enter the following values. This passes in the location of the customization DLL, the location of the deployment manifest and the location of the Microsoft Office document. Note that the following text has been wrapped.

		Copy
<pre>/assemblyLocation="[TARGETDIR]ExcelWorkbookProject.dll"/deploymentManifestLocation="[TARGETDIR]ExcelWorkbookProject.vsto"/documentLocation="[TARGETDIR]ExcelWorkbookProject.xlsx"</pre>		

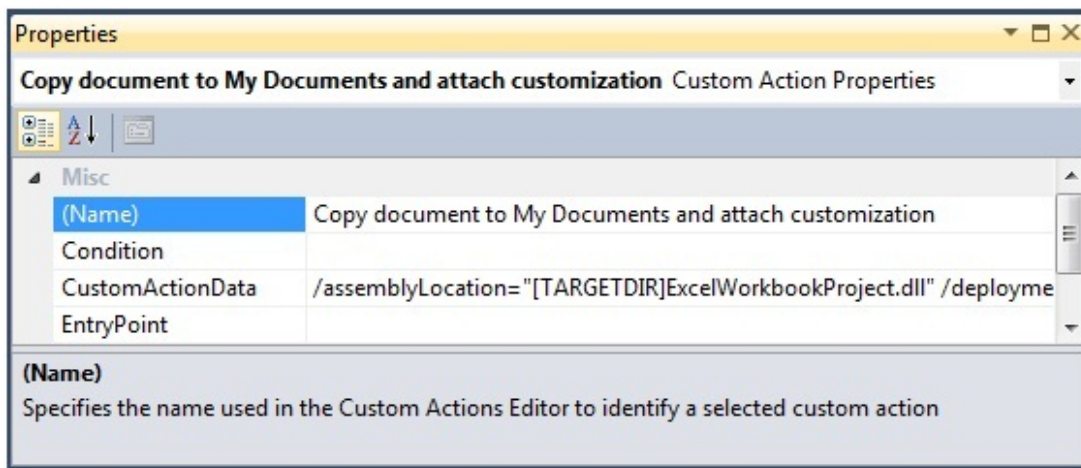


Figure 20: Custom Action to Copy Document to My Documents

12. Compile and deploy the **ExcelWorkbookSetup** project.
13. Look in **My Documents** folder, and open the **ExcelWorkbookProject.xlsx** file.

Conclusion

This article provides an overview of deploying a Visual Studio Tools for Office solution using a Visual Studio 2010

Windows Installer package. This article provides a walkthrough to show how you can install the prerequisites (the

.NET Framework, the VSTO 2010 runtime), add the project output and manifests, create registry keys to load an

application-level add-in, and use a custom action to copy document-level solutions to the My Documents folder.

Additional Resources

[Deploying Solutions for the 2007 Office System with ClickOnce Using Visual Studio 2008 Professional](#)

[How to: Install the Visual Studio Tools for Office Runtime](#)

[Office Primary Interop Assemblies](#)

[Registry Entries for Application-Level Add-Ins](#)

[Custom Document Properties Overview](#)

[Specifying Form Regions in the Windows Registry](#)

[Granting Trust to Documents](#)

About the Authors

Wouter van Vugt is a Microsoft MVP with Office Open XML technologies and an independent consultant focusing on

creating Office Business Applications (OBAs) with SharePoint, Microsoft Office, and related .NET technologies.

Wouter is a frequent contributor to developer community sites such as [OpenXmlDeveloper.org](#) and [MSDN](#).

He has

published several white papers and articles as well a book available on line titled Open XML: Explained e-book.

Wouter is the founder of Code-Counsel, a Dutch company focusing on delivering cutting-edge technical content

through a variety of channels. You can find out more about Wouter by reading his blog and visiting the

[Code-Counsel Web site](#).

Ted Pattison is a SharePoint MVP, author, trainer and the founder of Ted Pattison Group. In the fall of 2005, Ted

was hired by Microsoft's Developer Platform Evangelism group to author the Ascend developer training

curriculum

for Windows SharePoint Services 3.0 and Microsoft Office SharePoint Server 2007. Since that time, Ted has been

entirely focused on educating professional developers on SharePoint 2007 technologies. Ted has just finished

writing a book for Microsoft Press titled Inside Windows SharePoint Services 3.0 that focuses on how to use SharePoint as a development platform for building business solutions. Ted also writes a developer-focused column

for MSDN Magazine titled Office Space.