ClickOnce is designed with simple, straightforward applications in mind. It's particularly suitable for line-of-business applications and internal company software. Typically, these applications perform their work with the data and services on middle-tier server computers. As a result, they don't need privileged access to the local computer. These applications are also deployed in enterprise environments that may include thousands of workstations. In these environments, the cost of application deployment and updating isn't trivial, especially if it needs to be handled by an administrator. As a result, it's more important to provide a simple, streamlined setup process than to pack in features.

ClickOnce may also make sense for consumer applications that are deployed over the Web, particularly if these applications are updated frequently and don't have extensive installation requirements. However, the limitations of ClickOnce (such as the lack of flexibility for customizing the setup wizard) don't make it practical for sophisticated consumer applications that have detailed setup requirements or need to guide the user through a set of proprietary configuration steps. In these cases, you'll need to create a custom setup application.

■ **Note** For ClickOnce to install a WPF application, the computer must already have the .NET Framework runtime. When you first launch a ClickOnce setup, a bootstrapper runs that verifies this requirement. If the .NET Framework runtime isn't installed, the bootstrapper shows a message box that explains the issue and prompts the user to install .NET from Microsoft's website.

The ClickOnce Installation Model

Although ClickOnce supports several types of deployment, the overall model is designed to make web deployment practical and easy. Here's how it works: You use Visual Studio to publish your ClickOnce application to a web server. Then, the user surfs to an automatically generated web page (named publish.htm) that provides a link to install the application. When the user clicks that link, the application is downloaded, installed, and added to the Start menu. Figure 33-1 shows this process.

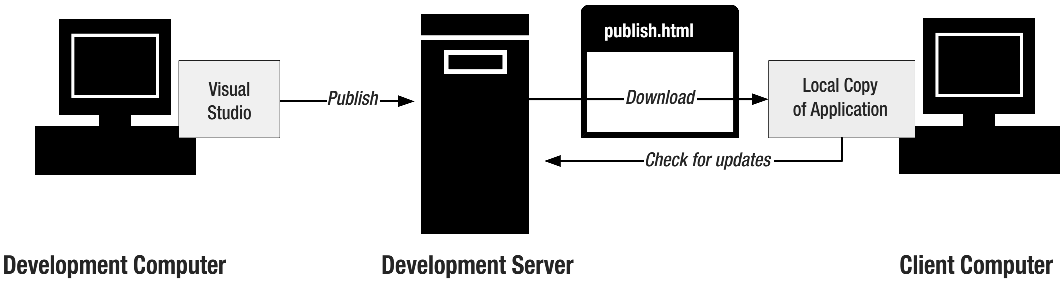


Figure 33-1. Installing a ClickOnce application

Although ClickOnce is ideal for web deployment, the same basic model lends itself to other scenarios, including the following:

- Deploying your application from a network file share
- Deploying your application from a CD or DVD

- Deploying your application to a web server or network file share and then sending a link to the setup program via e-mail

The installation web page isn't created when deploying to a network share, a CD, or a DVD. Instead, in these cases users must install the application by running the setup.exe program directly.

The most interesting part of a ClickOnce deployment is the way it supports updating. Essentially, you (the developer) have control over several update settings. For example, you can configure the application to check for updates automatically or periodically at certain intervals. When users launch your application, they actually run a shim that checks for newer versions and offers to download them.

You can even configure your application to use a web-like online-only mode. In this situation, the application must be launched from the ClickOnce web page. The application is still cached locally for optimum performance, but users won't be able to run the application unless they're able to connect to the site where the application was published. This ensures that users always run the latest, most up-to-date version of your application.

ClickOnce Limitations

ClickOnce deployment doesn't allow for much configuration. Many aspects of its behavior are completely fixed, either to guarantee a consistent user experience or to encourage enterprise-friendly security policies.

The limitations of ClickOnce include the following:

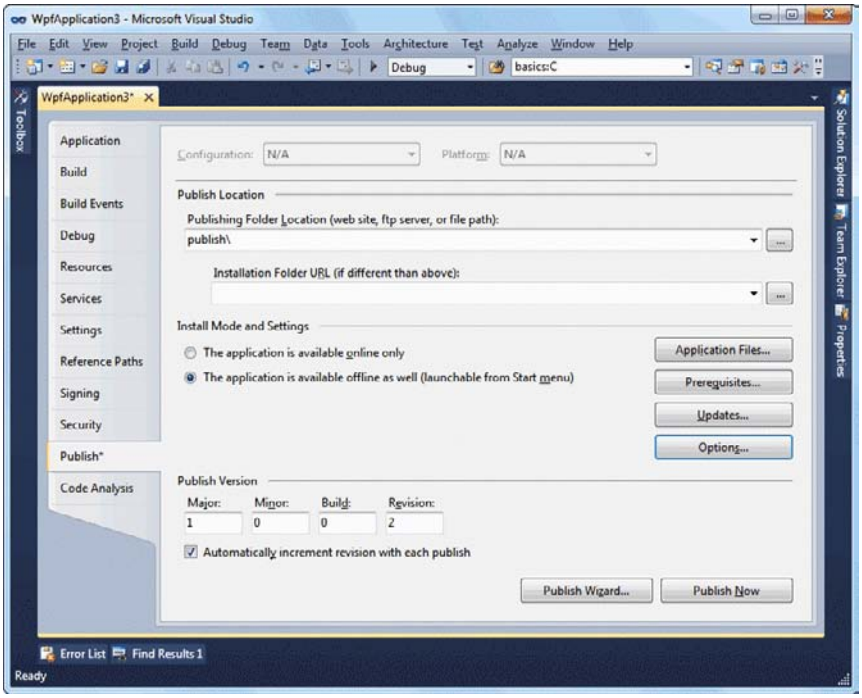
- ClickOnce applications are installed for a single user. You cannot install an application for all users on a workstation.
- ClickOnce applications are always installed in a system-managed user-specific folder. You cannot change or influence the folder where the application is installed.
- If ClickOnce applications are installed in the Start menu, you get just two shortcuts: one that launches the application and one that launches a help web page in the browser. You can't change this, and you can't add a ClickOnce application to the Startup group, the Favorites menu, and so on.
- You can't change the user interface of the setup wizard. That means you can't add new dialog boxes, change the wording of existing ones, and so on.
- You can't change the installation page that ClickOnce applications generate. However, you can edit the HTML by hand after it's generated.
- A ClickOnce setup can't install shared components in the global assembly cache (GAC).
- A ClickOnce setup can't perform custom actions (such as creating a database or configuring registry settings).

You can work around some of these issues. For example, you could configure your application to add registry settings the first time it's launched on a new computer. However, if you have complex setup requirements, you're much better off creating a full-fledged custom setup program. You can use a third-party tool like InstallShield, or you can create a setup project in Visual Studio.

Finally, it's worth noting that .NET makes it possible to build a custom installer that uses ClickOnce deployment technology. This gives you the option of designing an advanced setup application without sacrificing the automatic updating feature that ClickOnce provides. However, there are also some drawbacks. Not only does this approach force you to write (and debug) a fair bit of code, it also requires that you use the legacy classes from the Windows Forms toolkit to build your setup user interface. Mixing custom setup applications with ClickOnce is beyond the scope of this book, but if you're interested, you can get started with the example at `http://tinyurl.com/yf55qno`.

A Simple ClickOnce Publication

Before you get started with ClickOnce publishing, you need to set some basic information about your project. First, double-click the Properties node in the Solution Explorer, and then click the Publish tab. You'll see the settings shown in Figure 33-2.



Configuration: N/A Platform: N/A

Publish Location

Publishing Folder Location (web site, ftp server, or file path): publish\

Installation Folder URL (if different than above):

Install Mode and Settings

○ The application is available online only

● The application is available offline as well (launchable from Start menu)

Publish Version

Major:	Minor:	Build:	Revision:
1	0	0	2

☑ Automatically increment revision with each publish

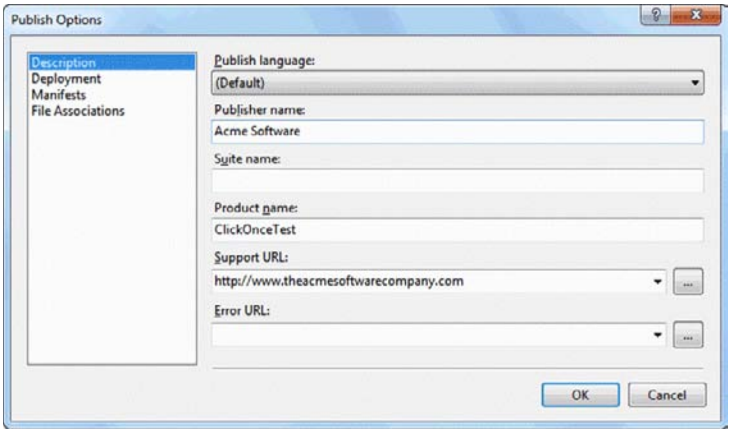
Figure 33-2. *ClickOnce project settings*

You'll learn your way around the settings in this window later in this chapter. But first, you need to supply some basic publication details.

Setting the Publisher and Production

Before you can install your application, it needs a basic identity, including a publisher name and a product name that can be used in the setup prompts and the Start menu shortcuts.

To supply this information, click the Options button to show the Publish Options dialog box. It displays a slew of additional settings, separated into several groups. Although Description is selected (in the list on the left), you'll see text boxes that allow you to supply three key details: the publisher name, suite name, and product name (see Figure 33-3).



Publish language: (Default)
Publisher name: Acme Software
Suite name:
Product name: ClickOnceTest
Support URL: http://www.theacmesoftwarecompany.com
Error URL:

Figure 33-3. *Supplying some basic information about your project*

These details are important because they're used to create the Start menu hierarchy. If you supply the optional suite name, ClickOnce creates a shortcut for your application in the form [Publisher Name] → [Suite Name] → [Product Name]. If you don't supply the suite name, ClickOnce creates the shortcut [Publisher Name] → [Product Name]. In the example shown in Figure 33-3, the shortcut will be generated as Acme Software → ClickOnceTest (Figure 33-4).

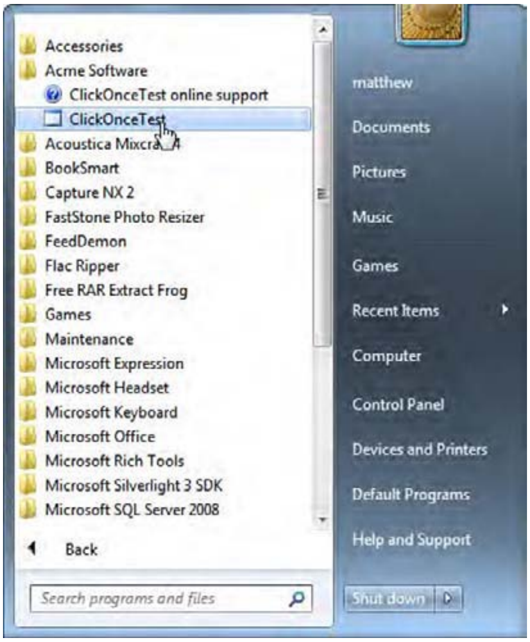


Figure 33-4. *The ClickOnce shortcuts (based on the information in Figure 33-3)*

If you specify a support URL, ClickOnce will create an additional shortcut named "[Product Name] online support." When the user clicks this shortcut, it launches the default web browser and sends it to the page you've specified.

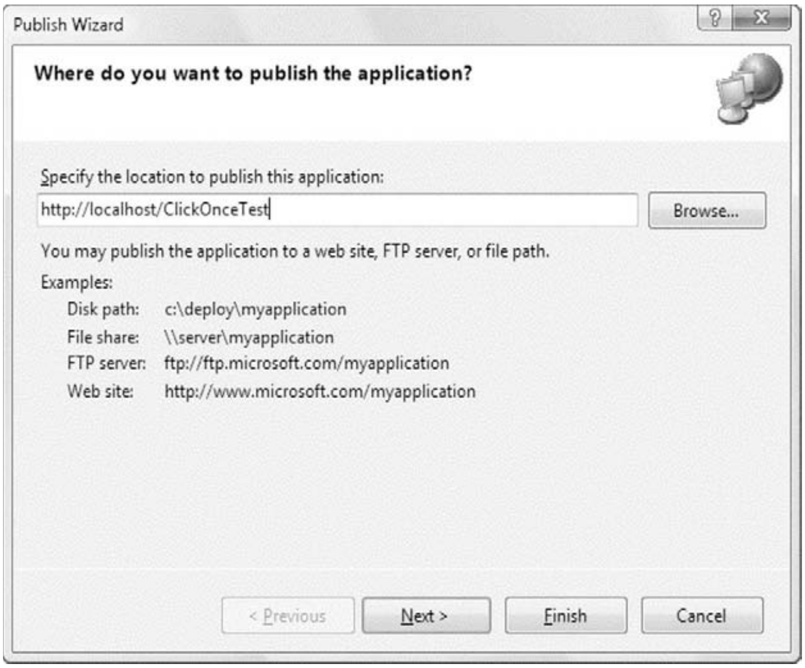
The error URL specifies a website link that will be shown (in a dialog box) if an error occurs while attempting to install the application.

You'll learn about the other groups of settings at the end of this chapter. For now, click OK once you've filled in the publisher name, product name, and any other details you choose to supply.

Starting the Publish Wizard

The easiest way to configure your ClickOnce settings is to click the Publish Wizard button at the bottom of the property page shown in Figure 33-3. This launches a wizard that walks you through a few short steps to gather the essential information. The wizard doesn't give you access to all the ClickOnce features you'll learn about in this chapter, but it's a quick way to get started.

The first choice you're faced with in the Publish Wizard is choosing the location where you want to publish the application (see Figure 33-5).


Publish Wizard

Where do you want to publish the application?

Specify the location to publish this application:

http://localhost/ClickOnceTest

You may publish the application to a web site, FTP server, or file path.

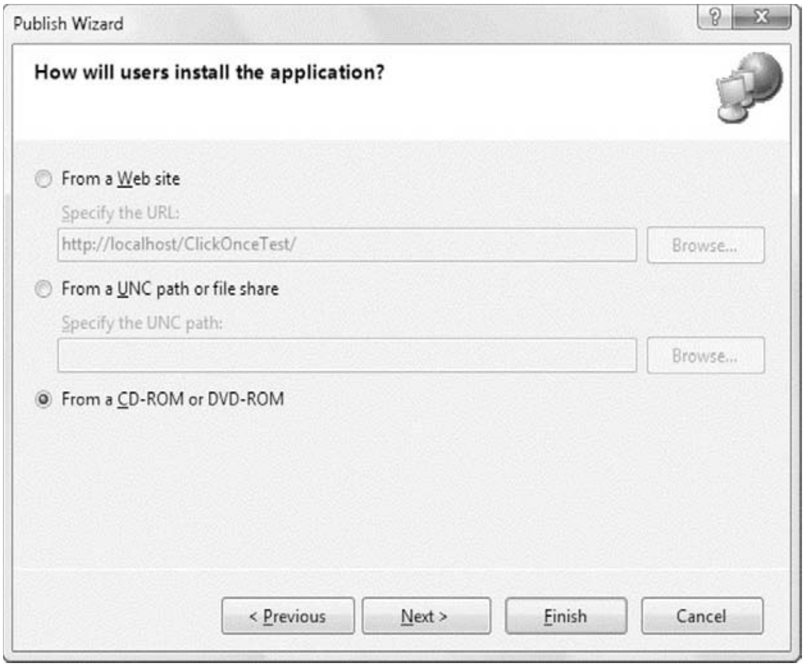
Examples:

Disk path:	c:\deploy\myapplication
File share:	\\server\myapplication
FTP server:	ftp://ftp.microsoft.com/myapplication
Web site:	http://www.microsoft.com/myapplication

Figure 33-5. *Choosing a publish location*

There's nothing particularly important about the location where you first publish your application, because this isn't necessarily the same location you'll use to host the setup files later. In other words, you could publish to a local directory and then transfer the files to a web server. The only caveat is that you need to know the ultimate destination of your files when you run the Publish Wizard, because you need to supply this information. Without it, the automatic update feature won't work.

Of course, you could choose to publish the application directly to its final destination, but it's not necessary. In fact, building the installation locally is often the easiest option. To get a better sense for how this works, start by choosing a local file path location (such as c:\Temp\ClickOnceApp). Then click Next. You're now faced with the real question—where users will go to install this application (see Figure 33-6).


Publish Wizard

How will users install the application?

○ From a Web site
Specify the URL:
http://localhost/ClickOnceTest/

○ From a UNC path or file share
Specify the UNC path:

◉ From a CD-ROM or DVD-ROM

Figure 33-6. *Choosing the installation type*

This bit is important, because it influences your update strategy. The choices you make are stored in a manifest file that's deployed with your application.

■ **Note** There is one case in which you won't see the dialog box in Figure 33-6. If you enter a virtual directory to a web server for the publish location (in other words, a URL starting with `http://`), the wizard assumes this is the final installation location.

In Figure 33-6, you have essentially three choices. You can create an installation for a network file share, a web server, or CD or DVD media. The following sections explain each approach.

Publishing for a Network File Share

In this case, all the users in your network will access the installation by browsing to a specific UNC path and running a file named setup.exe at that location.

A UNC path is a network path in the form \\ComputerName\ShareName. You can't use a networked drive, because networked drives depend on system settings (so different users might have their drives mapped differently). To provide automatic updates, the ClickOnce infrastructure needs to know exactly where it can find the installation files, because this is also the location where you'll deploy updates.

Publishing for a Web Server

You can create an installation for a web server on a local intranet or the Internet. Visual Studio will generate an HTML file named publish.htm that simplifies the process. Users request this page in a browser and click a link to download and install the application.

You have several options for transferring your files to a web server. If you want to take a two-step approach (publish the files locally and then transfer them to the right location), you simply need to copy the files from the local directory to your web server using the appropriate mechanism (such as FTP). Make sure you preserve the directory structure.

If you want to publish your files straight to the web server without any advance testing, you have two choices. If you are using IIS and the current account you're running has the necessary permissions to create a new virtual directory on the web server (or upload files to an existing one), you can publish files straight to your web server. Just supply the virtual directory path in the first step of the wizard. For example, you could use the publish location `http://ComputerName/VirtualDirectoryName` (in the case of an intranet) or `http://DomainName/VirtualDirectoryName` (for a server on the Internet).

You can also publish straight to a web server using FTP. This is often required in Internet (rather than intranet) scenarios. In this case, Visual Studio will contact your web server and transfer the ClickOnce files over FTP. You'll be prompted for user and password information when you connect.

■ **Note** FTP is used to transfer files—it's not used for the actual installation process. Instead, the idea is that the files you upload become visible on some web server, and users install the application from the publish.htm file on that web server. As a result, when you use an FTP path in the first step of the wizard (Figure 33-5), you'll still need to supply the corresponding web URL in the second step (Figure 33-6). This is important, because the ClickOnce publication needs to return to this location to perform its automatic update checks.

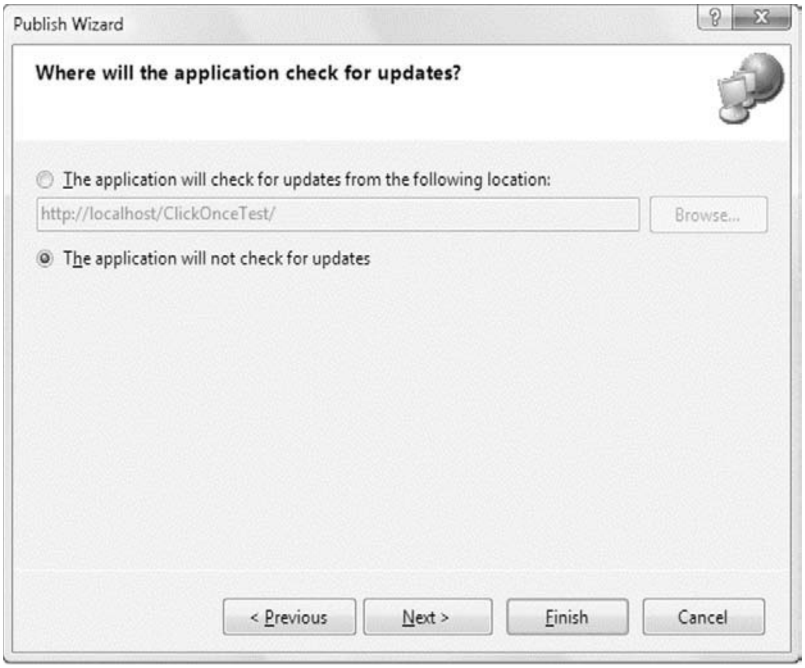
PUBLISHING TO THE LOCAL WEB SERVER

If you're publishing your application to a virtual directory on the local computer, you'll need to ensure that Internet Information Services (IIS) is installed using the Programs and Features entry in the Control Panel, which allows you to turn Windows features on or off. When you choose to install IIS, make sure you include the .NET Extensibility option and the IIS 6 Management Compatibility option (which allows Visual Studio to interact with IIS).

Additionally, if you're using Windows Vista or Windows 7, you need to run Visual Studio as an administrator before you can publish to a virtual directory. The easiest way to do this is to right-click the Microsoft Visual Studio 2010 shortcut in the Start menu and choose Run As Administrator. You can also configure your computer to always run Visual Studio as an administrator, which is a trade-off between convenience and security that needs to be weighed carefully. To put this in place, right-click the Visual Studio shortcut, choose Properties, and then head to the Compatibility tab, where you'll find an option named Run This Program As An Administrator.

Publishing for a CD or DVD

If you choose to publish to setup media such as a CD or DVD, you still need to decide whether you plan to support the automatic update feature. Some organizations will use CD-based deployment exclusively, while others will use it to supplement their existing web-based or networked-based deployment. You choose which option applies for use in the third step of the wizard (see Figure 33-7).


Publish Wizard

Where will the application check for updates?

( ) The application will check for updates from the following location:

http://localhost/ClickOnceTest/

(●) The application will not check for updates

Figure 33-7. Support for automatic updates

Here, you have three options:

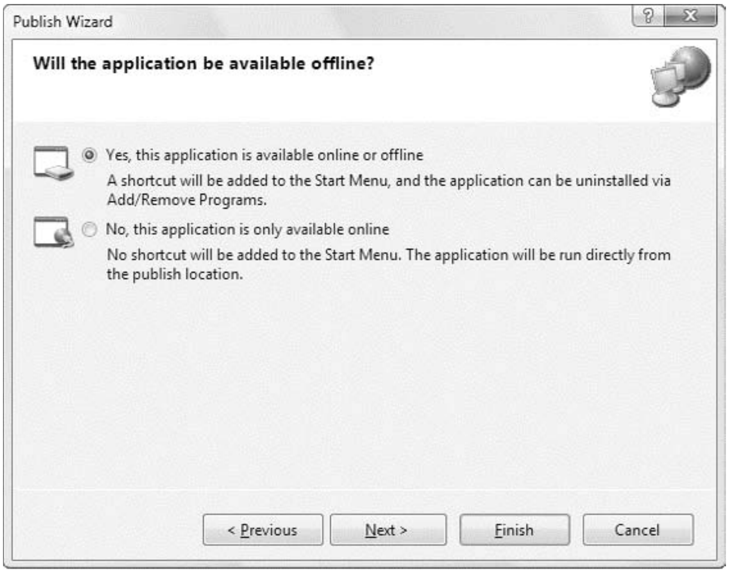
- You can supply a URL or UNC path that the application will check for updates. This assumes that you plan to publish the application to that location.

- You can omit this information and bypass the automatic update feature altogether.
- You can omit this information but tell the ClickOnce application to use the install location as the update location. For example, if you use this strategy and someone installs the application from \\CompanyServer-B\MyClickOnceApp, the application will automatically check this location (and only this location) for updates each time it runs. This looser approach gives you more flexibility, but it also risks causing problems (most commonly, not being able to find updated versions if users install it from the wrong path). You can't choose this behavior through the Publish Wizard. If you want it, you need to set the "Exclude deployment provider URL" setting, as described in the "Publish Options" section later in this chapter.

■ **Note** The Publish Wizard doesn't give you an option for how often to check for updates. By default, ClickOnce applications check for an update whenever they're launched. If a new version is found, .NET prompts the user to install it before launching the application. You'll learn how to change these settings in the "Updates" section later in this chapter.

Online or Offline

If you're creating a deployment for a web server or network share, you'll get one additional option, as shown in Figure 33-8.


Publish Wizard

Will the application be available offline?

(●) Yes, this application is available online or offline
A shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.

( ) No, this application is only available online
No shortcut will be added to the Start Menu. The application will be run directly from the publish location.

Figure 33-8. *Support for offline use*

The default choice is to create an online/offline application that runs whether or not the user can connect to the published location. In this case, a shortcut for the application is added to the Start menu.

If you choose to create an online-only application, the user needs to return to the published location to run the application. (To help make this clear, the publish.htm web page will show a button labeled Run instead of Install.) This ensures that an old version of the application can't be used after you roll out an update. This part of the deployment model is analogous to a web application.

When you create an online-only application, the application will still be downloaded (into a locally cached location) the first time it's launched. Thus, while startup times may be longer (because of the initial download), the application will still run as quickly as any other installed Windows application. However, the application can't be launched when the user isn't connected to the network or Internet, which makes it unsuitable for mobile users (such as laptop users who don't always have an Internet connection available).

If you choose to create an application that supports offline use, the setup program will add a Start menu shortcut. The user can launch the application from this shortcut, regardless of whether the computer is online or offline. If the computer is online, the application will check for new versions in the location where the application was published. If an update exists, the application will prompt the user to install it. You'll learn how to configure this policy later.

■ **Note** If you choose to publish for a CD installation, you don't have the option of creating an online-only application.

This is the final choice in the Publish Wizard. Click Next to see the final summary, and click Finish to generate the deployment files and copy them to the location you chose in step 1.

■ **Tip** From this point on, you can quickly republish your application by clicking the Publish Now button or by choosing Build → Publish [ApplicationName] from the menu.

The Deployed File Structure

ClickOnce uses a fairly straightforward directory structure. It creates a setup.exe file in the location you chose and a subdirectory for the application.

For example, if you deployed an application named ClickOnceTest to the location c:\ClickOnceTest, you'll end up with files like these:

```
c:\ClickOnceTest\setup.exe
c:\ClickOnceTest\publish.htm
c:\ClickOnceTest\ClickOnceTest.application
c:\ClickOnceTest\ClickOnceTest_1_0_0_0.application
c:\ClickOnceTest\ClickOnceTest_1_0_0_0\ClickOnceTest.exe.deploy
c:\ClickOnceTest\ClickOnceTest_1_0_0_0\ClickOnceTest.exe.manifest
```

The publish.htm file is present only if you're deploying to a web server. The .manifest and .application files store information about required files, update settings, and other details. (You can get a low-level look at these files and their XML file in the MSDN Help.) The .manifest and .application files are digitally signed at the time of publication, so these files can't be modified by hand. If you do make a change, ClickOnce will notice the discrepancy and refuse to install the application.

As you publish newer versions of your application, ClickOnce adds new subdirectories for each new version. For example, if you change the publish version of your application to 1.0.0.1, you'll get a new directory like this:

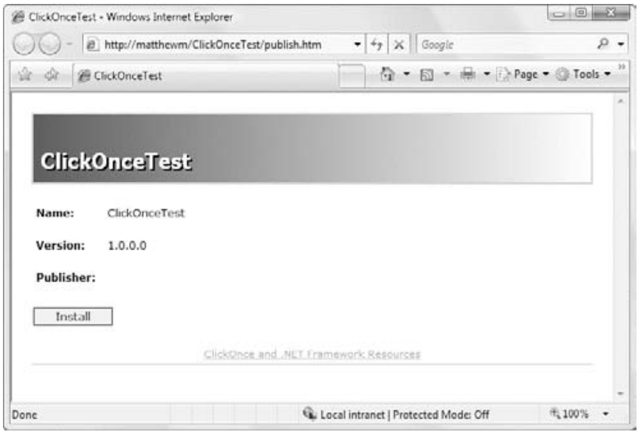
```
c:\ClickOnceTest\ClickOnceTest_1_0_0_1\ClickOnceTest.exe.deploy
c:\ClickOnceTest\ClickOnceTest_1_0_0_1\ClickOnceTest.exe.manifest
```

When you run the setup.exe program, it handles the process of installing any prerequisites (such as the .NET Framework) and then installs the most recent version of your application.

Installing a ClickOnce Application

To see ClickOnce in action with a web deployment, follow these steps:

0. Make sure you have the optional IIS web server component installed (as described in the "Publishing to the Local Web Server" sidebar earlier in this chapter).
2. Using Visual Studio, create a basic Windows application, and compile it.
3. Launch the Publish Wizard (by clicking the Publish Wizard button or choosing Build → Publish), and select `http://localhost/ClickOnceTest` for the publish location. The localhost portion of the URL points to the current computer. As long as IIS is installed and you are running with sufficient privileges, Visual Studio will be able to create this virtual directory.
4. Choose to create an online and offline application, and then click Finish to end the wizard. The files will be deployed to a folder named ClickOnceTest in the IIS web server root (by default, the directory c:\Inetpub\wwwroot).
5. Run the setup.exe program directly, or load up the publish.htm page (shown in Figure 33-9) and click Install. You'll receive a security message asking whether you want to trust the application (similar to when you download an ActiveX control in a web browser).


ClickOnceTest - Windows Internet Explorer
http://matthewm/ClickOnceTest/publish.htm
Google
ClickOnceTest
Page
Tools
Install
ClickOnce and .NET Framework Resources
Done
Local intranet | Protected Mode: Off
100%

ClickOnceTest

Name: ClickOnceTest

Version: 1.0.0.0

Publisher:

Figure 33-9. *The publish.htm installation page*

6. If you choose to continue, the application will be downloaded, and you'll be asked to verify that you want to install it.
7. Once the application is installed, you can run it from the Start menu shortcut or uninstall it using the Add/Remove Programs dialog box.

The shortcut for ClickOnce applications isn't the standard shortcut to which you're probably accustomed. Instead, it's an application reference—a text file with information about the application name and the location of the deployment files. The actual program files for your application are stored in a location that's difficult to find and impossible to control. The location follows this pattern:

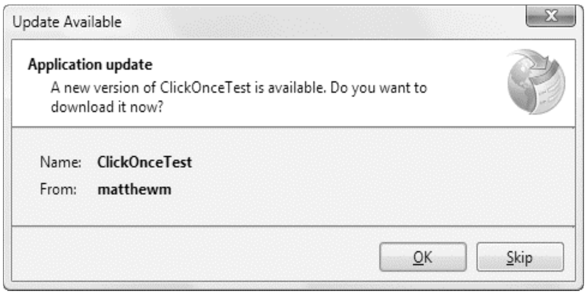
```
c:\Documents and Settings\[UserName]\Local Settings\Apps\2.0\[...]\[...]\[...]
```

The final three portions of this path are opaque, automatically generated strings like C6VLXKCE.828. Clearly, you aren't expected to access this directory directly.

Updating a ClickOnce Application

To see how a ClickOnce application can update itself automatically, follow these steps with the installation from the previous example:

1. Make a minor but noticeable change in the application (for example, adding a button).
2. Recompile the application, and republish it to the same location.
3. Run the application from the Start menu. The application will detect the new version and ask you whether you'd like to install it (see Figure 33-10).
4. Once you accept the update, the new version of the application will install and start.


Update Available

Application update

A new version of ClickOnceTest is available. Do you want to download it now?

Name: ClickOnceTest

From: matthewm

Figure 33-10. Detecting a newer version of a ClickOnce application

In the following sections, you'll learn how to customize some additional ClickOnce options.

■ **Note** The ClickOnce engine, dfsvc.exe, handles updates and downloads.

Additional ClickOnce Options

The Publish Wizard is a quick way to create a ClickOnce deployment, but it doesn't allow you to adjust all the possible options. For that, you need to take a closer look at the Publish tab in the application properties window shown earlier.

Many of the settings here duplicate details you've already seen in the wizard. For example, the first two text boxes allow you to set the publishing location (the place where the ClickOnce files will be placed, as set in step 1 of the wizard) and the installation location (the place from which the user will run the setup, as set in step 2 of the wizard). The Install Mode setting allows you to choose whether the application should be installed on the local computer or run in an online-only mode, as described earlier. However, there are also some settings you haven't already seen, which are discussed in the following sections.

The following sections discuss the settings you haven't already seen.

Publish Version

The Publish Version section sets the version of your application that's stored in the ClickOnce manifest file. This isn't the same as the assembly version, which you can set on the Application tab, although you might set both to match.

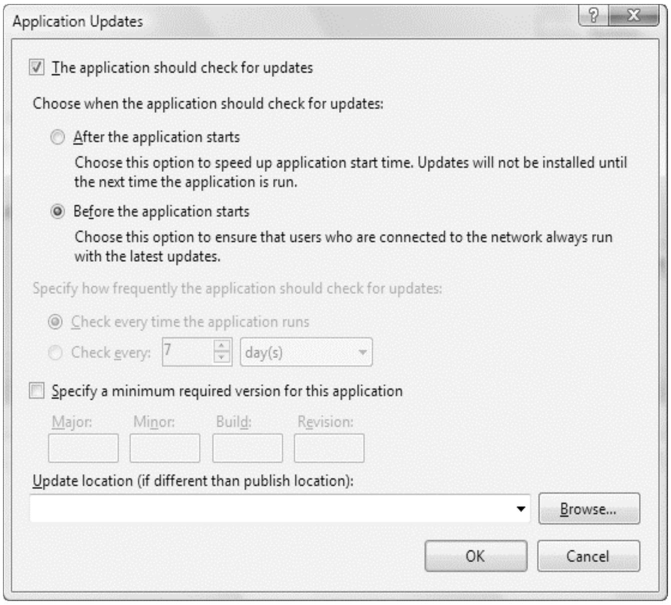
The key difference is that the publish version is the criteria that are used to determine whether a new update is available. If a user launches version 1.5.0.0 of an application and version 1.5.0.1 is available, the ClickOnce infrastructure will show the update dialog box shown in Figure 33-10.

By default, the Automatically Increment Revision with Each Publish check box is set, in which case the final part of the publish version (the revision number) is incremented by 1 after each publication, so 1.0.0.0 becomes 1.0.0.1, then 1.0.0.2, and so on. If you want to publish the same version of your application to multiple locations using Visual Studio, you should switch off this option. However, keep in mind that the automatic update feature springs into action only if it finds a higher version number. The date stamp on the deployed files has no effect (and isn't reliable).

It may seem horribly inelegant to track separate assembly and publication version numbers. However, sometimes it makes sense. For example, while testing an application, you may want to keep the assembly version number fixed without preventing testers from getting the latest version. In this case, you can use the same assembly version number but keep the autoincrementing publish version number. When you're ready to release an official update, you can set the assembly version and the publish version to match. Also, a published application might contain multiple assemblies with different version numbers. In this case, it wouldn't be realistic to use the assembly version number—instead, the ClickOnce infrastructure needs to consider a single version number to determine whether an update is warranted.

Updates

Click the Updates button to show the Application Updates dialog box (Figure 33-11), where you can choose your update strategy.



[✓] The application should check for updates

Choose when the application should check for updates:

○ After the application starts

Choose this option to speed up application start time. Updates will not be installed until the next time the application is run.

● Before the application starts

Choose this option to ensure that users who are connected to the network always run with the latest updates.

Specify how frequently the application should check for updates:

● Check every time the application runs

○ Check every: 7 day(s)

[ ] Specify a minimum required version for this application

Major: Minor: Build: Revision:

Update location (if different than publish location):

Figure 33-11. Setting update options

■ **Note** The Updates button isn't available if you're creating an online-only application. An online-only application always runs from its published location on a website or network share.

You first choose whether the application performs update checking. If it does, you can choose when updates are performed. You have two options:

- **Before the application starts.** If you use this model, the ClickOnce infrastructure checks for an application update (on the website or network share) every time the user runs the application. If an update is detected, it's installed, and *then* the application is launched. This option is a good choice if you want to make sure the user gets an update as soon as it's available.
- **After the application starts.** If you use this model, the ClickOnce infrastructure checks for a new update after the application is launched. If an updated version is detected, this version is installed the *next* time the user starts the application. This is the recommended option for most applications, because it improves load times.

If you choose to perform checks after the application starts, the check is performed in the background. You can choose to perform it every time the application is run (the default option) or in less frequent intervals. For example, you can limit checks to once per number of hours, days, or weeks.

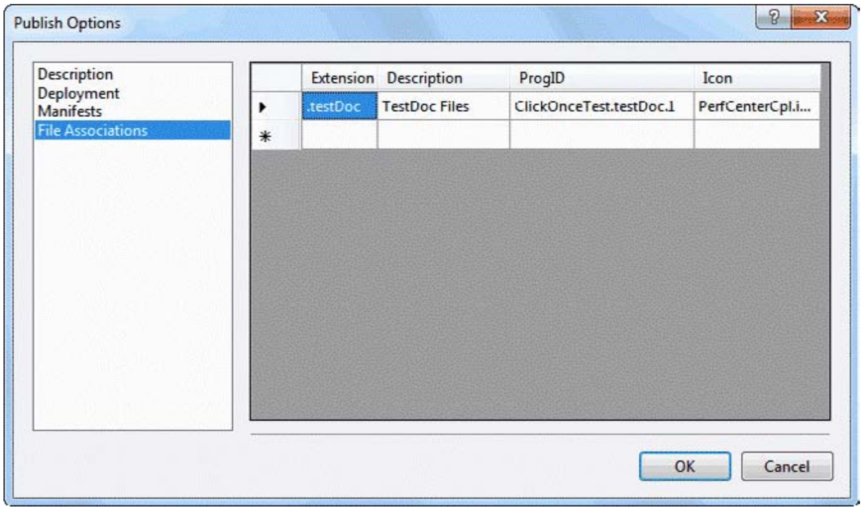
You can also specify a minimum required version. You can use this to make updates mandatory. For example, if you set the publish version to 1.5.0.1 and the minimum version to 1.5.0.0 and then publish your application, any user who has a version older than 1.5.0.0 will be forced to update before being allowed to run the application. (By default there is no minimum version, and all updates are optional.)

■ **Note** Even if you specify a minimum version and require the application to check for updates before starting, a user could end up running an old version of your application. This happens if the user is offline, in which case the update check will fail without an error. The only way around this limitation is to create an online-only application.

File Associations

ClickOnce allows you to set up to eight file associations. These are file types that will be linked to your application so that double-clicking a file of this type in Windows Explorer automatically launches your application.

To create a file association, begin by clicking the Options button in the Publish tab. This shows the Publish Options dialog box. Then, click File Associations in the list on the left. This shows a grid where you can enter the information for a file association, as shown in Figure 33-12.



Extension	Description	ProgID	Icon
.testDoc	TestDoc Files	ClickOnceTest.testDoc.1	PerfCenterCpl.i...

__Figure 33-12.__ Creating a file association

Every file association requires four pieces of information: the file extension, text description, ProgID, and icon file. The ProgID is a text code that uniquely identifies your file type. By convention, it should be based on the application name and version, as in MyApplication.testDoc.1.0, although the format doesn't really matter as long as it's unique. The icon points to a file in your project. In order for this file to be included in your ClickOnce setup, you must select it in the Solution Explorer and set the Build Action to Content.

■ **Note** One detail you don't need to specify for your file association is the name or path of your program. That's because ClickOnce already has this information.

There's one potential stumbling block when using file associations with ClickOnce. Contrary to what you might expect, when a user double-clicks a registered file, it isn't passed to your application as a command-line argument. Instead, you must retrieve it from the current AppDomain, as shown here:

```
string commandLineFile =
  AppDomain.CurrentDomain.SetupInformation.ActivationArguments.ActivationData[0];
```

Another tricky point is that the file location is passed in URI format, as in file:///c:\MyApp\MyFile.testDoc. That means you need code like this to get the true file path and clean up escaped spaces (which are translated in the %20 character in a URI):

```
Uri fileUri = new Uri(commandLineFile);
string filePath = Uri.UnescapeDataString(fileUri.AbsolutePath);
```

You can now check for the existence of the file and attempt to open it, as you normally would.

Publish Options

As you've already seen, you can click the Options button to see the Publish Options dialog box with even more options. You use the list on the left to pick the group of settings you want to tweak.

You've already looked through the settings in the Description and File Association groups. Table 33-1 describes the settings in the Deployment group, and Table 33-2 describes the settings in the Manifest group.

Table 33-1. *ClickOnce Deployment Settings*

Setting	Description
Deployment web page	Sets the name of the installation page in web deployments (which is publish.htm by default).
Automatically generate deployment web page after every publish	If set (the default), the web page is re-created during every publish operation.
Open deployment web page after publish	If set (the default), Visual Studio launched the installation page in your web browser after a successful publication so you can test it.
Use ".deploy" file extension	If set (the default), the installation web page always has the file extension .deploy. You shouldn't change this detail, because the .deploy file extension is registered on the IIS web server and locked down to prevent malicious users from snooping through it.

Setting	Description
For CD installations, automatically start Setup when CD is inserted	If set, Visual Studio generates an autorun.inf file to tell CD or DVD players to launch the setup program immediately when the CD is inserted into the drive.
Verify files uploaded to a web server	If set, the publish process downloads each file after publishing it to verify that it can be downloaded. If a file cannot be downloaded, you'll receive a notification that explains the problem.

Table 33-2. *ClickOnce Manifest Settings*

Setting	Description
Block application from being activated via a URL	If set, the user will be able to launch the application from the Start menu only after it's installed, not from the web browser.
Allow URL parameters to be passed to application	If set, this allows the application to receive URL information from the browser that launches it, such as query string arguments. You can retrieve the URI through the ApplicationDeployment class in the System.Deployment.Application namespace. Just use the ApplicationDeployment.CurrentDeployment.ActivationUri property.
Use application manifest for trust information	If set, you can re-sign the application manifest after you publish the application. Usually, you'll do this so you can use a certificate with your company name. This information will then appear in the trust message the user sees when installing the application.
Exclude deployment provider URL	If set, the application will automatically check its install location for updates. You can use this option if you don't know the exact deployment location but you still want to use ClickOnce automatic updating.
Create desktop shortcut	If set, the setup will create a desktop icon in addition to the Start menu icon.

The Last Word

This chapter gave a quick tour of the ClickOnce deployment model, which was introduced in .NET 2.0 and remains a good choice for deploying stand-alone WPF applications. As with XBAPs, ClickOnce entails certain compromises—for example, you need to accept that there are certain client configuration details you can't control. But now that most computers have web browsers that support ClickOnce, it's become a truly practical way to deploy applications that have modest setup requirements.

Index