msdn® training

Module 10: Deploying Applications

Contents

This course is based on the prerelease version (Beta 2) of Microsoft® Visual Studio® .NET Enterprise Edition. Content in the final release of the course may be different from the content included in this prerelease version. All labs in the course are to be completed with the Beta 2 version of Visual Studio .NET Enterprise Edition.

Microsoft®

Instructor Notes

Presentation:
75 Minutes

Labs:
45 Minutes

This module provides students with the skills necessary to deploy Microsoft® Visual Basic® .NET applications. They will learn what deployment choices are available to them and how to use the various deployment project templates to successfully deploy any type of application

After completing this module, students will be able to:

- Describe an assembly.
- List the different types of application deployment.
- Deploy a component assembly.
- Deploy an application based on Microsoft Windows®.
- Deploy a Web-based application.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2373A_10.ppt
- Module 10, "Deploying Applications"

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Read the instructor notes and the margin notes for the module.
- Practice the demonstration.
- Complete the labs.

Demonstration

This section provides demonstration procedures that do not fit in the margin notes or are not appropriate for the student notes.

Deploying a Web-Based Application

↙ To prepare for the demonstration

1. Open Visual Basic .NET, and then open Mod10.sln from *install folder*\DemoCode\Mod10 folder.
2. Quickly demonstrate the application. For an existing customer, use john@tailspintoys.msn.com with a password of **password**. For a new customer, use your own details.

   Students should be familiar with this application, because they created this application in an earlier lab.

↙ To create the deployment project

1. On the **File** menu, point to **Add Project**, and then click **New Project**.
2. In the Project Types pane, click **Setup and Deployment Projects**, and in the Templates pane, click **Web Setup Project**. Set the location to *install folder*\DemoCode\Mod10, and then click **OK**.
3. In the File System Editor, right-click the **Web Application Folder**, point to **Add**, and then click **Project Output**.
4. In the **Add Project Output Group** dialog box, select **Primary output** and **Content Files**, and then click **OK**.
5. In the File System Editor, select the **Web Application Folder**.
6. In the Properties window, change the **VirtualDirectory** property to **Demo10**.

↙ To specify launch conditions

1. Open the Launch Conditions Editor. Explain the existing conditions that verify that Internet Information Server (IIS) is present on the target server.
2. Right-click **Search Target Machine**, click **Add File Search**, and rename this **Check for Cargo Database**.
3. Change the properties of this search by using the information in the following table.

Property	Setting
Property	CARGOCHECK
Folder	[ProgramsFilesFolder]
FileName	Cargo.mdf
Depth	3

4. In the Launch Conditions Editor, right-click **Launch Conditions**, click **Add Launch Condition**, and then rename this **Cargo Condition**.
5. Change the properties of this condition by using the information in the following table.

Property	Setting
Condition	CARGOCHECK
Message	Cargo database is not installed

6. On the **Build** menu, click **Build WebSetup1**.

↙ To deploy and test the application

1. On the **Project** menu, click **Install**, and then wait for the deployment to finish.
2. Open Internet Services Manager, and check that a new application called Demo10 has been installed.
3. Open Internet Explorer, and browse to http://localhost/Demo10/DefaultPage.aspx. Show the students that this application works as expected.

Module Strategy

Use the following strategy to present this module:

- Describing Assemblies

  In this lesson, explain what assemblies are, how to create strong names, and how to version assemblies. You have discussed assemblies at various points throughout this course. Use this lesson to clarify any questions that have been raised throughout the previous modules.

  Students will have an opportunity to create a strong-named assembly in Lab10.1 later in this module.

- Selecting a Deployment Strategy

  In this lesson, explain to students the advantages gained by deploying applications in Visual Studio .NET. You will review the methods of deployment, from using XCOPY to creating a Windows Installer application. Be sure to stress that while XCOPY can simplify deployment, it cannot be used when registration of components is necessary. The Windows Installer is the best choice when deploying complex applications.

- Deploying Applications

  This lesson covers how to deploy both Windows-based and Web-based applications. Although these two types are quite disparate, the steps for deployment are actually very similar. You will review how to use the editor windows to customize your deployment, and you will show students how to use both the File System and Launch Conditions editors in the demonstration.

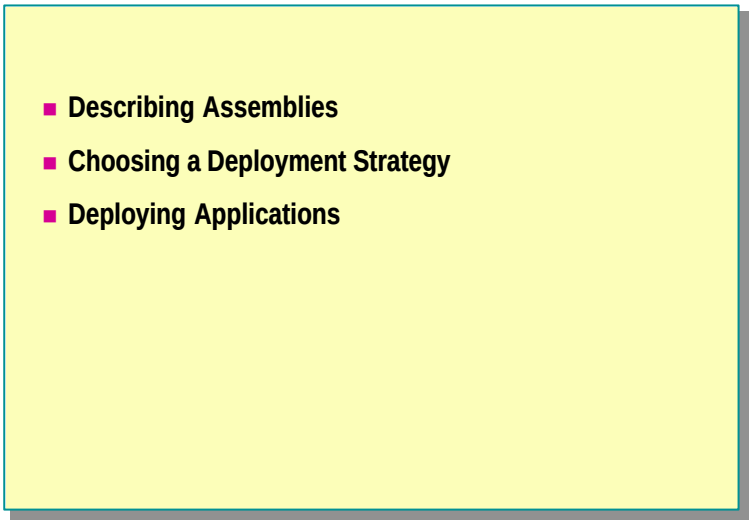
  Both of the labs for this module occur during this lesson. In the first one, students will package a component within a merge module, and in the second, they will include that in a setup project for a Windows-based application.

  You will also show the students how to deploy a Web-based application in the demonstration.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will learn about the options available to you when deploying Visual Basic .NET- based applications.



After you create and test an application, you will want to distribute it for use on other computers. The users may be end users running a Web application or an application based on Microsoft® Windows®, or other developers using a code library.

In this module, you will learn how to deploy assemblies for use by client applications, how to decide what type of distribution strategy to implement, and how to deploy Windows-based and Web-based applications.

After completing this module, you will be able to:

- Describe an assembly.
- List the different types of application deployment.
- Deploy a component assembly.
- Deploy an application based on Windows.
- Deploy a Web-based application.

◆ Describing Assemblies

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
This lesson examines assemblies and the specific requirements for creating components.

- **Assemblies Overview**
- **Benefits of Strong-Named Assemblies**
- **Creating Strong-Named Assemblies**
- **Versioning Strong-Named Assemblies**
- **Using the Global Assembly Cache**

In this lesson, you will learn about the role of assemblies in Microsoft Visual Basic® .NET version 7.0. You will learn about the benefits of strong-named assemblies and how to create them. Finally, you will learn how to version assemblies.

After completing this lesson, you will be able to:

- Describe the benefits of using strong-named assemblies.
- Create strong-named assemblies.
- Version assemblies.

Assemblies Overview

Topic Objective
To explain the function of assemblies.

Lead-in
An assemblies is the building block of a .NET-compatible application.

- **Contains Code, Resources, and Metadata**
- **Provides Security, Type, and Reference Scope**
- **Forms a Deployment Unit**
- **Versionable**
- **Side-by-Side Execution Allows Multiple Installed Versions**
- **Global Assembly Cache Allows Assembly Sharing**

An assembly is the building block of a Microsoft .NET-compatible application. It is a built, versioned, and deployed unit of functionality that can contain one or more files. An application can be composed of one or more assemblies.

You can think of an assembly as a collection of types and resources that form a logical unit of functionality and are built to work together. Using existing assemblies to add extra functionality to your application is similar to the way that you use Microsoft ActiveX® libraries in previous versions of Visual Basic. You also can create your own assemblies for other applications to use.

What makes assemblies different from .exe or .dll files in earlier versions of Windows is that they contain all the information you would find in a type library, in addition to information about everything else necessary to use the application or component.

Delivery Tip
Point out that all applications will also use the .NET Framework assemblies.

Assemblies Contain Code, Resources, and Metadata

An assembly contains:

- Intermediate language (IL) code to be executed
- Any required resources, such as pictures and assembly metadata, which exists in the form of the assembly manifest.
- Type metadata

  Type metadata provides information about available classes, interfaces, methods, and properties, similar to the way that a type library provides information about COM components.

An assembly can be grouped into a single portable executable (PE) file, such as an .exe or .dll file, or it can be made up of multiple PE files and external resource files, such as a bitmap.

The assembly manifest contains assembly metadata. It provides information about the assembly title, description, version information, and so on. It also provides information about linking to the other files in the assembly. This enables the assembly to be self describing, which allows you to distribute it using the XCOPY command. The information in the manifest is used at run time to resolve references and validate loaded assemblies.

The assembly manifest can be stored in a separate file but is usually compiled as part of one of the PE files.

Assemblies Provide Boundaries

Assemblies provide the following boundaries:

- Security boundary

  You set security permissions at an assembly level. You can use these permissions to request specific access to an application, such as file I/O permissions if the application must write to a disk. When the assembly is loaded at run–time, the permissions requested are entered into the security policy to determine if permissions can be granted.

- Type boundary

  An assembly provides a boundary for data types, because each type has the assembly name as part of its identity. As a result, two types can have the same name in different assemblies without any conflict.

- Reference scope boundary

  An assembly provides a reference scope boundary by using the assembly manifest for resolving type and resource requests. This metadata specifies which types and resources are exposed outside the assembly.

Assemblies Form a Deployment Unit

Assemblies are loaded by the client application when they are needed, allowing for a minimal download where appropriate.

Assemblies Are Versionable

An assembly is the smallest versionable unit in a .NET-compliant application. The assembly manifest describes the version information and any version dependencies specified for any dependent assemblies. You can only version assemblies that have a strong name.

Side-by-Side Execution Enables Multiple Installed Versions

Multiple versions of an assembly can run side-by-side simultaneously on the same computer or even in the same process. This ability greatly aids a client's compatibility with previous versions, because clients can specify which version they want to use regardless of how many new versions are deployed on the computer. This avoids the .dll conflicts that happen when a client application is expecting a particular version of an assembly but that version has been overwritten with an inappropriate version by another installation.

The Global Assembly Cache Enables Assembly Sharing

If an assembly is to be shared by several applications on a particular computer, you can install the assembly into the global assembly cache. Deploying assemblies into the cache can enhance performance because the operating system must only load one instance of the assembly. It also increases file security because only users with local Administrator privileges can delete assemblies in the global assembly cache.

Serviced component applications, such as COM+ applications, are often deployed into the global assembly cache so that all clients access only a single copy of the component assembly.

Note The .NET Framework assemblies are installed into the global assembly cache.

Benefits of Strong-Named Assemblies

Topic Objective
To explain the benefits of strong-named assemblies.

Lead-in
Strong-named assemblies provide enhanced features to...

- **Guaranteed Uniqueness**
  - No two strong names can be the same
- **Protected Version Lineage**
  - Only legitimate assembly versions can be loaded
- **Enforced Assembly Integrity**
  - Assemblies are tested for unauthorized modification before loading

---

You can use strong-named assemblies to ensure safe use of the components contained within the assembly. A strong-named assembly is a requirement for serviced components because only a single instance of the assembly is loaded regardless of the number of client applications.

Delivery Tip
You may need to elaborate on public and private key pairs.

Guaranteed Uniqueness

Strong names guarantee that an assembly name is unique and cannot be used by anyone else. You generate strong names through the use of public and private key pairs when the assembly is compiled.

Protected Version Lineage

By default, applications can only run with the version of the assembly that they were originally compiled with, unless a setting in a configuration file overrides it. If you want to update a component, you can use a publisher policy file to redirect an assembly binding request to the new version. This link ensures that a client application cannot use an incorrect component assembly unless the client application is explicitly recompiled.

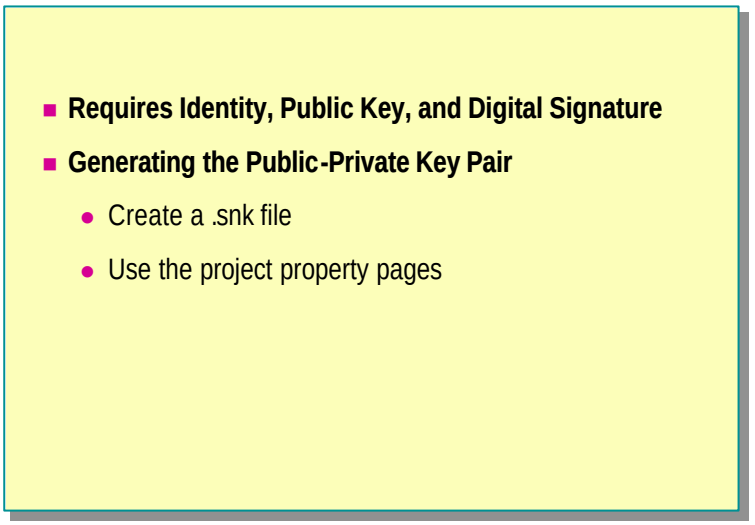
Enforced Assembly Integrity

The .NET Framework provides an integrity check that guarantees that strong-named assemblies have not been modified since they were built. This ensures that no unauthorized alterations can be made to the component assembly after the client application is compiled.

Creating Strong-Named Assemblies

Topic Objective
To explain the benefits of strong-named assemblies.

Lead-in
Strong-named assemblies provide enhanced features to.



The.NET Framework can create a strong-named assembly by combining the assembly identity (its name, version, and culture information), a public key, and a digital signature.

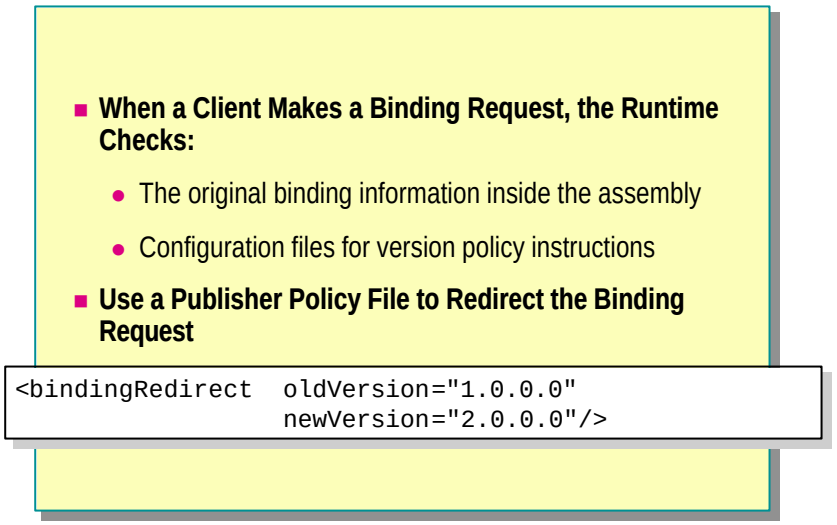
You must generate the strong name key file (.snk extension) that contains the public-private key pair before you build the assembly. You can do this manually by using the Strong Name tool (Sn.exe) utility or the Visual Basic .NET IDE.

In the property pages of the project, you can use the **Strong Name** section to automatically generate a strong name key file and add it to the project. The public key is inserted into the assembly manifest at compile time, and the private key is used to sign the assembly.

Versioning Strong-Named Assemblies

Topic Objective
To examine how to version assemblies.

Lead-in
Assemblies can be versioned by specifying a version number in the assembly manifest.



Often you will want to update a component without redeploying the client application that is using it. However, by default, an application only functions with the original component that it was compiled with. To overcome this behavior, you must ensure that your components have strong names, which enables you to version them at a later date.

When a client application makes a binding request, the runtime performs the following tasks:

- Checks the original assembly reference for the version to be bound
- Checks the configuration files for version policy instructions

You can use a publisher policy file to redirect a binding request to a newer instance of a component. The following example shows a publisher policy file. Note the **publicKeyToken** attribute, a hexadecimal value, which is used to identify the strong name of the assembly. This value can be obtained by using Sn.exe with the **–T** switch.

```
<configuration>
  <runtime>
      <assemblyBinding>
          <dependentAssembly>
              <assemblyIdentity name="myasm"
                  publicKeyToken="e9b4c4996039ede8"
                  culture="en-us"/>
              <bindingRedirect
                  oldVersion="1.0.0.0"
                  newVersion="2.0.0.0"/>
              <codeBase version="2.0.0.0"
                  href="http://www.Microsoft.com/Test.dll"/>
          </dependentAssembly>
      </assemblyBinding>
  </runtime>
</configuration>
```

You can compile this XML file into a publisher policy assembly, to be shipped with the new component, using the Assembly Generation tool (Al.exe). This signs the assembly with the strong name originally used, which confirms to the user that the updated component comes from a valid source.

Using the Global Assembly Cache

Topic Objective
To explain why to install assemblies into the global cache.

Lead-in
The global assembly cache offers many advantages for the administration and usage of your components.

- **Performance**
  - Quicker binding
  - Only one instance ever loaded
- **Shared Location**
  - Can use machine configuration file to redirect bindings
- **File Security**
  - Only administrators can delete files
- **Side-by-side Versioning**
  - Can install multiple copies using different version information

---

You can store your shared components inside the global assembly cache. To do this, you must create a strong name for the assembly, and when deploying the component, you must specify that it is stored in the global assembly cache, as opposed to the common files folder for the client application. Using the global assembly cache has the following benefits.

Performance

If the component is stored in the cache, the strong name does not need to be verified each time the component is loaded. This method also guarantees that only one instance of the component is loaded in memory, reducing the overhead on the target computer.

Shared Location

You can use the computer configuration file to redirect all bindings to the global assembly cache, providing simpler administration of assemblies.

File Security

Only users with administrative privileges can delete files from the cache.

Side-by-Side Versioning

You can install multiple copies of the same component, with different version information, into the cache.

◆ Choosing a Deployment Strategy

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
Applications can be deployed in a variety of ways.

- **Deployment Overview**
- **Copying Projects**
- **Deploying Projects**
- **Types of Deployment Projects**

There are a variety of options available when deploying Visual Basic.NET–based applications. Choosing what option to use depends on the type of application that you are deploying and the version of Windows you are deploying it to.

Before you can begin the distribution process, you must understand the differences in the various strategies available to you. Some of these simply involve copying the application to an appropriate place, some involve creating a deployment project to copy the application and register any components, and some involve creating a complete setup application for use by the end user.

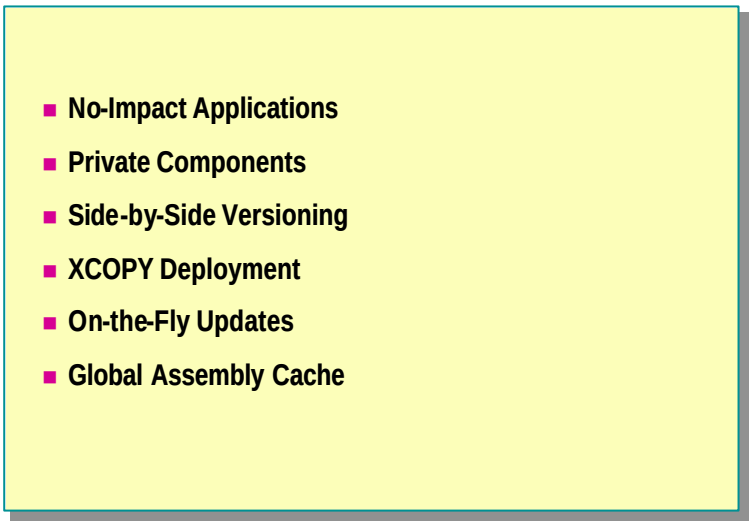
After completing this lesson, you will be able to:

- Describe the deployment options available to you.
- Match deployment options with specific scenarios.

Deployment Overview

Topic Objective
To discuss the advantages that Visual Studio .NET deployment offers over deployment in earlier versions.

Lead-in
The new deployment options in Visual Studio .NET offer distinct advantages over the options in the Package and Deployment Wizard in Visual Basic 6.



Application developers have traditionally faced many issues when deploying their applications. Use of the deployment features in Microsoft Visual Studio® .NET alleviates some of these issues.

The following table lists some of the advantages of using Visual Studio .NET deployment.

Feature	Description
No-impact applications	All applications are isolated, which results in fewer .dll conflicts.
Private components	By default, components are installed into the application directory. Therefore, you can only use it in that application.
Side-by-side versioning	You can have more than one copy of a component on a computer, which can prevent versioning problems.
XCOPY deployment	Self-describing components can just be copied to the target computer.
On-the-fly updates	.dlls are not locked when in use and can be updated by an administrator without stopping the application.
Global assembly cache	You can share assemblies between applications by installing them into the global assembly cache. This can also increase performance because only one copy of the assembly is loaded.

Copying Projects

Topic Objective
To discuss the simplest method of distributing an application.

Lead-in
Distributing applications can be as simple as using the MS-DOS XCOPY command. However, this method does have certain restrictions.

- **Copying a Project**
  - There is an extra menu command for Web applications
  - You can copy a project directly to a Web server
- **Using the XCOPY Command**
  - Use the DOS command
  - You can use it for any type of application

Deploying simple applications with no dependencies can be as easy as copying the application to the target computer. Using this method can be quick, although it does not take advantage of the all the new features available in Visual Studio .NET deployment.

Copying a Project

When you are working with a Web application, you have an extra menu item available, **Copy Project**, that allows you to copy the project directly to a Web server. You can specify the access method, for example with Microsoft FrontPage® Server extensions, and whether to copy just the necessary application files, the entire project, or all files in the project directory.

Consider the following facts when using the **Copy Project** command:

- Assemblies are not registered for unmanaged client access.
- The locations of assemblies are not verified.

Using the XCOPY Command

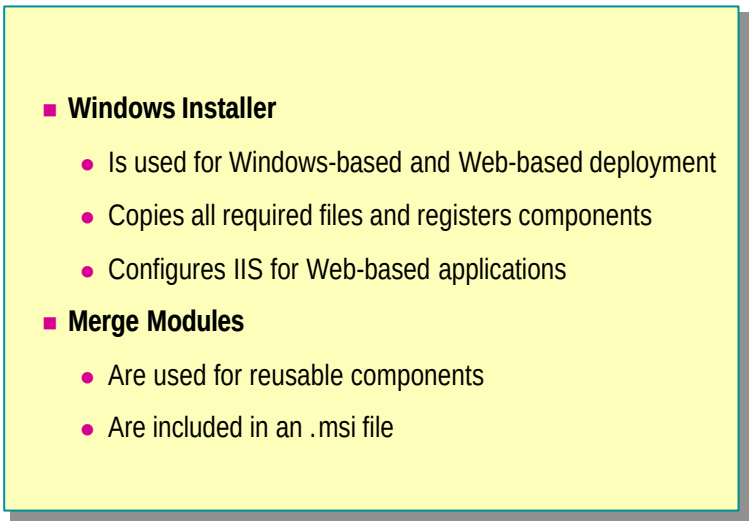
You can use the Microsoft MS-DOS® XCOPY command to deploy any type of application. Consider the following facts when deploying with the XCOPY command:

- Assemblies are not registered for unmanaged client access.
- The locations of assemblies are not verified.
- Project-to-project references are not copied.
- Internet Information Server (IIS) is not configured for Web applications.
- You cannot take advantage of the Zero Administration initiative for Windows feature in Microsoft Windows Installer.

Deploying Projects

Topic Objective
To provide an overview of the deployment options available.

Lead-in
The Windows Installer contains many features that enhance the deployment of your applications.



In general, you will create applications that have dependencies on other assemblies or components. In this situation, you must create a deployment package to ensure that the external references are correctly registered and located.

Windows Installer

You can use the Windows Installer to package all your data and installation instructions in one file, an .msi file, for easy distribution. Using the Windows Installer provides the following advantages:

- Support for the Zero Administration initiative for Windows

  This helps overcome the problems of overwriting shared components.
- Safe uninstall options

  Windows Installer provides an uninstall program that detects shared components and does not remove them.
- Rollback

  If the install fails before it is complete, for example, if the connection to the network share containing the source files is lost, then the Windows Installer will return the computer to its original state.

You can use the Windows Installer to package both Windows-based and Web-based applications.

Merge Modules

You can use merge module projects to package shared components that will be used by more than one application on the target computer. You can then incorporate these modules into .msi packages whenever that component is used in a solution. Using merge modules has the following advantages:

- Eliminates versioning problems
- Captures the dependencies of the component
- Creates reusable setup code

Types of Deployment Projects

Topic Objective
To discuss the types of deployment projects available in Visual Studio .NET.

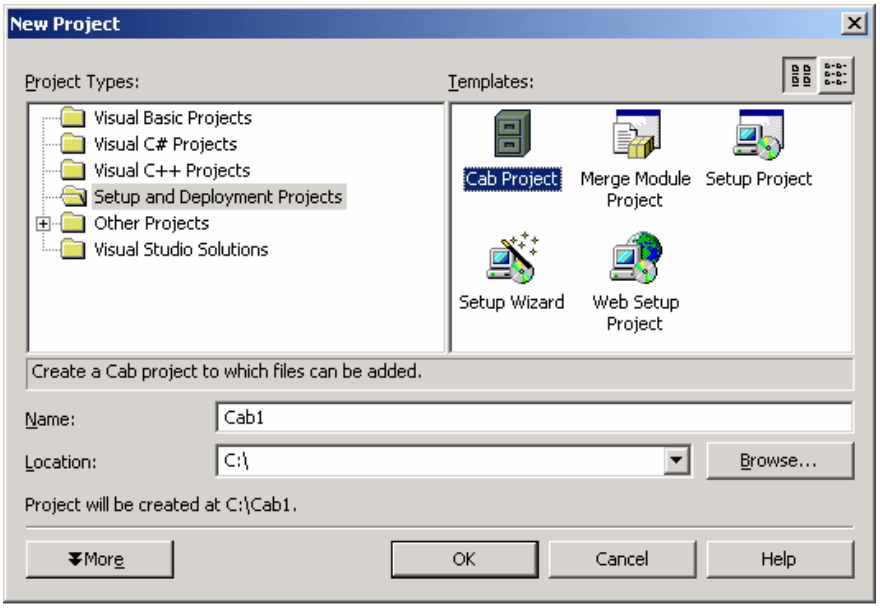
Lead-in
Choosing the correct type of deployment project is imperative.

- **Cab Project - For Downloading from Web Server to Legacy Browser**
- **Merge Module Project - For Shared Components**
- **Setup Project - For Windows-Based Applications**
- **Setup Wizard - To Walk Through Deployment Project Creation**
- **Web Setup Project - For Web-Based Applications**

There are five options available to you when creating a deployment project in Visual Studio .NET. The following table lists the types of projects and their uses.

Project type	Use
Cab project	Use this to create compressed CAB files for downloading from a Web server.
Merge module project	Use this to create a setup for a shared component.
Setup project	Use this to create a setup for a Windows-based application.
Setup Wizard	Use this to initiate the Setup Wizard that leads you through the steps of creating one of the four main deployment projects.
Web setup project	Use this to create a setup for a Web-based application.

To create a new deployment project, click **Setup and Deployment Projects** in the **New Project** dialog box, as shown below:



Deploying Applications

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
Deploying Windows-based and Web-based applications is actually a very similar process.

- **Creating a Merge Module Project**
- **Creating a Setup Project**
- **Using the Editors**
- **Creating Installation Components**
- **Deploying the Application**

To deploy applications, you need to create a setup project with all your installation preferences and build the project for distribution.

After completing this lesson, you will be able to:

- Describe the two types of setup projects used for Windows-based and Web-based applications.
- Configure your installation process by using the Visual Studio .NET editors.

Creating a Merge Module Project

Topic Objective
To explain how to create merge module projects for packaging components for distribution.

Lead-in
After you create the strong-named assembly, you need to package it within a merge module for further deployment with the client application.

- **Never Installed Directly - Included in Client Application Deployment Project**
- **Contains**
  - DLL
  - Any dependencies and resources
  - Information for the Windows Installer
- **Store One Component Only in One Merge Module**
- **If Component Changes, Create a New Merge Module**

After you create the strong-named assembly, you need to package it within a merge module for it to be included in the deployment project for a client application. The merge module provides you with a standard way of distributing components and ensuring that the correct version is installed.

The merge module is never installed directly, but it is distributed within a Windows Installer project. It includes the .dll file, any dependencies, any resources, and any setup logic. This method of deployment ensures that whenever this shared component is used, it is installed on the target computer in the same way. It also contains information that the Windows Installer database uses to determine when you can safely remove a component during application removal.

To package a component assembly, complete the following steps:

1. Create your component and build the .dll.
2. Add a merge module project to your solution.
3. Add the component to the Common Files folder or the global assembly cache.
4. Build the merge module project.
5. Add the merge module project to a Windows-based or Web-based setup project.

Lab 10.1: Packaging a Component Assembly

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will create a public-private key pair for an assembly, and then create a merge module project that will contain the assembly.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to create a merge module project.

Prerequisites

Before working on this lab, you must have:

- Knowledge of the merge module project template.
- Knowledge of the deployment editors.
- Knowledge of the setup project template for Windows-based applications.

Scenario

In this lab, you will create a merge module project to package a component assembly.

Estimated time to complete this lab: 15 minutes

Exercise 1
Packaging the Component Assembly

In this exercise, you will create a strong name for a component assembly and then package the component in a merge module project that is ready for deployment with the client application.

↙ To attach the key pair to the assembly

1. Open Visual Studio .NET.
2. On the **File** menu, point to **Open**, and then click **Project**.
3. Browse to the *install folder*\Labs\Lab101\Starter\Component folder, click **Component.sln**, and then click **Open**.
4. In Solution Explorer, right-click **Component**, and then click **Properties**.
5. Under **Strong Name**, click the **Generate strong name using** check box.
6. Click **Generate Key**, and then click **OK**.
7. Build the component.

↙ To create the merge module project

1. In Visual Studio .NET, on the **File** menu, point to **Add Project**, and then click **New Project**.
2. In the Project Types pane, click **Setup and Deployment Projects**.
3. In the Templates pane, click **Merge Module Project**. Set the location to *install folder*\Labs\Lab101\Starter, and then click **OK**.
4. In the File System Editor, right-click the **Common Files Folder**, point to **Add**, and then click **Project Output**.
5. In the **Add Project Output Group** dialog box, select **Primary output** and **Content Files**, and then click **OK**.
6. On the **Build** menu, click **Build MergeModule1**.
7. Save all files, and close Visual Studio .NET.

↙ To verify that the package has been created

- Open Windows Explorer, browse to the *install folder*\Labs\Lab101\Starter\MergeModule1\Debug folder, and verify that MergeModule1.msm has been created.

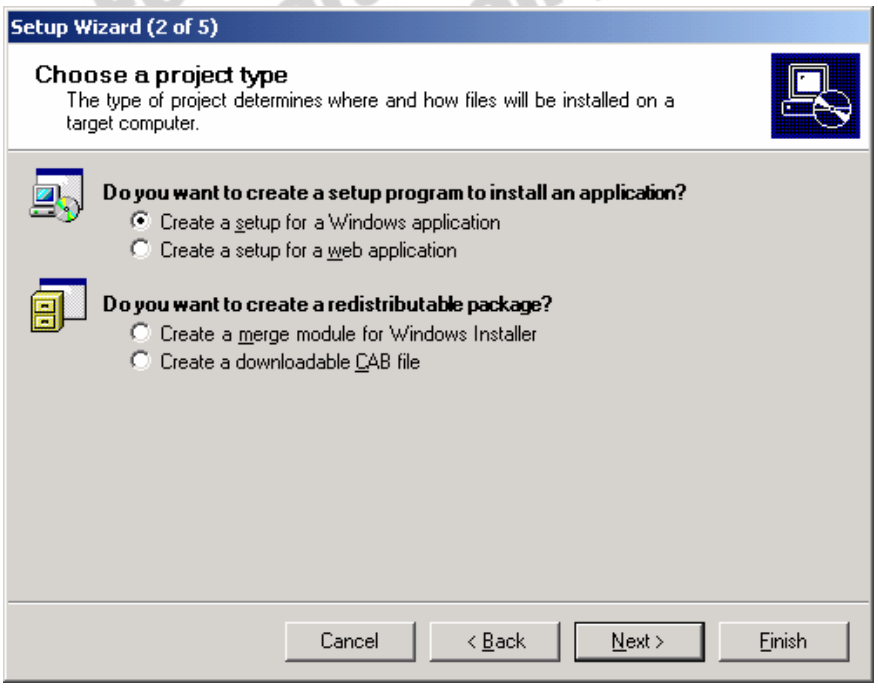
Creating a Setup Project

Topic Objective
To explain how to create a setup project.

Lead-in
You can use the **New Project** dialog box to select a project template for your setup project.

- **Creates a Blank Setup Project with Relevant Folders in File System Editor**
- **Windows-Based Application**
  - Application Folder
  - Global Assembly Cache
  - User's Desktop
  - User's Programs Menu
- **Web-Based Application**
  - Global Assembly Cache
  - Web Application Folder

You can use the Setup Project and Web Setup Project templates to create Windows Installer packages for Windows-based and Web-based applications, respectively. You specify which of these project types you want to use when you create a new project in Visual Studio .NET. You can also use the Setup Wizard to lead you through the process of gathering the necessary information resulting in the required project type.



Both project types start in the File System Editor window, which you use to specify where to install the included files on the target computer. You can allocate files, folders, shortcuts, and components to these folders. For example, you can include a ReadMe.htm or a merge module project containing a shared component in these folders. In addition to using the default folders, you can also use a predetermined set of special folders (for example the Windows folder), and you can also create your own subfolders.

Windows-Based Setup Project

When you create a setup project for a Windows-based application, you are given a set of default folders.

Folder name	Description
Application Folder	Where the user specifies that the application is to be installed on the target computer, the application is installed in the Program Files*Manufacturer**ProductName* folder by default. Use this folder for the standard files used to run the application.
Global Assembly Cache Folder	The cache used for shared components.
User's Desktop	Use this folder to create desktop shortcuts for the application. When the user installs the application, they can choose whether this application is for all users or just themselves, which determines where this folder is located.
User's Programs Menu	Use this folder to create **Start** menu shortcuts for the application. When users install this application, they can choose whether this application is for all users or just themselves, which determines where this folder is located.

Web Setup Project

A Web setup project also presents you with a set of folders, but these are different because of the differences in the base project type.

Folder name	Description
Global Assembly Cache Folder	The cache used for shared components
Web Application Folder	Use this to place files in the default folder for the Web application. By default this will be http://*ComputerName*/*ProductName* .

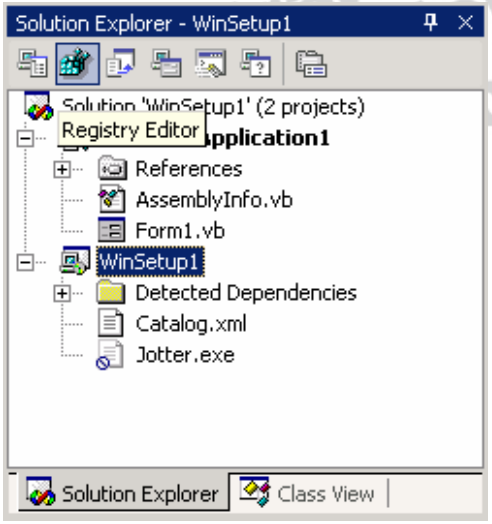
Using the Editors

Topic Objective
To discuss the different editors included in the setup projects for Windows-based and Web applications.

Lead-in
In addition to the File System Editor, there are other editors that allow you to further customize the setup of your application.

- **Registry**
- **File Types**
- **User Interface**
- **Custom Actions**
- **Launch Conditions**

In addition to the File System Editor, you can use a range of other editors to further define your setting for the installation process. You can access these editors by using the toolbar buttons in Solution Explorer, as shown below:



Registry

This gives you access to the commonly used registry hives and keys, such as **HKEY_CURRENT_USER\Software** and **HKEY_LOCAL_MACHINE\Software**. These vary according to whether your application is a Windows-based or Web-based application. In this editor, you can define your own keys and write their default values during the installation process.

File Types

This editor allows you to define new file types to be configured on the target computer and the actions associated with those types.

User Interface

This editor lists the windows in the Installation Wizard that the user sees and allows you to customize the messages and images displayed in them. You can customize both the standard and administrative installation programs.

You can also add extra dialog boxes to the installation process. For example, you can request user preferences with text boxes and option buttons, request user information for registration purposes, or display license agreements.

Custom Actions

This allows you to include custom actions within your main setup program. These can be actions performed at install, commit, rollback or uninstall time. They can include running any executable file, .dll, or script file; adding users to or removing users from a database; or adding a support contact to the address book in Microsoft Outlook®.

Launch Conditions

This editor allows you to define conditions for installing the application or performing custom actions. For example, if a database is not present on the server, you will not want to add users to it and may not want to install the application. You can check for files, registry keys, and Windows Installer installations. You can also customize the message given to the user if the condition is not satisfied.

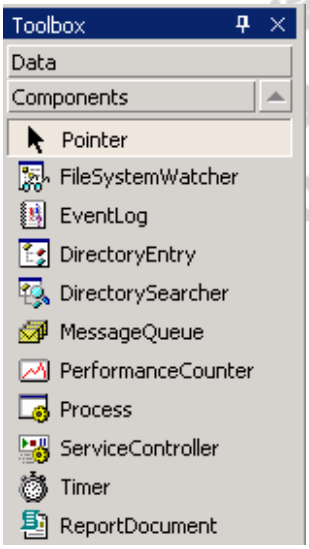
Creating Installation Components

Topic Objective
To explain how to create installation components.

Lead-in
You may also need to include Windows resources within your setup.

- **EventLog**
- **MessageQueue**
- **PerformanceCounter**
- **Service**
- **ServiceProcess**

When you are developing an application, you often use Windows resources such as event logs and message queues. These types of objects are available to you on the **Components** tab of the Visual Basic .NET toolbox.



The Visual Studio .NET installation process allows you to create these components on the target computer as part of your application installation process. You accomplish this by using installation components.

You can set the properties of any component. These properties include elements such as the name of an existing message queue or name of the log. When you want to deploy your application, you can create ProjectInstaller files that copy all the settings for your component and recreate it on the target computer at installation time.

Deploying the Application

Topic Objective
To learn how to deploy the application after the setup project is complete.

Lead-in
After you configure your setup using the editors, you are ready to compile your setup project and deploy the application.

- **Windows-Based Setup Project**
  - Copies all files
  - Registers components
  - Performs other installation tasks
- **Web Setup Project**
  - Copies all files
  - Registers components
  - Creates a Web application

After you configure settings using the editors in Visual Studio .NET for your custom setup program, you can build the project ready for deployment. Because this is a standard Visual Studio .NET project, you can build or deploy the project in the usual ways, although building a Visual Basic solution will not build any setup projects included in it.

Building the project creates an .msi file that you distribute to users so that they can run the setup program for your application. To install the application, you can run the .msi file on the target computer or click **Install** on the **Project** menu.

Windows-Based Setup Project

Installing a Windows-based application copies all the specified files to the appropriate locations on the target computer, creates any shortcuts that are specified, adds registry entries, creates file types, and creates any installation components included in the project.

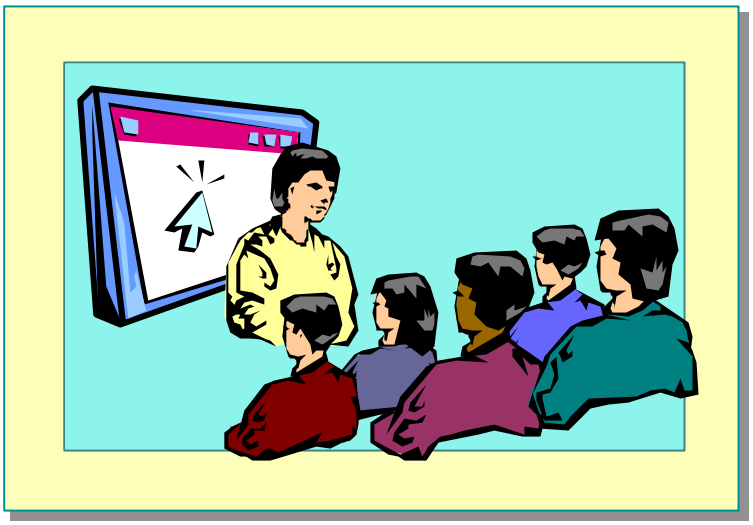
Web Setup Project

When you install a Web project, all the actions performed are the same as those performed during the installation of a Windows-based project. The Web application is also created and configured within IIS.

Demonstration: Deploying a Web-Based Application

Topic Objective
To demonstrate how to deploy a Web-based application.

Lead-in
This demonstration shows how to deploy a Web-based application.



Delivery Tip
The step-by-step instructions for this demonstration are in the instructor notes for this module.

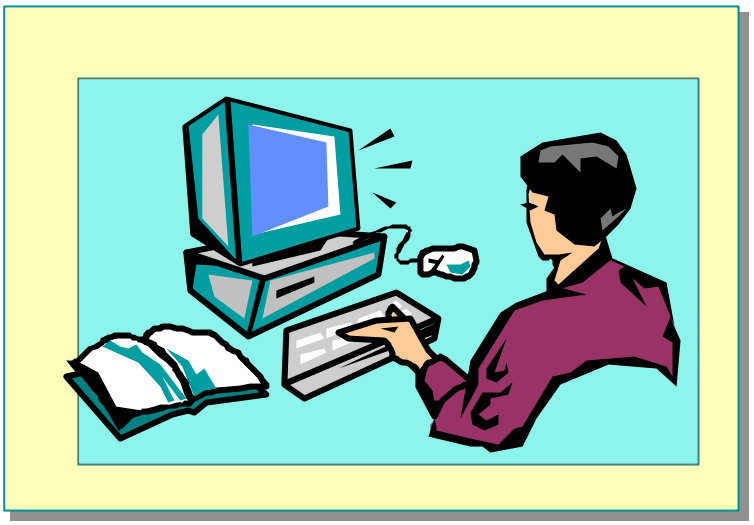
In this demonstration, you will learn how to use the Setup Wizard to create a deployment project for a Web-based application. You will also learn how to use the Launch Conditions Editor to verify that a database is present on the target computer before installing the application.

Delivery Tip
Ensure that students understand that this is a simple application created for demonstration purposes. It is the solution to Lab 7.1 of Course 2373A, *Programming with Microsoft Visual Basic .NET (Prerelease)*.

Lab 10.2: Deploying a Windows-Based Application

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will create a setup project for a Windows-based application and test the resulting Windows Installer file.



Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Create a Windows Installer project.
- Deploy a Windows-based application.

Prerequisites

Before working on this lab, you must have:

- Completed Lab 10.1.
- Knowledge of the deployment editors.
- Knowledge of the setup project template for Windows-based applications.

Scenario

In this lab, you will deploy a Windows-based application. You will begin by creating a Windows Installer package that includes the merge module that you created in the previous lab and the client application. Then, you will deploy the application and ensure that it installs all sections successfully.

Estimated time to complete this lab: 45 minutes

Exercise 1
Creating a Windows Installer Project

In this exercise, you will create a Windows Installer project for a client application. This will include the merge module that you created in the previous lab. You will create shortcuts for the application on the desktop and **Programs** menu and include a ReadMe file in the distribution.

↙ To reference the component

1. Open Visual Studio .NET.
2. On the **File** menu, point to **Open**, and then click **Project**.
3. Browse to the *install folder*\Labs\Lab102\Starter\Customers folder, click **Customers.sln**, and then click **Open**.
4. In Solution Explorer, right-click **Customers**, and then click **Add Reference**.
5. Click **Browse**, browse to the *install folder*\Labs\Lab102\Starter\Component\bin folder, click **Component.dll**, click **Open**, and then click **OK**.
6. View the properties of this component and verify that it is a strong-named assembly.
7. Run the application to test that it functions correctly.

↙ To create a Windows Installer project

1. On the **File** menu, point to **Add Project**, and then click **New Project**.
2. In the Project Types pane, click **Setup and Deployment Projects**.
3. In the Templates pane, click **Setup Project**. Set the location to *install folder*\Labs\Lab102\Starter, and then click **OK**.
4. In Solution Explorer, right-click **Setup1**, point to **Add**, and then click **Merge Module**.
5. Browse to the *install folder*\Labs\Lab102\Starter\MergeModule1\Debug folder, click **MergeModule1.msm**, and then click **Open**.
6. In the File System Editor, open the Application Folder. Right-click the folder, point to **Add**, and then click **File**. Browse to the *install folder*\Labs\Lab102\Starter\Customers\bin folder, click **Customers.exe**, and then click **Open**.

↙ To customize the installation

1. In the File System Editor, open the User's Desktop folder. Right-click in the right hand pane and click **Create New Shortcut**.
2. In the **Select Item in Project** dialog box, open the Application Folder, click **Customers.exe**, and then click **OK**.
3. In the Properties window, change the name of the shortcut to Customers.
4. Use the same method to create a shortcut in the User's Programs Menu.
5. Right-click the **Application Folder**, point to **Add**, and then click **File**.
6. Browse to the *install folder*\Labs\Lab102\Starter folder, click **ReadMe.rtf**, and then click **Open**.
7. On the Solution Explorer toolbar, click **User Interface Editor**.
8. Under **Install**, right-click **Start**, and then click **Add Dialog**.
9. In the **Add Dialog** dialog box, click **Read Me**, and then click **OK**.
10. Select the new **Read Me** dialog box in the editor.
11. In the Properties window, in the **ReadMeFile** property drop-down, click **(Browse...)**. In the **Select Item in Project** dialog box, open the **Application Folder**, click **ReadMe.rtf**, and then click **OK**.

↙ To build the project

1. On the **File** menu, click **Save All**.
2. On the **Build** menu, click **Build Setup1**.
3. When the project is successfully built, quit Visual Studio .NET.

Exercise 2
Running the Installation

In this exercise, you will run the Windows Installer project that you created in the previous exercise and verify that it installs correctly.

↙ To run the installation program

1. Open Windows Explorer.
2. Browse to the *install folder*\Labs\Lab102\Starter\Setup1\Debug folder, and then double-click **Setup1.msi**.
3. Follow the setup program, accepting the default options.

↙ To verify the installation

1. Minimize all windows, and check that there is a shortcut to your application on the desktop.
2. On the **Programs** menu, click **Customers**, and verify that the application functions correctly.
3. Quit the application.

↙ To remove the application

1. Open Control Panel, and then double-click **Add/Remove Programs**.
2. Click **Setup1**, and then click **Remove**. Confirm that you want to remove the application.
3. After the removal is complete, verify that the shortcuts and application folder have been removed.

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Describing Assemblies
- Choosing a Deployment Strategy
- Deploying Applications

---

1. Name the four ways of distributing a Visual Studio .NET project and describe what each is used for.

   XCOPY deployment – for simple stand-alone applications

   Copy Project deployment—for copying a Web project directly to the Web server

   Windows Installer—for deploying Windows-based and Web-based applications

   Merge Module—for packaging reusable, shared components

2. How do you create a strong-named assembly?

   Use Sn.exe to create a public-private key pair and apply it to the assembly through the project property pages or use the Generate Key button in the project property pages.

3. Describe the use of the Launch Conditions Editor.

   The Launch Conditions Editor allows you to define certain conditions under which the installation of an application fails, for example, the absence of a required database on a server.