msdn® training

Module 11: Application Settings and Deployment

Contents

Overview	1
Lesson: Working with Application Settings	2
Lesson: Deploying Applications	18
Review	28
Lab 11.1: Deploying an Application	29
Lab 11.2 (optional): Working with Application Settings	34

Microsoft®

Instructor Notes

Presentation:
60 minutes

This module provides students with an introduction to working with application settings and deployment.

Lab:
60 minutes

Important This module includes a Guided Practice. Each practice or the Lab may also include optional tasks to accommodate advanced learners.

This module also provides two labs from which you or your students may choose one to complete.

After completing this module, students will be able to:

- Work with application settings.
- Deploy an application.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2609A_11.ppt
- Module 11: Application Settings and Deployment

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the labs.

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Working with Application Settings

This section describes the instructional methods for teaching this lesson.

- It is recommended that you do not spend more than the allotted time on this module.
- This lesson is meant to provide students with a brief overview of the topics.
- You can begin by asking students, "What are user preferences?" and then state that these default settings must be stored somewhere.

Lesson: Deploying Applications

- The lesson on deployment is intended to show how easy it is to deploy applications by using the setup and deployment project templates in Microsoft® Visual Studio® .NET.
- It is recommended that you keep your discussion brief on this topic.

Review

The review questions are based mostly on conceptual understanding and procedures that were covered thoroughly in the module. You can use a discussion format to answer the questions so that everyone gets the benefit of knowing the right answers.

Lab 11.1: Deploying an Application

There are two labs provided for this module. The students may choose either one to complete. Lab 11.2, Working with User Preferences in the Registry, is the more challenging of the two.

Before beginning this lab, students should have completed all of the practices and answered the review questions. Students will need to be able to perform most of the tasks that they learned in the lessons and the practices. The lab is simple but comprehensive. It leads students through the process of deployment as described in the lessons of this module.

Lab 11.2 (optional): Working with User Preferences in the Registry

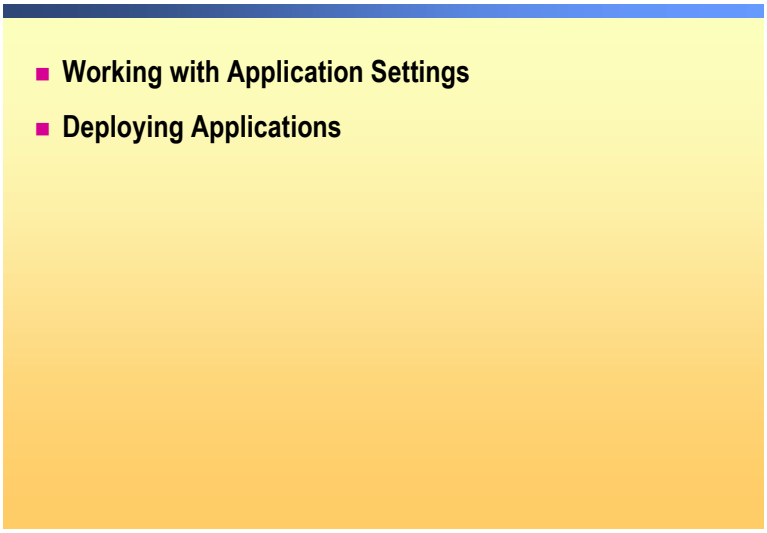
Before beginning this lab, students should have completed all of the practices and answered the review questions. Students will need to be able to perform most of the tasks that they learned in the lessons and the practices. The lab is simple but comprehensive. It leads students through the entire process of working with setting user preferences in the Registry as described in the lessons of this module.

Customization Information

This section identifies the lab setup requirements for a module and the configuration changes that occur on student computers during the labs. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The labs in this module are dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Automated Classroom Setup Guide* for Course 2609, *Introduction to C# Programming with Microsoft .NET*.

Overview



Introduction

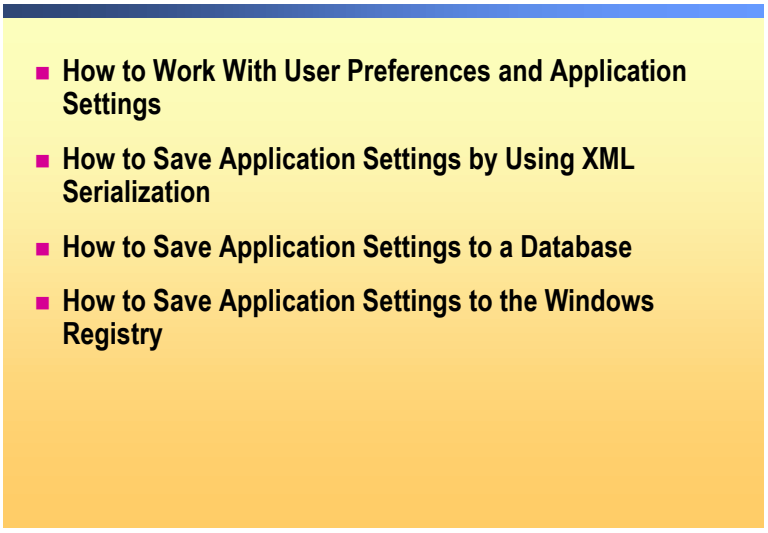
This module describes how to store user preferences and configure application settings. It also introduces the procedures that are involved in deploying a C# application by using Microsoft® Visual Studio® .NET. It explains how to deploy both Web-based applications and applications that are based on Microsoft Windows®.

Objectives

After completing this module, you will be able to:

- Work with application settings.
- Deploy an application.

Lesson: Working with Application Settings



Introduction

This lesson explains how to manage application configuration in a Windows-based application.

Lesson objectives

After completing this lesson, you will be able to store application settings by:

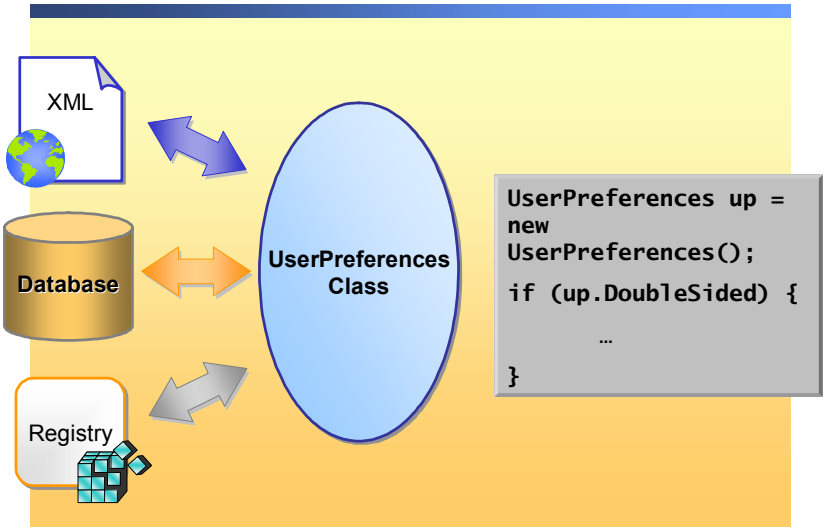
- Using XML serialization.
- Using a database.
- Using the Windows registry.

Lesson agenda

This lesson includes the following topics:

- How to Work with User Preferences and Application Settings
- How to Save Application Settings by Using XML Serialization
- How to Save Application Settings to a Database
- How to Save Application Settings to the Windows Registry
- Practice: Using the Windows Registry

How to Work with User Preferences and Application Settings



Introduction

When developing an application, you may establish application or user settings that you want to be persisted between sessions of the application. For example, you may want to store a color scheme or maintain the size of a window. These preferences must be stored somewhere so that they can be used the next time the application is run.

Options

The Microsoft .NET Framework provides several mechanisms for storing information about an application, its users, and default system settings. Depending upon the type of application that you are deploying, there are three primary ways that you can save settings and preferences:

- XML files

  For a Web application, an XML file may be a good choice for persisting information. XML files are also useful for persisting settings while an application is offline and therefore unable to connect to and set or retrieve settings from a database or other storage mechanism.

- Database files

  If your application depends upon data that is stored in a database, to the point where it will not function if a connection to the database is not established, then storing user preferences in the database is an appropriate decision.

- Windows registry

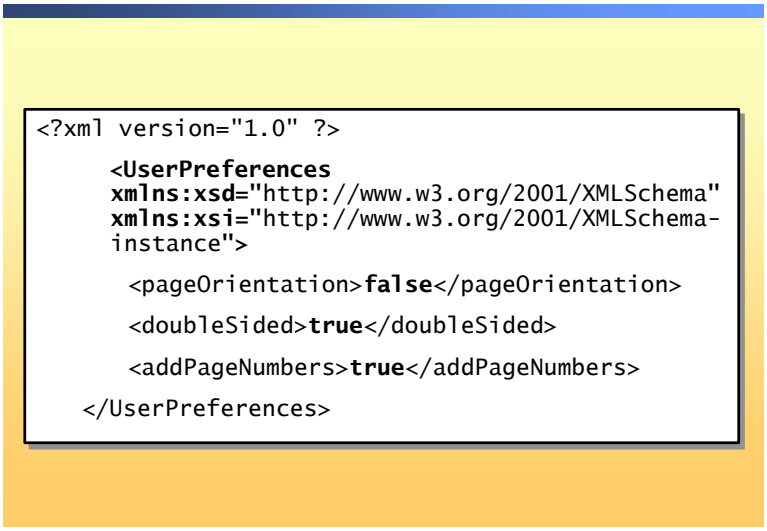
  For a Windows-based application, storing application settings in the Windows registry is often the preferred method.

An object-oriented solution to storing preferences

Regardless of where user preferences and application settings are stored, it is recommended that you create a class to access and manipulate this data. Using an object-oriented approach provides many benefits:

- *Encapsulation.* Because objects encapsulate both data and functionality together, only the data and methods that a user must access are made public, and not all the internal information about the object.
- *Scalability.* There is no limitation to the number and type of user preferences that you can save.
- *Portability*. If the application must store user preferences somewhere other than its current location, you can update the **UserPreferences** object without affecting any other code.

How to Save Application Settings by Using XML Serialization



Introduction

The .NET Framework provides classes that you can use to serialize an object. When you store your application or user settings in a class, serializing the class is straightforward.

Definition

XML serialization is the process of converting the public properties and fields of an object to an XML serial format for storage or transport. *Deserialization* re-creates the object in its original state from the XML output.

Note For more information about XML serialization, see Module 6, "Building .NET-based Applications with C#," in Course 2069, *Introduction to C# Programming with Microsoft .NET*.

Syntax

To serialize an object to a file, you must instantiate a new **XmlSerializer** object and then call the **Serialize** method, as shown in the following code:

```
MySerializableClass myObject = new MySerializableClass( );
XmlSerializer mySerializer = new
  XmlSerializer(typeof(MySerializableClass));
StreamWriter myWriter = new StreamWriter("myFileName.xml");
mySerializer.Serialize(myWriter, myObject);
```

To deserialize an object, you call the **Deserialize** method.

Example

The following code shows a class named **UserPreferences** that contains three properties, **pageOrientation**, **doubleSided**, and **addPageNumbers**. An instance of the class is created by calling the **UserPreferences.Load** static method. This method returns an instance of the class with the settings loaded from an XML file. The name of the XML file is the user domain and user name, which are available through the **System.Environment.UserDomainName** and **System.Environment.UserName** static properties.

```
public class UserPreferences {
  private bool pageorientation;
  private bool doublesided;
  private bool addpagenumbers;

  public bool pageOrientation {
      get {
          return pageorientation;
      }
      set {
          pageorientation=value;
      }
  }

  public bool doubleSided {
      get {
          return doublesided;
      }
      set {
          doublesided=value;
      }
  }

  public bool addPageNumbers {
      get{
          return addpagenumbers;
      }
      set {
          addpagenumbers=value;
      }
  }

  public UserPreferences() {
  }

  public static UserPreferences Load() {
      UserPreferences up;
      XmlSerializer myXmlSerializer = new
           XmlSerializer(typeof(UserPreferences));
      string filename = System.Environment.CurrentDirectory+
          "\\"+System.Environment.UserDomainName+
          System.Environment.UserName+".xml";
      if (File.Exists(filename)) {
          FileStream fs = new
              FileStream(filename,FileMode.Open);
          up=(UserPreferences)myXmlSerializer.Deserialize(fs);
          fs.Close();
      }
      else {
          up = new UserPreferences();
      }
      return up;
  }
```

Code continued on the following page.

```
  public void Save() {
      XmlSerializer myXmlSerializer = new
          XmlSerializer(typeof(UserPreferences));
      string filename =
          System.Environment.CurrentDirectory+
          "\\"+System.Environment.UserDomainName+
      System.Environment.UserName+".xml";
      FileStream fs = new FileStream(filename,
          FileMode.Create);
      myXmlSerializer.Serialize(fs,this);
      fs.Close();
  }
}
```

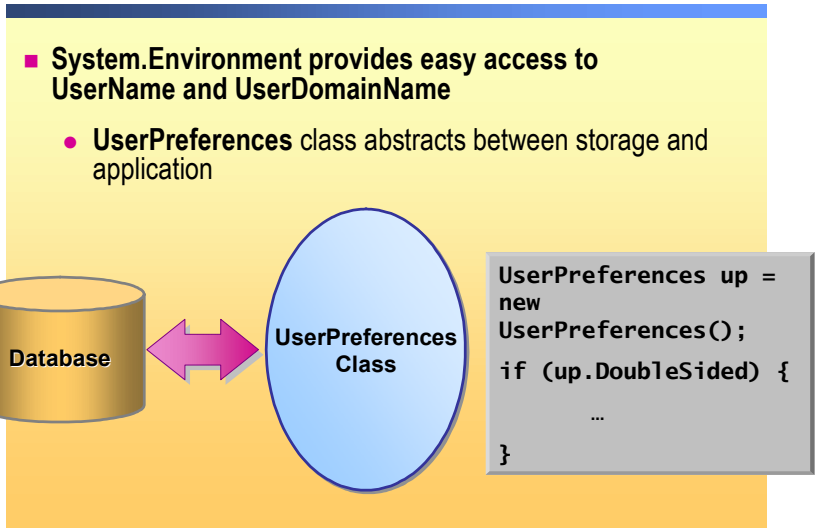
The **UserPreferences** object is created for the current user by using the static method of the **UserPreferences** class, as shown in the following code:

```
UserPreferences up = UserPreferences.Load();
```

The following XML code is created:

```
<?xml version="1.0" ?>
- <UserPreferences
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <pageOrientation>false</pageOrientation>
          <doubleSided>true</doubleSided>
          <addPageNumbers>true</addPageNumbers>
</UserPreferences>
```

How to Save Application Settings to a Database



Introduction

If the application that you develop depends upon data that is stored in a database, to the point where it will not function if a connection to the database is not established, then an appropriate place to store your user preferences is in the database.

In object-oriented programming, a good way to manage user preferences is to create a class that encapsulates the specific functions of retrieving and updating the user preferences.

Benefits of using a database

There are several benefits of using a database to store settings:

- *Central location.* When all user settings are saved in a database, administrative changes must be made in only one location instead of at each user's computer.
- *Global preferences.* User preferences are available to users on any computer that they use to run your application.
- *Regular backups.* User preferences are backed up with the database backup.

Example

In the following example, three user preferences are held as properties of a class and stored in a database. Notice the use of the **System.Environment** class, which is provided by the .NET Framework class library, to retrieve the user name and domain name for the currently logged-on user.

The class definition is shown in the following code:

```
public class UserPreferences {
  private bool pageorientation;
  private bool doublesided;
  private bool addpagenumbers;

  public bool pageOrientation {
      get {
          return pageorientation;
      }
      set {
          pageorientation=value;
      }
  }

  public bool doubleSided {
      get {
          return doublesided;
      }
      set {
          doublesided=value;
      }
  }

  public bool addPageNumbers {
      get {
          return addpagenumbers;
      }
      set {
          addpagenumbers=value;
      }
  }
```

The default constructor for the class

Notice in the remainder of the code how the default constructor for the **UserPreferences** class uses information from the **System.Environment** class to query the Microsoft SQL Server™ database for the user preferences for the logged-in user.

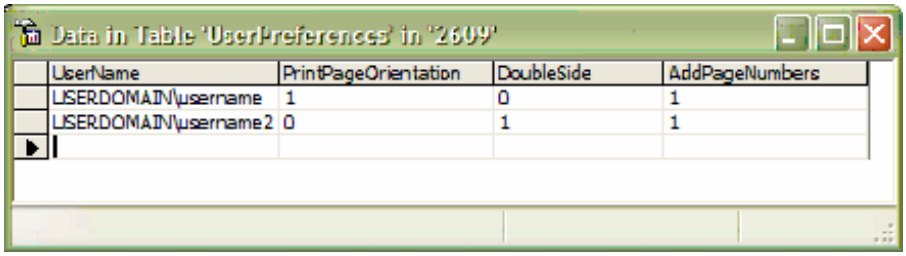
```
public UserPreferences() {
      SqlConnection sqlcon = new SqlConnection(
          "Data Source=localhost; Integrated "+
          "Security=SSPI;Initial Catalog=2609");
      sqlcon.Open();
      string sqlcomtext="SELECT * FROM  UserPreferences "+
          "WHERE UserName = N'" +
          System.Environment.UserDomainName + "\\" +
          System.Environment.UserName.ToString() + "')";
      SqlCommand sqlcom = new SqlCommand(sqlcomtext,sqlcon);
      SqlDataReader sqldr=sqlcom.ExecuteReader();
      while (sqldr.Read()) {
          pageorientation=System.Convert.ToBoolean
              (sqldr.GetInt32(1));
          doublesided=System.Convert.ToBoolean
              (sqldr.GetInt32(2));
          addpagenumbers=System.Convert.ToBoolean
              (sqldr.GetInt32(3));
      }
      sqldr.Close();
      sqlcon.Close();
  }
```

The save method

The **UserPreferences** class has only one public method, **Save**. In the remainder of the code, the preferences for this user are updated in the **UserPreferences** table in the database.

```
public void Save() {
      SqlConnection sqlcon = new SqlConnection(
          "Data source=localhost;Integrated "+
          "Security=SSPI;Initial Catalog=2609");
      sqlcon.Open();
      string sqlcomtext="UPDATE UserPreferences SET "+
          "DoubleSide=" + System.Convert.ToInt32
          (doublesided).ToString() + " ,AddPageNumbers=" +
          System.Convert.ToInt32(addpagenumbers).ToString()+
          ",PrintPageOrientation="+System.Convert.ToInt32
          (pageorientation).ToString()+" WHERE ("+
          "UserName = N'" + System.Environment.UserDomainName
          + "\\" + System.Environment.UserName.ToString() +
          "')";
      SqlCommand sqlcom = new SqlCommand(sqlcomtext,sqlcon);
      sqlcom.ExecuteNonQuery();
      sqlcon.Close();
  }
}
```

The following illustration shows how the User Preferences table looks in the database:

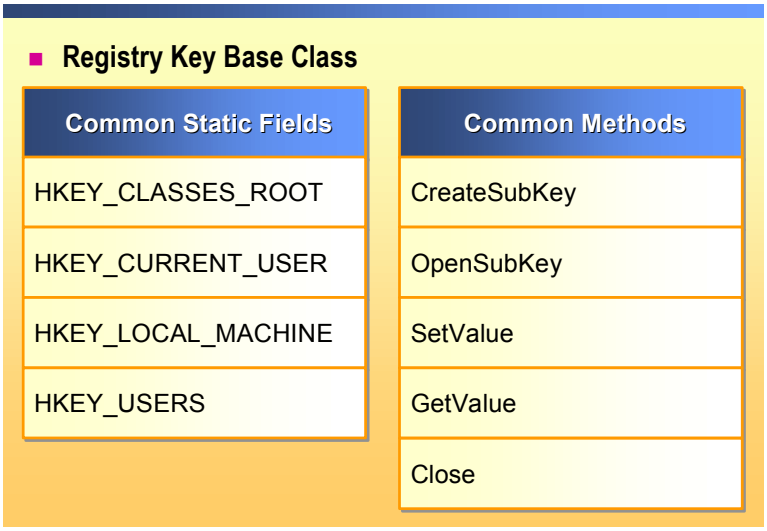


Example using the class

The following code uses the **UserPreferences** class defined above:

```
UserPreferences up = new UserPreferences();
If (up.AddPageNumbers) {
  // add page numbers code
}
```

How to Save Application Settings to the Windows Registry



Introduction

The Windows registry is a general-purpose mechanism for storing program information that must be maintained when a Windows-based application terminates.

Accessing the registry

The .NET Framework class library makes accessing the registry straightforward.

Registry static fields

The static read-only base keys that are exposed by the **Registry** class are shown in the following table.

Base key	Stores information about
HKEY_CLASSES_ROOT	Types and classes and their properties
HKEY_CURRENT_USER	User preferences
HKEY_LOCAL_MACHINE	Configuration of local computer
HKEY_USERS	Default user configuration

RegistryKey methods

The most common methods used to manipulate the **RegistryKey** class are shown in the following table.

RegistryKey method	Used for
CreateSubKey	Creating your own key for your application
OpenSubKey	Opening a key for reading and writing to the registry
SetValue	Modifying and storing an integer or a string to the registry
GetValue	Retrieving current values from the registry
Close	Exiting the registry

Example

The following code uses the **Registry.CurrentUser** static field to access the **HKEY_CURRENT_USER** registry base key. The **OpenSubKey** method of the **RegistryKey** class is then used to open the appropriate lower-level key.

Notice the use of the **GetValue** and **SetValue** methods to read and write values in the registry, respectively. Also, notice the use of the **CreateSubKey** method to create a new subkey.

```
public class UserPreferences {
  private bool pageorientation;
  private bool doublesided;
  private bool addpagenumbers;

  public bool pageOrientation {
      get {
          return pageorientation;
      }
      set {
          pageorientation=value;
      }
  }

  public bool doubleSided {
      get {
          return doublesided;
      }
      set {
          doublesided=value;
      }

  }

  public bool addPageNumbers {
      get {
          return addpagenumbers;
      }
      set {
          addpagenumbers=value;
      }
  }

  public UserPreferences() {
      RegistryKey uprk=Registry.CurrentUser.OpenSubKey
          ("SOFTWARE\\Microsoft\\Demo");
      if (uprk!=null) {
          pageorientation=System.Convert.ToBoolean
              (uprk.GetValue("PageOrientation"));
          doublesided=System.Convert.ToBoolean
              (uprk.GetValue("DoubleSided"));
          addpagenumbers=System.Convert.ToBoolean
              (uprk.GetValue("AddPageNumbers"));
      }
      else {
          pageorientation    =false;
          doublesided        =false;
          addpagenumbers     =false;
      }
  }
```

Code continued on the following page.

```
  public void Save() {
      RegistryKey uprk=Registry.CurrentUser.OpenSubKey
          ("SOFTWARE\\Microsoft\\Demo",true);
      if (uprk==null) {
          RegistryKey msrk=Registry.CurrentUser.OpenSubKey
              ("SOFTWARE\\Microsoft",true);
          uprk=msrk.CreateSubKey("Demo");
      }
      uprk.SetValue("PageOrientation",pageorientation);
      uprk.SetValue("DoubleSided",doublesided);
      uprk.SetValue("AddPageNumbers",addpagenumbers);
  }
}
```

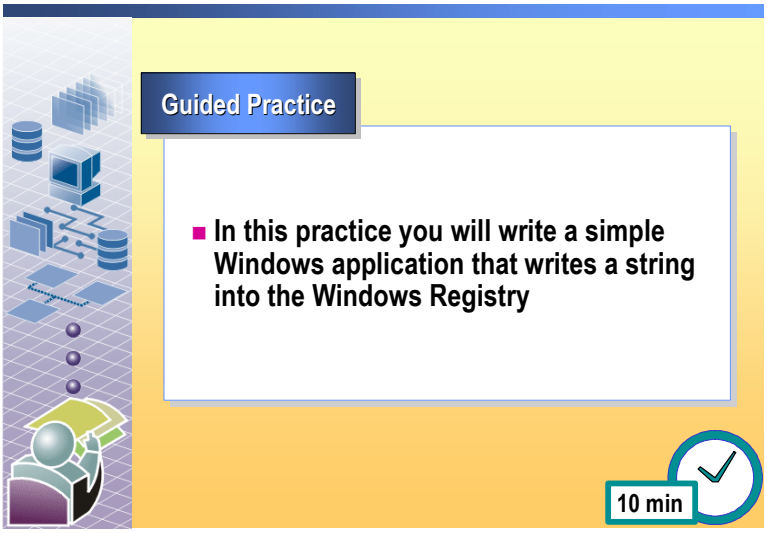
Example using the class

The following code creates a user preferences object tests the AddPageNumbers property:

```
UserPreferences up = new UserPreferences();
If (up.AddPageNumbers) {
  // add page numbers code
}
```

Notice that the preceding code is the same as the example for storing data in the database.

Practice: Using the Windows Registry



In this practice, you will write a simple Windows application that writes a string into the Windows Registry.

The solution for this practice is located at *install_folder*\Practices\Mod11\Registry_Solution\Registry_Solution.sln. Start a new instance of Visual Studio .NET before opening the solution.

Tasks	Detailed steps
1. Start Visual Studio and create a new project. Name: **Registry** Project Type: **Visual C# projects** Template: **Windows Application** Location: *install_folder***Practices****Mod11**	**a.** Start Visual Studio.NET. **b.** On the **File** menu, point to **New**, and then click **Project**. **c.** In the New Project window, under **Project Types**, click **Visual C# Projects**. **d.** Under **Templates**, click **Windows Application**. **e.** In the **Name** box, type **Registry** **f.** In the **Location** box, type *install_folder***Practices****Mod11** and then click **OK**.
2. Add a button to the form. Set the Text property of the button to **Add Info to the Registry**.	**a.** Hover over the toolbox, drag a button control from the toolbox and drop it onto the form. **b.** In the **Properties** window, click **Text** and then type **Add Info to the Registry**.

Tasks	Detailed steps
3. Add the code below into the button click event.	**a.** Double-click the button. **b.** In the **button1_Click** event procedure, add the code below.

Add the following code into the button1_Click event procedure.

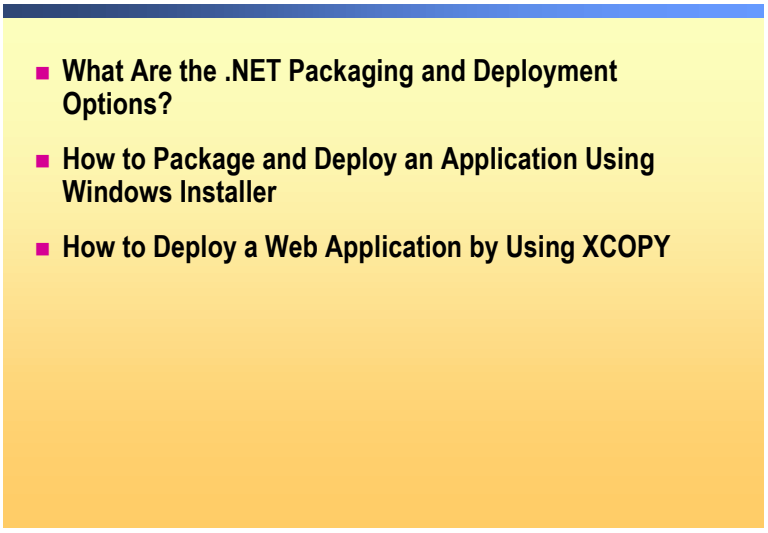
```
Microsoft.Win32.RegistryKey key = Microsoft.Win32.Registry.
  CurrentUser.CreateSubKey("SOFTWARE\\MSDNTraining");
key.SetValue("TestData","ABCDEF");
key.Close();
```

4. Run the application and click the button. Close the window.	**a.** Click **Start** on the standard toolbar. **b.** Click the button on the form. **c.** Close the window.
5. Open RegEdit and verify that the key **HKEY_CURRENT_USER \Software\MSDNTraining** contains a value for **TestData** of **ABCDEF**. Close the Registry Editor window, close Visual Studio.	**a.** Click **Start**, and then click **Run**. **b.** In the **Run** dialog box, in the **Open** box, type **RegEdit** and then click **OK**. **c.** In the **Registry Editor** window, expand **HKEY_CURRENT_USER**, expand **SOFTWARE**, and then click **MSDNTraining**. **d.** Verify that the string value **TestData** contains **ABCDEF**. **e.** Close the **Registry Editor** window. **f.** Close Visual Studio.

The first line of the code added into the button click event could be shortened. How would you accomplish this?

By including a using Microsoft.Win32 statement at the top of the Form1.cs file. The occurrences of Microsoft.Win32 in the code statement could then be removed.

Lesson: Deploying Applications



Introduction

This lesson explains how to package and deploy Windows-based and Web-based applications in the .NET environment.

Lesson objectives

After completing this lesson, you will be able to:

- Package and deploy a Windows-based application.
- Deploy a Web application.

Lesson agenda

This lesson includes the following topics:

- What Are the .NET Packaging and Deployment Options?
- How to Package and Deploy an Application Using Windows Installer
- How to Deploy a Web Application by Using XCOPY

What Are the .NET Packaging and Deployment Options?

- **Packaging Options**
  - As a set of executables and DLLs
  - Microsoft Windows Installer project
  - Cabinet files
- **Deployment Options Using Windows Installer**
  - **Merge Module Project**: packages files/components into a single module
  - **Setup Project**: builds an installer for a Windows-based application
  - **Web Setup Project**: builds an installer for Web application
  - **Cab Project**: creates a cabinet file for downloading to legacy

Introduction

This lesson explores the packaging and deployment options that are available for applications that you develop on the .NET Framework. Your choices depend primarily upon whether you are deploying a Web server application or a Windows-based desktop application.

Packaging

Packaging is the act of creating a package that can install your application on the user's computer.

There are several methods that you can use to package .NET Framework applications:

- As copied files.
- The easiest way to package your application is simply to copy the files directly. For example, you can put all the files on a CD-ROM and write a batch file to copy the files to a directory on the user's hard disk. The user can then just run the application. To uninstall the application, you simply delete the files.
- As cabinet (.cab) files.

  This option is typically used for Internet download scenarios to compress files and reduce download time.
- As a Microsoft Windows Installer 2.0 package.

  With this option, you create .msi files for use with Windows Installer.

Deployment

Deployment is the act of distributing a finished application to the computer and setting up the application so it will run correctly.

Deployment in the .NET Framework differs from traditional setup and deployment in many respects. The .NET Framework provides the following options for deploying applications:

- Use XCOPY or FTP.

  Because common language runtime applications are self-describing and require no registry entries, you can use XCOPY or FTP to simply copy the application to an appropriate directory. The application can then be run from that directory. For all but the simplest cases, it is recommended that you deploy your project rather than use XCOPY.

- Use code download.

  If you distribute your application over the Internet or through a corporate intranet, you can simply download the code to a computer and run the application there.

- Use no-touch deployment.

  Windows Forms allow no-touch deployment, in which applications can be downloaded, installed, and run directly on the user's computer without any alteration of the registry. The application removes itself from the user's computer when the application is closed.

- Use an installer program, such as Windows Installer 2.0.

  The Microsoft Windows Installer that is provided with Visual Studio .NET includes templates for four types of deployment projects and a Setup wizard that guides you through the process of creating a deployment project.

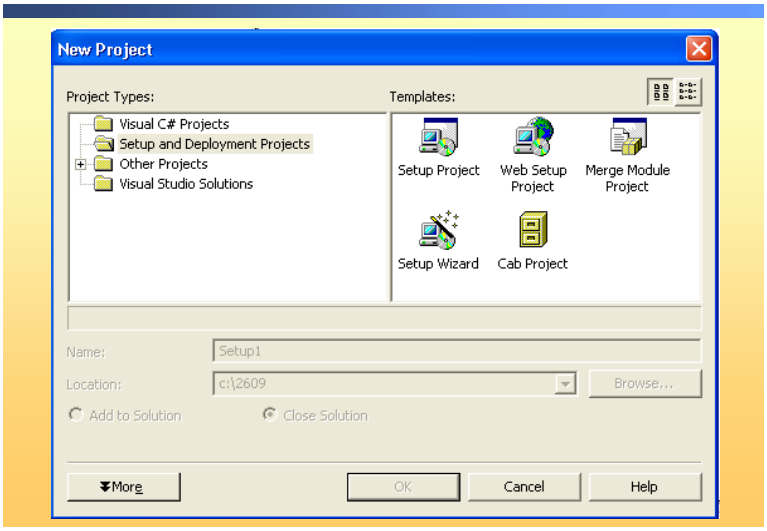
  The following four templates are available for your deployment project:

  - *Merge Module Project*. Packages components that may be shared by multiple applications. Merge Module projects allow you to package files or components into a single module to facilitate sharing. You can use the resulting .msm files in any other deployment project.
  - *Setup Project*. Builds an installer for a Windows-based application.
  - *Web Setup Project*. Builds an installer for a Web application.
  - *Cab Project*. Creates a cabinet file for downloading to an earlier version of a Web browser.

  The distinction between the Setup and Web Setup projects is where the installer deploys the files:

  - For a Setup project, the installer installs files into a Program Files directory on a target computer.
  - For a Web Setup project, the installer installs files into a virtual root directory on a Web server.

How to Package and Deploy an Application Using Windows Installer



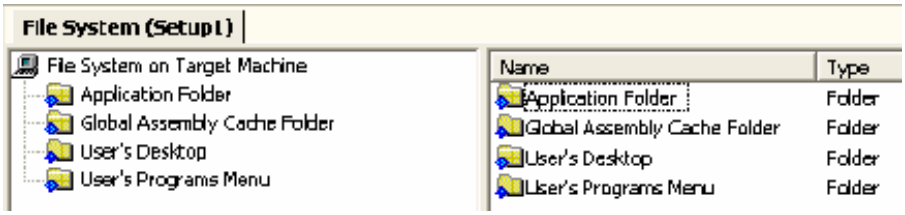
**********ILLEGAL FOR NON-TRAINER USE**********

Introduction

This lesson explains how to use the Windows Installer Setup Project template to customize and deploy an application. In addition to using Windows Installer to package your application as an .msi file and copy the file to the target computer's hard disk, you can also use the installer to add information in the registry, add shortcuts to the application, and create uninstall files.

Components of a Windows Installer Setup project

The following table describes the uses for the various components of a Windows Installer Setup project.



Setup component	Description
Application Folder	Used to store the application itself
Global Assembly Cache Folder	Used to install shared assemblies
User's Desktop	Used to install shortcuts on the desktop
User's Programs Menu	Used to install a shortcut in the **Programs** menu

A note about the bootstrapping application

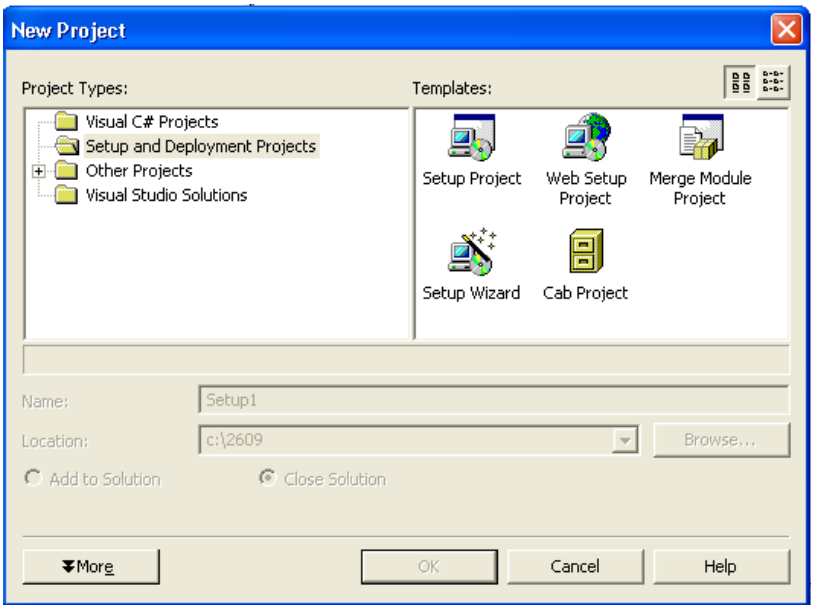
When you deploy a Windows-based application on a version of Windows earlier than Microsoft Windows XP, you must include a *bootstrapping* application, which will execute Windows Installer on your target system before installing the application. To do this, on the **Visual Studio .NET** menu, click **Project**, click **Properties**, and then on the **Bootstrapper** menu, click **Windows Installer Bootstrapper**.

Note For more information about deploying applications in the .NET Framework and including a bootstrapping application, see the MSDN® article, *.NET Framework Deployment Guide*.

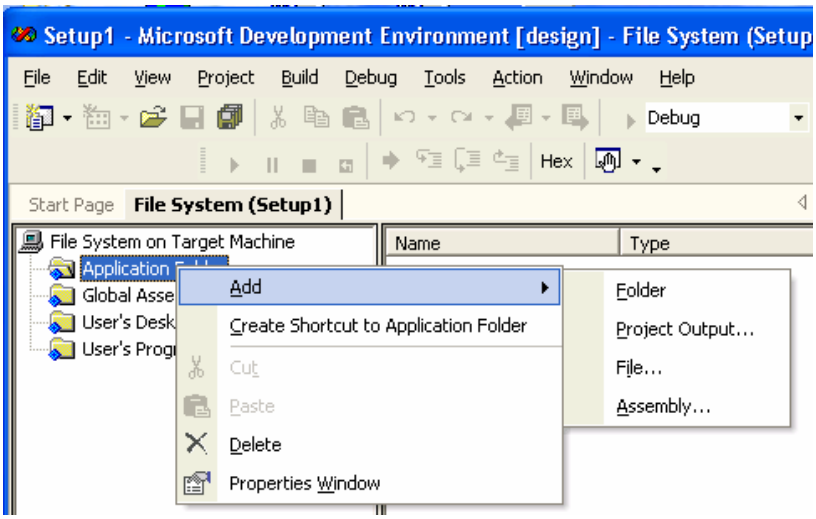
Procedure: Using a Windows Installer Setup Project

To use the Setup Project template to create a Windows Installer package:

1. Start Visual Studio .NET, and then click **New Project**.
2. In the **New Project** dialog box, under **Project Types**, click **Setup and Deployment Projects**, and then under **Templates**, double-click **Setup Project**.

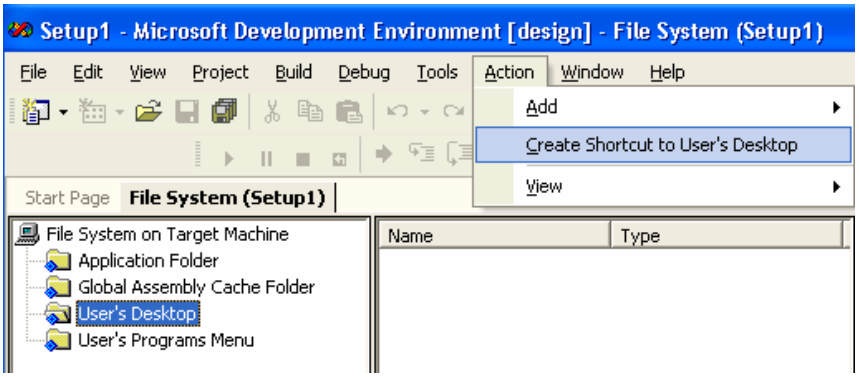


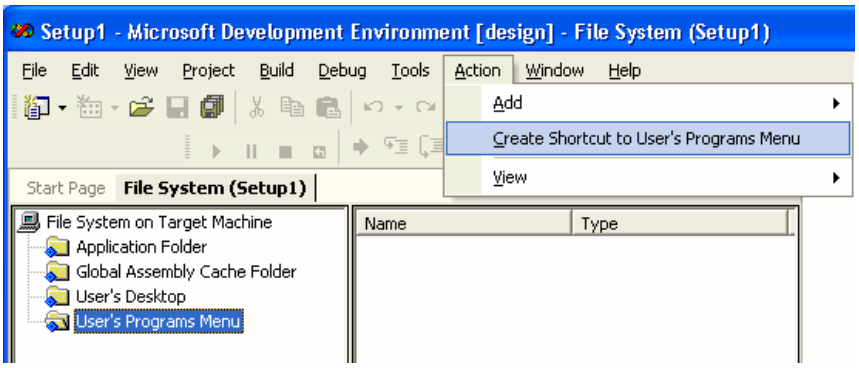
3. In the Properties window, set project properties such as **Author**, **Description**, **Manufacturer**, **ManufacturerUrl**, **ProductName**, **Title**, and **Version**.
4. To add the application files to be installed in the application folder, right-click the **Application Folder** in the left pane, click **Add**, and then click **Folder** or **File**, as appropriate.



5. If you want to change the location where the contents of the Application folder will be installed, change the **DefaultLocation** property of the Application Folder.

6. To add the icons for your application to the setup project, in the left pane, right-click **User's Desktop** or **User's Program Menu**, and then click **Create Shortcut to User's Desktop** or **Create Shortcut to User's Programs Menu**, as appropriate.





7. Build the setup project.

   The output is an .msi file.

8. Using Windows Explorer, locate and double-click the .msi file to install the application on your computer.

How to Deploy a Web Application by Using XCOPY

- **Copy command**
  - On the **Project** menu, click **Copy Project**.
  - Select the destination project folder.
  - Select the Web access method.
  - Select the files to be copied.
- **XCOPY command**
  - Type **xcopy/?** in a command prompt window

Introduction

In the .NET Framework, you can deploy Web applications by using the **Copy Project** or **XCOPY** commands. The **Copy Project** command is available on the **Project** menu of Visual Studio .NET. To use the **XCOPY** command, type **xcopy/?** in a command prompt window.

Pros and cons

Copying a project, rather than deploying it, is the simpler way to move the content of your project to a target Web server. Copying does not automatically configure the directory settings of Microsoft Internet Information Services (IIS). Therefore, it is recommended that you deploy your project in most cases, because it allows you to take advantage of extensive deployment project management features, such as registration and IIS configuration.

There are three major steps that are required to move your application from the development environment to a production server. You must first build the application, then you remove all unnecessary files, and finally you copy the files to the production environment.

Build the application

The first step is to build (compile) your Web application. This process creates the dynamic-link libraries (DLLs) that contain the code for the Web application.

Remove unnecessary files

The second step in deploying a Web application is to remove all unnecessary files from the directory that contains the Web application. This increases the security of your production site by not exposing uncompiled code.

The files that are not needed on the production server include:

- C# solution files (.csproj, and so on)
- Resource (.resx) files
- Code-behind pages (.cs)

The files that are required on the production server include:

- The \bin directory and the DLL files within it
- All Web form, user control, and XML Web service files (.aspx, .ascx, .asmx)
- Configuration files, including Web.config and global.asax
- Any additional support files that are in the directory, such as XML files

After you compile the Web application and remove all unnecessary files, you simply copy all of the remaining Web application files in the development directory to the production directory.

Using Copy Project

These are the typical steps for copying a project to a server:

1. On the **Project** menu, click **Copy Project**.
2. Select the destination project folder.
3. Select the Web access method.
4. Select the files to be copied.

By default, the **Copy Project** command creates a new Web application on the target server and copies only the files that are required to run to the application. Alternatively, you can deploy all project files or all files in the project folder. Note that Microsoft FrontPage® Server Extensions must be installed on the target server to use the **Copy Project** command.

Using XCOPY

The **XCOPY** command copies both files and directories, including subdirectories, to the target computer. Use the following syntax:

XCOPY *source* [*destination*] *options*

The source specifies the location and names of the files that you want to copy and must include either a drive or a path.

The destination specifies the location where you want to copy the files to. The destination parameter can include a drive letter, a directory name, a file name, or a combination of these. If you omit a destination, the **XCOPY** command copies the files to the current directory.

XCOPY options

The following table lists a few of the options that you can use when deploying an assembly by using the **XCOPY** command.

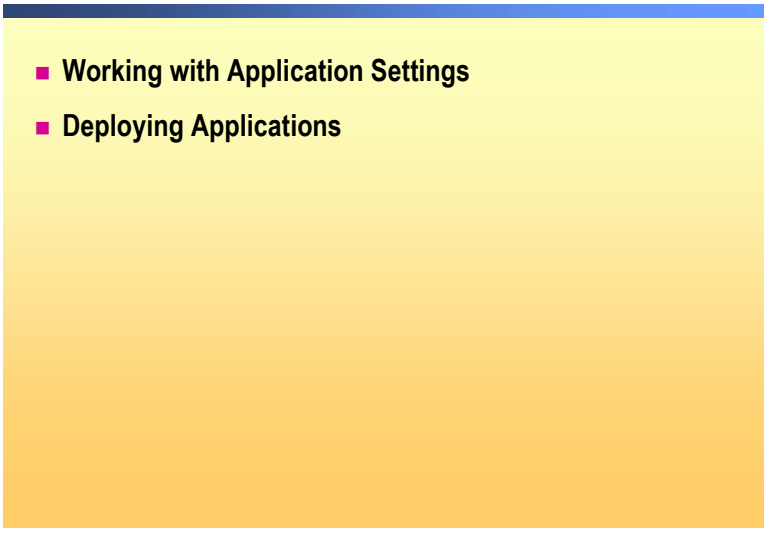
Option	Description
/p	Prompts you to confirm whether you want to create each destination file.
/q	Suppresses display of XCOPY messages.
/e	Copies all subdirectories, even if they are empty.
/s	Copies directories and subdirectories, unless they are empty. If you omit this option, XCOPY works within a single directory.

If you save Web application-specific information in the Machine.config file and transfer the Web application to a different server, you may need to edit the new Machine.config file. Deploying an application by using XCOPY or FTP will not copy the settings in the Machine.config file.

Securing Web servers

When you use the **Copy Project** or **XCOPY** commands, securing your servers is critical. As with any server, it is imperative that you keep up with the latest security updates from Microsoft at http://www.microsoft.com/security. Also, it is recommended that you turn off or disable all services on your Web servers that are not used, particularly those services that allow you to access the file system, such as File Transfer Protocol (FTP) and Web-based Distributed Authoring and Versioning (WebDAV).

Review



1. What are some benefits of storing user preferences in a database?

   Central location

   Global preferences

   Regular backups

2. What are the four deployment templates that are available in Visual Studio .NET?

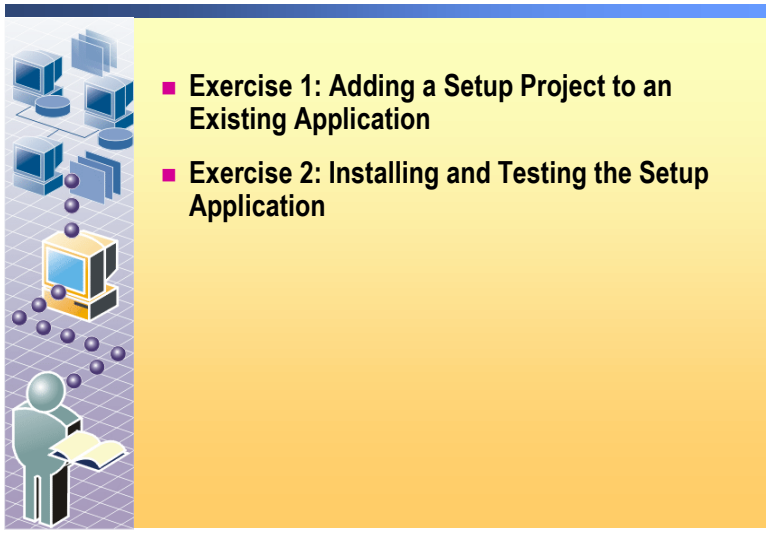
   Merge Module Project

   Setup Project

   Web Setup Project

   Cab Project

Lab 11.1: Deploying an Application



Objectives

After completing this lab, you will be able to:

- Add a deployment project to an existing solution.
- Set registry settings during the setup of an application.
- Under **All Programs** on the **Start** menu, create a shortcut to the installed application.

Prerequisites

Before working on this lab, you must have:

- Knowledge of deploying an application with Visual Studio.NET.

Scenario

In this lab, you will create a setup application to deploy the Zoo Information application. You will customize the setup project created to set some registry settings and add a shortcut under **All Programs** on the **Start** menu for the installed application.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

Task	Detailed steps
▪ Log on to Windows as **Student** with a password of **P@ssw0rd**.	▪ Log on to Windows with the following account: • User name: **Student** • Password: **P@ssw0rd**

Exercise 1
Adding a Setup Project to an Existing Application

In this exercise, you will use the Setup Wizard to add a Setup project to the existing Zoo Information application.

Tasks	Detailed steps
1. Start Visual Studio .NET, and then open the project in the *install_folder*\Labfiles\Lab11_1 folder .	**a.** Start a new instance of Visual Studio .NET. **b.** On the **Start Page**, click **Open Project**. **c.** In the Open Project window, browse to *install_folder*\Labfiles\Lab11_1, click **animals.sln**, and then click **Open**.
2. Use Solution Explorer to add a Setup Wizard template project to the existing solution. Name: **Setup** Location: *install_folder***Labfiles\Lab11_1**	**a.** In Solution Explorer, right-click **Solution 'animals'**, point to **Add**, and then click **New Project**. **b.** In the Add New Project window, under **Project Types**, click **Setup and Deployment Projects**. **c.** Under **Templates**, click **Setup Wizard**. **d.** In the **Name** box, type **Setup** **e.** In the **Location** box, type *install_folder***Labfiles\Lab11_1** and then click **OK**.
3. Complete the setup wizard using the following information: Project Type: **Create a setup for a Windows application**. Include: **Primary output from animals**. Additional Files: **AnimalData.xml**, **Antelope.jpg**, **Elephant.jpg**, **Lion.jpg** from the folder *install_folder***Labfiles\Lab11_1**	**a.** In the Setup Wizard, on the **Welcome to the Setup Project Wizard** page, click **Next**. **b.** On the **Choose a project type** page, click **Next**. **c.** On the **Choose project outputs to include** page, under **Which project output groups do you want to include?** Click **Primary output from animals**, and then click **Next**. **d.** On the **Choose files to include** page, click **Add**. **e.** In the **Add Files** dialog box, change **Look In** to *install_path***Labfiles\Lab11_1**. **f.** Press and hold the CTRL key, and then click **AnimalData.xml**. With the CTRL key still held down, click **Antelope.jpg**, click **Elephant.jpg**, click **Lion.jpg**, and then click **Open**. **g.** On the **Choose files to include** page, click **Next**. **h.** On the **Create Project** page, click **Finish**.
4. Set the following properties of the Setup project: Author: **AdventureWorks** Manufacturer: **AdventureWorks** Title: **Zoo Information** ProductName: **Zoo Information**	**a.** In Solution Explorer, click **Setup**. **b.** In the Properties window, click **Author**, and then type **AdventureWorks** **c.** Click **Manufacturer**, and then type **AdventureWorks** **d.** Click **Title**, and then type **Zoo Information** **e.** Click **ProductName**, and then type **Zoo Information**

Tasks	Detailed steps
5. Create a shortcut to the primary output, and then move this shortcut to a folder under the users Programs Menu. Program Sub-menu name: **Zoo Applications**.	**a.** In the File System (Setup) editor window, under **File System on Target Machine**, right-click **User's Programs Menu**, point to **Add**, and then click **Folder**. **b.** Type **Zoo Applications** **c.** Under **File System on Target Machine**, click **Application Folder**. **d.** In the contents of the **Application Folder**, right-click **Primary output from animals**, and then click **Create Shortcut to Primary output from animals (Active)**. **e.** Type **Zoo Information** **f.** Drag this new shortcut icon from the Application Folder to the Zoo Applications folder under **User's Programs Menu**.
6. Use the Registry editor of the Setup project to create a sub-key under **HKEY_CURRENT_USER\Software\[Manufacturer]** called **ZooInformation**. Add two string values to this key: AutoLoad = **False** ShowTitle = **True**	**a.** In Solution Explorer, right-click **Setup**, point to **View**, and then click **Registry**. **b.** In the **Registry (Setup)** editor window, under **Registry on Target Machine**, under **HKEY_CURRENT_USER**, under **Software**, right-click **[Manufacturer]**, point to **New**, and then click **Key**. **c.** Type **ZooInformation** **d.** Right-click **ZooInformation**, point to **New**, and then click **String Value**. **e.** Type **AutoLoad** **f.** In the Properties window, click **Value**, and then enter **False**. **g.** Right-click **ZooInformation**, point to **New**, and then click **String Value**. **h.** Type **ShowTitle** **i.** In the Properties window, click **Value**, and then enter **True**.
7. Build the Setup project.	▪ In Solution Explorer, right-click **Setup**, and then click **Build**.

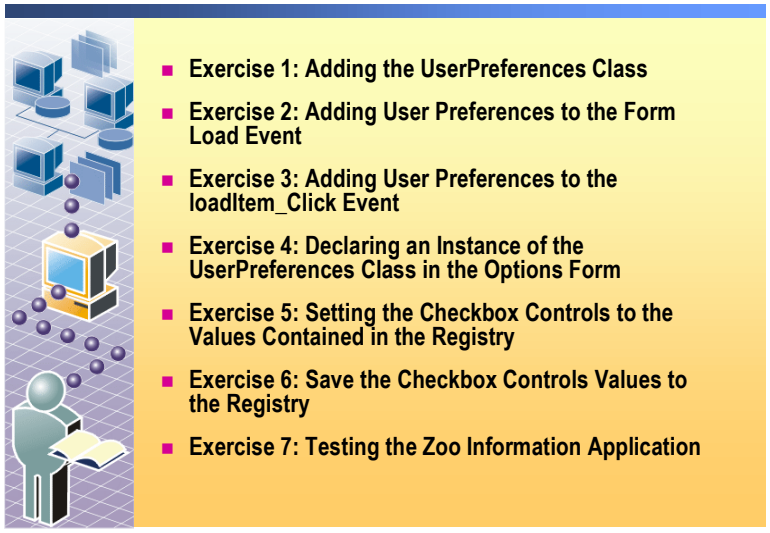
Exercise 2

Installing and Testing the Setup Application

In this exercise, you will install the application on your computer. You will then run the application.

Tasks	Detailed steps
1. Install the setup.	**a.** In Solution Explorer, right-click **Setup**, and then click **Install**. **b.** Accept the defaults provided by the setup wizard.
2. Run the Zoo Information application.	**a.** Click **Start**, point to **All Programs**, point to **Zoo Applications**, and then click **Zoo Information**. **b.** In the Zoo Information window, click **File**, and then click **Open**. **c.** In the **Open** dialog box, click **AnimalData.xml**, and then click **Open**.
3. Close the application, save changes to your solution, and then quit Visual Studio .NET.	**a.** Close the Zoo Information window. **b.** In Visual Studio .NET, on the **File** menu, click **Save All**. **c.** On the **File** menu, click **Exit**.

Lab 11.2 (optional): Working with Application Settings



Objectives

After completing this lab, you will be able to:

- Read data from the registry.
- Write data to the registry.

Prerequisites

Before working on this lab, you must have:

- Knowledge of working with application settings including the **RegistryKey** class of the .Net Framework Class Library.

Scenario

In this lab, you will enhance the existing code of the Zoo Information application, which is already written, to store the following simple user preferences in the Windows registry:

- *ShowTitle preference.* Controls whether to display the name of the animal above the text in the user interface.
- *AutoLoad preference.* Controls whether the application automatically attempts to open the data file that it last opened. The AutoLoad functionality requires that the last file opened by the application is stored in the registry.

The solution for this lab is provided in install_folder\Labfiles\Lab11_1\Solution_Code\Animals.sln. Start a new instance of Visual Studio .NET before opening the solution.

Estimated time to complete this lab: 30 minutes

Exercise 0
Lab Setup

Task	Detailed steps
▪ Log on to Windows as **Student** with a password of **P@ssw0rd**.	▪ Log on to Windows with the following account: • User name: **Student** • Password: **P@ssw0rd**

Exercise 1
Adding the UserPreferences Class

The first step in modifying the application is adding the **UserPreferences** class definition to the application. The **UserPreferences** class has three public fields: *showtitle*, *autoload*, and *lastfilename*. Information is retrieved from the registry during the execution of the class constructor and saved to the registry in a method called **Save**.

Tasks	Detailed steps
1. Start Visual Studio .NET, and the open *install_folder*\Labfiles \Lab11_2\animals.sln.	**a.** Start a new instance of Visual Studio .NET. **b.** On the **Start Page**, click **Open Project**. **c.** In the **Open Project** dialog box, browse to *install_folder*\Labfiles\Lab11_2, click **animals.sln**, and then click **Open**.
2. View the code of Form1.cs.	▪ In Solution Explorer, right-click **Form1.cs**, and then click **View Code**.

3. Add the **UserPreferences** class definition at the bottom of the existing code in Form1.cs.	**a.** Scroll to the bottom of Form1.cs.
	b. On the line before the last brace, **},** add the following class definition:
	`public class UserPreferences{`
	
	`}`

4. In the **UserPreferences** class, create three public fields: • showtitle (Boolean) • autoload (Boolean) • lastfilename (String)	▪ In the class definition that you created in the preceding step, add the following three public fields: • showtitle (Boolean) • autoload (Boolean) • lastfilename (String)

Tasks	Detailed steps
5. Add a default constructor to the **UserPreferences** class. You should be able to modify the code examples in this module to solve this problem.	**a.** In this constructor, open the registry key **HKEY_CURRENT_USER\Software\AdventureWorks\ ZooInformation**. **Note:** If this key does not exist, your application should default the values of the fields shown in the following step. **b.** Set the public fields that you created in the preceding step from the values held in the key: **ShowTitle** (default true) **AutoLoad** (default false) **LastFileName** (default "")
6. Add a **Save** method to the **UserPreference** class. You should be able to modify the code examples in this module to solve this problem.	**a.** In this method, open the registry key **HKEY_CURRENT_USER\Software\AdventureWorks\ ZooInformation** for writing. **Note:** If this key does not exist, your application should create it. **b.** Store the public fields in the registry values: **ShowTitle** **AutoLoad** **LastFileName**

Exercise 2

Adding User Preferences to the Form Load Event

In this exercise, you will use the **UserPreferences** class to read the user preferences and determine if the application should automatically load the last file opened.

Task	Detailed steps
▪ Locate the **Form1_Load** event in Form1.cs. Complete the instructions in the comments for Lab11.2 Exercise 2.	**a.** Locate the **Form1_Load** event. **b.** Add the required code as described in the comments titled Lab11.2 Exercise 2.

Exercise 3

Adding User Preferences to the loadItem_Click Event

In this exercise, you will use the **UserPreferences** class to update the **LastFileName** value held in the registry after a user selects a file to open.

Task	Detailed steps
▪ Locate the **loadItem_Click** event in Form1.cs. Scroll through the code in this event until you locate the Exercise 3 instructions. Complete the instructions.	**a.** Locate the loadItem_Click event. **b.** Scroll through the code until you locate the Exercise 3 instructions. **c.** Add the code as described in the instructions.

Exercise 4

Declaring an Instance of the UserPreferences Class in the Options Form

In this exercise, you will declare an instance of the **UserPreferences** class as private for use in the Options form.

Tasks	Detailed steps
1. Open the Options.cs code editor.	▪ In Solution Explorer, right-click **Options.cs**, and then click **View Code**.
2. Within the definition of the class Options, locate and then follow the Lab11.2 Exercise 4 instructions.	**a.** Locate the **Options** class within the code. **b.** Locate and then follow the Lab11.2 Exercise 4 instructions.

Exercise 5

Setting the Checkbox Controls to the Values Contained in the Registry

In this exercise, you will create an instance of the **UserPreferences** class and use the properties of the class to set the check boxes on the form. You will place this code into the **Options_Load** event.

Task	Detailed step
▪ Locate the **Options_Load** event, and then follow the Lab11.2 Exercise 5 instructions.	▪ Locate the **Options_Load** event, and then follow the Lab11.2 Exercise 5 instructions.

Exercise 6

Save the Checkbox Controls Values to the Registry

In this exercise, you will set the values of the check boxes on the form into the properties of the **UserPreferences** instance that you created in Exercise 5, and then call the **Save** method.

Task	Detailed step
▪ Locate the **ok_Click** event, and then follow the Lab11.2 Exercise 6 instructions.	▪ Locate the **ok_Click** event, and then follow the Lab11.2 Exercise 6 instructions.

Exercise 7

Testing the Zoo Information Application

In this exercise, you will run the Zoo Information application and test that the user preferences are stored correctly in the registry.

Tasks	Detailed steps
1. Run the Zoo Information application.	▪ On the Standard toolbar, click **Start**.
2. In the Zoo Information window, open the file AnimalData.xml.	**a.** In the Zoo Information window, on the **File** menu, click **Open**. **b.** In the **Open** dialog box, click **AnimalData.xml**, and then click **Open**.
3. On the **View** menu, click **Options**, and then click **Autoload Last File on Startup**.	**a.** On the **View** menu, click **Options**. **b.** In the **Options** window, click **Autoload Last File on Startup**, and then click **Open**.
4. Close the Zoo Information window.	▪ Close the Zoo Information window.
5. Run the application.	▪ On the Standard toolbar, click **Start**.
Note: The application should automatically load the AnimalData.xml file and display the Antelope information page. If the application displays an error or this functionality does not appear, debug your application.	
6. Close the Zoo Information window.	▪ Close the Zoo Information window.
7. Save the changes to the solution, and then quit Visual Studio .NET.	**a.** In Visual Studio .NET, on the **File** menu, click **Save All**. **b.** On the **File** menu, click **Exit**.