Visual Studio Installer Deployment

Article • 11/01/201211 minutes to read

Windows Installer deployment enables you to create installer packages to be distributed to users; the user runs the setup file and steps through a wizard to install the application. This is accomplished by adding a Setup project to your solution. When built, the project creates a setup file that you distribute to users; the user runs the setup file and steps through a wizard to install the application.

Microsoft Windows Installer is a data-driven installation and configuration service that ships as a part of the Windows operating system. Windows Installer maintains a database of information about every application that is installed, including files, registry keys, and components. When an application is uninstalled, the database is checked to make sure that no other applications rely on a file, registry key, or component before removing it. This prevents the removal of one application from breaking another.

> **Note**
>
> The Express Editions do not include Windows Installer technology. For information about the deployment technology used in the Express Editions, see **ClickOnce Security and Deployment**.

By using the deployment tools in Visual Studio with the functionality provided by Windows Installer, you can deploy and maintain your applications.

Creating an Installer Package

You should use an installer package to deploy your application and its prerequisites. Typically, applications have dependencies on the .NET Framework, SQL Server Express, or even a custom EXE or DLL. However, it is uncertain whether end-user computers have the particular version of the .NET Framework or other dependencies that your application relies on. For this reason, copying your application to end-user computer is not recommended.

Installation Location

Your end-users can install your application from the Web, CD, network file share, or other alternatives. The installation location affects the project template that you can use. For example, if you want end-users to install from the Web, you can use the Web Setup project template. To install from a CD or network, use the Setup project template. For more information about deployment project templates, see Setup and Deployment Projects.

Files and Folders

You can use the **File System Editor** to control where and how deployment files are installed. The organization of the file system can differ from one computer to another and folder names can also differ; the **File System Editor** uses the concept of abstract folders to ensure that files are installed where you want them. For more information, see File Installation Management in Deployment.

Virtual folders represent Windows system folders. For example, the **Desktop Folder** is the equivalent of the system folder Desktop. Windows tracks the location of system folders, so no matter where the folder is located or what it is named, files placed in the **Desktop Folder** always end up in the Desktop system folder. For more information, see Special Folders and Custom Folders.

You can also create your own folders and place them in a location beneath any system folder. For example, you might create an Application Data folder beneath the **Application Folder —** no matter where the **Application Folder** is located on a target computer, files placed in your Application Data folder will always be installed in the same relative location. For more information, see How to: Add and Remove Folders in the File System Editor.

Folders in the **File System Editor** can contain files, project outputs, and assemblies. Project outputs represent the items contained in another project within the solution and can include primary built output (for example, an executable file), localized resources, symbolic debug information, content files (for example, HTML pages), and the project source files. Each of these outputs is referred to as a project output group; a project output group contains the primary output (also known as the key output) plus any additional outputs and dependencies. For more information, see How to: Add and Remove Project Outputs in the File System Editor and How to: Add Items to a Deployment Project.

In addition, conditions can be placed on any file or folder using the **Condition** property. This allows you to customize the installation of files based on conditions that exist on a target computer during installation. For example, you might choose to install different files based on the operating system version on the target computer. For more information, see Condition Property.

The **File System Editor** also supports the creation of shortcuts, allowing you to place files in one folder and point to them from a shortcut on the desktop or in another folder. For more information, see How to: Add and Remove Shortcuts in the File System Editor.

File Associations

When you deploy an application, you often want to associate a file type with that application. For example, if your application creates and uses files with an extension of .myfile, you want your application to be associated with the .myfile file type so that when a user double-clicks a .myfile file, it opens in your application.

The deployment tools in Visual Studio include a **File Types Editor**, which allows you to specify document types and associate them with file extensions. In addition, you can specify the verbs or actions for each document type and specify MIME types for the document types for use in browsers. For more information, see File Types Management in Deployment.

During installation, the settings that you specify in the **File Types Editor** are updated on the target computer.

Registry

Often an integral part of deploying an application involves accessing the registry, setting registry values, or creating registry keys. The deployment tools in Visual Studio provide this functionality.

The **Registry Editor** in Visual Studio is similar to the Windows Registry Editor: both tools provide a hierarchical representation of the registry on a target computer. The standard registry roots are represented. You can change values for existing keys, add values for new keys, and specify default keys. For more information, see Registry Settings Management in Deployment.

During installation, keys and values specified in the **Registry Editor** are written to the registry of the target computer.

In addition, you can place conditions on any registry key or value by using the **Condition** property. This allows you to customize the registry based on conditions that exist on a target computer during installation. For example, you might want to enter a different registry value depending on the operating system version on the target computer.

Authenticode Signing

You may want to sign an application or component so that users can verify who published it and verify that it is safe. We recommend that you sign Cab files and installers that are downloaded through a Web browser.

You can use Visual Studio deployment tools to sign an installer, a merge module, or a Cab file by using Microsoft Authenticode technology. You must first obtain a digital certificate to sign your application or components.

To use Authenticode signing, you must enable signed ClickOnce manifests in your deployment project. For more information, see Signing Page, Project Designer.

Global Assembly Cache

The global assembly cache is a code cache provided by the .NET Framework that is used to store assemblies that need to be shared by several applications. To be installed in the global assembly cache, the assembly must be strong-named, which gives an application or component a unique identity that other software can use to identify and refer explicitly to it. For more information, see How to: Sign an Assembly (Visual Studio).

To install an assembly to the global assembly cache, add the assembly or the project output group for the assembly to the **Global Assembly Cache** folder in the **File System Editor**. To open this editor, on the **View** menu, point to **Editor**, and click **File System Editor**.

The **Global Assembly Cache** folder is unlike the other folders in the **File System Editor**. It has no properties that are settable, and you cannot create shortcuts to the folder or assemblies in the folder.

Selecting Prerequisites

To successfully deploy an application, you must also deploy all components referenced by the application. For example, most applications created with Visual Studio have a dependency on the .NET Framework. A required version of the common language runtime must be present on the target computer before the application is installed. The deployment tools in Visual Studio enable you to install the .NET Framework and other components as a part of your installation. The process of installing prerequisite components is also known as bootstrapping.

For more information, see How to: Install Prerequisites in Windows Installer Deployment

Installing with Administrative Privileges

Administrative installation is a feature of Microsoft Windows Installer that allows you to install a source image of an application on a network share. Users in a workgroup who have access to the network share can then install the application from the source image.

The **User Interface Editor** allows you to specify a different set of installation dialog boxes that are displayed when an administrator installs the application to a network share via the command line using the **/a** command-line option (**msiexec /a**InstallerName). For more information, see User Interface Management in Deployment.

> **Note**
>
> When installing an application through administrative installation, bootstrapping application files (the files that install Windows Installer, if needed) are not copied to the server even if the **Bootstrapper** property is set to **Windows Installer Bootstrapper**. If the bootstrapping application files are required for installation, you need to manually copy the files Instmsia.msi, Instmsiw.msi, Setup.exe, and Setup.ini to the server. These files can be found in the same directory as the .msi file for your application.

For more information, see the Windows Installer SDK documentation at Administrative Installation (Windows Installer) .

Windows and Elevation

Windows Installer technology supports software installation on the Windows Vista and Windows 7 operating systems. The end user installing applications should receive prompts only for each component installation that requires elevation, even when the user's computer runs under User Account Control (UAC).

Application Elevation

Typically, Setup.exe (also known as the bootstrapper) does not run as elevated; it runs at the current user's permission level. Therefore, the installation does not prompt for elevation when the final application installation starts. However, note that an .msi file usually prompts the user, whereas Setup.exe does not.

In the embedded UAC manifest of the bootstrapper, the requestedExecutionLevel node specifies that the installation run as the current user (asInvoker):

<requestedExecutionLevel level="asInvoker" />

However, you can elevate the application installation if you have to. For example, modifying Internet Information Services (IIS) settings in a Web Setup project requires administrative privileges, as does installing assemblies to the global assembly cache. The elevation prompt occurs after the prerequisite installations but before the application installation.

To elevate permissions for an installation, open the project (.vdproj) file. In the project file's MsiBootstrapper section, set the RequiresElevation property to True. This property is not made available through the Visual Studio integrated development environment (IDE). Therefore, you must use the project file. For more information, see RequiresElevation Property.

Administrator-Assisted Elevation

Windows Installer supports administrator-assisted elevation on Windows Vista and Windows 7. In this scenario, the user is prompted for administrator credentials, and the administrator enters the password for the user. To support this scenario, the bootstrapper sets the AdminUser property to True when your computer is running on Windows Vista or a later version of Windows.

> **Note**
>
> If you are running Windows Vista on a computer that does not use UAC and you are not an administrator, AdminUser will still be set to True. Therefore, .exe installers (such as SQLExpress32.exe) should be written to detect appropriate permissions and to generate a specific exit code in the case of insufficient permission. You should author Setup.exe to catch this exit code and display a message stating that an administrator is required.

Prerequisite Elevation

Windows Vista and Windows 7 elevates prerequisite component installation when it is necessary. The bootstrapper itself performs no elevation; when Windows Vista or Windows 7 runs under UAC, it issues a prompt for each prerequisite component that has to be elevated, unless it is already installed. If a package elevation fails, the bootstrapper fails and sends an appropriate error message.

Custom Action Elevation

Custom actions that you create in the Custom Actions Editor run as elevated. Custom actions should not access user-specific data, such as the registry or file system, because the custom action will not run in the invoking user's account.

By default, custom actions run elevated because the default setting of the NoImpersonate property is True in the Custom Actions Editor. Changing NoImpersonate to False would force the custom action to impersonate the invoking user, who might have reduced permissions.

Differences Between Visual Studio Versions

Also note that there will be differences between the way that Visual Studio 2005 and Visual Studio 2008 Setup projects will run under UAC.

Windows Vista or Windows 7 built-in installer detection prompts for consent when you run under UAC. A bootstrapper (Setup.exe) built with Visual Studio 2005 always prompts for consent, regardless of what it is installing. Because Setup.exe and all of its processes run with an administrator token on Windows Vista and Windows 7, the final application installation will be installed with elevated privileges. If a user runs Setup.exe with administrator-assisted elevation, the application will be installed under the elevated user's profile (not the administrator's profile).

In Visual Studio 2008 and Visual Studio 2010, the Setup.exe does not prompt for elevation when it is started. To prevent the elevation prompt, the embedded manifest of the bootstrapper specifies that Setup.exe run with a requested execution level of asInvoker. This provides the benefit of the final application installation not being run as elevated, although still enabling the installation of prerequisite components to be elevated as necessary. The bootstrapper calls ShellExecute to launch prerequisites. Windows Vista or Windows 7 receives this call, performs installation detection, and issues a user prompt before installation.

The disadvantage to this change is that a prompt is issued for each prerequisite component that has to be installed, in addition to the application itself. However, if all prerequisites are already on the computer, the installation might not cause any prompts. Also, you should not have external checks that require elevation. External checks will work, but the user receives several elevation prompts for each external check, in addition to prompts for the installer.

See Also

Tasks

Troubleshooting Setup and Deployment Projects

Reference

Deployment Errors and Support

Concepts

Setup and Deployment Projects

Other Resources

Deploying Applications and Components

Deployment Editors

Deployment Tasks and Walkthroughs

Deployment Dialog Boxes

Deployment Samples and Walkthroughs