ClickOnce Security and Deployment

Visual Studio 2015

ClickOnce is a deployment technology that enables you to create self-updating Windows-based applications that can be installed and run with minimal user interaction. Visual Studio provides full support for publishing and updating applications deployed with ClickOnce technology if you have developed your projects with Visual Basic and Visual C#. For information about deploying Visual C++ applications, see ClickOnce Deployment for Visual C++ Applications.

ClickOnce deployment overcomes three major issues in deployment:

- **Difficulties in updating applications.** With Microsoft Windows Installer deployment, whenever an application is updated, the user can install an update, an msp file, and apply it to the installed product; with ClickOnce deployment, you can provide updates automatically. Only those parts of the application that have changed are downloaded, and then the full, updated application is reinstalled from a new side-by-side folder.
- **Impact to the user's computer.** With Windows Installer deployment, applications often rely on shared components, with the potential for versioning conflicts; with ClickOnce deployment, each application is self-contained and cannot interfere with other applications.
- **Security permissions.** Windows Installer deployment requires administrative permissions and allows only limited user installation; ClickOnce deployment enables non-administrative users to install and grants only those Code Access Security permissions necessary for the application.

In the past, these issues sometimes caused developers to decide to create Web applications instead of Windows-based applications, sacrificing a rich user interface for ease of installation. By using applications deployed using ClickOnce, you can have the best of both technologies.

What Is a ClickOnce Application?

A ClickOnce application is any Windows Presentation Foundation (.xbap), Windows Forms (.exe), console application (.exe), or Office solution (.dll) published using ClickOnce technology. You can publish a ClickOnce application in three different ways: from a Web page, from a network file share, or from media such as a CD-ROM. A ClickOnce application can be installed on an end user's computer and run locally even when the computer is offline, or it can be run in an online-only mode without permanently installing anything on the end user's computer. For more information, see Choosing a ClickOnce Deployment Strategy.

ClickOnce applications can be self-updating; they can check for newer versions as they become available and automatically replace any updated files. The developer can specify the update behavior; a network administrator can also control update strategies, for example, marking an update as mandatory. Updates can also be rolled back to an earlier version by the end user or by an administrator. For more information, see Choosing a ClickOnce Update Strategy.

Because ClickOnce applications are isolated, installing or running a ClickOnce application cannot break existing applications. ClickOnce applications are self-contained; each ClickOnce application is installed to and run from a secure per-user, per-application cache. ClickOnce applications run in the Internet or Intranet security zones. If necessary, the application can request elevated security permissions. For more information, see Securing ClickOnce Applications.

How ClickOnce Security Works

The core ClickOnce security is based on certificates, code access security policies, and the ClickOnce trust prompt.

Certificates

Authenticode certificates are used to verify the authenticity of the application's publisher. By using Authenticode for application deployment, ClickOnce helps prevent a harmful program from portraying itself as a legitimate program coming from an established, trustworthy source. Optionally, certificates can also be used to sign the application and deployment manifests to prove that the files have not been tampered with. For more information, see ClickOnce and Authenticode. Certificates can also be used to configure client computers to have a list of trusted publishers. If an application comes from a trusted publisher, it can be installed without any user interaction. For more information, see Trusted Application Deployment Overview.

Code Access Security

Code access security helps limit the access that code has to protected resources. In most cases, you can choose the Internet or Local Intranet zones to limit the permissions. Use the **Security** page in the **Project Designer** to request the zone appropriate for the application. You can also debug applications with restricted permissions to emulate the end-user experience. For more information, see Code Access Security for ClickOnce Applications.

ClickOnce Trust Prompt

If the application requests more permissions than the zone allows, the end user can be prompted to make a trust decision. The end user can decide if ClickOnce applications such as Windows Forms applications, Windows Presentation Foundation applications, console applications, XAML browser applications, and Office solutions are trusted to run. For more information, see How to: Configure the ClickOnce Trust Prompt Behavior.

How ClickOnce Deployment Works

The core ClickOnce deployment architecture is based on two XML manifest files: an application manifest and a deployment manifest. The files are used to describe where the ClickOnce applications are installed from, how they are updated, and when they are updated.

Publishing ClickOnce Applications

The application manifest describes the application itself. This includes the assemblies, the dependencies and files that make up the application, the required permissions, and the location where updates will be available. The application developer authors the application manifest by using the Publish Wizard in Visual Studio or the Manifest Generation and Editing Tool (Mage.exe) in the Windows Software Development Kit (SDK). For more information, see How to: Publish a ClickOnce Application using the Publish Wizard.

The deployment manifest describes how the application is deployed. This includes the location of the application manifest, and the version of the application that clients should run.

Deploying ClickOnce Applications

After it is created, the deployment manifest is copied to the deployment location. This can be a Web server, network file share, or media such as a CD. The application manifest and all the application files are also copied to a deployment location that is specified in the deployment manifest. This can be the same as the deployment location, or it can be a different location. When using the **Publish Wizard** in Visual Studio, the copy operations are performed automatically.

Installing ClickOnce Applications

After it is deployed to the deployment location, end users can download and install the application by clicking an icon representing the deployment manifest file on a Web page or in a folder. In most cases, the end user is presented with a simple dialog box asking the user to confirm installation, after which installation proceeds and the application is started without additional intervention. In cases where the application requires elevated permissions or if the application is not signed by a trusted certificate, the dialog box also asks the user to grant permission before the installation can continue. Though ClickOnce installs are per-user, permission elevation may be required if there are prerequisites that require administrator privileges. For more information about elevated permissions, see Securing ClickOnce Applications.

Certificates can be trusted at the machine or enterprise level, so that ClickOnce applications signed with a trusted certificate can install silently. For more information about trusted certificates, see Trusted Application Deployment Overview.

The application can be added to the user's **Start** menu and to the **Add or Remove Programs** group in the **Control Panel**. Unlike other deployment technologies, nothing is added to the **Program Files** folder or the registry, and no administrative rights are required for installation

Note

It is also possible to prevent the application from being added to the **Start** menu and **Add or Remove Programs** group, in effect making it behave like a Web application. For more information, see Choosing a ClickOnce Deployment Strategy.

Updating ClickOnce Applications

When the application developers create an updated version of the application, they generate a new application manifest and copy files to a deployment location—usually a sibling folder to the original application deployment folder. The administrator updates the deployment manifest to point to the location of the new version of the

application.

Note
The **Publish Wizard** in Visual Studio can be used to perform these steps.

In addition to the deployment location, the deployment manifest also contains an update location (a Web page or network file share) where the application checks for updated versions. ClickOnce **Publish** properties are used to specify when and how often the application should check for updates. Update behavior can be specified in the deployment manifest, or it can be presented as user choices in the application's user interface by means of the ClickOnce APIs. In addition, **Publish** properties can be employed to make updates mandatory or to roll back to an earlier version. For more information, see Choosing a ClickOnce Update Strategy.

Third Party Installers

You can customize your ClickOnce installer to install third-party components along with your application. You must have the redistributable package (.exe or .msi file) and describe the package with a language-neutral product manifest and a language-specific package manifest. For more information, see Creating Bootstrapper Packages.

ClickOnce Tools

The following table shows the tools that you can use to generate, edit, sign, and re-sign the application and deployment manifests.

Tool	Description
Security Page, Project Designer	Signs the application and deployment manifests.
Publish Page, Project Designer	Generates and edits the application and deployment manifests for Visual Basic and Visual C# applications.
Mage.exe (Manifest Generation and Editing Tool)	Generates the application and deployment manifests for Visual Basic, Visual C#, and Visual C++ applications. Signs and re-signs the application and deployment manifests. Can be run from batch scripts and the command prompt.
MageUI.exe (Manifest Generation and Editing Tool, Graphical Client)	Generates and edits the application and deployment manifests. Signs and re-signs the application and deployment

	manifests.
GenerateApplicationManifest Task	Generates the application manifest. Can be run from MSBuild. For more information, see MSBuild Reference.
GenerateDeploymentManifest Task	Generates the deployment manifest. Can be run from MSBuild. For more information, see MSBuild Reference.
SignFile Task	Signs the application and deployment manifests. Can be run from MSBuild. For more information, see MSBuild Reference.
Microsoft.Build.Tasks.Deployment.ManifestUtilities	Develop your own application to generate the application and deployment manifests.

The following table shows the .NET Framework version required to support ClickOnce applications in these browsers.

Browser	**.NET Framework version**
Internet Explorer	2.0, 3.0, 3.5, 3.5 SP1, 4
Firefox	2.0 SP1, 3.5 SP1, 4

See Also

ClickOnce Deployment on Windows Vista
Publishing ClickOnce Applications
Securing ClickOnce Applications
Deploying COM Components with ClickOnce
Building ClickOnce Applications from the Command Line
Debugging ClickOnce Applications That Use System.Deployment.Application

Choosing a ClickOnce Deployment Strategy

Visual Studio 2015

There are three different strategies for deploying a ClickOnce application; the strategy that you choose depends primarily on the type of application that you are deploying. The three deployment strategies are as follows:

- Install from the Web or a Network Share
- Install from a CD
- Start the application from the Web or a Network Share

> **Note**
>
> In addition to selecting a deployment strategy, you will also want to select a strategy for providing application updates. For more information, see Choosing a ClickOnce Update Strategy.

Install from the Web or a Network Share

When you use this strategy, your application is deployed to a Web server or a network file share. When an end user wants to install the application, he or she clicks an icon on a Web page or double-clicks an icon on the file share. The application is then downloaded, installed, and started on the end user's computer. Items are added to the **Start** menu and **Add or Remove Programs** in **Control Panel**.

Because this strategy depends on network connectivity, it works best for applications that will be deployed to users who have access to a local-area network or a high-speed Internet connection.

If you deploy the application from the Web, you can pass arguments into the application when it is activated using a URL. For more information, see How to: Retrieve Query String Information in an Online ClickOnce Application. You cannot pass arguments into an application that is activated by using any of the other methods described in this document.

To enable this deployment strategy in Visual Studio, click **From the Web** or **From a UNC path or file share** on the **How Installed** page of the Publish Wizard.

This is the default deployment strategy.

Install from a CD

When you use this strategy, your application is deployed to removable media such as a CD-ROM or DVD. As with the previous option, when the user chooses to install the application, it is installed and started, and items are added to the **Start** menu and **Add or Remove Programs** in **Control Panel**.

This strategy works best for applications that will be deployed to users without persistent network connectivity or with

low-bandwidth connections. Because the application is installed from removable media, no network connection is necessary for installation; however, network connectivity is still required for application updates.

To enable this deployment strategy in Visual Studio, click **From a CD-ROM or DVD-ROM** on the **How Installed** page of the Publish Wizard.

To enable this deployment strategy manually, change the **deploymentProvider** tag in the deployment manifest. (In Visual Studio, this property is exposed as **Installation URL** on the **Publish** page of the Project Designer. In Mage.exe it is **Start Location**.)

Start the Application from the Web or a Network Share

This strategy is like the first, except the application behaves like a Web application. When the user clicks a link on a Web page (or double-clicks an icon on the file share), the application is started. When users close the application, it is no longer available on their local computer; nothing is added to the **Start** menu or **Add or Remove Programs** in **Control Panel**.

Note

Technically, the application is downloaded and installed to an application cache on the local computer, just as a Web application is downloaded to the Web cache. As with the Web cache, the files are eventually scavenged from the application cache. However, the perception of the user is that the application is being run from the Web or file share.

This strategy works best for applications that are used infrequently—for example, an employee-benefits tool that is typically run only one time each year.

To enable this deployment strategy in Visual Studio, click **Do not install the application** on the **Install or Run From Web** page of the Publish Wizard.

To enable this deployment strategy, manually, change the **install** tag in the deployment manifest. (Its value can be **true** or **false**. In Mage.exe, use the **Online Only** option in the **Application Type** list.)

Web Browser Support

Applications that target .NET Framework 3.5 can be installed using any browser.

Applications that target .NET Framework 2.0 require Internet Explorer.

See Also

ClickOnce Security and Deployment
Choosing a ClickOnce Update Strategy
How to: Publish a ClickOnce Application using the Publish Wizard
Securing ClickOnce Applications

ClickOnce Cache Overview

Visual Studio 2015

All ClickOnce applications, whether they are installed locally or hosted online, are stored on the client computer in a ClickOnce application *cache*. A ClickOnce cache is a family of hidden directories under the Local Settings directory of the current user's Documents and Settings folder. This cache holds all the application's files, including the assemblies, configuration files, application and user settings, and data directory. The cache is also responsible for migrating the application's data directory to the latest version. For more information about data migration, see Accessing Local and Remote Data in ClickOnce Applications.

By providing a single location for application storage, ClickOnce takes over the task of managing the physical installation of an application from the user. The cache also helps isolate applications by keeping the assemblies and data files for all applications and their distinct versions separate from one another. For example, when you upgrade a ClickOnce application, that version and its data resources are supplied with their own directories in the cache.

Cache Storage Quota

ClickOnce applications that are hosted online are restricted in the amount of space they can occupy by a quota that constrains the size of the ClickOnce cache. The cache size applies to all the user's online applications; a single partially-trusted, online application is limited to occupying half of the quota space. Installed applications are not limited by the cache size and do not count against the cache limit. For all ClickOnce applications, the cache retains only the current version and the previously installed version.

By default, client computers have 250 MB of storage for online ClickOnce applications. Data files do not count toward this limit. A system administrator can enlarge or reduce this quota on a particular client computer by changing the registry key, HKEY_CURRENT_USER\Software\Classes\Software\Microsoft\Windows\CurrentVersion\Deployment\OnlineAppQuotaInKB, which is a DWORD value that expresses the cache size in kilobytes. For example, in order to reduce the cache size to 50 MB, you would change this value to 51200.

See Also

Accessing Local and Remote Data in ClickOnce Applications

ClickOnce and Application Settings

Visual Studio 2015

Application settings for Windows Forms makes it easy to create, store, and maintain custom application and user preferences on the client. The following document describes how application settings files work in a ClickOnce application, and how ClickOnce migrates settings when the user upgrades to the next version.

The information below applies only to the default application settings provider, the LocalFileSettingsProvider class. If you supply a custom provider, that provider will determine how it stores its data and how it upgrades its settings between versions. For more information on application settings providers, see Application Settings Architecture.

Application Settings Files

Application settings consumes two files: *app*.exe.config and user.config, where *app* is the name of your Windows Forms application. user.config is created on the client the first time your application stores user-scoped settings. *app*.exe.config, by contrast, will exist prior to deployment if you define default values for settings. Visual Studio will include this file automatically when you use its **Publish** command. If you create your ClickOnce application using Mage.exe or MageUI.exe, you must make sure this file is included with your application's other files when you populate your application manifest.

In a Windows Forms applications not deployed using ClickOnce, an application's *app*.exe.config file is stored in the application directory, while the user.config file is stored in the user's **Documents and Settings** folder. In a ClickOnce application, *app*.exe.config lives in the application directory inside of the ClickOnce application cache, and user.config lives in the ClickOnce data directory for that application.

Regardless of how you deploy your application, application settings ensures safe read access to *app*.exe.config, and safe read/write access to user.config.

In a ClickOnce application, the size of the configuration files used by application settings is constrained by the size of the ClickOnce cache. For more information, see ClickOnce Cache Overview.

Version Upgrades

Just as each version of a ClickOnce application is isolated from all other versions, the application settings for a ClickOnce application are isolated from the settings for other versions as well. When your user upgrades to a later version of your application, application settings compares most recent (highest-numbered) version's settings against the settings supplied with the updated version and merges the settings into a new set of settings files.

The following table describes how application settings decides which settings to copy.

Type of Change	Upgrade Action
Setting added to *app*.exe.config	The new setting is merged into the current version's *app*.exe.config

Setting removed from *app*.exe.config	The old setting is removed from the current version's *app*.exe.config
Setting's default changed; local setting still set to original default in user.config	The setting is merged into the current version's user.config with the new default as the value
Setting's default changed; setting set to non-default in user.config	The setting is merged into the current version's user.config with the non-default value retained

If you have created your own application settings wrapper class and wish to customize the update logic, you can override the Upgrade method.

ClickOnce and Roaming Settings

ClickOnce does not work with roaming settings, which allows your settings file to follow you across machines on a network. If you need roaming settings, you will need either to implement an application settings provider that stores settings over the network, or develop your own custom settings classes for storing settings on a remote computer. For more information in settings providers, see Application Settings Architecture.

See Also

ClickOnce Security and Deployment
Application Settings Overview
ClickOnce Cache Overview
Accessing Local and Remote Data in ClickOnce Applications

ClickOnce Deployment on Windows Vista

Visual Studio 2015

Building applications in Visual Studio for User Account Control (UAC) on Windows Vista normally generates an embedded manifest, encoded as binary XML data in the application's executable file. Because ClickOnce and Registration-Free COM applications require an external manifest, Visual Studio generates a file for these types of projects containing the UAC data instead of an embedded manifest. By default, Visual Studio uses information from a file called app.manifest to generate external UAC manifest information (for ClickOnce and Registration-Free COM deployment), or to embed it in the application's executable file (for all other cases). Visual Studio provides the following options for manifest generation:

- Use an embedded manifest. Embed UAC data in the application's executable file and run as normal user.

  This is the default setting (unless you use ClickOnce). This setting will support the usual manner in which Visual Studio operates on Windows Vista; that is, the generation of an internal and external manifest, both using **AsInvoker**.

- Use an external manifest. Generate an external manifest by using app.manifest.

  This generates only the external manifest by using the information in app.manifest. When you publish an application by using ClickOnce or Registration-Free COM, Visual Studio adds app.manifest to the project and adds this option.

- Use no manifest. Create the application without a manifest.

  This approach is also known as *virtualization*. Use this option for compatibility with existing applications from earlier versions of Visual Studio.

The new properties are available on the **Application** page of the Project Designer (for Visual C# projects only) and in the MSBuild project file format.

Note that the method for configuring UAC manifest generation in the Visual Studio IDE differs depending on project type (Visual C# and Visual Basic).

For information about configuring Visual C# projects for manifest generation, see Application Page, Project Designer (C#).

For information about configuring Visual Basic projects for manifest generation, see Application Page, Project Designer (Visual Basic).

See Also

ClickOnce Security and Deployment
d5c55084-1e7b-4b61-b478-137db01c0fc0
Application Page, Project Designer (C#)
Application Page, Project Designer (Visual Basic)

Localizing ClickOnce Applications

Visual Studio 2015

Localization is the process of making your application appropriate for a specific culture. This process involves translating user interface (UI) text to a region-specific language, using correct date and currency formatting, adjusting the size of controls on a form, and mirroring controls from right to left if necessary.

Localizing your application results in the creation of one or more satellite assemblies. Each assembly contains UI strings, images, and other resources specific to a given culture. (Your application's main executable file contains the strings for the default culture for your application.)

This topic describes three ways to deploy a ClickOnce application for other cultures:

- Include all satellite assemblies in a single deployment.
- Generate one deployment for each culture, with a single satellite assembly included in each.
- Download satellite assemblies on demand.

Including All Satellite Assemblies in a Deployment

Instead of publishing multiple ClickOnce deployments, you can publish a single ClickOnce deployment that contains all of the satellite assemblies.

This method is the default in Visual Studio. To use this method in Visual Studio, you do not have to do any additional work.

To use this method with MageUI.exe, you must set the culture for your application to **neutral** in MageUI.exe. Next, you must manually include all of the satellite assemblies in your deployment. In MageUI.exe, you can add the satellite assemblies by using the **Populate** button on the **Files** tab of your application manifest.

The benefit of this approach is that it creates a single deployment and simplifies your localized deployment story. At run time, the appropriate satellite assembly will be used, depending on the default culture of the user's Windows operating system. A drawback of this approach is that it downloads all satellite assemblies whenever the application is installed or updated on a client computer. If your application has a large number of strings, or your customers have a slow network connection, this process can affect performance during application update.

Note

This approach assumes that your application adjusts the height, width, and position of controls automatically to accommodate different text string sizes in different cultures. Windows Forms contains a variety of controls and technologies that enable you to design your form to make it easily localizable, including the FlowLayoutPanel and TableLayoutPanel controls as well as the AutoSize property. Also see How to: Support Localization on Windows Forms Using AutoSize and the TableLayoutPanel Control.

Generate One Deployment for Each Culture

In this deployment strategy, you generate multiple deployments. In each deployment, you include only the satellite assembly needed for a specific culture, and you mark the deployment as specific to that culture.

To use this method in Visual Studio, set the **Publish Language** property on the **Publish** tab to the desired region. Visual Studio will automatically include the satellite assembly required for the region you select, and will exclude all other satellite assemblies from the deployment.

You can accomplish the same thing by using the MageUI.exe tool in the Microsoft Windows Software Development Kit (SDK). Use the **Populate** button on the **Files** tab of your application manifest to exclude all other satellite assemblies from the application directory, and then set the **Culture** field on the **Name** tab for your deployment manifest in MageUI.exe. These steps not only include the correct satellite assembly, but they also set the `language` attribute on the `assemblyIdentity` element in your deployment manifest to the corresponding culture.

After publishing the application, you must repeat this step for each additional culture your application supports. You must make sure that you publish to a different Web server directory or file share directory every time, because each application manifest will reference a different satellite assembly, and each deployment manifest will have a different value for the `language` attribute.

Downloading Satellite Assemblies on Demand

If you decide to include all satellite assemblies in a single deployment, you can improve performance by using

on-demand downloading, which enables you to mark assemblies as optional. The marked assemblies will not be downloaded when the application is installed or updated. You can install the assemblies when you need them by calling the DownloadFileGroup method on the ApplicationDeployment class.

Downloading satellite assemblies on demand differs slightly from downloading other types of assemblies on demand. For more information and code examples on how to enable this scenario using the Windows SDK tools for ClickOnce, see Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API.

You can also enable this scenario in Visual Studio. Also see Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API Using the Designer or Walkthrough: Downloading Satellite Assemblies on Demand with the ClickOnce Deployment API Using the Designer.

Testing Localized ClickOnce Applications Before Deployment

A satellite assembly will be used for a Windows Forms application only if the CurrentUICulture property for the main thread of the application is set to the satellite assembly's culture. Customers in local markets will probably already be running a localized version of Windows with their culture set to the appropriate default.

You have three options for testing localized deployments before you make your application available to customers:

- You can run your ClickOnce application on the appropriate localized versions of Windows.
- You can set the CurrentUICulture property programmatically in your application. (This property must be set before you call the Run method.)

See Also

<assemblyIdentity> Element (ClickOnce Deployment)
ClickOnce Security and Deployment
Globalizing Windows Forms

How to: Publish a Project That Has a Specific Locale

Visual Studio 2015

It is not uncommon for an application to contain components that have different locales. In this scenario, you would create a solution that has several projects, and then publish separate projects for each locale. This procedure shows how to use a macro to publish the first project in a solution by using the 'en' locale. If you want to try this procedure with a locale other than 'en', make sure to set `localeString` in the macro to match the locale that you are using (for example, 'de' or 'de-DE').

Note

When you use this macro, the Publish Location should be a valid URL or Universal Naming Convention (UNC) share. Also, Internet Information Services (IIS) has to be installed on your computer. To install IIS, on the **Start** menu, click **Control Panel**. Double-click **Add or Remove Programs**. In **Add or Remove Programs**, click **Add/Remove Windows Components**. In the **Windows Components Wizard**, select the **Internet Information Services (IIS)** check box in the **Components** list. Then click **Finish** to close the wizard.

To create the publishing macro

1. To open the Macro Explorer, on the **Tools** menu, point to **Macros**, and then click **Macro Explorer**.
2. Create a new macro module. In the Macro Explorer, select **MyMacros**. On the **Tools** menu, point to **Macros**, and then click **New Macro Module**. Name the module **PublishSpecificCulture**.
3. In the Macro Explorer, expand the **MyMacros** node, and then open the **PublishAllProjects** module by double-clicking it (or, from the **Tools** menu, point to **Macros**, and then click **Macros IDE**).
4. In the Macros IDE, add the following code to the module, after the **Import** statements:

VB

```
Module PublishSpecificCulture
    Sub PublishProjectFirstProjectWithEnLocale()
        ' Note: You should publish projects by using the IDE at least once
        ' before you use this macro. Items such as the certficate and the
        ' security zone must be set.
        Dim localeString As String = "en"

        ' Get first project.
        Dim proj As Project = DTE.Solution.Projects.Item(1)
        Dim publishProperties As Object = proj.Properties.Item("Publish").Value

        ' GenerateManifests and SignManifests must always be set to
        ' True for publishing to work.
```

```
proj.Properties.Item("GenerateManifests").Value = True
proj.Properties.Item("SignManifests").Value = True

'Set the publish language.
'This will set the deployment language and pick up all
' en resource dlls:
Dim originalTargetCulture As String = _
    publishProperties.Item("TargetCulture").Value
publishProperties.Item("TargetCulture").Value = localeString

'Append 'en' to end of publish, install, and update URLs if needed:
Dim originalPublishUrl As String = _
    publishProperties.Item("PublishUrl").Value
Dim originalInstallUrl As String = _
    publishProperties.Item("InstallUrl").Value
Dim originalUpdateUrl As String = _
    publishProperties.Item("UpdateUrl").Value
publishProperties.Item("PublishUrl").Value = _
    AppendStringToUrl(localeString, New Uri(originalPublishUrl))
If originalInstallUrl <> String.Empty Then
    publishProperties.Item("InstallUrl").Value = _
        AppendStringToUrl(localeString, New Uri(originalInstallUrl))
End If
If originalUpdateUrl <> String.Empty Then
    publishProperties.Item("UpdateUrl").Value = _
        AppendStringToUrl(localeString, New Uri(originalUpdateUrl))
End If
proj.Save()

Dim slnbld2 As SolutionBuild2 = _
    CType(DTE.Solution.SolutionBuild, SolutionBuild2)
slnbld2.Clean(True)

slnbld2.BuildProject( _
proj.ConfigurationManager.ActiveConfiguration.ConfigurationName, _
proj.UniqueName, True)

' Only publish if build is successful.
If slnbld2.LastBuildInfo <> 0 Then
    MsgBox("Build failed for " & proj.UniqueName)
Else
    slnbld2.PublishProject( _
    proj.ConfigurationManager.ActiveConfiguration.ConfigurationName, _
    proj.UniqueName, True)
    If slnbld2.LastPublishInfo = 0 Then
        MsgBox("Publish succeeded for " & proj.UniqueName _
        & vbCrLf & "." _
        & " Publish Language was '" & localeString & "'.")
    Else
        MsgBox("Publish failed for " & proj.UniqueName)
    End If
End If

' Return URLs and target culture to their previous state.
```

```
            publishProperties.Item("PublishUrl").Value = originalPublishUrl
            publishProperties.Item("InstallUrl").Value = originalInstallUrl
            publishProperties.Item("UpdateUrl").Value = originalUpdateUrl
            publishProperties.Item("TargetCulture").Value = originalTargetCulture
            proj.Save()
        End Sub

        Private Function AppendStringToUrl(ByVal str As String, _
        ByVal baseUri As Uri) As String
            Dim returnValue As String = baseUri.OriginalString
            If baseUri.IsFile OrElse baseUri.IsUnc Then
                returnValue = IO.Path.Combine(baseUri.OriginalString, str)
            Else
                If Not baseUri.ToString.EndsWith("/") Then
                    returnValue = baseUri.OriginalString & "/" & str
                Else
                    returnValue = baseUri.OriginalString & str
                End If
            End If
            Return returnValue
        End Function
    End Module
```

5. Close the Macros IDE. The focus will return to Visual Studio.

To publish a project for a specific locale

1. To create a Visual Basic Windows Application project, on the **File** menu, point to **New**, and then click **Project**.

2. In the **New Project** dialog box, select **Windows Application** from the **Visual Basic** node. Name the project **PublishLocales**.

3. Click Form1. In the **Properties** window, under **Design**, change the **Language** property from **(Default)** to **English**. Change the **Text** property of the form to **MyForm**.

   Note that the localized resource DLLs are not created until they are needed. For example, they are created when you change the text of the form or one of its controls after you have specified the new locale.

4. Publish PublishLocales by using the Visual Studio IDE.

   In **Solution Explorer**, select PublishLocales. On the **Project** menu, select **Properties**. In the Project Designer, on the **Publish** page, specify a publishing location of **http://localhost/PublishLocales**, and then click **Publish Now**.

   When the publish Web page appears, close it. (For this step, you only have to publish the project; you do not have to install it.)

5. Publish PublishLocales again by invoking the macro in the Visual Studio Command Prompt window. To view the Command Prompt window, on the **View** menu, point to **Other Windows** and then click **Command Window**, or press CTRL+ALT+A. In the Command Prompt window, type **macros**; auto-complete will provide a list of available macros. Select the following macro and press ENTER:

   Macros.MyMacros.PublishSpecificCulture.PublishProjectFirstProjectWithEnLocale

6. When the publish process succeeds, it will generate a message that says "Publish succeeded for PublishLocales\PublishLocales.vbproj. Publish language was 'en'." Click **OK** in the message box. When the publish Web page appears, click **Install**.

7. Look in C:\Inetpub\wwwroot\PublishLocales\en. You should see the installed files such as the manifests, setup.exe, and the publish Web page file, in addition to the localized resource DLL. (By default ClickOnce appends a .deploy extension on EXEs and DLLs; you can remove this extension after deployment.)

See Also

Publishing ClickOnce Applications
d23105d8-34fe-4ad9-8278-fae2c660aeac
762169e6-f83f-44b4-bffa-d0f107cae9a3
6716f820-1feb-48ad-a718-27eb6b473c5a

Securing ClickOnce Applications

Visual Studio 2015

ClickOnce applications are subject to code access security constraints in the .NET Framework to help limit the access that code has to protected resources and operations. For that reason, it is important that you understand the implications of code access security to write your ClickOnce applications accordingly. Your applications can use Full Trust or use partial zones, such as the Internet and Intranet zones, to limit access.

Additionally, ClickOnce uses certificates to verify the authenticity of the application's publisher, and to sign the application and deployment manifests to prove that the files have not been tampered with. Signing is an optional step, which makes it easier to change the application files after the manifests are generated. However, without signed manifests, it is difficult to ensure that the application installer is not tampered in man-in-the-middle security attacks. For this reason, we recommend that you sign your application and deployment manifests to help secure your applications.

Zones

Applications that are deployed using ClickOnce technology are restricted to a set of permissions and actions that are defined by the security zone. Security zones are defined in Internet Explorer, and are based on the location of the application. The following table lists the default permissions based on the deployment location:

Deployment Location	Security Zone
Run from Web	Internet Zone
Install from Web	Internet Zone
Install from network file share	Local Intranet Zone
Install from CD-ROM	Full Trust

The default permissions are based on the location from which the original version of the application was deployed; updates to the application will inherit those permissions. If the application is configured to check for updates from a Web or network location and a newer version is available, the original installation can receive permissions for the Internet or Intranet zone instead of full-trust permissions. To prevent users from being prompted, a System Administrator can specify a ClickOnce deployment policy that defines a specific application publisher as a trusted source. For computers on which this policy is deployed, permissions will be granted automatically and the user will not be prompted. For more information, see Trusted Application Deployment Overview. To configure trusted application deployment, the certificate can be installed to the machine or enterprise level. For more information, see How to: Add a Trusted Publisher to a Client Computer for ClickOnce Applications.

Code Access Security Policies

Permissions for an application are determined by the settings in the <trustInfo> Element (ClickOnce Application) element of the application manifest. Visual Studio automatically generates this information based on the settings on the project's **Security** property page. A ClickOnce application is granted only the specific permissions that it requests. For example, where file access requires full-trust permissions, if the application requests file-access permission, it will only be granted file-access permission, not full-trust permissions. When developing your ClickOnce application, you should make sure that you request only the specific permissions that the application needs. In most cases, you can use the Internet or Local Intranet zones to limit your application to partial trust. For more information, see How to: Set a Security Zone for a ClickOnce Application. If your application requires custom permissions, you can create a custom zone. For more information, see How to: Set Custom Permissions for a ClickOnce Application.

Including a permission that is not part of the default permission set for the zone from which the application is deployed will cause the end user to be prompted to grant permission at install or update time. To prevent users from being prompted, a system administrator can specify a ClickOnce deployment policy that defines a specific application publisher as a trusted source. On computers where this policy is deployed, permissions will automatically be granted and the user will not be prompted.

As a developer, it is your responsibility to make sure that your application will run with the appropriate permissions. If the application requests permissions outside of a zone during run time, a security exception may appear. Visual Studio enables you to debug your application in the target security zone. and provides help in developing secure applications. For more information, see How to: Debug a ClickOnce Application with Restricted Permissions.

For more information about code access security and ClickOnce, see Code Access Security for ClickOnce Applications.

Code-Signing Certificates

To publish an application by using ClickOnce deployment, you can sign the application and deployment manifests for the application by using a public/private key pair. The tools for signing a manifest are available on the **Signing** page of the **Project Designer**. For more information, see Signing Page, Project Designer. Alternatively, you can sign the manifests with a key file during the publishing process, using the Publish Wizard.

After the manifests are signed, the publisher information based on the Authenticode signature will be displayed to the user in the permissions dialog box during installation, to show the user that the application originated from a trusted source.

For more information about ClickOnce and certificates, see ClickOnce and Authenticode.

ASP.NET Form-Based Authentication

If you want to control which deployments each user can access, you should not enable anonymous access to ClickOnce applications deployed on a Web server. Rather, you would enable users access to the deployments you have installed based on a user's identity using Windows authentication.

ClickOnce does not support ASP.NET forms-based authentication because it uses persistent cookies; these present a security risk because they reside in the Internet Explorer cache and can be hacked. Therefore, if you are deploying ClickOnce applications, any authentication scenario besides Windows authentication is unsupported.

Passing Arguments

An additional security consideration occurs if you have to pass arguments into a ClickOnce application. ClickOnce enables developers to supply a query string to applications deployed over the Web. The query string takes the form of a series of name-value pairs at the end of the URL used to start the application:

http://servername.adatum.com/WindowsApp1.application?username=joeuser

By default, query-string arguments are disabled. To enable them, the attribute `trustUrlParameters` must be set in the application's deployment manifest. This value can be set from Visual Studio and from MageUI.exe. For detailed steps on how to enable passing query strings, see How to: Retrieve Query String Information in an Online ClickOnce Application.

You should never pass arguments retrieved through a query string to a database or to the command line without checking the arguments to make sure that they are safe. Unsafe arguments are ones that include database or command line escape characters that could allow a malicious user to manipulate your application into executing arbitrary commands.

Note

Query-string arguments are the only way to pass arguments to a ClickOnce application at startup. You cannot pass arguments to a ClickOnce application from the command line.

Deploying Obfuscated Assemblies

You might want to obfuscate your application by using Dotfuscator to prevent others from reverse engineering the code. However, assembly obfuscation is not integrated into the Visual Studio IDE or the ClickOnce deployment process. Therefore, you will have to perform the obfuscation outside of the deployment process, perhaps using a post-build step. After you build the project, you would perform the following steps manually, outside of Visual Studio:

1. Perform the obfuscation by using Dotfuscator.
2. Use Mage.exe or MageUI.exe to generate the ClickOnce manifests and sign them. For more information, see Mage.exe (Manifest Generation and Editing Tool) and MageUI.exe (Manifest Generation and Editing Tool, Graphical Client).
3. Manually publish (copy) the files to your deployment source location (Web server, UNC share, or CD-ROM).

See Also

ClickOnce Security and Deployment
Choosing a ClickOnce Deployment Strategy

ClickOnce and Authenticode

Visual Studio 2015

Authenticode is a Microsoft technology that uses industry-standard cryptography to sign application code with digital certificates that verify the authenticity of the application's publisher. By using Authenticode for application deployment, ClickOnce reduces the risk of a Trojan horse. A Trojan horse is a virus or other harmful program that a malicious third party misrepresents as a legitimate program coming from an established, trustworthy source. Signing ClickOnce deployments with a digital certificate is an optional step to verify that the assemblies and files are not tampered.

The following sections describe the different types of digital certificates used in Authenticode, how certificates are validated using Certificate Authorities (CAs), the role of time-stamping in certificates, and the methods of storage available for certificates.

Authenticode and Code Signing

A *digital certificate* is a file that contains a cryptographic public/private key pair, along with metadata describing the publisher to whom the certificate was issued and the agency that issued the certificate.

There are various types of Authenticode certificates. Each one is configured for different types of signing. For ClickOnce applications, you must have an Authenticode certificate that is valid for code signing. If you attempt to sign a ClickOnce application with another type of certificate, such as a digital e-mail certificate, it will not work. For more information, see Introduction to Code Signing.

You can obtain a certificate for code signing in one of three ways:

- Purchase one from a certificate vendor.
- Receive one from a group in your organization responsible for creating digital certificates.
- Generate your own certificate with MakeCert.exe, which is included with the Windows Software Development Kit (SDK).

How Using Certificate Authorities Helps Users

A certificate generated using the MakeCert.exe utility is commonly called a *self-cert* or a *test cert*. This kind of certificate works much the same way that a .snk file works in the .NET Framework. It consists solely of a public/private cryptographic key pair, and contains no verifiable information about the publisher. You can use self-certs to deploy ClickOnce applications with high trust on an intranet. However, when these applications run on a client computer, ClickOnce will identify them as coming from an Unknown Publisher. By default, ClickOnce applications signed with self-certs and deployed over the Internet cannot utilize Trusted Application Deployment.

By contrast, if you receive a certificate from a CA, such as a certificate vendor, or a department within your enterprise, the certificate offers more security for your users. It not only identifies the publisher of the signed software, but it verifies that identity by checking with the CA that signed it. If the CA is not the root authority, Authenticode will also "chain" back to the root authority to verify that the CA is authorized to issue certificates. For greater security, you should use a certificate issued by a CA whenever possible.

For more information about generating self-certs, see Makecert.exe (Certificate Creation Tool).

Timestamps

The certificates used to sign ClickOnce applications expire after a certain length of time, typically twelve months. In order to remove the need to constantly re-sign applications with new certificates, ClickOnce supports timestamp. When an application is signed with a timestamp, its certificate will continue to be accepted even after expiration, provided the timestamp is valid. This allows ClickOnce applications with expired certificates, but valid timestamps, to download and run. It also allows installed applications with expired certificates to continue to download and install updates.

To include a timestamp in an application server, a timestamp server must be available. For information about how to select a timestamp server, see How to: Sign Application and Deployment Manifests.

Updating Expired Certificates

In earlier versions of the .NET Framework, updating an application whose certificate had expired could cause that application to stop functioning. To resolve this problem, use one of the following methods:

- Update the .NET Framework to version 2.0 SP1 or later on Windows XP, or version 3.5 or later on Windows Vista.
- Uninstall the application, and reinstall a new version with a valid certificate.
- Create a command-line assembly that updates the certificate. Step-by-step information about this process can be found at Microsoft Support Article 925521.

Storing Certificates

- You can store certificates as a .pfx file on your file system, or you can store them inside of a key container. A user on a Windows domain can have a number of key containers. By default, MakeCert.exe will store certificates in your personal key container, unless you specify that it should save it to a .pfx instead. Mage.exe and MageUI.exe, the Windows SDK tools for creating ClickOnce deployments, enable you to use certificates stored in either fashion.

See Also

ClickOnce Security and Deployment
Securing ClickOnce Applications
Trusted Application Deployment Overview
Mage.exe (Manifest Generation and Editing Tool)

Trusted Application Deployment Overview

Visual Studio 2015

This topic provides an overview of how to deploy ClickOnce applications that have elevated permissions by using the Trusted Application Deployment technology.

Trusted Application Deployment, part of the ClickOnce deployment technology, makes it easier for organizations of any size to grant additional permissions to a managed application in a safer, more secure manner without user prompting. With Trusted Application Deployment, an organization can just configure a client computer to have a list of trusted publishers, who are identified using Authenticode certificates. Thereafter, any ClickOnce application signed by one of these trusted publishers receives a higher level of trust.

Note

Trusted Application Deployment requires one-time configuration of a user's computer. In managed desktop environments, this configuration can be performed by using global policy. If this is not what you want for your application, use permission elevation instead. For more information, see Securing ClickOnce Applications.

Trusted Application Deployment Basics

The following table shows the objects and roles that are involved in Trusted Application Deployment.

Object or role	Description
administrator	The organizational entity responsible for updating and maintaining client computers
trust manager	The subsystem within the common language runtime (CLR) responsible for enforcing client application security.
publisher	The entity that writes and maintains the application.
deployer	The entity that packages and distributes the application to users.
certificate	A cryptographic signature that consists of a public and private key; generally issued by a certification authority (CA) that can vouch for its authenticity.
Authenticode certificate	A certificate with embedded metadata describing, among other things, the uses for which the certificate can be employed.
certification authority	An organization that verifies the identity of publishers and issues them certificates embedded with the publisher's metadata.

root authority	A certification authority that authorizes other Certificate Authorities to issue certificates.
key container	A logical storage space in Microsoft Windows for storing certificates.
trusted publisher	A publisher whose Authenticode certificate has been added to a certificate trust list (CTL) on a client computer.

In larger organizations, the publisher and deployer are frequently two separate entities:

- The publisher is the group that creates the ClickOnce application.
- The deployer is the group, typically the information technology (IT) department, that distributes ClickOnce application to corporate enterprise desktop computers.

You must follow these steps to take advantage of Trusted Application Deployment:

1. Obtain a certificate for the publisher.
2. Add the publisher to the trusted publishers store on all clients.
3. Create your ClickOnce application.
4. Sign the deployment manifest with the publisher's certificate.
5. Publish the application deployment to client computers.

Obtain a Certificate for the Publisher

Digital certificates are a core component of the Microsoft Authenticode authentication and security system. Authenticode is a standard part of the Windows operating system. All ClickOnce applications must be signed with a digital certificate, regardless of whether they participate in Trusted Application Deployment. For a full explanation of how Authenticode works with ClickOnce, see ClickOnce and Authenticode.

Add the Publisher to the Trusted Publishers Store

For your ClickOnce application to receive a higher level of trust, you must add your certificate as a trusted publisher to each client computer on which the application will run. Performing this task is a one-time configuration. After it is completed, you can deploy as many ClickOnce applications signed with your publisher's certificate as you want, and they will all run with high trust.

If you are deploying your application in a managed desktop environment; for example, a corporate intranet running the Windows operating system; you can add trusted publishers to a client's store by creating a new certificate trust list (CTL) with Group Policy. For more information, see Create a certificate trust list for a Group Policy object.

If you are not deploying your application in a managed desktop environment, you have the following options for adding a certificate to the trusted publisher store:

- The System.Security.Cryptography namespace.
- CertMgr.exe, which is a component of Internet Explorer and therefore exists on Windows 98 and all later versions. For more information, see Certmgr.exe (Certificate Manager Tool).

Create a ClickOnce Application

A ClickOnce application is a .NET Framework client application combined with manifest files that describe the application and supply installation parameters. You can turn your program into a ClickOnce application by using the **Publish** command in Visual Studio. Alternatively, you can generate all the files required for ClickOnce deployment by using tools that are included with the Windows Software Development Kit (SDK). For detailed steps about ClickOnce deployment, see Walkthrough: Manually Deploying a ClickOnce Application.

Trusted Application Deployment is specific to ClickOnce, and can only be used with ClickOnce applications.

Sign the Deployment

After obtaining your certificate, you must use it to sign your deployment. If you are deploying your application by using the Visual Studio Publish wizard, the wizard will automatically generate a test certificate for you if you have not specified a certificate yourself. You can also use the Visual Studio Project Designer window, however, to supply a certificate provided by a CA. Also see How to: Publish a ClickOnce Application using the Publish Wizard or How to: Publish a ClickOnce Application using the Publish Wizard.

Caution
We do not recommend that the application be deployed with a test certificate.

You can also sign the application by using the Mage.exe or MageUI.exe SDK tools. For more information, see Walkthrough: Manually Deploying a ClickOnce Application. For a full list of command-line options related to deployment signing, see Mage.exe (Manifest Generation and Editing Tool).

Publish the Application

As soon as you have signed your ClickOnce manifests, the application is ready to publish to your install location. The installation location can be a Web server, a file share, or the local disk. When a client accesses the deployment manifest for the first time, the trust manager must choose whether the ClickOnce application has been granted authority or not to run at a higher level of trust by an installed trusted publisher. The trust manager makes this choice by comparing the certificate used to sign the deployment with the certificates stored in the client's trusted publisher store. If the trust manager finds a match, the application runs with high trust.

Trusted Application Deployment and Permission Elevation

If the current publisher is not a trusted publisher, trust manager will use Permission Elevation to query the user about whether he or she wants to grant your application elevated permissions. If permission elevation is disabled by the administrator, however, the application cannot obtain permission to run. The application will not run and no notification will be displayed to the user. For more information about Permission Elevation, see Securing ClickOnce Applications.

Limitations of Trusted Application Deployment

You can use Trusted Application Deployment to grant elevated trust to ClickOnce applications deployed over the Web or through an enterprise file share. You do not have to use Trusted Application Deployment for ClickOnce applications distributed on a CD, because, by default, these applications are granted full trust.

See Also

Mage.exe (Manifest Generation and Editing Tool)
Walkthrough: Manually Deploying a ClickOnce Application

Code Access Security for ClickOnce Applications

Visual Studio 2015

ClickOnce applications are based on the .NET Framework and are subject to code access security constraints. For this reason, it is important that you understand the implications of code access security and write your ClickOnce applications accordingly.

Code access security is a mechanism in the .NET Framework that helps limit the access that code has to protected resources and operations. You should configure the code access security permissions for your ClickOnce application to use the zone appropriate for the location of the application installer. In most cases, you can choose the **Internet** zone for a limited set of permissions or the **Local Intranet** zone for a greater set of permissions.

Default ClickOnce Code Access Security

By default, a ClickOnce application receives Full Trust permissions when it is installed or run on a client computer.

- An application that has Full Trust permissions has unrestricted access to resources such as the file system and the registry. This potentially allows your application (and the end user's system) to be exploited by malicious code.
- When an application requires Full Trust permissions, the end user may be prompted to grant permissions to the application. This means that the application does not truly provide a ClickOnce experience, and the prompt can potentially be confusing to less experienced users.

> **Note**
>
> When installing an application from removable media such as a CD-ROM, the user is not prompted. In addition, a network administrator can configure network policy so that users are not prompted when they install an application from a trusted source. For more information, see Trusted Application Deployment Overview.

To restrict the permissions for a ClickOnce application, you can modify the code access security permissions for your application to request the zone that best fits the permissions that your application requires. In most cases, you can select the zone from which the application is being deployed. For example, if your application is an enterprise application, you can use the **Local Intranet** zone. If your application is an internet application, you can use the **Internet** zone.

Configuring Security Permissions

You should always configure your ClickOnce application to request the appropriate zone to limit the code access security permissions. You can configure security permissions on the **Security** page of the **Project Designer**.

The **Security** page in the **Project Designer** contains an **Enable ClickOnce Security Settings** check box. When this check

box is selected, security permission requests are added to the deployment manifest for your application. At installation time, the user will be prompted to grant permissions if the requested permissions exceed the default permissions for the zone from which the application is deployed. For more information, see How to: Enable ClickOnce Security Settings.

Applications deployed from different locations are granted different levels of permissions without prompting. For example, when an application is deployed from the Internet, it receives a highly restrictive set of permissions. When installed from a local Intranet, it receives more permissions, and when installed from a CD-ROM, it receives Full Trust permissions.

As a starting point for configuring permissions, you can select a security zone from the **Zone** list on the **Security** page. If your application will potentially be deployed from more than one zone, select the zone with the least permissions. For more information, see How to: Set a Security Zone for a ClickOnce Application.

The properties that can be set vary by permission set; not all permission sets have configurable properties. For more information about the full list of permissions that your application can request, see System.Security.Permissions. For more information about how to set permissions for a custom zone, see How to: Set Custom Permissions for a ClickOnce Application.

Debugging an Application That Has Restricted Permissions

As a developer, you most likely run your development computer with Full Trust permissions. Therefore, you do not see the same security exceptions when you debug the application that users may see when they run it with restricted permissions.

In order to catch these exceptions, you have to debug the application with the same permissions as the end user. Debugging with restricted permissions can be enabled on the **Security** page of the **Project Designer**.

When you debug an application with restricted permissions, exceptions will be raised for any code security demands that have not been enabled on the **Security** page. An exception helper will appear, providing suggestions about how to modify your code to prevent the exception.

In addition, when you write code, the IntelliSense feature in the Code Editor will disable any members that are not included in the security permissions that you have configured.

For more information, see How to: Debug a ClickOnce Application with Restricted Permissions.

Security Permissions for Browser-Hosted Applications

Visual Studio provides the following project types for Windows Presentation Foundation (WPF) applications:

- WPF Windows Application
- WPF Web Browser Application
- WPF Custom Control Library
- WPF Service Library

Of these project types, only WPF Web Browser Applications are hosted in a Web browser and therefore require special

deployment and security settings. The default security settings for these applications are as follows:

- **Enable ClickOnce Security Settings**
- **This is a partial trust application**
- **Internet zone** (with default permission set for WPF Web Browser Applications selected)

In the **Advanced Security Settings** dialog box, the **Debug this application with the selected permission set** check box is selected and disabled. This is because Debug In Zone cannot be turned off for browser-hosted applications.

See Also

Securing ClickOnce Applications
How to: Enable ClickOnce Security Settings
How to: Set a Security Zone for a ClickOnce Application
How to: Set Custom Permissions for a ClickOnce Application
How to: Debug a ClickOnce Application with Restricted Permissions
Trusted Application Deployment Overview
Security Page, Project Designer

How to: Enable ClickOnce Security Settings

Visual Studio 2015

Code access security for ClickOnce applications must be enabled in order to publish the application. This is done automatically when you publish an application using the Publish wizard.

In some cases, enabling code access security can impact performance when building or debugging your application; in these cases, you may wish to temporarily disable the security settings.

ClickOnce security settings can be enabled or disabled on the **Security** page of the **Project Designer**.

To enable ClickOnce security settings

1. With a project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.
2. Click the **Security** tab.
3. Select the **Enable ClickOnce Security Settings** check box.

   You can now customize the security settings for your application on the Security page.

Note
This check box is automatically selected each time the application is published with the **Publish** wizard.

To disable ClickOnce security settings

1. With a project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.
2. Click the **Security** tab.
3. Clear the **Enable ClickOnce Security Settings** check box.

   Your application will be run with the full trust security settings; any settings on the **Security** page will be ignored.

Note
Each time the application is published with the Publish wizard, this check box will be selected; you must clear it again after each successful publish.

See Also

Securing ClickOnce Applications
Code Access Security for ClickOnce Applications
Securing ClickOnce Applications

How to: Set a Security Zone for a ClickOnce Application

Visual Studio 2015

When setting code access security permissions for a ClickOnce application, you need to start with a base set of permissions on the **Security** page of the **Project Designer**.

In most cases, you can also choose the **Internet** zone which contains a limited set of permissions, or the **Local Intranet** zone which contains a greater set of permissions. If your application requires custom permissions, you can do so by choosing the **Custom** security zone. For more information about setting custom permissions, see How to: Set Custom Permissions for a ClickOnce Application.

To set a security zone

1. With a project selected in **Solution Explorer**, on the **Project** menu click **Properties**.
2. Click the **Security** tab.
3. Select the **Enable ClickOnce Security Settings** check box.
4. Select the **This is a partial trust application** option button.

   The controls in the **ClickOnce security permissions** section are enabled.
5. In the **Zone your application will be installed from** drop-down list, select a security zone.

See Also

How to: Set Custom Permissions for a ClickOnce Application
Securing ClickOnce Applications
Code Access Security for ClickOnce Applications
Securing ClickOnce Applications

How to: Set Custom Permissions for a ClickOnce Application

Visual Studio 2015

You can deploy a ClickOnce application that uses default permissions for the Internet or Local Intranet zones. Alternatively, you can create a custom zone for the specific permissions that the application needs. You can do this by customizing the security permissions on the **Security** page of the **Project Designer**.

To customize a permission

1. With a project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.
2. Click the **Security** tab.
3. Select the **Enable ClickOnce Security Settings** check box.
4. Select the **This is a partial trust application** option button.

   The controls in the **ClickOnce security permissions** section are enabled.
5. From the **Zone your application will be installed from** drop-down list, click **(Custom)**.
6. Click **Edit Permissions XML**.

   The app.manifest file opens in the XML Editor.
7. Before the `</applicationRequestMinimum>` element, add XML code for permissions that your application requires.

> **Note**
>
> You can use the `ToXml` method of a permission set to generate the XML code for the application manifest. For example, to generate the XML for the EnvironmentPermission permission set, call the ToXml method. For more information about the structure of the permission set XML, see NIB: How to: Import a Permission Set by Using an XML File.

See Also

Securing ClickOnce Applications
Code Access Security for ClickOnce Applications
Securing ClickOnce Applications

How to: Add a Trusted Publisher to a Client Computer for ClickOnce Applications

Visual Studio 2015

With Trusted Application Deployment, you can configure client computers so that your ClickOnce applications run with a higher level of trust without prompting the user. The following procedures show how to use the command-line tool CertMgr.exe to add a publisher's certificate to the Trusted Publishers store on a client computer.

The commands you use vary slightly depending on whether the certificate authority (CA) that issued your certificate is part of a client's trusted root. If a Windows client computer is part of a domain, it will contain, in a list, CAs that are considered trusted roots. This list is usually configured by the system administrator. If your certificate was issued by one of these trusted roots, or by a CA that chains to one of these trusted roots, you can add the certificate to the client's trusted root store. If, on the other hand, your certificate was not issued by one of these trusted roots, you must add the certificate to both the client's Trusted Root store and Trusted Publisher store.

Note

You must add certificates this way on every client computer to which you plan to deploy a ClickOnce application that requires elevated permissions. You add the certificates either manually or through an application you deploy to your clients. You only need to configure these computers once, after which you can deploy any number of ClickOnce applications signed with the same certificate.

You may also add a certificate to a store programmatically using the X509Store class.

For an overview of Trusted Application Deployment, see Trusted Application Deployment Overview.

To add a certificate to the Trusted Publishers store under the trusted root

1. Obtain a digital certificate from a CA.
2. Export the certificate into the Base64 X.509 (.cer) format. For more information about certificate formats, see Export a Certificate.
3. From the command prompt on client computers, run the following command:

   certmgr.exe -add certificate.cer -c -s -r localMachine TrustedPublisher

To add a certificate to the Trusted Publishers store under a different root

1. Obtain a digital certificate from a CA.
2. Export the certificate into the Base64 X.509 (.cer) format. For more information about certificate formats, see Export a Certificate.
3. From the command prompt on client computers, run the following command:

   certmgr.exe -add good.cer -c -s -r localMachine Root

   certmgr.exe -add good.cer -c -s -r localMachine TrustedPublisher

See Also

Walkthrough: Manually Deploying a ClickOnce Application
Securing ClickOnce Applications
Code Access Security for ClickOnce Applications
ClickOnce and Authenticode
Trusted Application Deployment Overview
How to: Enable ClickOnce Security Settings
How to: Set a Security Zone for a ClickOnce Application
How to: Set Custom Permissions for a ClickOnce Application
How to: Debug a ClickOnce Application with Restricted Permissions
How to: Add a Trusted Publisher to a Client Computer for ClickOnce Applications
How to: Re-sign Application and Deployment Manifests
How to: Configure the ClickOnce Trust Prompt Behavior

How to: Re-sign Application and Deployment Manifests

Visual Studio 2015

After you make changes to deployment properties in the application manifest for Windows Forms applications, Windows Presentation Foundation applications (xbap), or Office solutions, you must re-sign both the application and deployment manifests with a certificate. This process helps ensure that tampered files are not installed on end user computers.

Another scenario where you might re-sign the manifests is when your customers want to sign the application and deployment manifests with their own certificate.

Re-signing the Application and Deployment Manifests

This procedure assumes that you have already made changes to your application manifest file (.manifest). For more information, see How to: Change Deployment Properties.

To re-sign the application and deployment manifests with Mage.exe

1. Open a **Visual Studio Command Prompt** window.
2. Change directories to the folder that contains the manifest files that you want to sign.
3. Type the following command to sign the application manifest file. Replace ManifestFileName with the name of your manifest file plus the extension. Replace Certificate with the relative or fully qualified path of the certificate file and replace Password with the password for the certificate.

```
mage -sign ManifestFileName.manifest -CertFile Certificate -Password Password
```

For example, you could run the following command to sign an application manifest for an add-in, a Windows Form application, or a Windows Presentation Foundation browser application. Temporary certificates created by Visual Studio are not recommended for deployment into production environments.

```
mage -sign WindowsFormsApplication1.exe.manifest -CertFile
..\WindowsFormsApplication1_TemporaryKey.pfx
mage -sign ExcelAddin1.dll.manifest -CertFile ..\ExcelAddIn1_TemporaryKey.pfx
mage -sign WpfBrowserApplication1.exe.manifest -CertFile
..\WpfBrowserApplication1_TemporaryKey.pfx
```

4. Type the following command to update and sign the deployment manifest file, replacing the placeholder names as in the previous step.

```
mage -update DeploymentManifest -appmanifest ApplicationManifest -CertFile
Certificate -Password Password
```

For example, you could run the following command to update and sign a deployment manifest for an Excel add-in, a Windows Forms application, or a Windows Presentation Foundation browser application.

```
mage -update WindowsFormsApplication1.application -appmanifest
WindowsFormsApplication1.exe.manifest -CertFile
..\WindowsFormsApplication1_TemporaryKey.pfx
mage -update ExcelAddin1.vsto -appmanifest ExcelAddin1.dll.manifest -CertFile
..\ExcelAddIn1_TemporaryKey.pfx
mage -update WpfBrowserApplication1.xbap -appmanifest
WpfBrowserApplication1.exe.manifest -CertFile
..\WpfBrowserApplication1_TemporaryKey.pfx
```

5. Optionally, copy the master deployment manifest (publish*appname*.application) to your version deployment directory (publish\Application Files*appname_version*).

Updating and Re-signing the Application and Deployment Manifests

This procedure assumes that you have already made changes to your application manifest file (.manifest), but that there are other files that were updated. When files are updated, the hash that represents the file must also be updated.

To update and re-sign the application and deployment manifests with Mage.exe

1. Open a **Visual Studio Command Prompt** window.
2. Change directories to the folder that contains the manifest files that you want to sign.
3. Remove the .deploy file extension from the files in the publish output folder.
4. Type the following command to update the application manifest with the new hashes for the updated files and sign the application manifest file. Replace ManifestFileName with the name of your manifest file plus the extension. Replace Certificate with the relative or fully qualified path of the certificate file and replace Password with the password for the certificate.

```
mage -update ManifestFileName.manifest -CertFile Certificate -Password Password
```

For example, you could run the following command to sign an application manifest for an add-in, a Windows Form application, or a Windows Presentation Foundation browser application. Temporary certificates created by Visual Studio are not recommended for deployment into production environments.

```
mage -update WindowsFormsApplication1.exe.manifest -CertFile
..\WindowsFormsApplication1_TemporaryKey.pfx
mage -update ExcelAddin1.dll.manifest -CertFile ..\ExcelAddIn1_TemporaryKey.pfx
mage -update WpfBrowserApplication1.exe.manifest -CertFile
..\WpfBrowserApplication1_TemporaryKey.pfx
```

5. Type the following command to update and sign the deployment manifest file, replacing the placeholder names as in the previous step.

```
mage -update DeploymentManifest -appmanifest ApplicationManifest -CertFile
Certificate -Password Password
```

For example, you could run the following command to update and sign a deployment manifest for an Excel add-in, a Windows Forms application, or a Windows Presentation Foundation browser application.

```
mage -update WindowsFormsApplication1.application -appmanifest
WindowsFormsApplication1.exe.manifest -CertFile
..\WindowsFormsApplication1_TemporaryKey.pfx
mage -update ExcelAddin1.vsto -appmanifest ExcelAddin1.dll.manifest -CertFile
..\ExcelAddIn1_TemporaryKey.pfx
mage -update WpfBrowserApplication1.xbap -appmanifest
WpfBrowserApplication1.exe.manifest -CertFile
..\WpfBrowserApplication1_TemporaryKey.pfx
```

6. Add the .deploy file extension back to the files, except the application and deployment manifest files.

7. Optionally, copy the master deployment manifest (publish*appname*.application) to your version deployment directory (publish\Application Files*appname_version*).

See Also

Securing ClickOnce Applications
Code Access Security for ClickOnce Applications
ClickOnce and Authenticode
Trusted Application Deployment Overview
How to: Enable ClickOnce Security Settings
How to: Set a Security Zone for a ClickOnce Application
How to: Set Custom Permissions for a ClickOnce Application
How to: Debug a ClickOnce Application with Restricted Permissions
How to: Add a Trusted Publisher to a Client Computer for ClickOnce Applications
How to: Configure the ClickOnce Trust Prompt Behavior

How to: Configure the ClickOnce Trust Prompt Behavior

Visual Studio 2015

You can configure the ClickOnce trust prompt to control whether end users are given the option of installing ClickOnce applications, such as Windows Forms applications, Windows Presentation Foundation applications, console applications, WPF browser applications, and Office solutions. You configure the trust prompt by setting registry keys on each end user's computer.

The following table shows the configuration options that can be applied to each of the five zones (Internet, UntrustedSites, MyComputer, LocalIntranet, and TrustedSites).

Option	Registry setting value	Description
Enable the trust prompt.	**Enabled**	The ClickOnce trust prompt is display so that end users can grant trust to ClickOnce applications.
Restrict the trust prompt.	**AuthenticodeRequired**	The ClickOnce trust prompt is only displayed if ClickOnce applications are signed with a certificate that identifies the publisher.
Disable the trust prompt.	**Disabled**	The ClickOnce trust prompt is not displayed for any ClickOnce applications that are not signed with an explicitly trusted certificate.

The following table shows the default behavior for each zone. The Applications column refers to Windows Forms applications, Windows Presentation Foundation applications, WPF browser applications, and console applications.

Zone	Applications	Office solutions
MyComputer	**Enabled**	**Enabled**
LocalIntranet	**Enabled**	**Enabled**
TrustedSites	**Enabled**	**Enabled**
Internet	**Enabled**	**AuthenticodeRequired**
UntrustedSites	**Disabled**	**Disabled**

You can override these settings by enabling, restricting, or disabling the ClickOnce trust prompt.

Enabling the ClickOnce Trust Prompt

Enable the trust prompt for a zone when you want end users to be presented with the option of installing and running any ClickOnce application that comes from that zone.

To enable the ClickOnce trust prompt by using the registry editor

1. Open the registry editor:

    a. Click **Start**, and then click **Run**.

    b. In the **Open** box, type **regedit32**, and then click **OK**.

2. Find the following registry key:

    \HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\.NETFramework\Security\TrustManager\PromptingLevel

    If the key does not exist, create it.

3. Add the following subkeys as **String Value**, if they do not already exist, with the associated values shown in the following table.

String Value subkey	Value
Internet	**Enabled**
UntrustedSites	**Disabled**
MyComputer	**Enabled**
LocalIntranet	**Enabled**
TrustedSites	**Enabled**

For Office solutions, **Internet** has the default value **AuthenticodeRequired** and **UntrustedSites** has the value **Disabled**. For all others, **Internet** has the default value **Enabled**.

To enable the ClickOnce trust prompt programmatically

1. Create a Visual Basic or Visual C# console application in Visual Studio.

2. Open the Program.vb or Program.cs file for editing and add the following code.

VB

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.LocalMachine.CreateSubKey("SOFTWARE\MICROSOFT
\.NETFramework\Security\TrustManager\PromptingLevel")
```

```
key.SetValue("MyComputer", "Enabled")
key.SetValue("LocalIntranet", "Enabled")
key.SetValue("Internet", "Enabled")
key.SetValue("TrustedSites", "Enabled")
key.SetValue("UntrustedSites", "Disabled")
key.Close()
```

3. Build and run the application.

Restricting the ClickOnce Trust Prompt

Restrict the trust prompt so that solutions must be signed with Authenticode certificates that have known identity before users are prompted for a trust decision.

To restrict the ClickOnce trust prompt by using the registry editor

1. Open the registry editor:

   a. Click **Start**, and then click **Run**.

   b. In the **Open** box, type **regedit**, and then click **OK**.

2. Find the following registry key:

   \HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\.NETFramework\Security\TrustManager\PromptingLevel

   If the key does not exist, create it.

3. Add the following subkeys as **String Value**, if they do not already exist, with the associated values shown in the following table.

String Value subkey	Value
UntrustedSites	**Disabled**
Internet	**AuthenticodeRequired**
MyComputer	**AuthenticodeRequired**
LocalIntranet	**AuthenticodeRequired**
TrustedSites	**AuthenticodeRequired**

To restrict the ClickOnce trust prompt programmatically

1. Create a Visual Basic or Visual C# console application in Visual Studio.
2. Open the Program.vb or Program.cs file for editing and add the following code.

VB

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.LocalMachine.CreateSubKey("SOFTWARE\MICROSOFT
\.NETFramework\Security\TrustManager\PromptingLevel")
key.SetValue("MyComputer", "AuthenticodeRequired")
key.SetValue("LocalIntranet", "AuthenticodeRequired")
key.SetValue("Internet", "AuthenticodeRequired")
key.SetValue("TrustedSites", "AuthenticodeRequired")
key.SetValue("UntrustedSites", "Disabled")
key.Close()
```

3. Build and run the application.

Disabling the ClickOnce Trust Prompt

You can disable the trust prompt so that end users are not given the option to install solutions that are not already trusted in their security policy.

To disable the ClickOnce trust prompt by using the registry editor

1. Open the registry editor:
   a. Click **Start**, and then click **Run**.
   b. In the **Open** box, type **regedit**, and then click **OK**.
2. Find the following registry key:

   \HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\.NETFramework\Security\TrustManager\PromptingLevel

   If the key does not exist, create it.
3. Add the following subkeys as **String Value**, if they do not already exist, with the associated values shown in the following table.

String Value subkey	Value
UntrustedSites	**Disabled**
Internet	**Disabled**
MyComputer	**Disabled**

LocalIntranet	**Disabled**
TrustedSites	**Disabled**

To disable the ClickOnce trust prompt programmatically

1. Create a Visual Basic or Visual C# console application in Visual Studio.
2. Open the Program.vb or Program.cs file for editing and add the following code.

VB

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.LocalMachine.CreateSubKey("SOFTWARE\MICROSOFT
\.NETFramework\Security\TrustManager\PromptingLevel")
key.SetValue("MyComputer", "Disabled")
key.SetValue("LocalIntranet", "Disabled")
key.SetValue("Internet", "Disabled")
key.SetValue("TrustedSites", "Disabled")
key.SetValue("UntrustedSites", "Disabled")
key.Close()
```

3. Build and run the application.

See Also

Securing ClickOnce Applications
Code Access Security for ClickOnce Applications
ClickOnce and Authenticode
Trusted Application Deployment Overview
How to: Enable ClickOnce Security Settings
How to: Set a Security Zone for a ClickOnce Application
How to: Set Custom Permissions for a ClickOnce Application
How to: Debug a ClickOnce Application with Restricted Permissions
How to: Add a Trusted Publisher to a Client Computer for ClickOnce Applications
How to: Re-sign Application and Deployment Manifests

How to: Sign Setup Files with SignTool.exe (ClickOnce)

Visual Studio 2015

You can use SignTool.exe to sign a Setup program (setup.exe). This process helps ensure that tampered files are not installed on end-user computers.

By default, ClickOnce has signed manifests and a signed Setup program. However, if you want to change the parameters of the Setup program later, you must sign the Setup program later. If you change the parameters after the Setup program is signed, the signature becomes corrupted.

The following procedure generates unsigned manifests and an unsigned Setup program. Then, ClickOnce signing is enabled in Visual Studio to generate signed manifests. The Setup program is left unsigned so that the customer can sign the executable with their own certificate.

To generate an unsigned Setup program and sign later

1. On the development computer, install the certificate that you want the sign the manifests with.
2. Select the project in **Solution Explorer**.
3. On the **Project** menu, click *ProjectName* **Properties**.
4. In the **Signing** page, clear **Sign the ClickOnce manifests**.
5. In the **Publish** page, click **Prerequisites**.
6. Verify that all the prerequisites are selected, and then click **OK**.
7. In the **Publish** page, verify the publish settings and then click **Publish Now**.

   The solution publishes the unsigned application manifest, unsigned deployment manifest, version-specific files, and unsigned Setup program to the publishing folder location.
8. In the **Publish** page, click **Prerequisites**.
9. In the **Prerequisites** dialog box, clear **Create setup program to install prerequisite components**.
10. In the **Publish** page, verify the publish settings and then click **Publish Now**.

    The solution publishes the signed application manifest, signed deployment manifest, and version-specific files to the publishing folder location. The unsigned Setup program is not overwritten by the publish process.
11. At the customer site, open a command prompt.
12. Change to the directory that contains the .exe file.
13. Sign the .exe file with the following command:

```
signtool sign /sha1 CertificateHash Setup.exe
signtool sign /f CertFileName Setup.exe
```

For example, to sign the Setup program, use one of the following commands:

```
signtool sign /sha1 CCB... Setup.exe
signtool sign /f CertFileName Setup.exe
```

See Also

How to: Re-sign Application and Deployment Manifests

Choosing a ClickOnce Update Strategy

Visual Studio 2015

ClickOnce can provide automatic application updates. A ClickOnce application periodically reads its deployment manifest file to see whether updates to the application are available. If available, the new version of the application is downloaded and run. For efficiency, only those files that have changed are downloaded.

When designing a ClickOnce application, you have to determine which strategy the application will use to check for available updates. There are three basic strategies that you can use: checking for updates on application startup, checking for updates after application startup (running in a background thread), or providing a user interface for updates.

In addition, you can determine how often the application will check for updates, and you can make updates required.

Note

Application updates require network connectivity. If a network connection is not present, the application will run without checking for updates, regardless of the update strategy that you choose.

Note

In .NET Framework 2.0 and .NET Framework 3.0, any time your application checks for updates, before or after startup, or by using the System.Deployment.Application APIs, you must set `deploymentProvider` in the deployment manifest. The `deploymentProvider` element corresponds in Visual Studio to the **Update location** field on the **Updates** dialog box of the **Publish** tab. This rule is relaxed in .NET Framework 3.5. For more information, see Deploying ClickOnce Applications For Testing and Production Servers without Resigning.

Checking for Updates After Application Startup

By using this strategy, the application will attempt to locate and read the deployment manifest file in the background while the application is running. If an update is available, the next time that the user runs the application, he will be prompted to download and install the update.

This strategy works best for low-bandwidth network connections or for larger applications that might require lengthy downloads.

To enable this update strategy, click **After the application starts** in the **Choose when the application should check for updates** section of the **Application Updates** dialog box. Then specify an update interval in the section **Specify how frequently the application should check for updates**.

This is the same as changing the **Update** element in the deployment manifest as follows:

```
<!-- When to check for updates -->
<subscription>
   <update>
      <expiration maximumAge="6" unit="hours" />
   </update>
</subscription>
```

Checking for Updates Before Application Startup

The default strategy is to try to locate and read the deployment manifest file before the application starts. By using this strategy, the application will attempt to locate and read the deployment manifest file every time that the user starts the application. If an update is available, it will be downloaded and started; otherwise, the existing version of the application will be started.

This strategy works best for high-bandwidth network connections; the delay in starting the application may be unacceptably long over low-bandwidth connections.

To enable this update strategy, click **Before the application starts** in the **Choose when the application should check for updates** section of the **Application Updates** dialog box.

This is the same as changing the **Update** element in the deployment manifest as follows:

```
<!-- When to check for updates -->
<subscription>
   <update>
      <beforeApplicationStartup />
   </update>
</subscription>
```

Making Updates Required

There may be occasions when you want to require users to run an updated version of your application. For example, you might make a change to an external resource such as a Web service that would prevent the earlier version of your application from working correctly. In this case, you would want to mark your update as required and prevent users from running the earlier version.

Note

Although you can require updates by using the other update strategies, checking **Before the application starts** is the only way to guarantee that an older version cannot be run. When the mandatory update is detected on startup, the user must either accept the update or close the application.

To mark an update as required, click **Specify a minimum required version for this application** in the **Application**

Updates dialog box, and then specify the publish version (**Major**, **Minor**, **Build**, **Revision**), which specifies the lowest version number of the application that can be installed.

This is the same as setting the **minimumRequiredVersion** attribute of the **Deployment** element in the deployment manifest; for example:

```
<deployment install="true" minimumRequiredVersion="1.0.0.0">
```

Specifying Update Intervals

You can also specify how often the application checks for updates. To do this, specify that the application check for updates after startup as described in "Checking for Updates After Application Startup" earlier in this topic.

To specify the update interval, set the **Specify how frequently the application should check for updates** properties in the **Application Updates** dialog box.

This is the same as setting the **maximumAge** and **unit** attributes of the **Update** element in the deployment manifest.

For example, you may want to check each time the application runs, or one time a week, or one time a month. If a network connection is not present at the specified time, the update check is performed the next time that the application runs.

Providing a User Interface for Updates

When using this strategy, the application developer provides a user interface that enables the user to choose when or how often the application will check for updates. For example, you might provide a "Check for Updates Now" command, or an "Update Settings" dialog box that has choices for different update intervals. The ClickOnce deployment APIs provide a framework for programming your own update user interface. For more information, see the System.Deployment.Application namespace.

If your application uses deployment APIs to control its own update logic, you should block update checking as described in "Blocking Update Checking" in the following section.

This strategy works best when you need different update strategies for different users.

Blocking Update Checking

It is also possible to prevent your application from ever checking for updates. For example, you might have a simple application that will never be updated, but you want to take advantage of the ease of installation provide by ClickOnce deployment.

You should also block update checking if your application uses deployment APIs to perform its own updates; see "Providing a User Interface for Updates" earlier in this topic.

To block update checking, clear the **The application should check for updates** check box in the Application Updates Dialog Box.

You can also block update checking by removing the `<Subscription>` tag from the deployment manifest.

Permission Elevation and Updates

If a new version of a ClickOnce application requires a higher level of trust to run than the previous version, ClickOnce will prompt the user, asking him if he wants the application to be granted this higher level of trust. If the user declines to grant the higher trust level, the update will not install. ClickOnce will prompt the user to install the application again when it is next restarted. If the user declines to grant the higher level of trust at this point, and the update is not marked as required, the old version of the application will run. However, if the update is required, the application will not run again until the user accepts the higher trust level.

No prompting for trust levels will occur if you use Trusted Application Deployment. For more information, see Trusted Application Deployment Overview.

See Also

System.Deployment.Application
ClickOnce Security and Deployment
Choosing a ClickOnce Deployment Strategy
Securing ClickOnce Applications
How ClickOnce Performs Application Updates
How to: Manage Updates for a ClickOnce Application

How ClickOnce Performs Application Updates

Visual Studio 2015

ClickOnce uses the file version information specified in an application's deployment manifest to decide whether to update the application's files. After an update begins, ClickOnce uses a technique called *file patching* to avoid redundant downloading of application files.

File Patching

When updating an application, ClickOnce does not download all of the files for the new version of the application unless the files have changed. Instead, it compares the hash signatures of the files specified in the application manifest for the current application against the signatures in the manifest for the new version. If a file's signatures are different, ClickOnce downloads the new version. If the signatures match, the file has not changed from one version to the next. In this case, ClickOnce copies the existing file and uses it in the new version of the application. This approach prevents ClickOnce from having to download the entire application again, even if only one or two files have changed.

File patching also works for assemblies that are downloaded on demand using the DownloadFileGroup and DownloadFileGroupAsync methods.

If you use Visual Studio to compile your application, it will generate new hash signatures for all files whenever you rebuild the entire project. In this case, all assemblies will be downloaded to the client, although only a few assemblies may have changed.

File patching does not work for files that are marked as data and stored in the data directory. These are always downloaded regardless of the file's hash signature. For more information on the data directory, see Accessing Local and Remote Data in ClickOnce Applications.

See Also

Choosing a ClickOnce Update Strategy
Choosing a ClickOnce Deployment Strategy

How to: Check for Application Updates Programmatically Using the ClickOnce Deployment API

Visual Studio 2015

ClickOnce provides two ways to update an application once it is deployed. In the first method, you can configure the ClickOnce deployment to check automatically for updates at certain intervals. In the second method, you can write code that uses the ApplicationDeployment class to check for updates based on an event, such as a user request.

The following procedures show some code for performing a programmatic update and also describe how to configure your ClickOnce deployment to enable programmatic update checks.

In order to update a ClickOnce application programmatically, you must specify a location for updates. This is sometimes referred to as a deployment provider. For more information on setting this property, see Choosing a ClickOnce Update Strategy.

Note

You can also use the technique described below to deploy your application from one location but update it from another. For more information, see How to: Specify an Alternate Location for Deployment Updates.

To check for updates programmatically

1. Create a new Windows Forms application using your preferred command-line or visual tools.
2. Create whatever button, menu item, or other user interface item you want your users to select to check for updates. From that item's event handler, call the following method to check for and install updates.

VB

```
Private Sub InstallUpdateSyncWithInfo()
    Dim info As UpdateCheckInfo = Nothing

    If (ApplicationDeployment.IsNetworkDeployed) Then
        Dim AD As ApplicationDeployment = ApplicationDeployment.CurrentDeployment

        Try
            info = AD.CheckForDetailedUpdate()
        Catch dde As DeploymentDownloadException
            MessageBox.Show("The new version of the application cannot be downloaded
at this time. " + ControlChars.Lf & ControlChars.Lf & "Please check your network
connection, or try again later. Error: " + dde.Message)
```

```
            Return
        Catch ioe As InvalidOperationException
            MessageBox.Show("This application cannot be updated. It is likely not a
ClickOnce application. Error: " & ioe.Message)
            Return
        End Try

        If (info.UpdateAvailable) Then
            Dim doUpdate As Boolean = True

            If (Not info.IsUpdateRequired) Then
                Dim dr As DialogResult = MessageBox.Show("An update is available.
Would you like to update the application now?", "Update Available",
MessageBoxButtons.OKCancel)
                If (Not System.Windows.Forms.DialogResult.OK = dr) Then
                    doUpdate = False
                End If
            Else
                ' Display a message that the app MUST reboot. Display the minimum
required version.
                MessageBox.Show("This application has detected a mandatory update
from your current " & _
                    "version to version " & info.MinimumRequiredVersion.ToString() &
_
                    ". The application will now install the update and restart.", _
                    "Update Available", MessageBoxButtons.OK, _
                    MessageBoxIcon.Information)
            End If

            If (doUpdate) Then
                Try
                    AD.Update()
                    MessageBox.Show("The application has been upgraded, and will now
restart.")
                    Application.Restart()
                Catch dde As DeploymentDownloadException
                    MessageBox.Show("Cannot install the latest version of the
application. " & ControlChars.Lf & ControlChars.Lf & "Please check your network
connection, or try again later.")
                    Return
                End Try
            End If
        End If
        End If
End Sub
```

3. Compile your application.

Using Mage.exe to deploy an application that checks for updates programmatically

- Follow the instructions for deploying your application using Mage.exe as explained in Walkthrough: Manually Deploying a ClickOnce Application. When calling Mage.exe to generate the deployment manifest, make sure to use the command-line switch `providerUrl`, and to specify the URL where ClickOnce should check for updates. If your application will update from http://www.adatum.com/MyApp, for example, your call to generate the deployment manifest might look like this:

```
mage -New Deployment -ToFile WindowsFormsApp1.application -Name "My App 1.0"
-Version 1.0.0.0 -AppManifest 1.0.0.0\MyApp.manifest -providerUrl
http://www.adatum.com/MyApp/MyApp.application
```

Using MageUI.exe to deploy an application that checks for updates programmatically

- Follow the instructions for deploying your application using Mage.exe as explained in Walkthrough: Manually Deploying a ClickOnce Application. On the **Deployment Options** tab, set the **Start Location** field to the application manifest ClickOnce should check for updates. On the **Update Options** tab, clear the **This application should check for updates** check box.

.NET Framework Security

Your application must have full-trust permissions to use programmatic updating.

See Also

How to: Specify an Alternate Location for Deployment Updates
Choosing a ClickOnce Update Strategy
Publishing ClickOnce Applications

How to: Specify an Alternate Location for Deployment Updates

Visual Studio 2015

You can install your ClickOnce application initially from a CD or a file share, but the application must check for periodic updates on the Web. You can specify an alternate location for updates in your deployment manifest so that your application can update itself from the Web after its initial installation.

Note

Your application must be configured to install locally to use this feature. For more information, see Walkthrough: Manually Deploying a ClickOnce Application. In addition, if you install a ClickOnce application from the network, setting an alternate location causes ClickOnce to use that location for both the initial installation and all subsequent updates. If you install your application locally (for example, from a CD), the initial installation is performed using the original media, and all subsequent updates will use the alternate location.

Specifying an alternate location for updates by using MageUI.exe (Windows Forms-based utility)

1. Open a .NET Framework command prompt and type:

   mageui.exe

2. On the **File** menu, choose **Open** to open your application's deployment manifest.

3. Select the **Deployment Options** tab.

4. In the text box named **Launch Location**, enter the URL to the directory that will contain the deployment manifest for application updates.

5. Save the deployment manifest.

Specifying an alternate location for updates by using Mage.exe

1. Open a .NET Framework command prompt.

2. Set the update location using the following command. In this example, **HelloWorld.exe.application** is the path to your ClickOnce application manifest, which always has the .application extension, and **http://adatum.com /Update/Path** is the URL that ClickOnce will check for application updates.

   Mage -Update HelloWorld.exe.application -ProviderUrl http://adatum.com/Update/Path

3. Save the file.

> **Note**
>
> You now need to re-sign the file with Mage.exe. For more information, see Walkthrough: Manually Deploying a ClickOnce Application.

.NET Framework Security

If you install your application from an offline medium such as a CD, and the computer is online, ClickOnce first checks the URL specified by the `<deploymentProvider>` tag in the deployment manifest to determine if the update location contains a more recent version of the application. If it does, ClickOnce installs the application directly from there, instead of from the initial installation directory, and the common language runtime (CLR) determines your application's trust level using `<deploymentProvider>`. If the computer is offline, or `<deploymentProvider>` is unreachable, ClickOnce installs from the CD, and the CLR grants trust based on the installation point; for a CD install, this means your application receives full trust. All subsequent updates will inherit that trust level.

All ClickOnce applications that use `<deploymentProvider>` should explicitly declare the permissions they need in their application manifest, so that the application does not receive different levels of trust on different computers.

See Also

- Walkthrough: Manually Deploying a ClickOnce Application
- ClickOnce Deployment Manifest
- Securing ClickOnce Applications
- Choosing a ClickOnce Update Strategy

Troubleshooting ClickOnce Deployments

Visual Studio 2015

This topic helps you diagnose and resolve the most common issues with ClickOnce deployments.

In most cases, a ClickOnce application will download to a user's computer and run without any problems. There are some cases, however, where Web server or application configuration issues can cause unforeseen problems.

How to: Set a Custom Log File Location for ClickOnce Deployment Errors
: Describes how to redirect all ClickOnce activation failures on a machine to a single log file.

How to: Specify Verbose Log Files for ClickOnce Deployments
: Describes how to increase the detail that ClickOnce writes to log files.

Server and Client Configuration Issues in ClickOnce Deployments
: Describes various issues with the configuration of your Web server that could cause difficulty downloading ClickOnce applications.

Security, Versioning, and Manifest Issues in ClickOnce Deployments
: Describes miscellaneous issues surrounding ClickOnce deployments.

Troubleshooting Specific Errors in ClickOnce Deployments
: Describes specific scenarios in which a ClickOnce deployment cannot succeed, and provides steps for resolving them.

Debugging ClickOnce Applications That Use System.Deployment.Application
: Describes a technique for debugging ClickOnce applications that use System.Deployment.Application.

See Also

ClickOnce Deployment Manifest
ClickOnce Application Manifest

How to: Set a Custom Log File Location for ClickOnce Deployment Errors

Visual Studio 2015

ClickOnce maintains activation log files for all deployments. These logs document any errors pertaining to installing and initializing a ClickOnce deployment. By default, ClickOnce creates one log file for each deployment activation. It stores these log files in the Temporary Internet Files folder. The log file for a deployment is displayed to the user when an activation failure occurs, and the user clicks **Details** in the resulting error dialog box.

You can change this behavior for a specific client by using Registry Editor (**regedit.exe**) to set a custom log file path. In this case, ClickOnce logs activation successes and failures for all deployments in a single file.

Caution

If you use Registry Editor incorrectly, you may cause serious problems that may require you to reinstall your operating system. Use Registry Editor at your own risk.

Note

You will need to truncate or delete the log file occasionally to prevent it from growing too large.

The following procedure describes how to set a custom log file location for a single client.

To set a custom log file location

1. Open **Regedit.exe**.
2. Navigate to the node **HKCU\Software\Classes\Software\Microsoft\Windows\CurrentVersion\Deployment**.
3. Set the string value **LogFilePath** to the full path and filename of your preferred custom log location.

   This location must be in a directory to which the user has write access. For example, on Windows Vista, create the following folder structure and set **LogFilePath** to C:\Users\<username>\Documents\Logs\ClickOnce\installation.log.

See Also

Troubleshooting ClickOnce Deployments

How to: Specify Verbose Log Files for ClickOnce Deployments

Visual Studio 2015

ClickOnce maintains activity log files for all deployments. These logs document details pertaining to installing, initializing, updating, and uninstalling a ClickOnce deployment. To increase the detail that ClickOnce writes to these log files, use Registry Editor (**regedit.exe**) to specify the verbosity level.

> **Caution**
>
> If you use Registry Editor incorrectly, you may cause serious problems that may require you to reinstall the operating system. Use Registry Editor at your own risk.

The following procedure describes how to specify the verbosity level for ClickOnce log files for the current user. To reduce the level of verbosity, remove this registry value.

To specify verbose log files

1. Open **Regedit.exe**.
2. Navigate to the node **HKEY_CURRENT_USER\Software\Classes\Software\Microsoft\Windows\CurrentVersion\Deployment**.
3. If necessary, create a new string value named **LogVerbosityLevel**.
4. Set the **LogVerbosityLevel** value to **1**.

See Also

Troubleshooting ClickOnce Deployments

Server and Client Configuration Issues in ClickOnce Deployments

Visual Studio 2015

If you use Internet Information Services (IIS) on Windows Server, and your deployment contains a file type that Windows does not recognize, such as a Microsoft Word file, IIS will refuse to transmit that file, and your deployment will not succeed.

Additionally, some Web servers and Web application software, such as ASP.NET, contain a list of files and file types that you cannot download. For example, ASP.NET prevents the download of all Web.config files. These files may contain sensitive information such as user names and passwords.

Although this restriction should cause no problems for downloading core ClickOnce files such as manifests and assemblies, this restriction may prevent you from downloading data files included as part of your ClickOnce application. In ASP.NET, you can resolve this error by removing the handler that prohibits downloading of such files from the IIS configuration manager. See the IIS server documentation for additional details.

Some Web servers might block files with extensions such as .dll, .config, and .mdf. Windows-based applications typically include files with some of these extensions. If a user attempts to run a ClickOnce application that accesses a blocked file on a Web server, an error will result. Rather than unblocking all file extensions, ClickOnce publishes every application file with a ".deploy" file extension by default. Therefore, the administrator only needs to configure the Web server to unblock the following three file extensions:

- .application
- .manifest
- .deploy

However, you can disable this option by clearing the **Use ".deploy" file extension** option on the fd9baa1b-7311-4f9e-8ffb-ae50cf110592, in which case you must configure the Web server to unblock all file extensions used in the application.

You will have to configure .manifest, .application, and .deploy, for example, if you are using IIS where you have not installed the .NET Framework, or if you are using another Web server (for example, Apache).

ClickOnce and Secure Sockets Layer (SSL)

A ClickOnce application will work fine over SSL, except when Internet Explorer raises a prompt about the SSL certificate. The prompt can be raised when there is something wrong with the certificate, such as when the site names do not match or the certificate has expired. To make ClickOnce work over an SSL connection, make sure that the certificate is up-to-date, and that the certificate data matches the site data.

ClickOnce and Proxy Authentication

ClickOnce provides support for Windows Integrated proxy authentication starting in .NET Framework 3.5. No specific

machine.config directives are required. ClickOnce does not provide support for other authentication protocols such as Basic or Digest.

You can also apply a hotfix to .NET Framework 2.0 to enable this feature. For more information, see http://go.microsoft.com/fwlink/?LinkId=158730.

For more information, see <defaultProxy> Element (Network Settings).

ClickOnce and Web Browser Compatibility

Currently, ClickOnce installations will launch only if the URL to the deployment manifest is opened using Internet Explorer. A deployment whose URL is launched from another application, such as Microsoft Office Outlook, will launch successfully only if Internet Explorer is set as the default Web browser.

Note

Mozilla Firefox is supported if the deployment provider is not blank or the Microsoft .NET Framework Assistant extension is installed. This extension is packaged with .NET Framework 3.5 SP1. For XBAP support, the NPWPF plug-in is activated when needed.

Activating ClickOnce Applications Through Browser Scripting

If you have developed a custom Web page that launches a ClickOnce application using Active Scripting, you may find that the application will not launch on some machines. Internet Explorer contains a setting called **Automatic prompting for file downloads**, which affects this behavior. This setting is available on the **Security** Tab in its **Options** menu that affects this behavior. It is called **Automatic prompting for file downloads**, and it is listed underneath the **Downloads** category. The property is set to **Enable** by default for intranet Web pages, and to **Disable** by default for Internet Web pages. When this setting is set to **Disable**, any attempt to activate a ClickOnce application programmatically (for example, by assigning its URL to the `document.location` property) will be blocked. Under this circumstance, users can launch applications only through a user-initiated download, for example, by clicking a hyperlink set to the application's URL.

Additional Server Configuration Issues

Administrator Permissions Required

You must have Administrator permissions on the target server if you are publishing with HTTP. IIS requires this permissions level. If you are not publishing using HTTP, you only need write permission on the target path.

Server Authentication Issues

When you publish to a remote server that has "Anonymous Access" turned off, you will receive the following warning:

```
"The files could not be downloaded from http://<remoteserver>/<myapplication>/.  The
remote server returned an error: (401) Unauthorized."
```

Note

You can make NTLM (NT challenge-response) authentication work if the site prompts for credentials other than your default credentials, and, in the security dialog box, you click **OK** when you are prompted if you want to save the supplied credentials for future sessions. However, this workaround will not work for basic authentication.

Using Third-Party Web Servers

If you are deploying a ClickOnce application from a Web server other than IIS, you may experience a problem if the server is returning the incorrect content type for key ClickOnce files, such as the deployment manifest and application manifest. To resolve this problem, see your Web server's Help documentation about how to add new content types to the server, and make sure that all the file name extension mappings listed in the following table are in place.

File name extension	Content type
.application	application/x-ms-application
.manifest	application/x-ms-manifest
.deploy	application/octet-stream
.msu	application/octet-stream
.msp	application/octet-stream

ClickOnce and Mapped Drives

If you use Visual Studio to publish a ClickOnce application, you cannot specify a mapped drive as the installation location. However, you can modify the ClickOnce application to install from a mapped drive by using the Manifest Generator and Editor (Mage.exe and MageUI.exe). For more information, see Mage.exe (Manifest Generation and Editing Tool) and MageUI.exe (Manifest Generation and Editing Tool, Graphical Client).

FTP Protocol Not Supported for Installing Applications

ClickOnce supports installing applications from any HTTP 1.1 Web server or file server. FTP, the File Transfer Protocol, is

not supported for installing applications. You can use FTP to publish applications only. The following table summarizes these differences:

URL Type	Description
ftp://	You can publish a ClickOnce application by using this protocol.
http://	You can install a ClickOnce application by using this protocol.
https://	You can install a ClickOnce application by using this protocol.
file://	You can install a ClickOnce application by using this protocol.

Windows XP SP2: Windows Firewall

By default, Windows XP SP2 enables the Windows Firewall. If you are developing your application on a computer that has Windows XP installed, you are still able to publish and run ClickOnce applications from the local server that is running IIS. However, you cannot access that server that is running IIS from another computer unless you open the Windows Firewall. See Windows Help for instructions on managing the Windows Firewall.

Windows Server: Enable FrontPage server extensions

FrontPage Server Extensions from Microsoft is required for publishing applications to a Windows Web server that uses HTTP.

By default, Windows Server does not have FrontPage Server Extensions installed. If you want to use Visual Studio to publish to a Windows Server Web server that uses HTTP with FrontPage Server Extensions, you must install FrontPage Server Extensions first. You can perform the installation by using the Manage Your Server administration tool in Windows Server.

Windows Server: Locked-Down Content Types

IIS on Windows Server 2003 locks down all file types except for certain known content types (for example, .htm, .html, .txt, and so on). To enable deployment of ClickOnce applications using this server, you need to change the IIS settings to allow downloading files of type .application, .manifest, and any other custom file types used by your application.

If you deploy using an IIS server, run inetmgr.exe and add new File Types for the default Web page:

- For the .application and .manifest extensions, the MIME type should be "application/x-ms-application." For other file types, the MIME type should be "application/octet-stream."
- If you create a MIME type with extension "*" and the MIME type "application/octet-stream," it will allow files of unblocked file type to be downloaded. (However, blocked file types such as .aspx and .asmx cannot be

downloaded.)

For specific instructions on configuring MIME types on Windows Server, refer to Microsoft Knowledge Base article KB326965, "IIS 6.0 Does Not Serve Unknown MIME Types" at http://support.microsoft.com/default.aspx?scid=kb;en-us;326965.

Content Type Mappings

When publishing over HTTP, the content type (also known as MIME type) for the .application file should be "application/x-ms-application." If you have .NET Framework 2.0 installed on the server, this will be set for you automatically. If this is not installed, then you need to create a MIME type association for the ClickOnce application vroot (or entire server).

If you deploy using an IIS server, run inetmgr.exe and add a new content type of "application/x-ms-application" for the .application extension.

HTTP Compression Issues

With ClickOnce, you can perform downloads that use HTTP compression, a Web server technology that uses the GZIP algorithm to compress a data stream before sending the stream to the client. The client—in this case, ClickOnce—decompresses the stream before reading the files.

If you are using IIS, you can easily enable HTTP compression. However, when you enable HTTP compression, it is only enabled for certain file types—namely, HTML and text files. To enable compression for assemblies (.dll), XML (.xml), deployment manifests (.application), and application manifests (.manifest), you must add these file types to the list of types for IIS to compress. Until you add the file types to your deployment, only text and HTML files will be compressed.

For detailed instructions for IIS, see How to specify additional document types for HTTP compression.

See Also

Troubleshooting ClickOnce Deployments
Choosing a ClickOnce Deployment Strategy
Application Deployment Prerequisites

Security, Versioning, and Manifest Issues in ClickOnce Deployments

Visual Studio 2015

There are a variety of issues with ClickOnce security, application versioning, and manifest syntax and semantics that can cause a ClickOnce deployment not to succeed.

ClickOnce and Windows Vista User Account Control

In Windows Vista, applications by default run as a standard user, even if the current user is logged in with an account that has administrator permissions. If an application must perform an action that requires administrator permissions, it tells the operating system, which then prompts the user to enter their administrator credentials. This feature, which is named User Account Control (UAC), prevents applications from making changes that may affect the entire operating system without a user's explicit approval. Windows applications declare that they require this permission elevation by specifying the `requestedExecutionLevel` attribute in the `trustInfo` section of their application manifest.

Due to the risk of exposing applications to security elevation attacks, ClickOnce applications cannot request permission elevation if UAC is enabled for the client. Any ClickOnce application that attempts to set its `requestedExecutionLevel` attribute to `requireAdministrator` or `highestAvailable` will not install on Windows Vista.

In some cases, your ClickOnce application may attempt to run with administrator permissions because of installer detection logic on Windows Vista. In this case, you can set the `requestedExecutionLevel` attribute in the application manifest to `asInvoker`. This will cause the application itself to run without elevation. Visual Studio 2008 automatically adds this attribute to all application manifests.

If you are developing an application that requires administrator permissions for the entire lifetime of the application, you should consider deploying the application by using Windows Installer (MSI) technology instead. For more information, see Windows Installer Basics.

Online Application Quotas and Partial Trust Applications

If your ClickOnce application runs online instead of through an installation, it must fit within the quota set aside for online applications. Also, a network application that runs in partial trust, such as with a restricted set of security permissions, cannot be larger than half of the quota size.

For more information, and instructions about how to change the online application quota, see ClickOnce Cache Overview.

Versioning Issues

You may experience problems if you assign strong names to your assembly and increment the assembly version number to reflect an application update. Any assembly compiled with a reference to a strong-named assembly must itself be

recompiled, or the assembly will try to reference the older version. The assembly will try this because the assembly is using the old version value in its bind request.

For example, say that you have a strong-named assembly in its own project with version 1.0.0.0. After compiling the assembly, you add it as a reference to the project that contains your main application. If you update the assembly, increment the version to 1.0.0.1, and try to deploy it without also recompiling the application, the application will not be able to load the assembly at run time.

This error can occur only if you are editing your ClickOnce manifests manually; you should not experience this error if you generate your deployment using Visual Studio 2005.

Specifying Individual .NET Framework Assemblies in the Manifest

Your application will fail to load if you have manually edited a ClickOnce deployment to reference an older version of a .NET Framework assembly. For example, if you added a reference to the System.Net assembly for a version of the .NET Framework prior to the version specified in the manifest, then an error would occur. In general, you should not attempt to specify references to individual .NET Framework assemblies, as the version of the .NET Framework against which your application runs is specified as a dependency in the application manifest.

Manifest Parsing Issues

The manifest files that are used by ClickOnce are XML files, and they must be both well-formed and valid: they must obey the XML syntax rules and only use elements and attributes defined in the relevant XML schema.

Something that can cause problems in a manifest file is selecting a name for your application that contains a special character, such as a single or double quotation mark. The application's name is part of its ClickOnce identity. ClickOnce currently does not parse identities that contain special characters. If your application fails to activate, make sure that you are using only alphabetical and numeric characters for the name, and attempt to deploy it again.

If you have manually edited your deployment or application manifests, you may have unintentionally corrupted them. Corrupted manifest will prevent a correct ClickOnce installation. You can debug such errors at run time by clicking **Details** on the **ClickOnce Error** dialog box, and reading the error message in the log. The log will list one of the following messages:

- A description of the syntax error, and the line number and character position where the error occurred.
- The name of an element or attribute used in violation of the manifest's schema. If you have added XML manually to your manifests, you will have to compare your additions to the manifest schemas. For more information, see ClickOnce Deployment Manifest and ClickOnce Application Manifest.
- An ID conflict. Dependency references in deployment and application manifests must be unique in both their `name` and `publicKeyToken` attributes. If both attributes match between any two elements within a manifest, manifest parsing will not succeed.

Precautions When Manually Changing Manifests or Applications

When you update an application manifest, you must re-sign both the application manifest and the deployment manifest. The deployment manifest contains a reference to the application manifest that includes that file's hash and its digital signature.

Precautions with Deployment Provider Usage

The ClickOnce deployment manifest has a **deploymentProvider** property which points to the full path of the location from where the application should be installed and serviced:

```
<deploymentProvider codebase="http://myserver/myapp.application" />
```

This path is set when ClickOnce creates the application and is compulsory for installed applications. The path points to the standard location where the ClickOnce installer will install the application from and search for updates. If you use the **xcopy** command to copy a ClickOnce application to a different location, but do not change the **deploymentProvider** property, ClickOnce will still refer back to the original location when it tries to download the application.

If you want to move or copy an application, you must also update the **deploymentProvider** path, so that the client actually installs from the new location. Updating this path is mostly a concern if you have installed applications. For online applications that are always launched through the original URL, setting the **deploymentProvider** is optional. If **deploymentProvider** is set, it will be honored; otherwise, the URL used to start the application will be used as the base URL to download application files.

Note
Every time that you update the manifest you must also sign it again.

See Also

Troubleshooting ClickOnce Deployments
Securing ClickOnce Applications
Choosing a ClickOnce Deployment Strategy

Troubleshooting Specific Errors in ClickOnce Deployments

Visual Studio 2015

This topic lists the following common errors that can occur when you deploy a ClickOnce application, and provides steps to resolve each problem.

General Errors

When you try to locate an .application file, nothing occurs, or XML renders in Internet Explorer, or you receive a Run or Save As dialog box

This error is likely caused by content types (also known as MIME types) not being registered correctly on the server or client.

First, make sure that the server is configured to associate the .application extension with content type "application/x-ms-application".

If the server is configured correctly, ensure that the .NET Framework 2.0 is installed on your computer. If the .NET Framework 2.0 is installed, and you are still seeing this problem, try uninstalling and reinstalling the .NET Framework 2.0 to re-register the content type on the client.

Error message says, "Unable to retrieve application. Files missing in deployment" or "Application download has been interrupted, check for network errors and try again later"

This message indicates that one or more files being referenced by the ClickOnce manifests cannot be downloaded. The easiest way to debug this error is to try to download the URL that ClickOnce says it cannot download. Here are some possible causes:

- If the log file says "(403) Forbidden" or "(404) Not found," verify that the Web server is configured so that it does not block download of this file. For more information, see Server and Client Configuration Issues in ClickOnce Deployments.
- If the .config file is being blocked by the server, see the section "Download error when you try to install a ClickOnce application that has a .config file" later in this topic.
- Determine whether this occurred because the **deploymentProvider** URL in the deployment manifest is pointing to a different location than the URL used for activation.
- Ensure that all files are present on the server; the ClickOnce log should tell you which file was not found.
- See whether there are network connectivity issues; you can receive this message if your client computer went offline during the download.

Download error when you try to install a ClickOnce application that has a .config file

By default, a Visual Basic Windows-based application includes an App.config file. There will be a problem when a user tries to install from a Web server that uses Windows Server 2003, because that operating system blocks the installation of .config files for security reasons. To enable the .config file to be installed, click **Use ".deploy" file extension** in the **Publish Options** dialog box.

You also must set the content types (also known as MIME types) appropriately for .application, .manifest, and .deploy files. For more information, see your Web server documentation.

For more information, see "Windows Server 2003: Locked-Down Content Types" in Server and Client Configuration Issues in ClickOnce Deployments.

Error message: "Application is improperly formatted;" Log file contains "XML signature is invalid"

Ensure that you updated the manifest file and signed it again. Republish your application by using Visual Studio or use Mage to sign the application again.

You updated your application on the server, but the client does not download the update

This problem might be solved by completing one of the following tasks:

- Examine the **deploymentProvider** URL in the deployment manifest. Ensure that you are updating the bits in the same location that **deploymentProvider** points to.
- Verify the update interval in the deployment manifest. If this interval is set to a periodic interval, such as one time every six hours, ClickOnce will not scan for an update until this interval has passed. You can change the manifest to scan for an update every time that the application starts. Changing the update interval is a

convenient option during development time to verify updates are being installed, but it slows down application activation.

- Try starting the application again on the Start menu. ClickOnce may have detected the update in the background, but will prompt you to install the bits on the next activation.

During update you receive an error that has the following log entry: "The reference in the deployment does not match the identity defined in the application manifest"

This error may occur because you have manually edited the deployment and application manifests, and have caused the description of the identity of an assembly in one manifest to become out of sync with the other. The identity of an assembly consists of its name, version, culture, and public key token. Examine the identity descriptions in your manifests, and correct any differences.

First time activation from local disk or CD-ROM succeeds, but subsequent activation from Start Menu does not succeed

ClickOnce uses the Deployment Provider URL to receive updates for the application. Verify that the location that the URL is pointing to is correct.

Error: "Cannot start the application"

This error message usually indicates that there is a problem installing this application into the ClickOnce store. Either the application has an error or the store is corrupted. The log file might tell you where the error occurred.

You should do the following:

- Verify that the identity of the deployment manifest, identity of application manifest, and identity of the main application EXE are all unique.
- Verify that your file paths are not longer than 100 characters. If your application contains file paths that are too long, you may exceed the limitations on the maximum path you can store. Try shortening the paths and reinstall.

PrivatePath settings in application config file are not honored

To use PrivatePath (Fusion probing paths), the application must request full trust permission. Try changing the application manifest to request full trust, and then try again.

During uninstall a message appears saying, "Failed to uninstall application"

This message usually indicates that the application has already been removed or the store is corrupted. After you click **OK**, the **Add/Remove Program** entry will be removed.

During installation, a message appears that says that the platform dependencies are not installed

You are missing a prerequisite in the GAC (global assembly cache) that the application needs in order to run.

Publishing with Visual Studio

Publishing in Visual Studio fails

Ensure that you have the right to publish to the server that you are targeting. For example, if you are logged in to a terminal server computer as an ordinary user, not as an administrator, you probably will not have the rights required to publish to the local Web server.

If you are publishing with a URL, ensure that the destination computer has FrontPage Server Extensions enabled.

Error Message: Unable to create the Web site '<site>'. The components for communicating with FrontPage Server Extensions are not installed.

Ensure that you have the Microsoft Visual Studio Web Authoring Component installed on the machine that you are publishing from. For Express users, this component is not installed by default. For more information, see http://go.microsoft.com/fwlink/?LinkId=102310.

Error Message: Could not find file 'Microsoft.Windows.Common-Controls, Version=6.0.0.0, Culture=*, PublicKeyToken=6595b64144ccf1df, ProcessorArchitecture=*, Type=win32'

This error message appears when you attempt to publish a WPF application with visual styles enabled. To resolve this issue, see How to: Publish a WPF Application with Visual Styles Enabled.

Using Mage

You tried to sign with a certificate in your certificate store and a received blank message box

In the **Signing** dialog box, you must:

- Select **Sign with a stored certificate**, and
- Select a certificate from the list; the first certificate is not the default selection.

Clicking the "Don't Sign" button causes an exception

This issue is a known bug. All ClickOnce manifests are required to be signed. Just select one of the signing options, and then click **OK**.

Additional Errors

The following table shows some common error messages that a client-computer user may receive when the user installs a ClickOnce application. Each error message is listed next to a description of the most probable cause for the error.

Error message	Description
Application cannot be started. Contact the application publisher. Cannot start the application. Contact the application vendor for assistance.	These are generic error messages that occur when the application cannot be started, and no other specific reason can be found. Frequently this means that the application is somehow corrupted, or that the ClickOnce store is corrupted.
Cannot continue. The application is improperly formatted. Contact the application publisher for assistance. Application validation did not succeed. Unable to continue. Unable to retrieve application files. Files corrupt in deployment.	One of the manifest files in the deployment is syntactically not valid, or contains a hash that cannot be reconciled with the corresponding file. This error may also indicate that the manifest embedded inside an assembly is corrupted. Re-create your deployment and recompile your application, or find and fix the errors manually in your manifests.
Cannot retrieve application. Authentication error. Application installation did not succeed. Cannot locate applications files on the server. Contact the application publisher or your administrator for assistance.	One or more files in the deployment cannot be downloaded because you do not have permission to access them. This can be caused by a 403 Forbidden error being returned by a Web server, which may occur if one of the files in your deployment ends with an extension that makes the Web server treat it as a protected file. Also, a directory that contains one or more of the application's files might require a username and password in order to access.
Cannot download the application. The application is missing required files. Contact the application vendor or your system administrator for assistance.	One or more of the files listed in the application manifest cannot be found on the server. Verify that you have uploaded all the deployment's dependent files, and try again.
Application download did not succeed. Check your network connection, or contact your system administrator or network service provider.	ClickOnce cannot establish a network connection to the server. Examine the server's availability and the state of your network.

URLDownloadToCacheFile failed with HRESULT '<number>'. An error occurred trying to download '<file>'.	If a user has set Internet Explorer Advanced Security option "Warn if changing between secure and not secure mode" on the deployment target computer, and if the setup URL of the ClickOnce application being installed is redirected from a non-secure to a secure site (or vice-versa), the installation will fail because the Internet Explorer warning interrupts it. To resolve this, you can do one of the following: • Clear the security option. • Make sure that the setup URL is not redirected in such a way that changes security modes. • Remove the redirection completely and point to the actual setup URL.
An error has occurred writing to the hard disk. There might be insufficient space available on the disk. Contact the application vendor or your system administrator for assistance.	This may indicate insufficient disk space for storing the application, but it may also indicate a more general I/O error when you are trying to save the application files to the drive.
Cannot start the application. There is not enough available space on the disk.	The hard disk is full. Clear off space and try to run the application again.
Too many deployed activations are attempting to load at once.	ClickOnce limits the number of different applications that can start at the same time. This is largely to help protect against malicious attempts to instigate denial-of-service attacks against the local ClickOnce service; users who try to start the same application repeatedly, in rapid succession, will only end up with a single instance of the application.
Shortcuts cannot be activated over the network.	Shortcuts to a ClickOnce application can only be started on the local hard disk. They cannot be started by opening a URL that points to a shortcut file on a remote server.
The application is too large to run online in partial trust. Contact the application vendor or your system administrator for assistance.	An application that runs in partial trust cannot be larger than half of the size of the online application quota, which by default is 250 MB.

See Also

ClickOnce Security and Deployment
Troubleshooting ClickOnce Deployments

Debugging ClickOnce Applications That Use System.Deployment.Application

Visual Studio 2015

In Visual Studio, ClickOnce deployment allows you to configure how an application is updated. However, if you need to use and customize advanced ClickOnce deployment features, you will need to access the deployment object model provided by System.Deployment.Application. You can use the System.Deployment.Application APIs for advanced tasks such as:

- Creating an "Update Now" option in your application
- Conditional, on-demand downloads of various application components
- Updates integrated directly into the application
- Guaranteeing that the client application is always up-to-date

Because the System.Deployment.Application APIs work only when an application is deployed with ClickOnce technology, the only way to debug them is to deploy the application using ClickOnce, attach to it, then debug it. It can be difficult to attach the debugger early enough, because this code often runs when the application starts up and executes before you can attach the debugger. A solution is to place breaks (or stops, for Visual Basic projects) before your update check code or on-demand code.

The recommended debugging technique is as follows:

1. Before you start, make sure the symbol (.pdb) and source files are archived.
2. Deploy version 1 of the application.
3. Create a new blank solution. From the **File** menu, click **New**, then **Project**. In the **New Project** dialog box, open the **Other Project Types** node, then select the **Visual Studio Solutions** folder. In the **Templates** pane, select **Blank Solution**.
4. Add the archived source location to the properties for this new solution. In **Solution Explorer**, right-click the solution node, then click **Properties**. In the **Property Pages** dialog box, select **Debug Source Files**, then add the directory of the archived source code. Otherwise, the debugger will find the out-of-date source files, since the source file paths are recorded in the .pdb file. If the debugger uses out-of-date source files, you see a message telling you that the source does not match.
5. Make sure the debugger can find the .pdb files. If you have deployed them with your application, the debugger finds them automatically. It always looks next to the assembly in question first. Otherwise, you will need to add the archive path to the **Symbol file (.pdb) locations** (to access this option, from the **Tools** menu, click **Options**, then open the **Debugging** node, and click **Symbols**).
6. Debug what happens between the **CheckForUpdate** and **Download/Update** method calls.

   For example, the update code might be as follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    If My.Application.Deployment.IsNetworkDeployed Then

        If (My.Application.Deployment.CheckForUpdate()) Then

            My.Application.Deployment.Update()
            Application.Restart()

        End If

    End If
End Sub
```

7. Deploy version 2.

8. Attempt to attach the debugger to the version 1 application as it downloads an update for version 2. Alternatively you can use the **System.Diagnostics.Debugger.Break** method or simply **Stop** in Visual Basic. Of course, you should not leave these method calls in production code.

   For example, assume you are developing a Windows Forms application, and you have an event handler for this method with the update logic in it. To debug this, simply attach before the button is pressed, then set a breakpoint (make sure that you open the appropriate archived file and set the breakpoint there).

Use the IsNetworkDeployed property to invoke the System.Deployment.Application APIs only when the application is deployed; the APIs should not be invoked during debugging in Visual Studio.

See Also

System.Deployment.Application

ClickOnce Reference

Visual Studio 2015

The following pages describe the structure of the XML files used to represent ClickOnce applications.

In This Section

ClickOnce Application Manifest
: Lists and describes the elements and attributes that make up an application manifest.

ClickOnce Deployment Manifest
: Lists and describes the elements and attributes that make up a deployment manifest.

Product and Package Schema Reference
: Lists product and package file elements.

ClickOnce Unmanaged API Reference
: Lists unmanaged public APIs from dfshim.dll.

Reference

Related Sections

ClickOnce Security and Deployment
: Provides detailed conceptual information about ClickOnce deployment.

System.Deployment.Application
: Provides links to reference documentation of the public classes that support ClickOnce within managed code.

Publishing ClickOnce Applications
: Provides walkthroughs and how-to's that perform ClickOnce tasks.

ClickOnce Application Manifest

Visual Studio 2015

A ClickOnce application manifest is an XML file that describes an application that is deployed using ClickOnce.

ClickOnce application manifests have the following elements and attributes.

Element	Description	Attributes
<assembly> Element (ClickOnce Application)	Required. Top-level element.	manifestVersion
<assemblyIdentity> Element (ClickOnce Application)	Required. Identifies the primary assembly of the ClickOnce application.	name version publicKeyToken processorArchitecture language
<trustInfo> Element (ClickOnce Application)	Identifies the application security requirements.	None
<entryPoint> Element (ClickOnce Application)	Required. Identifies the application code entry point.	name
<dependency> Element (ClickOnce Application)	Required. Identifies each dependency required for the application to run. Optionally identifies assemblies that need to be preinstalled.	None
<file> Element (ClickOnce Application)	Optional. Identifies each nonassembly file that is used by the application. Can include Component Object Model (COM) isolation data associated with the file.	name size group optional writeableType
<fileAssociation> Element (ClickOnce Application)	Optional. Identifies a file extension to be associated with the application.	extension

		`description`
		`progid`
		`defaultIcon`

Remarks

The ClickOnce application manifest file identifies an application deployed using ClickOnce. For more information about ClickOnce, see ClickOnce Security and Deployment.

File Location

A ClickOnce application manifest is specific to a single version of a deployment. For this reason, they should be stored separately from deployment manifests. The common convention is to place them in a subdirectory named after the associated version.

The application manifest always must be signed prior to deployment. If you change an application manifest manually, you must use mage.exe to re-sign the application manifest, update the deployment manifest, and then re-sign the deployment manifest. For more information, see Walkthrough: Manually Deploying a ClickOnce Application.

File Name Syntax

The name of a ClickOnce application manifest file should be the full name and extension of the application as identified in the `assemblyIdentity` element, followed by the extension .manifest. For example, an application manifest that refers to the Example.exe application would use the following file name syntax.

```
example.exe.manifest
```

Example

The following code example shows an application manifest for a ClickOnce application.

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1
assembly.adaptive.xsd" manifestVersion="1.0" xmlns:asmv3="urn:schemas-microsoft-
com:asm.v3" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" xmlns:co.v2="urn:schemas-
microsoft-com:clickonce.v2" xmlns="urn:schemas-microsoft-com:asm.v2"
xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-microsoft-
com:asm.v2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:co.v1="urn:schemas-microsoft-com:clickonce.v1">
  <asmv1:assemblyIdentity name="My Application Deployment.exe" version="1.0.0.0"
publicKeyToken="43cb1e8e7a352766" language="neutral" processorArchitecture="x86"
type="win32" />
```

```
  <application />
  <entryPoint>
    <assemblyIdentity name="MyApplication" version="1.0.0.0" language="neutral"
processorArchitecture="x86" />
    <commandLine file="MyApplication.exe" parameters="" />
  </entryPoint>
  <trustInfo>
    <security>
      <applicationRequestMinimum>
        <PermissionSet Unrestricted="true" ID="Custom" SameSite="site" />
        <defaultAssemblyRequest permissionSetReference="Custom" />
      </applicationRequestMinimum>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!--
          UAC Manifest Options
          If you want to change the Windows User Account Control level replace the
          requestedExecutionLevel node with one of the following.

        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />

         If you want to utilize File and Registry Virtualization for backward
         compatibility then delete the requestedExecutionLevel node.
    -->
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>
  <dependency>
    <dependentOS>
      <osVersionInfo>
        <os majorVersion="4" minorVersion="10" buildNumber="0" servicePackMajor="0" />
      </osVersionInfo>
    </dependentOS>
  </dependency>
  <dependency>
    <dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">
      <assemblyIdentity name="Microsoft.Windows.CommonLanguageRuntime"
version="4.0.20506.0" />
    </dependentAssembly>
  </dependency>
  <dependency>
    <dependentAssembly dependencyType="install" allowDelayedBinding="true"
codebase="MyApplication.exe" size="4096">
      <assemblyIdentity name="MyApplication" version="1.0.0.0" language="neutral"
processorArchitecture="x86" />
      <hash>
        <dsig:Transforms>
          <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
        </dsig:Transforms>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <dsig:DigestValue>DpTW7RzS9IeT/RBSLj54vfTEzNg=</dsig:DigestValue>
      </hash>
```

```
      </dependentAssembly>
    </dependency>
  <publisherIdentity name="CN=DOMAINCONTROLLER\UserMe"
  issuerKeyHash="18312a18a21b215ecf4cdb20f5a0e0b0dd263c08" /><Signature
  Id="StrongNameSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
  …
  </Signature></r:issuer></r:license></msrel:RelData></KeyInfo></Signature></asmv1:assembly>
```

See Also

Publishing ClickOnce Applications

<assembly> Element (ClickOnce Application)

Visual Studio 2015

The top-level element for the application manifest.

Syntax

```
      <assembly
   manifestVersion
/>
```

Elements and Attributes

The `assembly` element is the root element and is required. Its first contained element must be an `assemblyIdentity` element. The manifest elements must be in one of the following namespaces:

`urn:schemas-microsoft-com:asm.v1`

`urn:schemas-microsoft-com:asm.v2`

`http://www.w3.org/2000/09/xmldsig#`

Child elements of the assembly must also be in these namespaces, by inheritance or by tagging.

The `assembly` element has the following attribute.

Attribute	Description
`manifestVersion`	Required. The `manifestVersion` attribute must be set to **1.0**.

Example

The following code example illustrates an `assembly` element in an application manifest for a ClickOnce application. This code example is part of a larger example provided in ClickOnce Application Manifest.

```
<asmv1:assembly
```

```
    xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd"
    manifestVersion="1.0"
    xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
    xmlns:dsig=http://www.w3.org/2000/09/xmldsig#
    xmlns:co.v2="urn:schemas-microsoft-com:clickonce.v2"
    xmlns="urn:schemas-microsoft-com:asm.v2"
    xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
    xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:co.v1="urn:schemas-microsoft-com:clickonce.v1">
```

See Also

ClickOnce Application Manifest
<assembly> Element (ClickOnce Deployment)

<assemblyIdentity> Element (ClickOnce Application)

Visual Studio 2015

Identifies the application deployed in a ClickOnce deployment.

Syntax

```
      <assemblyIdentity
    name
    version
    publicKeyToken
    processorArchitecture
    language
/>
```

Elements and Attributes

The `assemblyIdentity` element is required. It contains no child elements and has the following attributes.

Attribute	Description
`Name`	Required. Identifies the name of the application. If `Name` contains special characters, such as single or double quotes, the application may fail to activate.
`Version`	Required. Specifies the version number of the application in the following format: **major.minor.build.revision**
`publicKeyToken`	Optional. Specifies a 16-character hexadecimal string that represents the last 8 bytes of the `SHA-1` hash value of the public key under which the application or assembly is signed. The public key that is used to sign the catalog must be 2048 bits or greater. Although signing an assembly is recommended but optional, this attribute is required. If an assembly is unsigned, you should copy a value from a self-signed assembly or use a "dummy" value of all zeros.

`processorArchitecture`	Required. Specifies the processor. The valid values are `msil` for all processors, `x86` for 32-bit Windows, `IA64` for 64-bit Windows, and `Itanium` for Intel 64-bit Itanium processors.
`language`	Required. Identifies the two part language codes (for example, **en-US**) of the assembly. This element is in the `asmv2` namespace. If unspecified, the default is **neutral**.

Examples

Description

The following code example illustrates an `assemblyIdentity` element in a ClickOnce application manifest. This code example is part of a larger example provided in ClickOnce Application Manifest.

Code

```
<asmv1:assemblyIdentity
  name="My Application Deployment.exe"
  version="1.0.0.0"
  publicKeyToken="43cb1e8e7a352766"
  language="neutral"
  processorArchitecture="x86"
  type="win32" />
```

See Also

ClickOnce Application Manifest
<assemblyIdentity> Element (ClickOnce Deployment)

<trustInfo> Element (ClickOnce Application)

Visual Studio 2015

Describes the minimum security permissions required for the application to run on the client computer.

Syntax

```
      <trustInfo>
   <security>
      <applicationRequestMinimum>
         <PermissionSet
            ID
            Unrestricted>
            <IPermission
               class
               version
               Unrestricted
            />
         </PermissionSet>
         <defaultAssemblyRequest
            permissionSetReference
         />
         <assemblyRequest
            name
            permissionSetReference
         />
      </applicationRequestMinimum>
      <requestedPrivileges>
         <requestedExecutionLevel
            level
            uiAccess
         />
      </requestedPrivileges>
   </security>
</trustInfo>
```

Elements and Attributes

The `trustInfo` element is required and is in the `asm.v2` namespace. It has no attributes and contains the following elements.

security

Required. This element is a child of the `trustInfo` element. It contains the `applicationRequestMinimum` element and has no attributes.

applicationRequestMinimum

Required. This element is a child of the `security` element and contains the `PermissionSet`, `assemblyRequest`, and `defaultAssemblyRequest` elements. This element has no attributes.

PermissionSet

Required. This element is a child of the `applicationRequestMinimum` element and contains the `IPermission` element. This element has the following attributes.

- `ID`

  Required. Identifies the permission set. This attribute can be any value. The ID is referenced in the `defaultAssemblyRequest` and `assemblyRequest` attributes.

- `version`

  Required. Identifies the version of the permission. Normally this value is **1**.

IPermission

Optional. This element is a child of the `PermissionSet` element. The `IPermission` element fully identifies a permission class in the .NET Framework. The `IPermission` element has the following attributes, but can have additional attributes that correspond to properties on the permission class. To find out the syntax for a specific permission, see the examples listed in the Security.config file.

- `class`

  Required. Identifies the permission class by strong name. For example, the following code identifies the `FileDialogPermission` type.

  ```
  System.Security.Permissions.FileDialogPermission, mscorlib, Version=1.2.3300.0,
  Culture=neutral, PublicKeyToken=b77a5c561934e089
  ```

- `version`

  Required. Identifies the version of the permission. Usually this value is **1**.

- `Unrestricted`

  Required. Identifies whether the application needs an unrestricted grant of this permission. If **true**, the permission grant is unconditional. If **false**, or if this attribute is undefined, it is restricted according to the permission-specific attributes defined on the `IPermission` tag. Take the following permissions:

```
<IPermission
  class="System.Security.Permissions.EnvironmentPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
  version="1"
  Read="USERNAME" />
<IPermission
  class="System.Security.Permissions.FileDialogPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
  version="1"
  Unrestricted="true" />
```

  In this example, the declaration for EnvironmentPermission restricts the application to reading only the environment variable USERNAME, whereas the declaration for FileDialogPermission gives the application unrestricted use of all FileDialog classes.

defaultAssemblyRequest

Optional. Identifies the set of permissions granted to all assemblies. This element is a child of the `applicationRequestMinimum` element and has the following attribute.

- `permissionSetReference`

  Required. Identifies the ID of the permission set that is the default permission. The permission set is declared in the `PermissionSet` element.

assemblyRequest

Optional. Identifies permissions for a specific assembly. This element is a child of the `applicationRequestMinimum` element and has the following attributes.

- `Name`

  Required. Identifies the assembly name.

- `permissionSetReference`

  Required. Identifies the ID of the permission set that this assembly requires. The permission set is declared in the

`PermissionSet` element.

requestedPrivileges

Optional. This element is a child of the `security` element and contains the `requestedExecutionLevel` element. This element has no attributes.

requestedExecutionLevel

Optional. Identifies the security level at which the application requests to be executed. This element has no children and has the following attributes.

- `Level`

  Required. Indicates the security level the application is requesting. Possible values are:

  asInvoker, requesting no additional permissions. This level requires no additional trust prompts.

  highestAvailable, requesting the highest permissions available to the parent process.

  requireAdministrator, requesting full administrator permissions.

  ClickOnce applications will only install with a value of **asInvoker**. Installing with any other value will fail.

- `uiAccess`

  Optional. Indicates whether the application requires access to protected user interface elements. Values are either **true** or **false**, and the default is false. Only signed applications should have a value of true.

Remarks

If a ClickOnce application asks for more permissions than the client computer will grant by default, the common language runtime's Trust Manager will ask the user if she wants to grant the application this elevated level of trust. If she says no, the application will not run; otherwise, it will run with the requested permissions.

All permissions requested using `defaultAssemblyRequest` and `assemblyRequest` will be granted without user prompting if the deployment manifest has a valid Trust License.

For more information about Permission Elevation, see Securing ClickOnce Applications. For more information about policy deployment, see Trusted Application Deployment Overview.

Examples

The following three code examples illustrate `trustInfo` elements for the default named security zones—Internet,

LocalIntranet, and FullTrust—for use in a ClickOnce deployment's application manifest.

The first example illustrates the `trustInfo` element for the default permissions available in the Internet security zone.

```
<trustInfo>
    <security>
      <applicationRequestMinimum>
        <PermissionSet ID="Internet">
          <IPermission
            class="System.Security.Permissions.FileDialogPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Access="Open" />
          <IPermission
           class="System.Security.Permissions.IsolatedStorageFilePermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Allowed="DomainIsolationByUser"
            UserQuota="10240" />
          <IPermission
            class="System.Security.Permissions.SecurityPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Flags="Execution" />
          <IPermission
            class="System.Security.Permissions.UIPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Window="SafeTopLevelWindows"
            Clipboard="OwnClipboard" />
          <IPermission
            class="System.Drawing.Printing.PrintingPermission, System.Drawing,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
            version="1"
            Level="SafePrinting" />
        </PermissionSet>
        <defaultAssemblyRequest permissionSetReference="Internet" />
      </applicationRequestMinimum>
    </security>
  </trustInfo>
```

The second example illustrates the `trustInfo` element for the default permissions available in the LocalIntranet security zone.

```
<trustInfo>
    <security>
      <applicationRequestMinimum>
        <PermissionSet ID="LocalIntranet">
          <IPermission
```

```
            class="System.Security.Permissions.EnvironmentPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Read="USERNAME" />
          <IPermission
            class="System.Security.Permissions.FileDialogPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Unrestricted="true" />
          <IPermission
            class="System.Security.Permissions.IsolatedStorageFilePermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Allowed="AssemblyIsolationByUser"
            UserQuota="9223372036854775807"
            Expiry="9223372036854775807"
            Permanent="True" />
          <IPermission
            class="System.Security.Permissions.ReflectionPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Flags="ReflectionEmit" />
          <IPermission
            class="System.Security.Permissions.SecurityPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Flags="Assertion, Execution" />
          <IPermission
            class="System.Security.Permissions.UIPermission, mscorlib,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Unrestricted="true" />
          <IPermission
            class="System.Net.DnsPermission, System, Version=1.2.3300.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1"
            Unrestricted="true" />
          <IPermission
            class="System.Drawing.Printing.PrintingPermission, System.Drawing,
Version=1.2.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
            version="1"
            Level="DefaultPrinting" />
          <IPermission
            class="System.Diagnostics.EventLogPermission, System, Version=1.2.3300.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
            version="1" />
        </PermissionSet>
        <defaultAssemblyRequest permissionSetReference="LocalIntranet" />
      </applicationRequestMinimum>
    </security>
</trustInfo>
```

The third example illustrates the `trustInfo` element for the default permissions available in the FullTrust security zone.

```
<trustInfo>
  <security>
    <applicationRequestMinimum>
      <PermissionSet ID="FullTrust" Unrestricted="true" />
      <defaultAssemblyRequest permissionSetReference="FullTrust" />
    </applicationRequestMinimum>
  </security>
</trustInfo>
```

See Also

Trusted Application Deployment Overview
ClickOnce Application Manifest

<entryPoint> Element (ClickOnce Application)

Visual Studio 2015

Identifies the assembly that should be executed when this ClickOnce application is run on a client computer.

Syntax

```
      <entryPoint
   name
>
   <assemblyIdentity
      name
      version
      processorArchitecture
      language
   />
   <commandLine
      file
      parameters
   />
   <customHostRequired />
   <customUX />
</entryPoint>
```

Elements and Attributes

The `entryPoint` element is required and is in the `urn:schemas-microsoft-com:asm.v2` namespace. There may only be one `entryPoint` element defined in an application manifest.

The `entryPoint` element has the following attribute.

Attribute	Description
`name`	Optional. This value is not used by .NET Framework.

`entryPoint` has the following elements.

assemblyIdentity

Required. The role of `assemblyIdentity` and its attributes is defined in <assemblyIdentity> Element (ClickOnce Application).

The `processorArchitecture` attribute of this element and the `processorArchitecture` attribute defined in the `assemblyIdentity` elsewhere in the application manifest must match.

commandLine

Required. Must be a child of the `entryPoint` element. It has no child elements and has the following attributes.

Attribute	Description
`file`	Required. A local reference to the startup assembly for the ClickOnce application. This value cannot contain forward slash (/) or backslash (\) path separators.
`parameters`	Required. Describes the action to take with the entry point. The only valid value is **run**; if a blank string is supplied, **run** is assumed.

customHostRequired

Optional. If included, specifies that this deployment contains a component that will be deployed inside of a custom host, and is not a stand-alone application.

If this element is present, the `assemblyIdentity` and `commandLine` elements must not also be present. If they are, ClickOnce will raise a validation error during installation.

This element has no attributes and no children.

customUX

Optional. Specifies that the application is installed and maintained by a custom installer, and does not create a Start menu entry, shortcut, or Add or Remove Programs entry.

```
<customUX xmlns="urn:schemas-microsoft-com:clickonce.v1" />
```

An application that includes the customUX element must provide a custom installer that uses the InPlaceHostingManager class to perform install operations. An application with this element cannot be installed by double-clicking its manifest or setup.exe prerequisite bootstrapper. The custom installer can create Start menu entries, shortcuts, and Add or Remove Programs entries. If the custom installer does not create an Add or Remove Programs entry, it must store the subscription identifier provided by the SubscriptionIdentity property and enable the user to uninstall the application later by calling the UninstallCustomUXApplication method. For more information, see Walkthrough: Creating a Custom Installer for a ClickOnce Application.

Remarks

This element identifies the assembly and entry point for the ClickOnce application.

You cannot use `commandLine` to pass parameters into your application at run time. You can access query string parameters for a ClickOnce deployment from the application's AppDomain. For more information, see How to: Retrieve Query String Information in an Online ClickOnce Application.

Example

The following code example illustrates an `entryPoint` element in an application manifest for a ClickOnce application. This code example is part of a larger example provided for the ClickOnce Application Manifest topic.

```
<!-- Identify the main code entrypoint. -->
<!-- This code runs the main method in an executable assembly. -->
  <entryPoint>
    <assemblyIdentity
      name="MyApplication"
      version="1.0.0.0"
      language="neutral"
```

```
        processorArchitecture="x86" />
      <commandLine file="MyApplication.exe" parameters="" />
    </entryPoint>
```

See Also

ClickOnce Application Manifest

<dependency> Element (ClickOnce Application)

Visual Studio 2015

Identifies a platform or assembly dependency that is required for the application.

Syntax

```
   <dependency>
<dependentOS
   supportURL
   description
>
   <osVersionInfo>
      <os
         majorVersion
         minorVersion
         buildNumber
         servicePackMajor
         servicePackMinor
         productType
         suiteType
      />
   </osVersionInfo>
</dependentOS>
<dependentAssembly
   dependencyType
   allowDelayedBinding
   group
   codeBase
   size
>
   <assemblyIdentity
      name
      version
      processorArchitecture
      language
   >
      <hash>
         <dsig:Transforms>
            <dsig:Transform
               Algorithm
         />
```

```
            </dsig:Transforms>
            <dsig:DigestMethod />
            <dsig:DigestValue>
            </dsig:DigestValue>
     </hash>

       </assemblyIdentity>
    </dependentAssembly>
 </dependency>
```

Elements and Attributes

The `dependency` element is required. There may be multiple instances of `dependency` in the same application manifest.

The `dependency` element has no attributes, and contains the following child elements.

dependentOS

Optional. Contains the `osVersionInfo` element. The `dependentOS` and `dependentAssembly` elements are mutually exclusive: one or the other must exist for a `dependency` element, but not both.

`dependentOS` supports the following attributes.

Attribute	Description
`supportUrl`	Optional. Specifies a support URL for the dependent platform. This URL is shown to the user if the required platform is found.
`description`	Optional. Describes, in human-readable form, the operating system described by the `dependentOS` element.

osVersionInfo

Required. This element is a child of the `dependentOS` element and contains the `os` element. This element has no attributes.

os

Required. This element is a child of the `osVersionInfo` element. This element has the following attributes.

Attribute	Description

majorVersion	Required. Specifies the major version number of the OS.
minorVersion	Required. Specifies the minor version number of the OS.
buildNumber	Required. Specifies the build number of the OS.
servicePackMajor	Required. Specifies the service pack major number of the OS.
servicePackMinor	Optional. Specifies the service pack minor number of the OS.
productType	Optional. Identifies the product type value. Valid values are server, workstation, and domainController. For example, for Windows 2000 Professional, this attribute value is workstation.
suiteType	Optional. Identifies a product suite available on the system, or the system's configuration type. Valid values are backoffice, blade, datacenter, enterprise, home, professional, smallbusiness, smallbusinessRestricted, and terminal. For example, for Windows 2000 Professional, this attribute value is professional.

dependentAssembly

Optional. Contains the assemblyIdentity element. The dependentOS and dependentAssembly elements are mutually exclusive: one or the other must exist for a dependency element, but not both.

dependentAssembly has the following attributes.

Attribute	Description
dependencyType	Required. Specifies the dependency type. Valid values are preprequisite and install. An install assembly is installed as part of the ClickOnce application. A prerequisite assembly must be present in the global assembly cache (GAC) before the ClickOnce application can install.
allowDelayedBinding	Required. Specifies whether the assembly can be loaded programmatically at runtime.
group	Optional. If the dependencyType attribute is set to install, designates a named group of assemblies that only install on demand. For more information, see Walkthrough: Downloading Assemblies on Demand with the ClickOnce Deployment API Using the Designer. If set to framework and the dependencyType attribute is set to prerequisite, designates the assembly as part of the .NET Framework. The global assemby cache (GAC) is not checked for this assembly when installing on .NET Framework 4 and later versions.

codeBase	Required when the dependencyType attribute is set to install. The path to the dependent assembly. May be either an absolute path, or a path relative to the manifest's code base. This path must be a valid URI in order for the assembly manifest to be valid.
size	Required when the dependencyType attribute is set to install. The size of the dependent assembly, in bytes.

assemblyIdentity

Required. This element is a child of the dependentAssembly element and has the following attributes.

Attribute	Description
name	Required. Identifies the name of the application.
version	Required. Specifies the version number of the application in the following format: **major.minor.build.revision**
publicKeyToken	Optional. Specifies a 16-character hexadecimal string that represents the last 8 bytes of the SHA-1 hash value of the public key under which the application or assembly is signed. The public key used to sign the catalog must be 2048 bits or more.
processorArchitecture	Optional. Specifies the processor. The valid values are **x86** for 32-bit Windows and **I64** for 64-bit Windows.
language	Optional. Identifies the two part language codes, such as EN-US, of the assembly.

hash

The hash element is an optional child of the assemblyIdentity element. The hash element has no attributes.

ClickOnce uses an algorithmic hash of all the files in an application as a security check, to ensure that none of the files were changed after deployment. If the hash element is not included, this check will not be performed. Therefore, omitting the hash element is not recommended.

dsig:Transforms

The dsig:Transforms element is a required child of the hash element. The dsig:Transforms element has no attributes.

dsig:Transform

The `dsig:Transform` element is a required child of the `dsig:Transforms` element. The `dsig:Transform` element has the following attributes.

Attribute	Description
`Algorithm`	The algorithm used to calculate the digest for this file. Currently the only value used by ClickOnce is **urn:schemas-microsoft-com:HashTransforms.Identity**.

dsig:DigestMethod

The `dsig:DigestMethod` element is a required child of the `hash` element. The `dsig:DigestMethod` element has the following attributes.

Attribute	Description
`Algorithm`	The algorithm used to calculate the digest for this file. Currently the only value used by ClickOnce is **http://www.w3.org/2000/09/xmldsig#sha1**.

dsig:DigestValue

The `dsig:DigestValue` element is a required child of the `hash` element. The `dsig:DigestValue` element has no attributes. Its text value is the computed hash for the specified file.

Remarks

All assemblies used by your application must have a corresponding `dependency` element. Dependent assemblies do not include assemblies that must be preinstalled in the global assembly cache as platform assemblies.

Example

The following code example illustrates `dependency` elements in a ClickOnce application manifest. This code example is part of a larger example provided for the ClickOnce Application Manifest topic.

```
<dependency>
  <dependentOS>
```

```
      <osVersionInfo>
        <os
          majorVersion="4"
          minorVersion="10"
          buildNumber="0"
          servicePackMajor="0" />
      </osVersionInfo>
    </dependentOS>
  </dependency>
  <dependency>
    <dependentAssembly
      dependencyType="preRequisite"
      allowDelayedBinding="true">
      <assemblyIdentity
        name="Microsoft.Windows.CommonLanguageRuntime"
        version="4.0.20506.0" />
    </dependentAssembly>
  </dependency>

  <dependency>
    <dependentAssembly
      dependencyType="install"
      allowDelayedBinding="true"
      codebase="MyApplication.exe"
      size="4096">
      <assemblyIdentity
        name="MyApplication"
        version="1.0.0.0"
        language="neutral"
        processorArchitecture="x86" />
      <hash>
        <dsig:Transforms>
          <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
        </dsig:Transforms>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <dsig:DigestValue>DpTW7RzS9IeT/RBSLj54vfTEzNg=</dsig:DigestValue>
      </hash>
    </dependentAssembly>
  </dependency>
```

See Also

ClickOnce Application Manifest
<dependency> Element (ClickOnce Deployment)

<file> Element (ClickOnce Application)

Visual Studio 2015

Identifies all nonassembly files downloaded and used by the application.

Syntax

```
<file
    name
    size
    group
    optional
    writeableType
>
    <typelib
        tlbid
        version
        helpdir
        resourceid
        flags
    />
    <comClass
        clsid
        description
        threadingModel
        tlbid
        progid
        miscStatus
        miscStatusIcon
        miscStatusContent
        miscStatusDocPrint
        miscStatusThumbnail
    />
    <comInterfaceExternalProxyStub
        iid
        baseInterface
        numMethods
        name
        tlbid
        proxyStubClass32
    />
    <comInterfaceProxyStub
        iid
        baseInterface
        numMethods
        name
```

```
            tlbid
            proxyStubClass32
        />
        <windowClass
            versioned
        />
    </file>
```

Elements and Attributes

The `file` element is optional. The element has the following attributes.

Attribute	Description
`name`	Required. Identifies the name of the file.
`size`	Required. Specifies the size, in bytes, of the file.
`group`	Optional, if the `optional` attribute is not specified or set to `false`; required if `optional` is `true`. The name of the group to which this file belongs. The name can be any Unicode string value chosen by the developer, and is used for downloading files on demand with the ApplicationDeployment class.
`optional`	Optional. Specifies whether this file must download when the application is first run, or whether the file should reside only on the server until the application requests it on demand. If **false** or undefined, the file is downloaded when the application is first run or installed. If `true`, a `group` must be specified for the application manifest to be valid. `optional` cannot be true if `writeableType` is specified with the value `applicationData`.
`writeableType`	Optional. Specifies that this file is a data file. Currently the only valid value is `applicationData`.

typelib

The `typelib` element is an optional child of the file element. The element describes the type library that belongs to the COM component. The element has the following attributes.

Attribute	Description
`tlbid`	Required. The GUID assigned to the type library.
`version`	Required. The version number of the type library.

`helpdir`	Required. The directory that contains the Help files for the component. May be zero-length.
`resourceid`	Optional. The hexadecimal string representation of the locale identifier (LCID). It is one to four hexadecimal digits without a 0x prefix and without leading zeros. The LCID may have a neutral sublanguage identifier.
`flags`	Optional. The string representation of the type library flags for this type library. Specifically, it should be one of "RESTRICTED", "CONTROL", "HIDDEN" and "HASDISKIMAGE".

comClass

The `comClass` element is an optional child of the `file` element, but is required if the ClickOnce application contains a COM component it intends to deploy using registration-free COM. The element has the following attributes.

Attribute	Description
`clsid`	Required. The class ID of the COM component expressed as a GUID.
`description`	Optional. The class name.
`threadingModel`	Optional. The threading model used by in-process COM classes. If this property is null, no threading model is used. The component is created on the main thread of the client and calls from other threads are marshaled to this thread. The following list shows the valid values: **Apartment**, **Free**, **Both**, and **Neutral**.
`tlbid`	Optional. GUID for the type library for this COM component.
`progid`	Optional. Version-dependent programmatic identifier associated with the COM component. The format of a `ProgID` is `<vendor>.<component>.<version>`.
`miscStatus`	Optional. Duplicates in the assembly manifest the information provided by the `MiscStatus` registry key. If values for the `miscStatusIcon`, `miscStatusContent`, `miscStatusDocprint`, or `miscStatusThumbnail` attributes are not found, the corresponding default value listed in `miscStatus` is used for the missing attributes. The value can be a comma-delimited list of the attribute values from the following table. You can use this attribute if the COM class is an OCX class that requires `MiscStatus` registry key values.
`miscStatusIcon`	Optional. Duplicates in the assembly manifest the information provided by DVASPECT_ICON. It can provide an icon of an object. The value can be a comma-delimited list of the attribute values from the following table. You can use this attribute if the COM class is an OCX class that requires `Miscstatus` registry key values.

`miscStatusContent`	Optional. Duplicates in the assembly manifest the information provided by DVASPECT_CONTENT. It can provide a compound document displayable for a screen or printer. The value can be a comma-delimited list of the attribute values from the following table. You can use this attribute if the COM class is an OCX class that requires `MiscStatus` registry key values.
`miscStatusDocPrint`	Optional. Duplicates in the assembly manifest the information provided by DVASPECT_DOCPRINT. It can provide an object representation displayable on the screen as if printed to a printer. The value can be a comma-delimited list of the attribute values from the following table. You can use this attribute if the COM class is an OCX class that requires `MiscStatus` registry key values.
`miscStatusThumbnail`	Optional. Duplicates in an assembly manifest the information provided by DVASPECT_THUMBNAIL. It can provide a thumbnail of an object displayable in a browsing tool. The value can be a comma-delimited list of the attribute values from the following table. You can use this attribute if the COM class is an OCX class that requires `MiscStatus` registry key values.

comInterfaceExternalProxyStub

The `comInterfaceExternalProxyStub` element is an optional child of the `file` element, but may be required if the ClickOnce application contains a COM component it intends to deploy using registration-free COM. The element contains the following attributes.

Attribute	Description
`iid`	Required. The interface ID (IID) which is served by this proxy. The IID must have braces surrounding it.
`baseInterface`	Optional. The IID of the interface from which the interface referenced by `iid` is derived.
`numMethods`	Optional. The number of methods implemented by the interface.
`name`	Optional. The name of the interface as it will appear in code.
`tlbid`	Optional. The type library that contains the description of the interface specified by the `iid` attribute.
`proxyStubClass32`	Optional. Maps an IID to a CLSID in 32-bit proxy DLLs.

comInterfaceProxyStub

The `comInterfaceProxyStub` element is an optional child of the `file` element, but may be required if the ClickOnce

application contains a COM component it intends to deploy using registration-free COM. The element contains the following attributes.

Attribute	Description
`iid`	Required. The interface ID (IID) which is served by this proxy. The IID must have braces surrounding it.
`baseInterface`	Optional. The IID of the interface from which the interface referenced by `iid` is derived.
`numMethods`	Optional. The number of methods implemented by the interface.
`Name`	Optional. The name of the interface as it will appear in code.
`Tlbid`	Optional. The type library that contains the description of the interface specified by the `iid` attribute.
`proxyStubClass32`	Optional. Maps an IID to a CLSID in 32-bit proxy DLLs.
`threadingModel`	Optional. Optional. The threading model used by in-process COM classes. If this property is null, no threading model is used. The component is created on the main thread of the client and calls from other threads are marshaled to this thread. The following list shows the valid values: **Apartment**, **Free**, **Both**, and **Neutral**.

windowClass

The `windowClass` element is an optional child of the `file` element, but may be required if the ClickOnce application contains a COM component it intends to deploy using registration-free COM. The element refers to a window class defined by the COM component that must have a version applied to it. The element contains the following attributes.

Attribute	Description
`versioned`	Optional. Controls whether the internal window class name used in registration contains the version of the assembly that contains the window class. The value of this attribute can be **yes** or **no**. The default is **yes**. The value **no** should only be used if the same window class is defined by a side-by-side component and an equivalent non-side-by-side component and you want to treat them as the same window class. Note that the usual rules about window class registration apply—only the first component that registers the window class will be able to register it, because it does not have a version applied to it.

hash

The `hash` element is an optional child of the `file` element. The `hash` element has no attributes.

ClickOnce uses an algorithmic hash of all the files in an application as a security check, to ensure that none of the files were changed after deployment. If the `hash` element is not included, this check will not be performed. Therefore, omitting the `hash` element is not recommended.

If a manifest contains a file that is not hashed, that manifest cannot be digitally signed, because users cannot verify the contents of an unhashed file.

dsig:Transforms

The `dsig:Transforms` element is a required child of the `hash` element. The `dsig:Transforms` element has no attributes.

dsig:Transform

The `dsig:Transform` element is a required child of the `dsig:Transforms` element. The `dsig:Transform` element has the following attributes.

Attribute	Description
`Algorithm`	The algorithm used to calculate the digest for this file. Currently the only value used by ClickOnce is **urn:schemas-microsoft-com:HashTransforms.Identity**.

dsig:DigestMethod

The `dsig:DigestMethod` element is a required child of the `hash` element. The `dsig:DigestMethod` element has the following attributes.

Attribute	Description
`Algorithm`	The algorithm used to calculate the digest for this file. Currently the only value used by ClickOnce is **http://www.w3.org/2000/09/xmldsig#sha1**.

dsig:DigestValue

The `dsig:DigestValue` element is a required child of the `hash` element. The `dsig:DigestValue` element has no

attributes. Its text value is the computed hash for the specified file.

Remarks

This element identifies all the nonassembly files that make up the application and, in particular, the hash values for file verification. This element can also include Component Object Model (COM) isolation data associated with the file. If a file changes, the application manifest file also must be updated to reflect the change.

Example

The following code example illustrates `file` elements in an application manifest for an application deployed using ClickOnce.

```
<file name="Icon.ico" size="9216">
  <hash>
    <dsig:Transforms>
      <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
    </dsig:Transforms>
    <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <dsig:DigestValue>lVoj+Rh6RQ/HPNLOdayQah5McrI=</dsig:DigestValue>
  </hash>
</file>
```

See Also

ClickOnce Application Manifest

<fileAssociation> Element (ClickOnce Application)

Visual Studio 2015

Identifies a file extension to be associated with the application.

Syntax

```
<fileAssociation
    xmlns="urn:schemas-microsoft-com:clickonce.v1"
    extension
    description
    progid
    defaultIcon
/>
```

Elements and Attributes

The `fileAssociation` element is optional. The element has the following attributes.

Attribute	Description
`extension`	Required. The file extension to be associated with the application.
`description`	Required. A description of the file type for use by the shell.
`progid`	Required. A name uniquely identifying the file type.
`defaultIcon`	Required. Specifies the icon to use for files with this extension. The icon file must be specified by using the <file> Element (ClickOnce Application) within the <assembly> Element (ClickOnce Application) that contains this element.

Remarks

This element must include an XML namespace reference to "urn:schemas-microsoft-com:clickonce.v1". If the

`<fileAssociation>` element is used, it must come after the `<application>` element in its parent <assembly> Element (ClickOnce Application).

ClickOnce will not overwrite existing file associations. However, a ClickOnce application can override the file extension for the current user only. After that ClickOnce application is uninstalled, ClickOnce deletes the file association for the user, and the per-machine association is active again.

Example

The following code example illustrates `fileAssociation` elements in an application manifest for a text editor application deployed using ClickOnce. This code example also includes the <file> Element (ClickOnce Application) required by the `defaultIcon` attribute.

```
<file name="text.ico" size="4286">
  <hash>
    <dsig:Transforms>
      <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
    </dsig:Transforms>
    <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <dsig:DigestValue>0joAqhmfeBb93ZneZv/oTMP2brY=</dsig:DigestValue>
  </hash>
</file>
<file name="writing.ico" size="9662">
  <hash>
    <dsig:Transforms>
      <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
    </dsig:Transforms>
    <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <dsig:DigestValue>2cL2U7cm13nG40v9MQdxYKazIwI=</dsig:DigestValue>
  </hash>
</file>
<fileAssociation xmlns="urn:schemas-microsoft-com:clickonce.v1" extension=".text"
description="Text  Document (ClickOnce)" progid="Text.Document" defaultIcon="text.ico" />
<fileAssociation xmlns="urn:schemas-microsoft-com:clickonce.v1" extension=".writing"
description="Writings (ClickOnce)" progid="Writing.Document" defaultIcon="writing.ico" />
```

See Also

ClickOnce Application Manifest

ClickOnce Deployment Manifest

Visual Studio 2015

A deployment manifest is an XML file that describes a ClickOnce deployment, including the identification of the current ClickOnce application version to deploy.

Deployment manifests have the following elements and attributes.

Element	Description	Attributes
<assembly> Element	Required. Top-level element.	manifestVersion
<assemblyIdentity> Element	Required. Identifies the application manifest for the ClickOnce application.	name version publicKeyToken processorArchitecture culture
<description> Element	Required. Identifies application information used to create a shell presence and the **Add or Remove Programs** item in Control Panel.	publisher product supportUrl
<deployment> Element	Optional. Identifies the attributes used for the deployment of updates and exposure to the system.	install minimumRequiredVersion mapFileExtensions disallowUrlActivation trustUrlParameters
<compatibleFrameworks> Element (ClickOnce Deployment)	Required. Identifies the versions of the .NET Framework where this application can install and run.	SupportUrl
<dependency> Element	Required. Identifies the version of the application to install for the deployment and the location of the application manifest.	preRequisite

		`visible`
		`dependencyType`
		`codebase`
		`size`

<publisherIdentity> Element (ClickOnce Deployment)	Required for signed manifests. Contains information about the publisher that signed this deployment manifest.	`Name` `issuerKeyHash`
<Signature> Element	Optional. Contains the necessary information to digitally sign this deployment manifest.	None
<customErrorReporting> Element (ClickOnce Deployment)	Optional. Specifies a URI to show when an error occurs.	Uri

Remarks

The deployment manifest file identifies a ClickOnce application deployment, including the current version and other deployment settings. It references the application manifest, which describes the current version of the application and all of the files contained within the deployment.

For more information, see ClickOnce Security and Deployment.

File Location

The deployment manifest file references the correct application manifest for the current version of the application. When you make a new version of an application deployment available, you must update the deployment manifest to refer to the new application manifest.

The deployment manifest file must be strongly named and can also contain certificates for publisher validation.

File Name Syntax

The name of a deployment manifest file must end with the .application extension.

Examples

The following code example illustrates a deployment manifest.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1
assembly.adaptive.xsd"
  manifestVersion="1.0"
  xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
  xmlns:dsig=http://www.w3.org/2000/09/xmldsig#
  xmlns:co.v1="urn:schemas-microsoft-com:clickonce.v1"
  xmlns:co.v2="urn:schemas-microsoft-com:clickonce.v2"
  xmlns="urn:schemas-microsoft-com:asm.v2"
  xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
  xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
  xmlns:xrml="urn:mpeg:mpeg21:2003:01-REL-R-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <assemblyIdentity
    name="My Application Deployment.app"
    version="1.0.0.0"
    publicKeyToken="43cb1e8e7a352766"
    language="neutral"
    processorArchitecture="x86"
    xmlns="urn:schemas-microsoft-com:asm.v1" />
  <description
    asmv2:publisher="My Company Name"
    asmv2:product="My Application"
    xmlns="urn:schemas-microsoft-com:asm.v1" />
  <deployment install="true">
    <subscription>
      <update>
        <expiration maximumAge="0" unit="days" />
      </update>
    </subscription>
    <deploymentProvider codebase="\\myServer\sampleDeployment
\MyApplicationDeployment.application" />
  </deployment>
  <compatibleFrameworks xmlns="urn:schemas-microsoft-com:clickonce.v2">
    <framework targetVersion="4.0" profile="Full" supportedRuntime="4.0.20506" />
    <framework targetVersion="4.0" profile="Client" supportedRuntime="4.0.20506" />
  </compatibleFrameworks>
  <dependency>
    <dependentAssembly
      dependencyType="install"
      codebase="1.0.0.0\My Application Deployment.exe.manifest"
      size="6756">
      <assemblyIdentity
        name="My Application Deployment.exe"
        version="1.0.0.0"
        publicKeyToken="43cb1e8e7a352766"
        language="neutral"
        processorArchitecture="x86"
        type="win32" />
      <hash>
        <dsig:Transforms>
          <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity"
/>
        </dsig:Transforms>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

```
          <dsig:DigestValue>E506x9FwNauks7UjQywmzgtd3FE=</dsig:DigestValue>
        </hash>
      </dependentAssembly>
    </dependency>
  <publisherIdentity name="CN=DOMAIN\MyUsername"
  issuerKeyHash="18312a18a21b215ecf4cdb20f5a0e0b0dd263c08" /><Signature
  Id="StrongNameSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
  …
  </Signature></asmv1:assembly>
```

See Also

Publishing ClickOnce Applications

<assembly> Element (ClickOnce Deployment)

Visual Studio 2015

The top-level element for the deployment manifest.

Syntax

```
      <assembly
   manifestVersion
/>
```

Elements and Attributes

The `assembly` element is the root element and is required. Its first contained element must be an `assemblyIdentity` element. The manifest elements must be in the following namespaces: **urn:schemas-microsoft-com:asm.v1**, **urn:schemas-microsoft-com:asm.v2**, and **http://www.w3.org/2000/09/xmldsig#**. Child elements of the assembly must also be in these namespaces, by inheritance or by tagging.

The `assembly` element has the following attribute.

Attribute	Description
`manifestVersion`	Required. This attribute must be set to **1.0**.

Example

The following code example illustrates an `assembly` element in a deployment manifest for an application deployed using ClickOnce. This code example is part of a larger example provided for the ClickOnce Deployment Manifest topic.

```
<asmv1:assembly
  xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd"
  manifestVersion="1.0"
  xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
```

```
    xmlns:dsig=http://www.w3.org/2000/09/xmldsig#
    xmlns:co.v1="urn:schemas-microsoft-com:clickonce.v1"
    xmlns:co.v2="urn:schemas-microsoft-com:clickonce.v2"
    xmlns="urn:schemas-microsoft-com:asm.v2"
    xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
    xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
    xmlns:xrml="urn:mpeg:mpeg21:2003:01-REL-R-NS"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

See Also

ClickOnce Deployment Manifest
<assembly> Element (ClickOnce Application)

<assemblyIdentity> Element (ClickOnce Deployment)

Visual Studio 2015

Identifies the primary assembly of the ClickOnce application.

Syntax

```
      <assemblyIdentity
    name
    version
    publicKeyToken
    processorArchitecture
     type
/>
```

Elements and Attributes

The `assemblyIdentity` element is required. It contains no child elements and has the following attributes.

Attribute	Description
`name`	Required. Identifies the human-readable name of the deployment for informational purposes. If `name` contains special characters, such as single or double quotes, the application may fail to activate.
`version`	Required. Specifies the version number of the assembly, in the following format: **major.minor.build.revision**. This value must be incremented in an updated manifest to trigger an application update.
`publicKeyToken`	Required. Specifies a 16-character hexadecimal string that represents the last 8 bytes of the SHA-1 hash value of the public key under which the deployment manifest is signed. The public key that is used to sign must be 2048 bits or greater. Although signing an assembly is recommended but optional, this attribute is required. If

	an assembly is unsigned, you should copy a value from a self-signed assembly or use a "dummy" value of all zeros.
`processorArchitecture`	Required. Specifies the processor. The valid values are `msil` for all processors, `x86` for 32-bit Windows, `IA64` for 64-bit Windows, and `Itanium` for Intel 64-bit Itanium processors.
`type`	Required. For compatibility with Windows side-by-side installation technology. The only allowed value is **win32**.

Remarks

Example

The following code example illustrates an `assemblyIdentity` element in a ClickOnce deployment manifest. This code example is part of a larger example provided for the ClickOnce Deployment Manifest topic.

```
<!-- Identify the deployment. -->
<assemblyIdentity
  name="My Application Deployment.app"
  version="1.0.0.0"
  publicKeyToken="43cb1e8e7a352766"
  language="neutral"
  processorArchitecture="x86"
  xmlns="urn:schemas-microsoft-com:asm.v1" />
```

See Also

ClickOnce Deployment Manifest
<assemblyIdentity> Element (ClickOnce Application)

<description> Element (ClickOnce Deployment)

Visual Studio 2015

Identifies application information used to create a shell presence and an **Add or Remove Programs** item in Control Panel.

Syntax

```
      <description
    publisher
    product
    suiteName
    supportUrl
/>
```

Elements and Attributes

The `description` element is required and is in the `urn:schemas-microsoft-com:asm.v1` namespace. It contains no child elements and has the following attributes.

Attribute	Description
`publisher`	Required. Identifies the company name used for icon placement in the Windows **Start** menu and the **Add or Remove Programs** item in Control Panel, when the deployment is configured for install.
`product`	Required. Identifies the full product name. Used as the title for the icon installed in the Windows **Start** menu.
`suiteName`	Optional. Identifies a subfolder within the `publisher` folder in the Windows **Start** menu.
`supportUrl`	Optional. Specifies a support URL that is shown in the **Add or Remove Programs** item in Control Panel. A shortcut to this URL is also created for application support in the Windows **Start** menu, when the deployment is configured for installation.

Remarks

The description element is required in all deployment configurations.

Example

The following code example illustrates a `description` element in a ClickOnce deployment manifest. This code example is part of a larger example provided for the ClickOnce Deployment Manifest topic.

```
<description
  asmv2:publisher="My Company Name"
  asmv2:product="My Application"
  xmlns="urn:schemas-microsoft-com:asm.v1" />
```

See Also

ClickOnce Deployment Manifest

<deployment> Element (ClickOnce Deployment)

Visual Studio 2015

Identifies the attributes used for the deployment of updates and exposure to the system.

Syntax

```
      <deployment
   install
   minimumRequiredVersion
   mapFileExtensions
   disallowUrlActivation
   trustUrlParameters
>
   <subscription>
         <update>
            <beforeApplicationStartup/>
            <expiration
               maximumAge
               unit
            />
         </update>
   </subscription>
   <deploymentProvider
      codebase
   />
</deployment>
```

Elements and Attributes

The `deployment` element is required and is in the `urn:schemas-microsoft-com:asm.v1` namespace. The element has the following attributes.

Attribute	Description
`install`	Required. Specifies whether this application defines a presence on the Windows **Start** menu and in the Control Panel **Add or Remove Programs** application. Valid values are

	true and **false**. If **false**, ClickOnce will always run the latest version of this application from the network, and will not recognize the `subscription` element.
`minimumRequiredVersion`	Optional. Specifies the minimum version of this application that can run on the client. If the version number of the application is less than the version number supplied in the deployment manifest, the application will not run. Version numbers must be specified in the format **N.N.N.N**, where **N** is an unsigned integer. If the `install` attribute is **false**, `minimumRequiredVersion` must not be set.
`mapFileExtensions`	Optional. Defaults to **false**. If **true**, all files in the deployment must have a .deploy extension. ClickOnce will strip this extension off these files as soon as it downloads them from the Web server. If you publish your application by using Visual Studio, it automatically adds this extension to all files. This parameter allows all the files within a ClickOnce deployment to be downloaded from a Web server that blocks transmission of files ending in "unsafe" extensions such as .exe.
`disallowUrlActivation`	Optional. Defaults to **false**. If **true**, prevents an installed application from being started by clicking the URL or entering the URL into Internet Explorer. If the `install` attribute is not present, this attribute is ignored.
`trustURLParameters`	Optional. Defaults to **false**. If **true**, allows the URL to contain query string parameters that are passed into the application, much like command-line arguments are passed to a command-line application. For more information, see How to: Retrieve Query String Information in an Online ClickOnce Application. If the `disallowUrlActivation` attribute is **true**, `trustUrlParameters` must either be excluded from the manifest, or explicitly set to **false**.

The `deployment` element also contains the following child elements.

subscription

Optional. Contains the `update` element. The `subscription` element has no attributes. If the `subscription` element does not exist, the ClickOnce application will never scan for updates. If the `install` attribute of the `deployment` element is **false**, the `subscription` element is ignored, because a ClickOnce application that is launched from the network always uses the latest version.

update

Required. This element is a child of the `subscription` element and contains either the `beforeApplicationStartup` or the `expiration` element. `beforeApplicationStartup` and `expiration` cannot both be specified in the same deployment manifest.

The `update` element has no attributes.

beforeApplicationStartup

Optional. This element is a child of the `update` element and has no attributes. When the `beforeApplicationStartup` element exists, the application will be blocked when ClickOnce checks for updates, if the client is online. If this element does not exist, ClickOnce will first scan for updates based on the values specified for the `expiration` element. `beforeApplicationStartup` and `expiration` cannot both be specified in the same deployment manifest.

expiration

Optional. This element is a child of the `update` element, and has no children. `beforeApplicationStartup` and `expiration` cannot both be specified in the same deployment manifest. When the update check occurs and an updated version is detected, the new version caches while the existing version runs. The new version then installs on the next launch of the ClickOnce application.

The `expiration` element supports the following attributes.

Attribute	Description
`maximumAge`	Required. Identifies how old the current update should become before the application performs an update check. The unit of time is determined by the `unit` attribute.
`unit`	Required. Identifies the unit of time for `maximumAge`. Valid units are **hours**, **days**, and **weeks**.

deploymentProvider

For the .NET Framework 2.0, this element is required if the deployment manifest contains a `subscription` section. For the .NET Framework 3.5 and later, this element is optional, and will default to the server and file path in which the deployment manifest was discovered.

This element is a child of the `deployment` element and has the following attribute.

Attribute	Description
`codebase`	Required. Identifies the location, as a Uniform Resource Identifier (URI), of the deployment manifest that is used to update the ClickOnce application. This element also allows for forwarding update locations for CD-based installations. Must be a valid URI.

Remarks

You can configure your ClickOnce application to scan for updates on startup, scan for updates after startup, or never

check for updates. To scan for updates on startup, ensure that the `beforeApplicationStartup` element exists under the `update` element. To scan for updates after startup, ensure that the `expiration` element exists under the `update` element, and that update intervals are provided.

To disable checking for updates, remove the `subscription` element. When you specify in the deployment manifest to never scan for updates, you can still manually check for updates by using the CheckForUpdate method.

For more information on how deploymentProvider relates to updates, see Choosing a ClickOnce Update Strategy.

Examples

The following code example illustrates a `deployment` element in a ClickOnce deployment manifest. The example uses a `deploymentProvider` element to indicate the preferred update location.

```
<deployment install="true" minimumRequiredVersion="2.0.0.0" mapFileExtension="true"
trustUrlParameters="true">
    <subscription>
      <update>
        <expiration maximumAge="6" unit="hours" />
      </update>
    </subscription>
    <deploymentProvider codebase="http://www.adatum.com/MyApplication.application" />
  </deployment>
```

See Also

ClickOnce Deployment Manifest

<compatibleFrameworks> Element (ClickOnce Deployment)

Visual Studio 2015

Identifies the versions of the .NET Framework where this application can install and run.

Note

MageUI.exe does not support the `compatibleFrameworks` element when saving an application manifest that has already been signed with a certificate using MageUI.exe. Instead, you must use Mage.exe.

Syntax

```
<compatibleFrameworks
      SupportUrl>
   <framework
      targetVersion
      profile
      supportedRuntime
   />
</ compatibleFrameworks>
```

Elements and Attributes

The `compatibleFrameworks` element is required for deployment manifests that target the ClickOnce runtime provided by .NET Framework 4 or later. The `compatibleFrameworks` element contains one or more `framework` elements that specify the .NET Framework versions on which this application can run. The ClickOnce runtime will run the application on the first available `framework` in this list.

The following table lists the attribute that the `compatibleFrameworks` element supports.

Attribute	Description
`SupportUrl`	Optional. Specifies a URL where the preferred compatible .NET Framework version can be downloaded.

framework

Required. The following table lists the attributes that the `framework` element supports.

Attribute	Description
`targetVersion`	Required. Specifies the version number of the target .NET Framework.
`profile`	Required. Specifies the profile of the target .NET Framework.
`supportedRuntime`	Required. Specifies the version number of the runtime associated with the target .NET Framework.

Remarks

Example

The following code example shows a `compatibleFrameworks` element in a ClickOnce deployment manifest. This deployment can run on the .NET Framework 4 Client Profile. It can also run on the .NET Framework 4 because it is a superset of the .NET Framework 4 Client Profile.

```
<compatibleFrameworks xmlns="urn:schemas-microsoft-com:clickonce.v2">
  <framework
      targetVersion="4.0"
      profile="Client"
      supportedRuntime="4.0.30319" />
</compatibleFrameworks>
```

See Also

ClickOnce Deployment Manifest

<dependency> Element (ClickOnce Deployment)

Visual Studio 2015

Identifies the version of the application to install, and the location of the application manifest.

Syntax

```
      <dependency>
   <dependentAssembly
      preRequisite
      visible
      dependencyType
      codeBase
      size
   >
      <assemblyIdentity
         name
         version
         publicKeyToken
         processorArchitecture
         language
         type
      />
      <hash>
         <dsig:Transforms>
            <dsig:Transform
                Algorithm
            />
         </dsig:Transforms>
         <dsig:DigestMethod />
         <dsig:DigestValue>
         </dsig:DigestValue>
      </hash>

   </dependentAssembly>
</dependency>
```

Elements and Attributes

The `dependency` element is required. It has no attributes. A deployment manifest can have multiple `dependency`

elements.

The `dependency` element usually expresses dependencies for the main application on assemblies contained within a ClickOnce application. If your Main.exe application consumes an assembly called DotNetAssembly.dll, then that assembly must be listed in a dependency section. Dependency, however, can also express other types of dependencies, such as dependencies on a specific version of the common language runtime, on an assembly in the global assembly cache (GAC), or on a COM object. Because it is a no-touch deployment technology, ClickOnce cannot initiate download and installation of these types of dependencies, but it does prevent the application from running if one or more of the specified dependencies do not exist.

dependentAssembly

Required. This element contains the `assemblyIdentity` element. The following table shows the attributes the `dependentAssembly` supports.

Attribute	Description
`preRequisite`	Optional. Specifies that this assembly should already exist in the GAC. Valid values are **true** and **false**. If **true**, and the specified assembly does not exist in the GAC, the application fails to run.
`visible`	Optional. Identifies the top-level application identity, including its dependencies. Used internally by ClickOnce to manage application storage and activation.
`dependencyType`	Required. The relationship between this dependency and the application. Valid values are: • **install**. Component represents a separate installation from the current application. • **preRequisite**. Component is required by the current application.
`codebase`	Optional. The full path to the application manifest.
`size`	Optional. The size of the application manifest, in bytes.

assemblyIdentity

Required. This element is a child of the `dependentAssembly` element. The content of `assemblyIdentity` must be the same as described in the ClickOnce application manifest. The following table shows the attributes of the `assemblyIdentity` element.

Attribute	Description
`Name`	Required. Identifies the name of the application.

`Version`	Required. Specifies the version number of the application, in the following format: **major.minor.build.revision**
`publicKeyToken`	Required. Specifies a 16-character hexadecimal string that represents the last 8 bytes of the SHA-1 hash of the public key under which the application or assembly is signed. The public key used to sign must be 2048 bits or greater.
`processorArchitecture`	Required. Specifies the microprocessor. The valid values are **x86** for 32-bit Windows and **IA64** for 64-bit Windows.
`Language`	Optional. Identifies the two part language codes of the assembly. For example, EN-US, which stands for English (U.S.). The default is **neutral**. This element is in the `asmv2` namespace.
`type`	Optional. For backwards compatibility with Windows side-by-side install technology. The only allowed value is **win32**.

hash

The `hash` element is an optional child of the `file` element. The `hash` element has no attributes.

ClickOnce uses an algorithmic hash of all the files in an application as a security check to ensure that none of the files were changed after deployment. If the `hash` element is not included, this check will not be performed. Therefore, omitting the `hash` element is not recommended.

dsig:Transforms

The `dsig:Transforms` element is a required child of the `hash` element. The `dsig:Transforms` element has no attributes.

dsig:Transform

The `dsig:Transform` element is a required child of the `dsig:Transforms` element. The following table shows the attributes of the `dsig:Transform` element.

Attribute	Description
`Algorithm`	The algorithm used to calculate the digest for this file. Currently the only value used by ClickOnce is **urn:schemas-microsoft-com:HashTransforms.Identity**.

dsig:DigestMethod

The `dsig:DigestMethod` element is a required child of the `hash` element. The following table shows the attributes of the `dsig:DigestMethod` element.

Attribute	Description
`Algorithm`	The algorithm used to calculate the digest for this file. Currently the only value used by ClickOnce is **http://www.w3.org/2000/09/xmldsig#sha1**.

dsig:DigestValue

The `dsig:DigestValue` element is a required child of the `hash` element. The `dsig:DigestValue` element has no attributes. Its text value is the computed hash for the specified file.

Remarks

Deployment manifests typically have a single `assemblyIdentity` element that identifies the name and version of the application manifest.

Example

The following code example shows a `dependency` element in a ClickOnce deployment manifest.

```
<!-- Identify the assembly dependencies -->
<dependency>
  <dependentAssembly dependencyType="install" allowDelayedBinding="true"
codebase="MyApplication.exe" size="16384">
    <assemblyIdentity name="MyApplication" version="0.0.0.0" cultural="neutral"
processorArchitecture="msil" />
    <hash>
      <dsig:Transforms>
        <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
      </dsig:Transforms>
      <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
       <dsig:DigestValue>YzXYZJAvj9pgAG3y8jXUjC7AtHg=</dsig:DigestValue>
    </hash>
  </dependentAssembly>
</dependency>
```

Example

The following code example specifies a dependency on an assembly already installed in the GAC.

```
<dependency>
  <dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">
    <assemblyIdentity name="GACAssembly" version="1.0.0.0" language="neutral"
processorArchitecture="msil" />
  </dependentAssembly>
</dependency>
```

Example

The following code example specifies a dependency on a specific version of the common language runtime.

```
<dependency>
  <dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">
    <assemblyIdentity name="Microsoft.Windows.CommonLanguageRuntime"
version="2.0.50215.0" />
  </dependentAssembly>
</dependency>
```

Example

The following code example specifies an operating system dependency.

```
<dependency>
   <dependentOS supportUrl="http://www.microsoft.com" description="Microsoft Windows
Operating System">
      <osVersionInfo>
         <os majorVersion="4" minorVersion="10" />
      </osVersionInfo>
   </dependentOS>
</dependency>
```

See Also

ClickOnce Deployment Manifest
<dependency> Element (ClickOnce Application)

<publisherIdentity> Element (ClickOnce Deployment)

Visual Studio 2015

Contains information about the publisher that signed this deployment manifest.

Syntax

```
<publisherIdentity
   name
   issuerKeyHash
/>
```

Elements and Attributes

The `publisherIdentity` element is required for signed manifests. The following table shows the attributes that the `publisherIdentity` element supports.

Attribute	Description
`name`	Required. Describes the identity of the party that published this application.
`issuerKeyHash`	Required. Contains the SHA-1 hash of the public key of the certificate issuer.

Parameters

Property Value/Return Value

Exceptions

Exception	Condition

Remarks

Requirements

Subhead

<Signature> Element (ClickOnce Deployment)

Visual Studio 2015

Contains the necessary information to digitally sign this deployment manifest.

Syntax

```
      <Signature>
   XML signature information
</Signature>
```

Remarks

Signing a deployment manifest using an envelope signature is optional, but recommended. For more information about signing XML files see the World Wide Web Consortium Recommendation, "XML-Signature Syntax and Processing," described at http://www.w3.org/TR/xmldsig-core/.

If you want to sign your manifest, hashes must be provided for all files. A manifest with files that are not hashed cannot be signed, because users cannot verify the contents of unhashed files.

Example

The following code example illustrates a `Signature` element in a deployment manifest used in a ClickOnce deployment.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm=
           "http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
    <SignatureMethod Algorithm=
           "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="">
      <Transforms>
        <Transform Algorithm=
           "http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

```
        <DigestValue>d2z5AE...</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
4PHj6SaopoLp...
    </SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>
MIIHnTCCBoWgAwIBAgIKJY9+nwAHAAB...
        </X509Certificate>
      </X509Data>
    </KeyInfo>
</Signature>
```

See Also

ClickOnce Deployment Manifest

<customErrorReporting> Element (ClickOnce Deployment)

Visual Studio 2015

Specifies a URI to show when an error occurs.

Syntax

```
<customErrorReporting
   uri
/>
```

Remarks

This element is optional. Without it, ClickOnce displays an error dialog box showing the exception stack. If the **customErrorReporting** element is present, ClickOnce will instead display the URI indicated by the *uri* parameter. The target URI will include the outer exception class, the inner exception class, and the inner exception message as parameters.

Use this element to add error reporting functionality to your application. Since the generated URI includes information about the type of error, your Web site can parse that information and display, for example, an appropriate troubleshooting screen.

Example

The following snippet shows the **customErrorReporting** element, together with the generated URI it might produce.

```
<customErrorReporting uri=http://www.contoso.com/applications/error.asp />

Example Generated Error:
http://www.contoso.com/applications/error.asp?
outer=System.Deployment.Application.InvalidDeploymentException&&
inner=System.Deployment.Application.InvalidDeploymentException&&
msg=The%20application%20manifest%20is%20signed,%20but%20the%20deployment%20manifest%20is%20u
```

See Also

ClickOnce Deployment Manifest

ClickOnce Unmanaged API Reference

Visual Studio 2015

ClickOnce unmanaged public APIs from dfshim.dll.

CleanOnlineAppCache

Cleans or uninstalls all online applications from the ClickOnce application cache.

Return Value

If successful, returns S_OK; otherwise, returns an HRESULT that represents the failure. If a managed exception occurs, returns 0x80020009 (DISP_E_EXCEPTION).

Remarks

Calling CleanOnlineAppCache will start the ClickOnce service if it is not already running.

GetDeploymentDataFromManifest

Retrieves deployment information from the manifest and activation URL.

Parameters

Parameter	Description	Type
pcwzActivationUrl	A pointer to the **ActivationURL**.	LPCWSTR
pcwzPathToDeploymentManifest	A pointer to the **PathToDeploymentManifest**.	LPCWSTR
pwzApplicationIdentity	A pointer to a buffer to receive a NULL-terminated string that specifies the full application identity returned.	LPWSTR
pdwIdentityBufferLength	A pointer to a DWORD that is the length of the *pwzApplicationIdentity* buffer, in WCHARs. This includes the space for the NULL termination character.	LPDWORD
pwzProcessorArchitecture	A pointer to a buffer to receive a NULL-terminated string that specifies the processor architecture of the application	LPWSTR

	deployment, from the manifest.	
pdwArchitectureBufferLength	A pointer to a DWORD that is the length of the *pwzProcessorArchitecture* buffer, in WCHARs.	LPDWORD
pwzApplicationManifestCodebase	A pointer to a buffer to receive a NULL-terminated string that specifies the codebase of the application manifest, from the manifest.	LPWSTR
pdwCodebaseBufferLength	A pointer to a DWORD that is the length of the *pwzApplicationManifestCodebase* buffer, in WCHARs.	LPDWORD
pwzDeploymentProvider	A pointer to a buffer to receive a NULL-terminated string that specifies the deployment provider from the manifest, if present. Otherwise, an empty string is returned.	LPWSTR
pdwProviderBufferLength	A pointer to a DWORD that is the length of the *pwzProviderBufferLength*.	LPDWORD

Return Value

If successful, returns S_OK; otherwise, returns an HRESULT that represents the failure. Returns HRESULTFROMWIN32(ERROR_INSUFFICIENT_BUFFER) if a buffer is too small.

Remarks

Pointers must not be null. *pcwzActivationUrl* and *pcwzPathToDeploymentManifest* must not be empty.

It is the caller's responsibility to clean up the activation URL. For example, adding escape characters where they are needed or removing the query string.

It is the caller's responsibility to limit the input length. For example, the maximum URL length is 2KB.

LaunchApplication

Launches or installs an application by using a deployment URL.

Parameters

Parameter	Description	Type

deploymentUrl	A pointer to a NULL-terminated string that contains the URL of the deployment manifest.	LPCWSTR
data	Reserved for future use. Must be NULL.	LPVOID
flags	Reserved for future use. Must be 0.	DWORD

Return Value

If successful, returns S_OK; otherwise, returns an HRESULT that represents the failure. If a managed exception occurs, returns 0x80020009 (DISP_E_EXCEPTION).

See Also

CleanOnlineAppCache

ClickOnce Deployment Samples and Walkthroughs

Visual Studio 2015

This section contains sample applications, example code, and step-by-step walkthroughs that illustrate the syntax, structure, and techniques used to deploy Windows Forms, WPF, and console applications.

The sample code is intended for instructional purposes, and should not be used in deployed solutions without modifications. In particular, security must be taken into greater consideration.

ClickOnce Deployment

Topic	Description
Deploying a ClickOnce Application Manually	Explains how to use .NET Framework utilities to deploy your ClickOnce application.
Downloading Assemblies on Demand with the ClickOnce Deployment API	Demonstrates how to mark certain assemblies in your application as "optional," and how to download them using classes in the System.Deployment.Application namespace.
Downloading Assemblies On Demand with the ClickOnce Deployment API Using the Designer	Explains how to download application assemblies only when they are first used by the application.

See Also

F5399A1F-2D3D-42FB-B989-134CCDA2159F
Visual Studio Samples