msdn® training

Module 2: Development Environment Features

Contents

For trainer preparation purposes only

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®

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be in ferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, ActiveX, BizTalk, FrontPage, IntelliSense, JScript, Microsoft Press, Outlook, PowerPoint, Visio, Visual Basic, Visual C++, Visual C#, Visual InterDev, Visual Studio, and Windows Media are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
60 Minutes

Lab:
45 Minutes

This module teaches students some of the overall benefits of using the new Microsoft® Visual Studio® .NET version 7.0 integrated development environment (IDE) and how to create Visual Basic® .NET projects. Students will try some of the tools that make the IDE so powerful for application development. They will learn how to debug and compile projects.

After completing this module, students will be able to:

- Describe the overall benefits of the new IDE.
- Describe the different types of Visual Basic .NET projects and their structures, including their file structures.
- Reference external applications from a project.
- View and set the properties of a project.
- Use the various windows in the IDE, including the Server Explorer, Object Browser, and Task List.
- Debug a simple application.
- Build and compile a simple application.

Materials and Preparation

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

For trainer preparation purposes only

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2373A_02.ppt
- Module 2, "Development Environment Features"

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 demonstrations.
- Complete the labs.

Demonstrations

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

Creating a Visual Basic .NET Project

↙ To prepare for the demonstration

- Open Visual Studio .NET.

↙ To use a project template

1. Create a new project and point out the various project templates available when creating a Visual Basic .NET project. Remind students of the general purpose of each template.
2. Create a Windows application project named SimpleProject in the *install folder*\DemoCode\Mod02\SimpleProject folder.

↙ To analyze the project

1. Examine the current project hierarchy. Point out the solution, project, and Form1 files.
2. Open the project **Property Pages** dialog box, and then explain all aspects of the **General** section under **Common Properties**, such as **Assembly name**, **Root namespace**, and **Output type**.
3. Examine the **Imports** section. Point out that these project-level imports do not require an **Imports** statement in the code.
4. Close the project **Property Pages** dialog box.

For training preparation purposes only

↙ To add a project reference

1. Examine the current list of project references in the Solution Explorer. Point out some of their properties, such as the location of the assembly .dll files.
2. Open the **Add Reference** dialog box for the project and examine the list of .NET and COM items listed in the list boxes. Add one of the .NET assemblies such as **System.EnterpriseServices** (the reference will not actually be used in the demonstration, but it can be seen in the Solution Explorer). Point out that if a COM reference is added, the COM interoperability mechanism is invoked and should be accepted. Also point out that there are currently no items on the **Projects** tab, because no other projects are loaded into this solution.
3. Save the project and close Visual Studio .NET.

Using the Visual Studio .NET IDE

↙ To prepare for the demonstration

- Open Visual Studio .NET, and then open the SimpleProject project created in the last demonstration (" Creating a Visual Basic .NET Project").

↙ To use the Solution Explorer

1. Add a class module to the project. Leave the default name Class1.vb.
2. Select the project in the Solution Explorer. On the **Project** menu, click **New Folder** to add a new folder to the project hierarchy. Rename the folder Classes.
3. Drag the Class1.vb file to the Classes folder.
4. Right-click **Class1.vb**, and then click **Exclude From Project**.
5. On the Solution Explorer toolbar, click **Show All Files**.
6. Right-click **Class1.vb**, and then click **Include In Project**.

↙ To use Server Explorer to create a data connection

1. Open Server Explorer.
2. Click **Auto Hide** on the Server Explorer toolbar to anchor Server Explorer to the screen.
3. To add a data connection to the project, click **Connect to Database** on the Server Explorer toolbar.
4. Enter the following values in the **Data Link Properties** dialog box.

Property	Value
Server name	localhost
Logon information	Windows NT Integrated security
Database	Cargo

5. Expand the new data connection. Point out the functional similarity to the Data View window found in previous versions of Microsoft Visual Basic.

For trainer preparation purposes only

↙ To use the Server Explorer to examine the local server

1. In Server Explorer, expand **Servers**.
2. Expand the instructor machine.
3. Expand and examine the server items, such as **Event Logs**, **SQL Servers**, **Performance Counters**, and **Services**.
4. To start the SQLServerAgent service, expand the **Services** item, right-click **SQLServerAgent**, and then click **Start**. Point out how to stop the service if required.
5. Click **Auto Hide** to hide Server Explorer.

↙ To edit an Extensible Markup Language (XML) file

1. Using Windows Explorer, navigate to the *install folder*\DemoCode\Mod02 folder.
2. Drag the Customers.xml file from Windows Explorer to the **SimpleProject** node of the tree view in Solution Explorer.

   If you drop it anywhere else, it may not be added to the project successfully.
3. Point out that this copy of the document is stored with the project files.
4. Double-click **Customers.xml** in the Solution Explorer to edit the document.
5. Examine the document in XML view and then Data view.
6. Change the first name of one of the customers by using the editable grid, and then press ENTER. Confirm the change has taken place by checking the data in XML view.
7. Save your project and close Visual Studio .NET.

Debugging a Project

↙ To prepare for the demonstration

- Open Visual Studio .NET, and then open the *install folder*\DemoCode\Mod02\Debugging\Starter\Debugging.sln solution.

↙ To set a conditional breakpoint

1. Open the code window for the form.
2. Explain the purpose of the code.
3. Set a breakpoint on the **Button1_Click** procedure definition.
4. Right click on the breakpoint itself, and in the **Breakpoint Properties** dialog box, click **Condition**. Set the following condition:

   ```
   iCounter = 4
   ```

5. Close the **Breakpoint Properties** dialog box.

For trainer preparation purposes only

↙ To step through the code

1. Run the project.
2. When the form appears, click the **Begin** button on the form. Verify that the Output window displays the debugging information as a result of the **Debug.WriteLine** statement.
3. To demonstrate that the condition is not initially met, click **Begin** three times until execution halts at the conditional breakpoint.
4. Point out the various debugging windows, including the Locals and Breakpoints windows, and the value of the *iCounter* variable.
5. Step through the entire code by using either the **Debug** menu or toolbar before closing the form and stopping the debugging process.

↙ To use the Command window

1. Ensure the Command window is activated and that the command prompt (>) is displayed. If it is not, type >**cmd** and press ENTER.
2. Use the **Debug.Start** command to start the debugger.
3. Click **Begin** three times to enter debugging mode, and then click the Command window tab to make it the active window.
4. Use the **immed** command to change to Immediate mode.
5. Type **?iCounter**to check the value of the *iCounter* variable.
6. Use the **>cmd** command to change back to Command mode.
7. Use the **Debug.StopDebugging** command to end the debugging session.
8. Use the **Exit** command to close Visual Studio .NET.

For trainer preparation purposes only

Module Strategy

Use the following strategy to present this module:

- Describing the Integrated Development Environment

  This lesson is an introduction to the general benefits of using the Visual Studio .NET IDE.

  The most important aspect of this lesson is that there is only one IDE that is required to create all types of projects. Students no longer must develop their components in the Visual Basic IDE and their Web pages in Microsoft Visual InterDev® or another Web development tool.

- Creating Visual Basic .NET Projects

  This lesson introduces some of the basic concepts required to create a Visual Basic .NET project. Many aspects of this lesson are similar to concepts covered in previous Visual Basic courses, such as project templates, structures, properties, and references.

  This lesson also introduces assemblies and namespaces, both of which are fundamental aspects of .NET-compatible development. Explain assemblies enough to give the students an overall understanding of their purpose, without going into too much depth. Concepts such as versioning and security will be covered in Module 10, "Deploying Applications," in Course 2373A, *Programming with Microsoft Visual Basic .NET (Prerelease).*

  The namespaces lesson has a small amount of simple code that you will need to explain to students. This code defines classes and creates objects based on those classes. This type of code will be easily understood by all Visual Basic developers, but advise them that this code will be explained in Module 5 "Object-Oriented Programming in Visual Basic .NET," in Course 2373A, *Programming with Microsoft Visual Basic .NET (Prerelease)*.

- Using Development Environment Features

  This lesson introduces the various features of the IDE such as the Solution Explorer, Server Explorer, Object Browser, Task List, Dynamic Help, XML editing, and macros. Some of these items are similar to those found in previous versions of Visual Basic, such as the Solution Explorer and Object Browser, so you will not have to explain them in detail.

  Students should have a basic amount of XML knowledge, but check that they understand the structure of the XML document, Bookstore.xml, shown in the demonstration.

For trainer preparation purposes only

- Debugging Applications

  This lesson begins by discussing breakpoints and how to set them. This will not be new to most students. However, explain how to set conditional breakpoints in detail, because this is handled differently in Visual Basic .NET than in previous versions of Visual Basic.

  This lesson discusses the various debugging techniques and debugging windows that are available. Again, because several of these features are based on previous versions of Visual Basic, point out the enhancements rather than explaining each feature in much detail.

  The Command window is also discussed, and the slide shows an example of the window's use. Step through the example and point out the various effects each statement has on the environment.

- Compiling in Visual Basic .NET

  This lesson begins by looking at how the Task List window assists you in tracking syntax errors during an attempted compilation. The last slide discusses the various compilation options available to the developer, including the possible configuration settings Debug and Release. Be sure students understand the difference between the two settings. This will affect the assembly that is generated.

For trainer preparation purposes only

Overview

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

Lead-in
In this module, you will learn about the Visual Studio .NET development environment and the many powerful features it provides for Visual Basic .NET developers.

- **Describing the Integrated Development Environment**
- **Creating Visual Basic .NET Projects**
- **Using Development Environment Features**
- **Debugging Applications**
- **Compiling in Visual Basic .NET**

The Microsoft® Visual Studio® .NET version 7.0 integrated development environment (IDE) provides you with enhancements to many tools found in previous versions of Microsoft Visual Basic®, combined with features found in other environments, such as Microsoft Visual C++®.

In this module, you will learn the overall benefits of using this new IDE. You will learn how to create Visual Basic .NET projects, and will try some tools of the new IDE. Finally, you will learn how to debug your projects and how to compile them.

After completing this module, you will be able to:

- Describe the overall benefits of the new IDE.
- Describe the different types of Visual Basic .NET projects and their structures, including their file structures.
- Reference external applications from your project.
- View and set the properties of a project.
- Use the various windows in the IDE, including Server Explorer, the Object Browser, and the Task List.
- Debug a simple application.
- Build and compile a simple application.

For training preparation purposes only

Describing the Integrated Development Environment

Topic Objective
To discuss some overall benefits of the IDE.

Lead-in
The Visual Basic .NET IDE provides some significant benefits over previous Visual Basic IDEs.

- **There Is One IDE for All .NET Projects**
- **Projects Can Contain Multiple Programming Languages**
  - Example: Visual Basic .NET and C# in the same project
- **The IDE Is Customizable Through "My Profile"**
- **The IDE Has a Built-in Internet Browser**

The Visual Studio .NET IDE provides some significant enhancements to previous IDEs for Visual Basic.

- There is one IDE for all Microsoft .NET projects

  The Visual Studio .NET IDE provides a single environment where you can develop all types of .NET applications, from simple applications based on Microsoft Windows®, to complex *n*-tier component systems and complete Internet applications. For example, you no longer need to create your components in a separate environment from your Internet pages and scripts.

- Projects can contain multiple programming languages

  You can incorporate multiple programming languages within one solution and edit all your code within the same IDE. This can aid team development of a system where parts of the solution are written in Visual Basic .NET, and other parts are written in C# or other .NET-compatible languages.

- The IDE is customizable through My Profile

  The IDE is fully customizable through the My Profile configuration section on the Visual Studio .NET Start Page.

  - You can select a preexisting profile such as the Visual Basic Developer, or you can modify each section manually.
  - You can specify how you want your IDE screen to look and how the keyboard behaves. This is particularly useful if you are used to Visual Basic version 6.0 keyboard shortcuts for various actions.
  - You can choose to filter help files based on your preferences.

- The IDE has a built-in Internet browser

  You can browse the Internet within the IDE, enabling you to look up online resources without moving between multiple application windows. This built-in browser can also display the Visual Studio .NET Help files for easy access to the relevant documentation.

For Trainer Preparation Purposes Only

◆ Creating Visual Basic .NET Projects

Topic Objective
To introduce the topics in this lesson.

Lead-in
Many concepts in Visual Basic .NET project creation will be familiar to you.

- Choosing a Project Template
- Analyzing Project Structures
- What Are Assemblies?
- Setting Project References
- What Are Namespaces?
- Creating Namespaces
- Importing Namespaces
- Setting Project Properties

Many aspects of project development in Visual Basic .NET are similar to those in previous versions of Visual Basic. You still have a range of project templates to choose from, you still need to reference other projects and applications, and you still need to set project properties. Visual Basic .NET provides enhancements to these and other aspects of project development.

In this lesson, you will become familiar with the project templates provided by Visual Basic .NET. After completing this lesson, you will be able to:

- Choose the correct template for your project.
- Explain how various Visual Basic .NET projects are structured.
- Explain what assemblies are and how to create them.
- Reference other code from your project.
- Create and use namespaces in your projects.
- Use the **Imports** statement to access objects.
- Set various project properties that affect how your application behaves.

For training preparation purposes only

Choosing a Project Template

Topic Objective
To discuss the various Visual Basic .NET project templates.

Lead-in
As in previous versions of Visual Basic, you can choose from a variety of project templates to assist you in the creation of a new project.

- **Windows Application**
- **Class Library**
- **Windows Control Library**
- **ASP .NET Web Application / Service / Control Library**
- **Console Application**
- **Windows Service**
- **Others**

Visual Basic developers are used to having multiple project templates to choose from when starting a new project. Visual Basic .NET provides many familiar templates along with a range of new ones.

Delivery Tip
Point out that students will use many of these project templates during the remainder of the course.

Template	Use this template to create:
Windows Application	Standard Windows-based applications.
Class Library	Class libraries that provide similar functionality to Microsoft ActiveX® dynamic-link libraries (DLLs) by creating classes accessible to other applications.
Windows Control Library	User-defined Windows control projects, which are similar to ActiveX Control projects in previous versions of Visual Basic.
ASP .NET Web Application	Web applications that will run from an Internet Information Services (IIS) server and can include Web pages and XML Web services.
ASP .NET Web Service	Web applications that provide XML Web Services to client applications.
Web Control Library	User-defined Web controls that can be reused on Web pages in the same way that Windows controls can be reused in Windows applications.
Console Application	Console applications that will run from a command line.
Windows Service	Windows services that will run continuously regardless of whether a user is logged on or not. Previous versions of Visual Basic require you to use third-party products or low-level application programming interface (API) calls to create these types of applications.
Other	Other templates exist for creating enterprise applications, deployment projects, and database projects.

For training preparation purposes only

Analyzing Project Structures

Topic Objective
To discuss the structure of Visual Basic .NET projects.

Lead-in
Visual Basic .NET projects contain various types of files that you use depending on the type of project you are creating.

- **Solution Files (.sln, .suo)**
- **Project Files (.vbproj)**
- **Local Project Items**
  - Classes, forms, modules, etc. (.vb)
- **Web Project Items**
  - XML Web services (.asmx)
  - Web forms (.aspx)
  - Global application classes (.asax)

Each project contains a variety of files unique to the type of project. To simplify management, the project files are usually stored within the same project directory.

- Solution files (.sln, .suo)

  The .sln extension is used for solution files that link one or more projects together, and are also used for storing certain global information. These files are similar to Visual Basic groups (.vbg files) in previous versions of Visual Basic. Solution files are automatically created within your Visual Basic .NET projects, even if you are only using one project in the solution.

  The .suo file extension is used for Solution User Options files that accompany any solution records and any customizations you make to your solution. This file saves your settings, such as breakpoints and task items, so that they are retrieved each time you open the solution.

- Project files (.vbproj)

  The project file is an Extensible Markup Language (XML) document that contains references to all project items, such as forms and classes, as well as project references and compilation options. Visual Basic .NET project files use a .vbproj extension, which allows you to differentiate between files written in other .NET-compatible languages (Microsoft C# uses .csproj). This makes it easy to include multiple projects that are based on different languages within the same solution.

For trainer preparation purposes only

Delivery Tip
Explain that you can store more than one item in the same file, such as two classes in a single .vb file. This allows you to keep related classes together for easy maintenance.

- Local project items (.vb)

  Previous versions of Visual Basic use different file extensions to distinguish between classes (.cls), forms (.frm), modules (.bas), and user controls (.ctl). Visual Basic .NET allows you to mix multiple types within a single .vb file. For example, you can create more than one item in the same file. You can have a class and some modules, a form and a class, or multiple classes all within the same file. This allows you to keep any strongly related items together in the same file; for example, the **Customer** and **Address** classes.

  Any files that are not based on a programming language have their own extension; for example, a Crystal Report file (.rpt) or text file (.txt).

- Web project items (.aspx., .asmx, .asax)

  Web projects store their items in a Web server virtual directory and in an offline cache. Like local project items, Web project items also use the .vb file extension for classes and modules. However, Web project items include Web-specific files, such as .aspx for Web Forms, .asmx for XML Web Services, and .asax for global application classes.

Note For more information about Web projects, see Module 7, "Building Web Applications," in Course 2373A, *Programming with Microsoft Visual Basic .NET (Prerelease)*.

For trainer preparation purposes only

What Are Assemblies?

Topic Objective
To explain how assemblies are created.

Lead-in
Assemblies are a key concept in the .NET Framework. They are the building block of all .NET-compatible applications.

- **An Assembly is One or More .exe or .dll Files That Make Up a Visual Studio .NET Application**
- **The .NET Framework Provides Predefined Assemblies**
- **Assemblies Are Created Automatically When You Compile Source Files**
  - Click **Build** on the Build menu
  - Use the command-line command **vbc.exe**

Delivery Tip
Point out that assemblies will be discussed in more detail later in the course but make sure students understand the basic concepts at this stage.

An assembly is one or more .exe or .dll files that make up a Visual Studio .NET application. Assemblies are a key concept in .NET development; they serve as a building block for all .NET applications. The .NET Framework provides many predefined assemblies for you to reference within your projects. These assemblies provide the classes and functionality of the common language runtime that enables your applications to work.

Assemblies are created automatically when you compile Visual Studio .NET source files. To create an assembly, compile your application by clicking **Build** on the **Build** menu. You can also use the command-line command **vbc.exe** to compile an assembly. Your assembly can then be referenced by other applications, in much the same way that ActiveX components can be referenced in previous versions of Visual Basic.

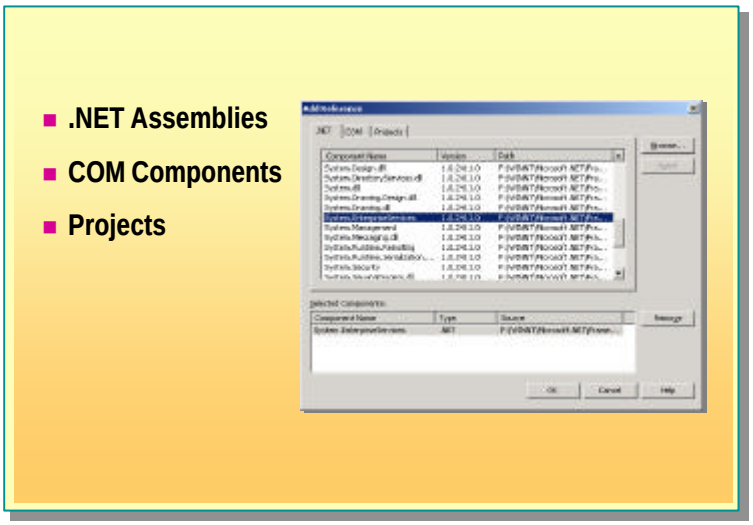
For instructor preparation purposes only

Note For more information about assemblies, see Module 10, " Deploying Applications," in Course 2373A, *Programming with Microsoft Visual Basic .NET (Prerelease)*.

Setting Project References

Topic Objective
To discuss how to reference external code by setting a project reference.

Lead-in
Most projects reference other applications or code libraries to use the functionality that they provide. Use the **Add Reference** dialog box to set project references.



You can set project references to other applications or code libraries in order to use the functionality these applications provide. You can set project references to other .NET assemblies, existing COM components, or other .NET projects within the same .NET solution.

To add a reference:

1. Select the current project in Solution Explorer. On the **Project** menu, click **Add Reference**.
2. In the **Add Reference** dialog box, select the appropriate type of reference by clicking the **.NET**, **COM**, or **Projects** tab.
3. Only projects within the same solution are displayed in the **Projects** tab.
4. Browse for the appropriate item if the item is not displayed in the list of available components.

After you set a reference, you can use it in the same way that you use COM components in previous versions of Visual Basic. You can view information about the reference in the Object Browser and create code that uses the functionality of the reference.

For training preparation purposes only

What Are Namespaces?

Topic Objective
To explain the role that namespaces play in .NET development.

Lead-in
Namespaces play an integral role in .NET development.

- **Namespaces Organize Objects Defined in an Assembly**
  - Group logically related objects together
- **Namespaces Create Fully Qualified Names for Objects**
  - Prevent ambiguity
  - Prevent naming conflicts in classes

Namespaces are used in .NET Framework assemblies to organize the objects of an assembly (classes, interfaces, and modules) into a structure that is easy to understand.

Namespaces group logically related objects together so that you can easily access them in your Visual Basic .NET code. For example, the **SQLClient** namespace defined within the **System.Data** assembly provides the relevant objects required to use a Microsoft SQL Server™ database.

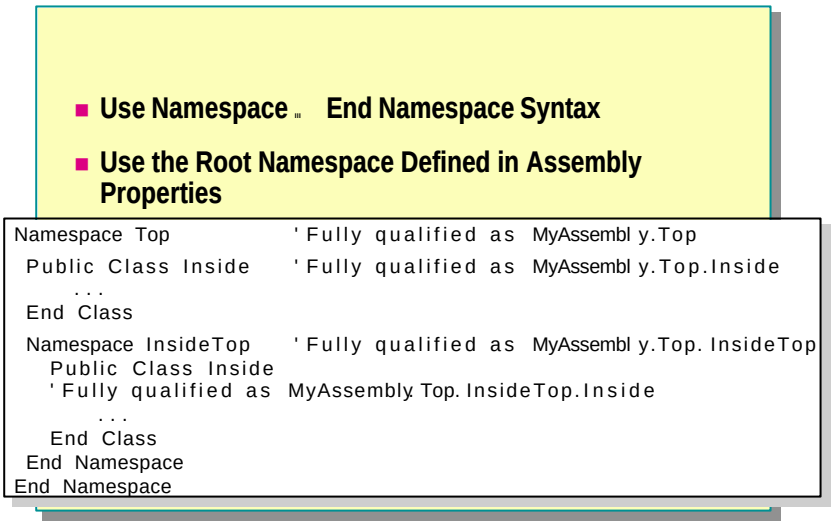
When you prefix an object with the namespace it belongs to, the object is considered to be fully qualified. Using unique, fully qualified names for objects in your code prevents ambiguity. You can declare two classes with the same name in different namespaces without conflict.

Courseware Preparation purposes only

Creating Namespaces

Topic Objective
To explain how to create namespaces.

Lead-in
You can create your own namespaces or use the namespaces that are defined in the assembly properties.



Delivery Tip
This is an animated slide. It begins by showing the bullet points only. Click the slide to reveal the following sections:
1. **Top** namespace
2. **Inside** class
3. **InsideTop** namespace

You can create your own namespaces in an assembly by creating a block of code that uses the **Namespace..End Namespace** syntax. The following example shows how to create a namespace named **Customers**:

```
Namespace Customers
  'Create classes, modules, and interfaces
  'Related to Customer information
End Namespace
```

For training preparation purposes only

The assembly usually defines a root namespace for the project that is set in the **Project Properties** dialog box. You can modify or delete this root namespace if you choose to. The following example shows code in an assembly that has a root namespace named **MyAssembly**:

```
Namespace Top
'Fully qualified as MyAssembly.Top

    Public Class Inside
    'Fully qualified as MyAssembly.Top.Inside
         ...
    End Class

    Namespace InsideTop
    'Fully qualified as MyAssembly.Top.InsideTop

        Public Class Inside
        'Fully qualified as MyAssembly.Top.InsideTop.Inside
            ...
        End Class
    End Namespace
End Namespace
```

The following example shows how code from the same assembly, but outside of the **Top** namespace, calls classes. Notice that the **MyAssembly** namespace is not required as part of the fully qualified name, because this code also resides in the **MyAssembly** namespace.

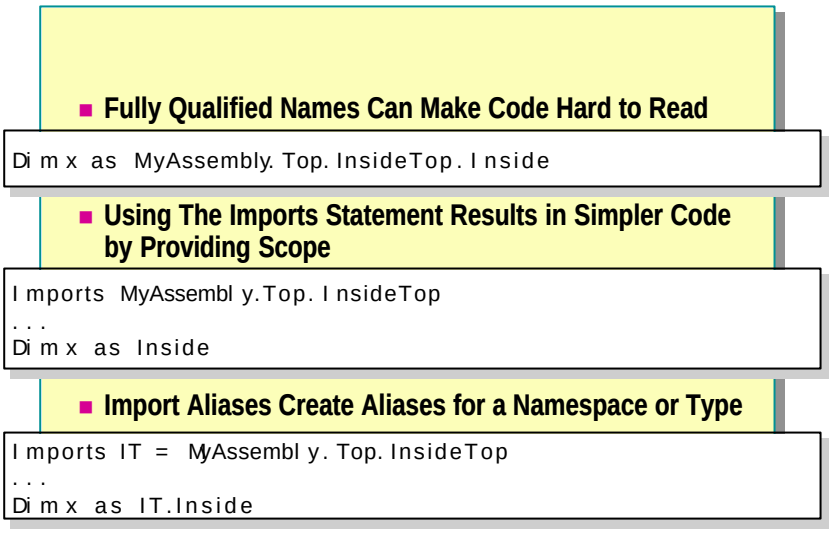
```
Public Sub Perform( )
    Dim x As New Top.Inside( )
    Dim y As New Top.InsideTop.Inside( )
    ...
End Sub
```

For trainer preparation purposes only

Importing Namespaces

Topic Objective
To explain how imports and aliases can simplify code.

Lead-in
Referencing the full namespace makes code difficult to read. You can avoid this by using the Imports statement and aliases.



You can access any object in an assembly by using a fully qualified name. The problem with this approach is that it makes your code difficult to read, because variable declarations must include the entire namespace hierarchy for you to access the desired class or interface.

Using the Imports Statement

You can simplify your code by using the **Imports** statement. The **Imports** statement allows you to access objects without using the fully qualified name. The **Imports** statement does not just point to namespaces in other assemblies. You can also use it to point to namespaces in the current assembly.

The following examples compare two methods for accessing the **InsideTop.Inside** class from an external assembly:

- Example using the fully qualified name:

```
Module ModMain
    Sub Perform( )
        'Fully qualified needed
        Dim x as New MyAssembly.Top.InsideTop.Inside( )
        ...
    End Sub
End Module
```

For training preparation purposes only

- Example using the **Imports** statement:

```
Imports MyAssembly.Top.InsideTop

Module ModMain
    Sub Perform( )
        Dim x As New Inside( )  'Fully qualified not needed
        ...
    End Sub
End Module
```

Import Aliases

You can use the **Imports** statement to create import aliases for parts of namespaces. Import aliases provide a convenient way to access items in a namespace. They prevent naming conflicts but still make code easy to write and understand.

The following example creates an import alias called **IT** for the **MyAssembly.Top.InsideTop** namespace. You can reference any item belonging to the namespace by using the **IT** import alias.

```
Imports IT = MyAssembly.Top.InsideTop

Module ModMain
    Sub Perform( )
        Dim x As New IT.Inside( )  'Alias used
        ...
    End Sub
End Module
```

For trainer preparation purposes only

Setting Project Properties

Topic Objective
To describe project properties and how to set them.

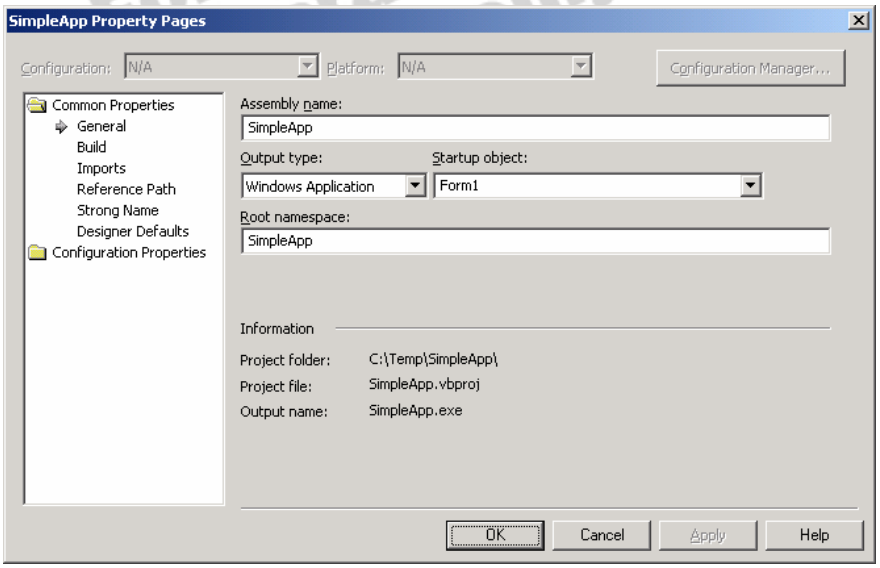
Lead-in
Visual Basic .NET provides many project properties that can affect how your project behaves.

- **Common Property Settings**
  - Defining assembly name
  - Root namespace
  - Project output type
  - Startup object
  - Importing project-level namespaces
- **Configuration Property Settings**
  - Debugging settings
  - Build options

Delivery Tip
The project property pages are shown during the next demonstration, Creating a Visual Basic .NET Project.

You can specify many project properties in the project **Property Pages** dialog box. These properties affect how the project behaves both in the IDE and after it is compiled.

The following screen shot shows the project **Property Pages** dialog box for an application named SimpleApp:



Some of the **Common Property** settings are listed below.

Property	Use this property to:
Assembly name	Specify the name of the assembly when compiled into an .exe or .dll file.
Root namespace	Change the root namespace without affecting the name of the assembly. (A default root namespace is created when you create the project.) This property affects any fully qualified names used for variable declaration.
Project output type	Choose what type of assembly is generated when your project is compiled. You can select Windows Application (.exe), Console Application (.exe), or Class Library (.dll).
Startup object	Select an entry point for your application. This is usually the main form of your application or a **Sub Main** procedure. Class libraries cannot have a startup object.
Importing project-level namespaces	Import multiple namespaces. They are then automatically accessible without forcing you to use the **Imports** statement in each file within the project.

Some of the frequently used **Configuration Property** settings are listed below.

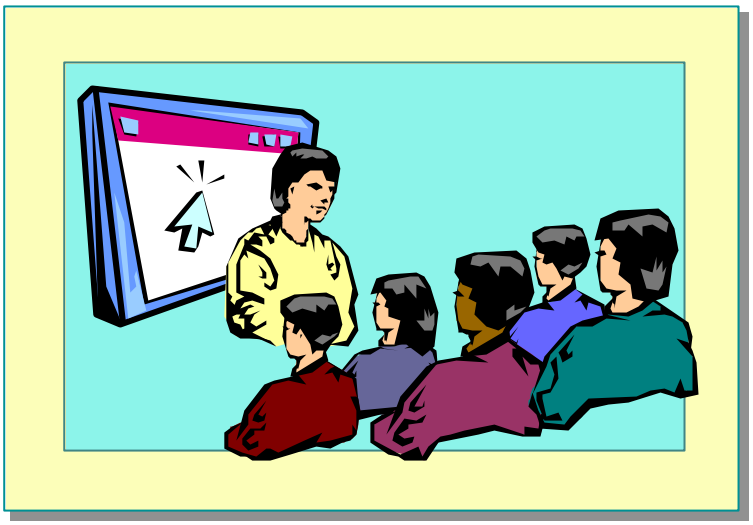
Property	Purpose
Debugging settings	These properties allow you to set debugging options, like for previous versions of Visual Basic. You can choose how your application starts up when debugging by simply starting the project, starting an external program that calls your code, or displaying a Web page from a URL that calls your code. You can also specify any command-line arguments your application needs for testing purposes.
Build options	You can specify an output directory for your compiled code (\bin is the default). You can also enable or disable the generation of debugging information contained in the .pdb file.

For trainer preparation purposes only

Demonstration: Creating a Visual Basic .NET Project

Topic Objective
To demonstrate how to create a Visual Basic .NET project.

Lead-in
This demonstration will show you how to create a Visual Basic .NET project by using the project templates.



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 create a Visual Basic .NET project based on the project templates. You will also learn about the files that comprise the project structure and how to create a reference to another assembly.

For training preparation purposes only

◆ Using Development Environment Features

Topic Objective
To introduce the topics covered in this lesson.

Lead-in
The development environment contains many enhanced features that make developing Visual Basic .NET projects faster and more efficient.

- **Using Solution Explorer**
- **Using Server Explorer**
- **Using the Object Browser**
- **Using the Task List**
- **Using Dynamic Help**
- **Using XML Features**
- **Recording and Using Macros**

Delivery Tip
Several of the IDE windows will be familiar to Visual Basic developers, so detailed discussion should not be required.

The Visual Studio .NET IDE contains several features that enable more efficient development of projects. Some of these features are enhancements of existing Visual Basic features. Others are amalgamated from other sources, such as Microsoft Visual InterDev®.

After completing this lesson, you will be able to:

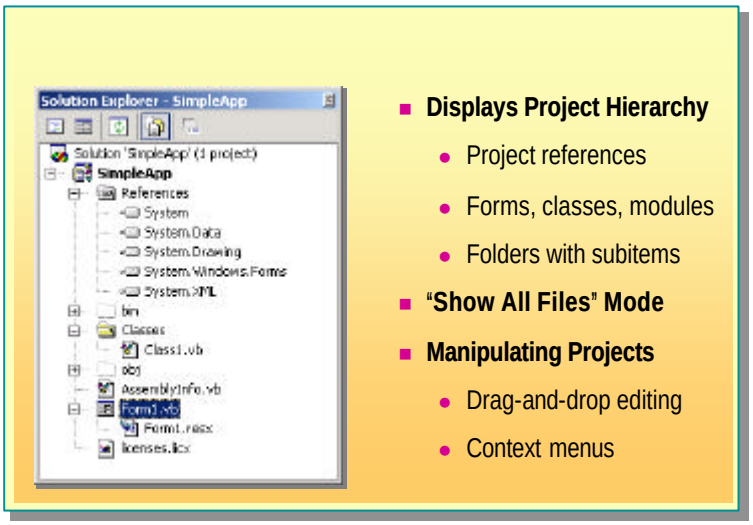
- Use IDE tools such as Solution Explorer, Server Explorer, Object Browser, and Task List.
- Use Dynamic Help while developing your Visual Basic .NET applications.
- Edit XML documents in the IDE.
- Record and use macros for repetitive tasks in your projects.

For training preparation purposes only

Using Solution Explorer

Topic Objective
To discuss how to use Solution Explorer.

Lead-in
Solution Explorer enhances the Project Explorer found in previous versions of Visual Basic.



Solution Explorer displays your project hierarchy, including all project references; project items such as forms, classes, modules, and so on; and any subfolders that contain project items. If your solution contains more than one project, you will see the same sort of hierarchy used in previous versions of Visual Basic when a project group exists.

For training preparation purposes only

"Show All Files" Mode

By default, Solution Explorer only shows some of the files stored in the project hierarchy. Certain files, which do not form an integral part of the solution, may be hidden or marked as excluded from the project, such as the files in the bin and obj folders on the slides. These files become visible when you click the **Show All Files** toolbar button. This option allows you to see items that are copied manually to the project folders. The slide associated with this topic shows a screen shot of this view of Solution Explorer.

Manipulating Projects

The following features allow you to manipulate your projects with Solution Explorer:

- Drag-and-drop editing

  You can use drag-and-drop editing to move existing project items between folders.

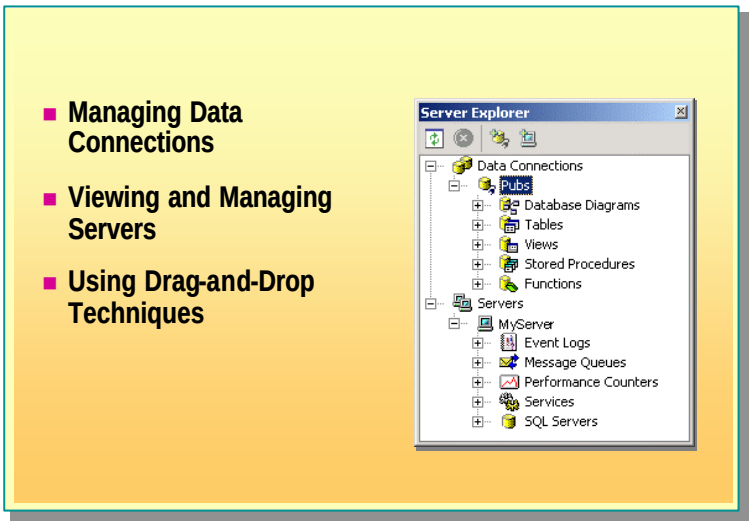
- Context menus

  Most items provide context menus that allow you to perform standard actions, such as adding to the project, deleting items from the project, and excluding items from the project, which removes the file from the project but does not delete the file. If you use Microsoft Visual SourceSafe®, you can add items to Visual SourceSafe from Solution Explorer.

Using Server Explorer

Topic Objective
To discuss Server Explorer and how it can assist in project development.

Lead-in
Server Explorer allows you to establish data connections similar to the Data View window in Visual Basic 6.0. However, Server Explorer also has the ability to manage and use specific aspects of a server.



In previous versions of Visual Basic, you can manipulate databases by using the Data View window. Server Explorer provides the same functionality and additional functionality for managing and using server components.

Managing Data Connections

To use Server Explorer to manipulate a database, add a connection to the server by clicking **Connect to Database** on the Server Explorer toolbar. This action brings up the **Data Link Properties** dialog box. After a connection is established, you can view and manipulate the database diagrams, tables, views, stored procedures, and functions.

For training preparation purposes only

Viewing and Managing Servers

You can also use Server Explorer to view and manage various server items from within the Visual Studio .NET IDE.

Server item	Purpose
Event Logs	View system event logs for application, security, and system events. The Properties window displays information about each particular event. You can use the context menu to clear the log.
Message Queues	Use message queues to send messages asynchronously between applications. You can view and manipulate any message queues located on the server by using the context menu for the item.
Performance Counters	Use the many performance counters provided by the Windows platform to monitor system-level and application-level interactions, such as the total number of logons to the server.
Services	Start and stop Windows services from Server Explorer by using context menus.
SQL Servers	View and manage Microsoft SQL Server™ databases directly from Server Explorer in the same way that you view and manage data connections.

Using Drag-and-Drop Techniques

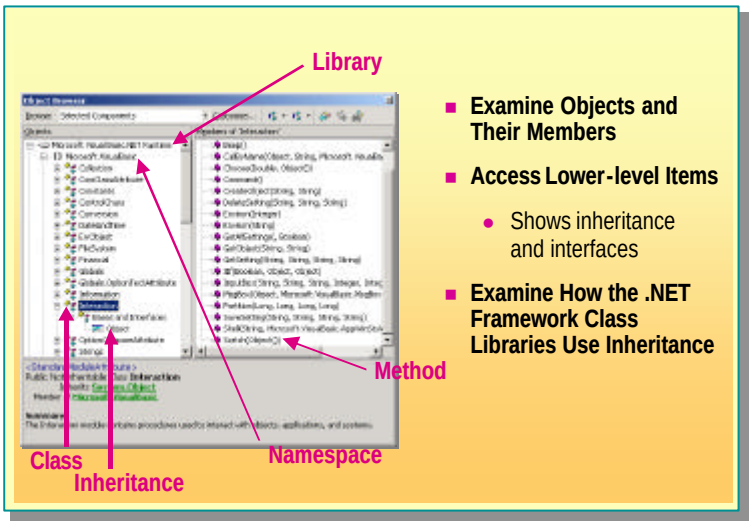
You do not use Server Explorer just for viewing and managing server items. You can use drag-and-drop techniques to place items (such as fields from a database) on your forms, or to manipulate server items (such as starting or stopping a Windows service) from within your Visual Basic .NET code.

For online preparation purposes only

Using the Object Browser

Topic Objective
To describe new features of the Object Browser.

Lead-in
Visual Basic .NET enhances the Object Browser found in previous versions of Visual Basic.

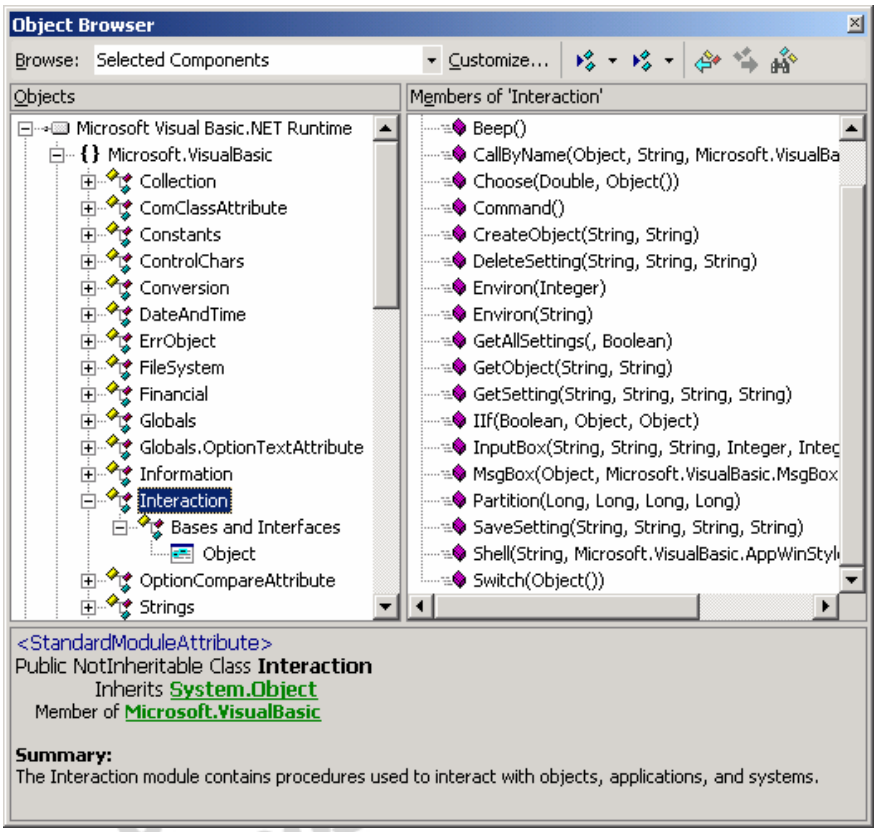


Visual Basic .NET enhances the Object Browser found in previous versions of Visual Basic. Previous versions of the Object Browser show only a high-level view of objects and their methods. Using the Visual Basic .NET Object Browser, you can:

- Examine objects and their members within a library, exploring the object hierarchy to find details about a particular method or item.
- Access lower-level items, such as interfaces and object inheritance details.
- Examine how the .NET Framework class libraries use inheritance in their object hierarchies.

For training and evaluation purposes only

The following screen shot shows the Microsoft Visual Basic .NET Runtime library and its various namespaces. This screen shot highlights the **Microsoft.VisualBasic** namespace and shows the classes it contains, including the highlighted class **Interaction**, which inherits characteristics from the **System.Object** class.

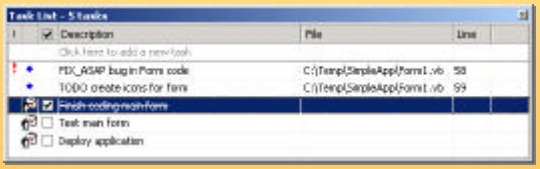


Using the Task List

Topic Objective
To describe how to use the Task List feature.

Lead-in
The IDE provides a Task List to track tasks that are awaiting completion and that are related to a particular solution.

- **Similar to the Tasks Feature in Microsoft Outlook**
- **Stored with the Solution in the .suo File**
- **Adding to the Task List**
  - You can add tasks manually by typing in appropriate field
  - Visual Basic .NET adds build errors, upgrade comments, etc.
  - You can use token strings to add comments in code



If you use Microsoft Outlook®, you may be familiar with the Tasks feature. You can use this feature to maintain a list of tasks that you are working on or tracking, and you can clear tasks when you complete them. Visual Studio .NET provides the same functionality through a Task List window, which keeps track of solution-level tasks that you must complete.

Tasks are kept in the .suo project file so that you do not lose information when you close your Visual Studio .NET session. Any stored tasks are available to all developers that use the same .suo project files.

Tasks can be added to your Task List in three ways:

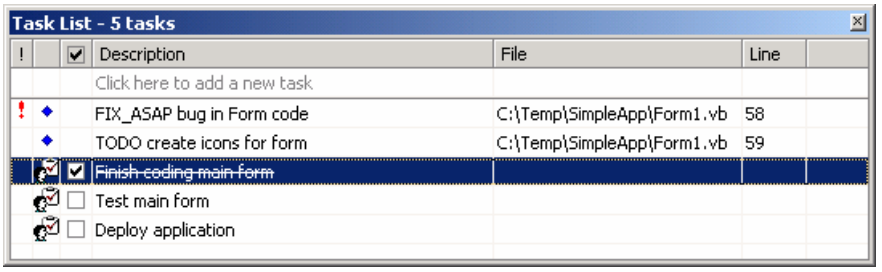
- You can manually add tasks to the task list by typing in the top row that is always visible in the Task List window.
- Visual Studio .NET automatically adds tasks to the list when you attempt to build your application, when you upgrade from a Visual Basic 6.0 project, or at various other stages during the project. This allows you to keep track of what you must do to successfully complete your project.
- You can add tasks by creating comments in your code that use specific token strings defined in the **Options** dialog box, which is accessible from the **Tools** menu. The TODO, HACK, and UNDONE tokens have been created for you, but you can define your own.

The following example shows a code section that uses the **TODO** token and a custom token named **FIX_ASAP**:

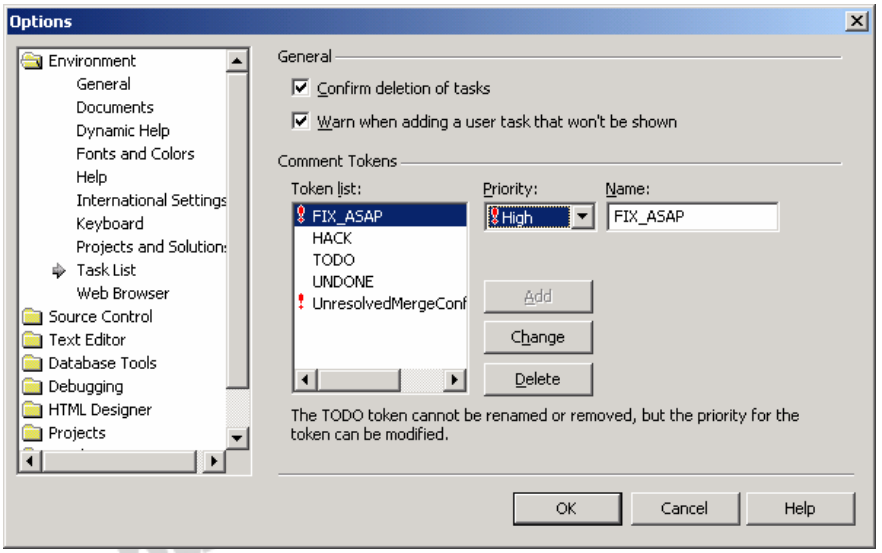
```
'TODO create icons for form
'FIX_ASAP bug in form code
```

For training preparation purposes only

The following screen shot shows how the Task List window displays information based on this example, with three extra items that have been added to the list manually:



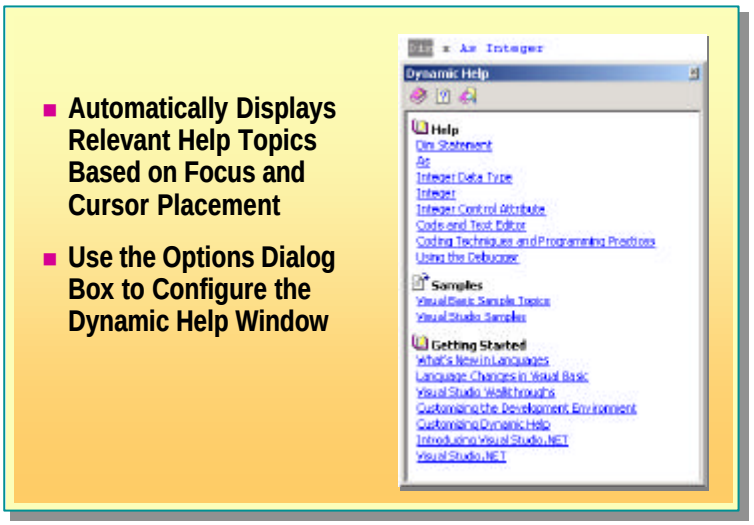
The following screen shot shows how to use the **Options** dialog box to create the **FIX_ASAP** token. Notice that the token has been created so that the items in the Task List display a High priority icon.



Using Dynamic Help

Topic Objective
To explain how to use Dynamic Help.

Lead-in
The Visual Studio .NET IDE introduces a new form of assistance that displays Help links dynamically, based on your current requirements.



Delivery Tip
Point out to students that the cursor position is not tracked if the Dynamic Help window is not displayed.

The Dynamic Help window automatically displays appropriate Help links to the .NET Help files, depending on where the cursor is and what text is highlighted. As you move from one window to another within the IDE, information displayed in the Dynamic Help window changes. If you are typing Visual Basic syntax, you see the appropriate Help topic for the syntax you are typing.

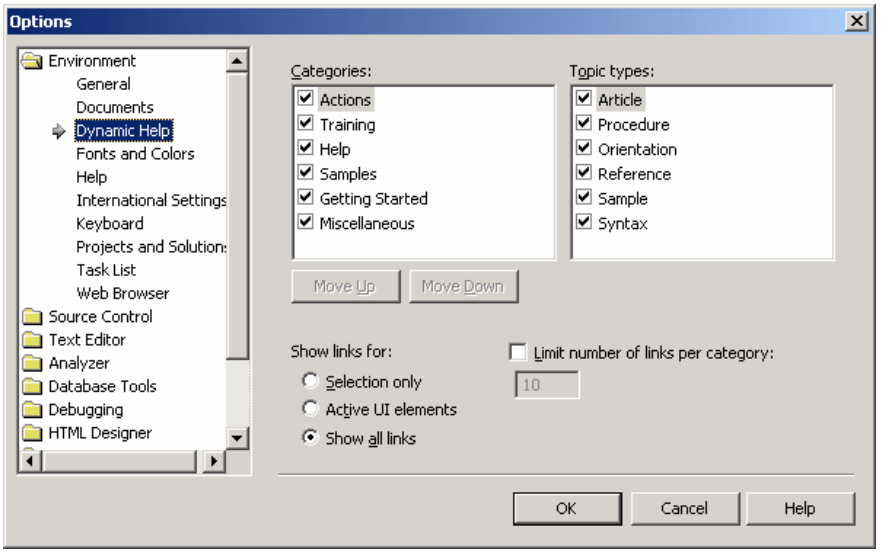
For training preparation purposes only

For example, the results that the Dynamic Help displays for the following statement vary depending on where the cursor is positioned:

```
Dim x As Integer
```

- If the cursor is positioned within the **Dim** keyword, the Dynamic Help window displays links relevant to the **Dim** keyword at the top of the list.
- If the cursor is positioned within the **Integer** keyword, the Dynamic Help window displays links relevant to integer types at the top of the list.

You can use the **Options** dialog box on the **Tools** menu to configure the items that the Dynamic Help window displays. The following screen shot shows how to use the **Options** dialog box to configure the Dynamic Help window:



For trainer preparation purposes only

Using XML Features

Topic Objective
To explain how to use the XML features provided by the IDE.

Lead-in
Many of your Visual Basic .NET applications will use XML documents to store or retrieve information. The IDE provides several XML features that make it easier for you to use these types of documents.

- **HTML and XML Document Outline Window**
- **AutoComplete**
- **Color-Coding**
- **Data View for Manipulating Data**

Delivery Tip
There is a Bookstore.xml file in the DemoCode folder for this module. You can use this file to demonstrate the Document Outline window and color-coding features.

Enterprise applications often use XML documents to specify information as part of the application architecture.

The Visual Studio .NET IDE provides several useful features for creating and editing XML documents, as described in the following table.

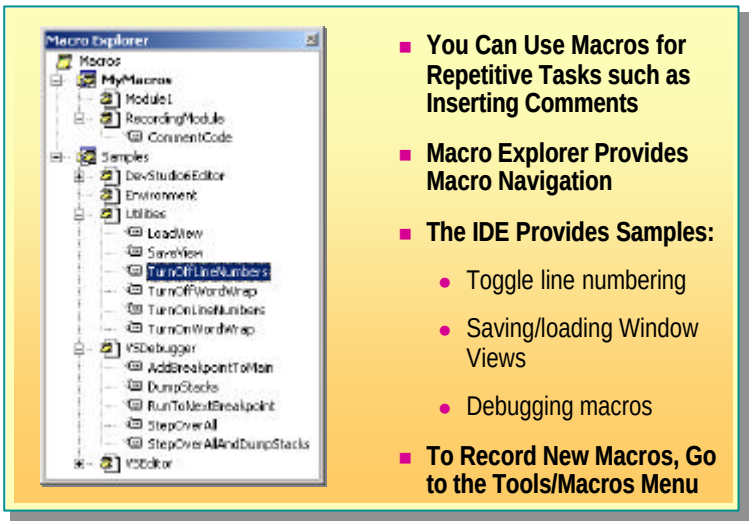
XML feature	Description
Hypertext Markup Language (HTML) and XML Document Outline window	Provides a view of the hierarchy of HTML and XML documents within the application.
AutoComplete	Automatically creates the closing tags when you create either HTML or XML starting tags. This feature can be switched off in the **Options** dialog box.
Color-coding	Assists in distinguishing tags from data.
Data View for manipulating data	Allows you to add items to your XML data hierarchy and edit existing information. Provides hyperlinks for navigation to lower-level items in the XML hierarchy.

For trainer preparation purposes only

Recording and Using Macros

Topic Objective
To explain how to record and use macros in the IDE.

Lead-in
You may be familiar with macros in Microsoft Word or Microsoft Excel. Now you can use macros in Visual Studio .NET.



Macros allow users to perform repetitive tasks with the click of a button or menu item. The Visual Studio .NET IDE provides macros, so you can automate tasks that require tedious work, such as inserting standard comments into your code.

The Macro Explorer allows you to edit, rename, delete, or run your macros within the IDE.

Several sample macros are included in the IDE, including the following:

- Toggle line numbering macros
- Saving or loading Window Views macros
- Debugging macros

You can use any of the sample macros in your projects by executing them in the Command window or placing them on menus or toolbars.

To record your own macros:

1. On the **Tools** menu, point to **Macros**, and then click **Record Temporary Macro**.
2. Perform the actions that you wish to record, such as inserting comments in the current module.
3. Click **Stop Recording** on the **Recorder** toolbar to stop recording your macro.
4. Your macro is saved with a temporary name visible in the Macro Explorer. You can rename the temporary macro to save your macro with an appropriate name.

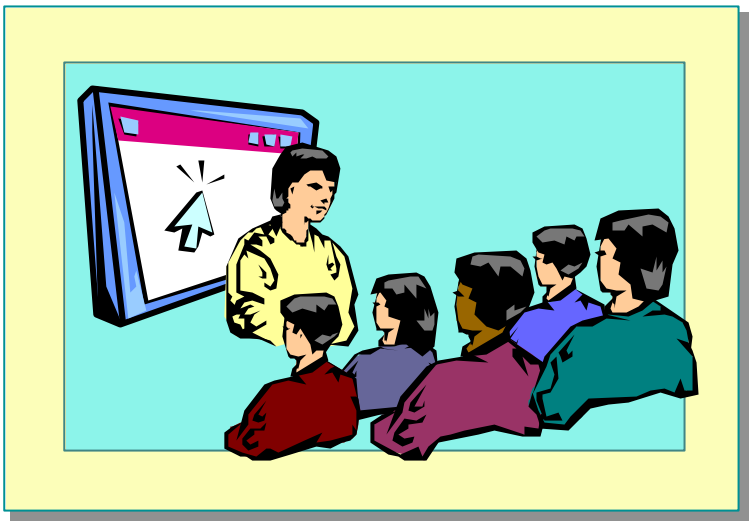
Any macros you create are stored in a subdirectory of the Visual Studio Projects folder in My Documents.

For training preparation purposes only

Demonstration: Using the Visual Studio .NET IDE

Topic Objective
To demonstrate how to use the windows that make up the Visual Studio .NET IDE.

Lead-in
This demonstration shows how to use Solution Explorer and Server Explorer, and how to edit an XML document in the Visual Studio .NET IDE.



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 several features of the Visual Studio .NET IDE, including Solution Explorer, Server Explorer, and XML editing tools.

For trainer preparation purposes only

◆ Debugging Applications

Topic Objective
To introduce the topics covered in this lesson.

Lead-in
The new IDE provides enhanced debugging features based on those found in previous versions of Visual Basic.

- **Setting Breakpoints**
- **Debugging Code**
- **Using the Command Window**

The Visual Studio .NET IDE provides enhanced versions of many of the debugging features found in previous versions of Visual Basic, along with several powerful features found in Visual C++.

After completing this lesson, you will be able to:

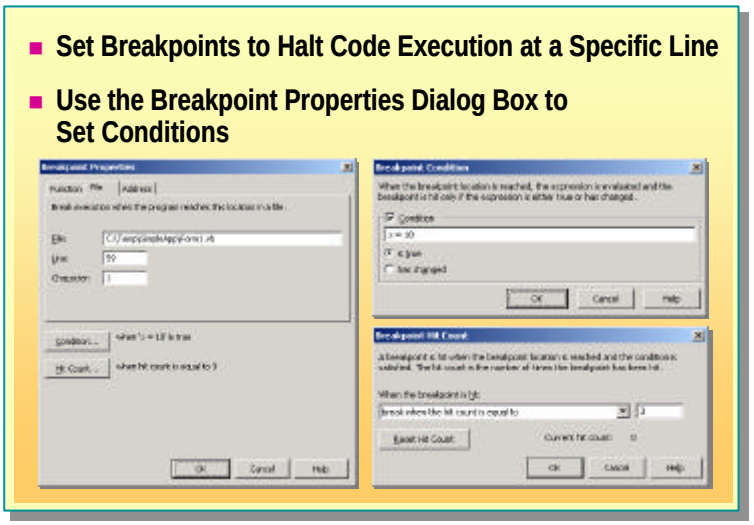
- Set breakpoints.
- Debug code in a Visual Basic .NET project.
- Use the Command window while designing and debugging applications.

For training preparation purposes only

Setting Breakpoints

Topic Objective
To explain how to set breakpoints.

Lead-in
Setting breakpoints in your project allows you to step into your code under a variety of conditions.



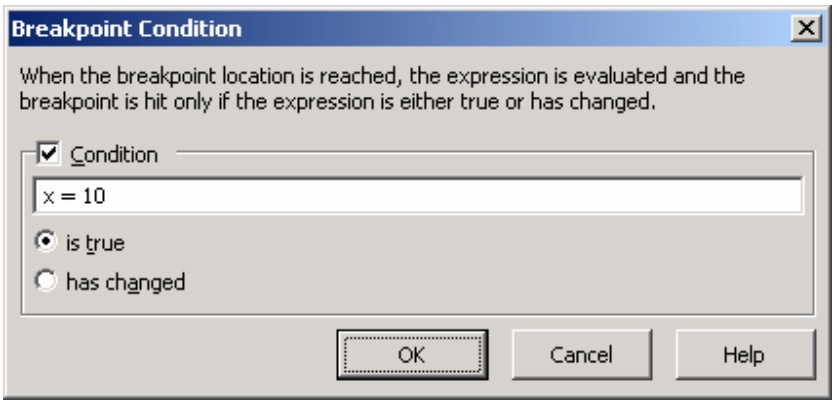
Breakpoints halt execution of code at a specific line. You can set breakpoints at design time or during a debugging session.

There are several ways you can set a breakpoint:

- Click the margin to the left of the code window on the line containing the statement where you want the debugger to halt.
- On the **Debug** menu, click **New Breakpoint**, and choose from the various options.
- Place the cursor on the line where you want the debugger to halt. Press **F9** to switch the breakpoint on or off.

You can use the **Breakpoint Properties** dialog box to make a conditional breakpoint. This feature works in a way similar to watch expressions in previous versions of Visual Basic. You set a breakpoint condition that only halts execution when a particular condition is true or when a variable has changed.

The following screenshot shows a breakpoint condition that only halts when a variable *x* has a value of *10*.



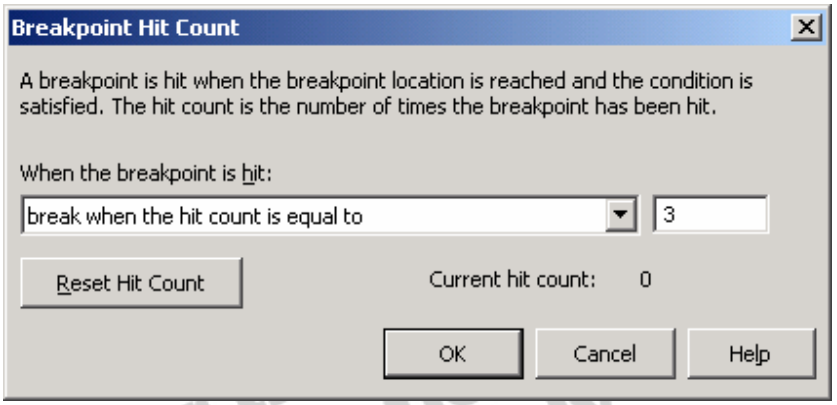
For trainer preparation purposes only

You may also want to halt execution only when the breakpoint has been reached and the breakpoint condition has been satisfied a specific number of times. This number is called the *hit count*.

To set a breakpoint hit count:

1. In the **Breakpoint Properties** dialog box, click **Hit Count**.
2. In the **Breakpoint Hit Count** dialog box, choose the type of hit count test that you want to perform from the drop-down combo box, enter the appropriate hit count value, and then click **OK**.

The following screen shot shows how you specify that you want execution to stop the third time that the breakpoint is reached and the breakpoint condition is satisfied:



For preparation purposes only

Debugging Code

Topic Objective
To discuss how to debug code and use the various debugging windows.

Lead-in
Several aspects of debugging in Visual Basic .NET will be familiar to Visual Basic developers.

- **Use the Debug Menu or Toolbar to Step Through Code**
- **Use the Debugging Windows:**
  - Locals: to view and modify local variables
  - Output: to view output from the compiler
  - Watch: to view watch expressions
  - Call Stack: to view call history, including parameter information
  - Breakpoints: to view, add, or temporarily disable breakpoints

Debugging your code in Visual Basic .NET is similar to debugging code in previous versions of Visual Basic. When code execution stops at the breakpoint, you can step through the code by using the **Debug** menu or toolbar.

All of the debugging windows found in previous versions of Visual Basic are available in Visual Basic .NET, but with some enhancements.

For training preparation purposes only

Debug window	Use this window to:
Locals	View and modify variables. This window provides explicit details about objects, such as inheritance information. The tree view of this window is particularly useful for viewing values in an object hierarchy.
Output	View output information from the compiler, such as the number of compilation errors that occurred and what libraries were loaded. You can use the **Debug.Writeline** statement to print information to the Output window. This statement replaces the **Debug.Print** statement in previous versions of Visual Basic.
Watch	View and manipulate any watch expressions. To add values to the Watch window, type in the **Name** column of an empty row, or click **Quick Watch** on the **Debug** menu. This allows you to quickly add watch expressions during your debugging session. Unlike in previous versions of Visual Basic, you cannot set watch conditions. These have been replaced by breakpoints conditions in Visual Basic .NET.

(*continued*)

Debug window	Use this window to:
Call Stack	View the history of calls to the line of code being debugged. This window displays the history of the call, including any parameters to procedures and their values.
Breakpoints	View a list of current breakpoints, including information such as how many times the breakpoint has been called, and the conditions the breakpoint has met. You can also add new breakpoints and temporarily disable breakpoints in this window.

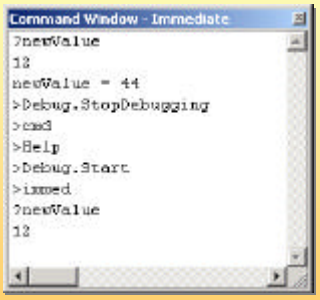
For trainer preparation purposes only

Using the Command Window

Topic Objective
To explain the purpose of the Command window.

Lead-in
The Command window combines features found in the Immediate window of previous versions of Visual Basic with a command-line utility.

- **Immediate Mode**
  - Similar to the Immediate window
- **Command Mode**
  - Use Visual Studio IDE features
- **Switching Modes**
  - Use **>cmd** to change to Command mode
  - Use **immed** to return to Immediate mode



In Immediate mode, the Command window in Visual Basic .NET provides functionality similar to that found in the Immediate window in previous versions of Visual Basic. You can query local variables while debugging and change their values under certain conditions. You can also run procedures in your code or other .NET Framework class libraries while you are in Immediate mode.

For training preparation purposes only

The Command window also has a second purpose. In Command mode, you can use features of the Visual Studio .NET IDE. The features you can use while in Command mode include the following:

- The **Debug.Start** command, to start debugging
- The **Help** command, to display the Visual Studio .NET documentation
- The **Exit** command, to quit the Visual Studio .NET IDE
- Any macros that you recorded
- Any macros that the IDE provides as samples

To switch between the two modes of the Command window:

- Use the **>cmd** command to switch from Immediate mode to Command mode.

  You can issue single commands in Immediate mode by prefixing your command with the > symbol.
- Use the **immed** command to switch from Command mode to Immediate mode.

The following example shows various commands in both Immediate and Command mode. The window is initially in Immediate mode during a debugging session.

```
?newValue
12
newValue=44
?newValue
44
>Debug.StopDebugging
>cmd
>help
>Debug.Start
>immed
?newValue
12
```

The following steps are executed in this code:

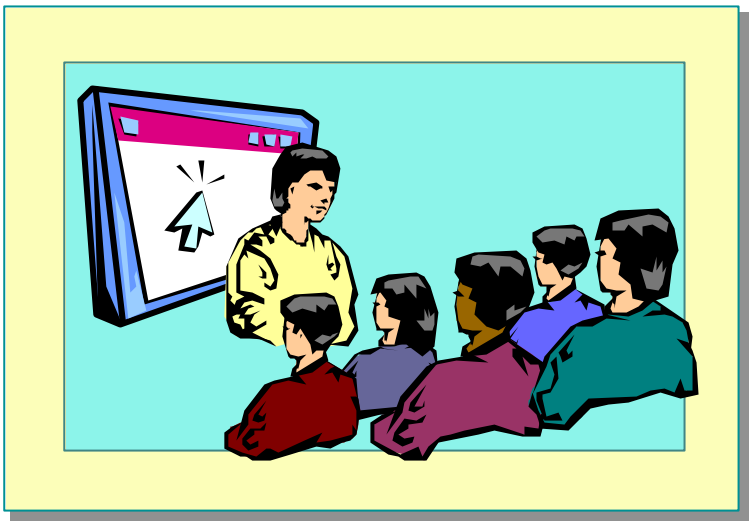
1. The example shows a local variable named *newValue* with a value of 12.
2. In Immediate mode, this value is changed to 44.
3. The variable is queried again to confirm the change.
4. A single command is issued to stop debugging.
5. The **cmd** command is used to switch to Command mode.
6. The **help** command is used to display the Visual Studio .NET documentation.
7. The **Debug.Start** command is used to start debugging.
8. The **immed** command is used to switch back to Immediate mode.
9. The *newValue* variable is tested again.

for trainer preparation purposes only

Demonstration: Debugging a Project

Topic Objective
To demonstrate how to debug a simple Visual Basic .NET project.

Lead-in
This demonstration shows how to debug a simple Visual Basic .NET project.



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 debugging features of the Visual Studio .NET IDE to debug a simple Visual Basic .NET project.

For trainer preparation purposes only

◆ Compiling in Visual Basic .NET

Topic Objective
To introduce the topics covered in this lesson.

Lead-in
Compiling an application in Visual Basic .NET.

- **Locating Syntax Errors**
- **Compilation Options**

After completing this lesson, you will be able to:

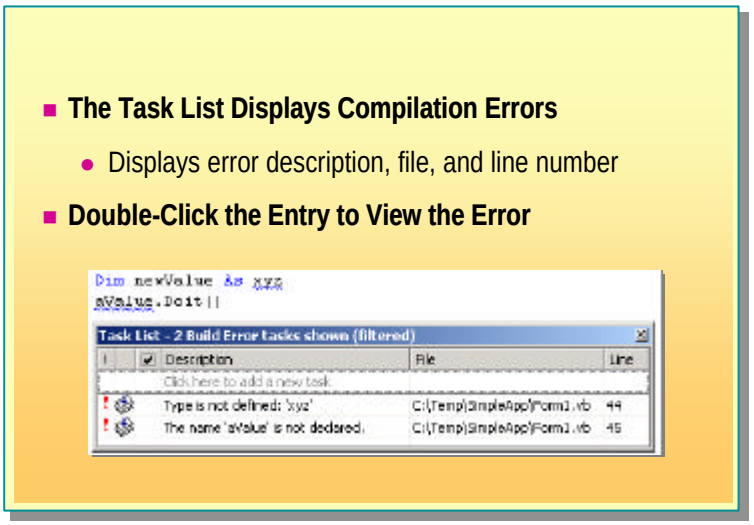
- Locate syntax errors when you attempt to build your application.
- Select the best compilation option for building your Visual Basic .NET projects.

For trainer preparation purposes only

Locating Syntax Errors

Topic Objective
To explain how to locate syntax errors when attempting to build an application.

Lead-in
You can immediately address syntax errors when you attempt to build your project.



Visual Basic .NET displays compilation errors as you type each statement in your application code. If you ignore these warnings and attempt to build your application, the Task List is displayed, with all build errors included on the list.

Information about the error includes the error description, the file in which the error occurred, and the line number. The error description is the same information that you see if you position the cursor over the highlighted part of your code in the code window.

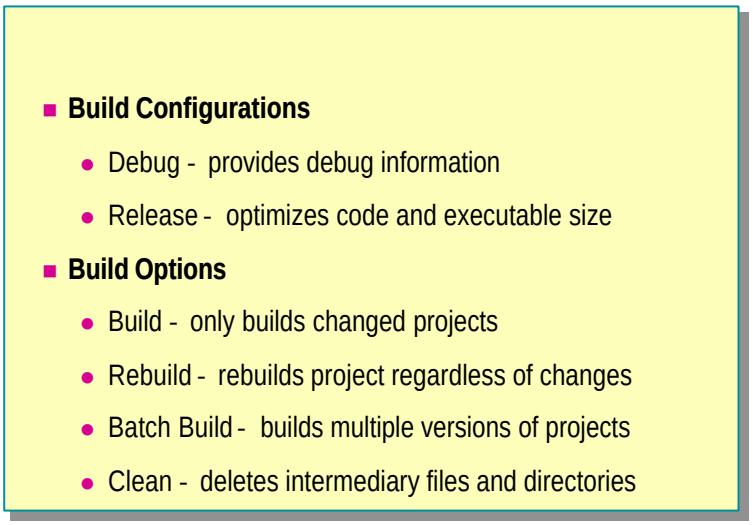
You can edit the errors by double-clicking the appropriate entry in the Task List. This positions the cursor in the correct file and exact line where the error is located, so you can make the required modifications. As soon as you complete your changes and you move off the modified line, the Task List entries are updated.

Preparation for classroom purposes only

Compilation Options

Topic Objective
To describe the options available when you compile a project.

Lead-in
There are several compilation options available to you when you compile your Visual Basic .NET projects.



The Visual Studio .NET IDE provides several compilation options for building your Visual Basic .NET projects.

For trainer preparation purposes only

Build Configurations

There are two types of build configurations for Visual Basic .NET projects:

- Debug

  During the development phase, you may want to build and test your applications by using compiled assemblies. The Debug configuration produces a .pdb file that contains debugging information. Other applications can use this file to debug your code. To assist these other applications, no optimizations are made to your code. Other applications have access to your complete and original code.

- Release

  After testing is completed, you will want to deploy your application to client computers. The Release configuration performs various code optimizations and attempts to minimize the size of the executable file. No debugging information is generated for a Release configuration build.

Build Options

You can choose what to build by selecting the appropriate **Build** menu options.

- **Build**

  The **Build** option only builds project items whose code has changed since they were last compiled.

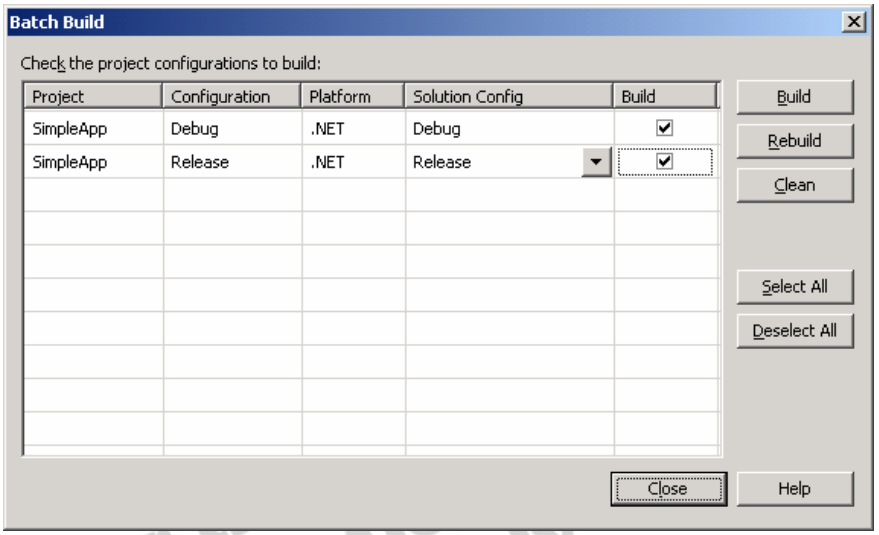
- **Rebuild**

  The **Rebuild** option compiles all project items even if they have not been modified since they were last compiled. Use this option when you want to be sure your application contains the latest code and resources.

- **Batch Build**

  To build more than one version of a project (or multiple projects if they are loaded in a solution), you can use the **Batch Build** option. Use this option if you want to build both Debug and Release versions of your project. The different builds will be created in the \\obj\Debug and \\obj\Release subdirectories of the project directory.

  The following illustration shows the **Batch Build** dialog box. It specifies that both the Debug and Release versions of the *SimpleApp* project will be built.



- **Clean**

  The **Clean** option allows you to create a Clean build of your projects by deleting all intermediary files and directories. You can use this option to make sure that you do not inadvertently leave previous versions of intermediary files and directories on your system.

For preparation purposes only

Lab 2.1: Exploring the Development Environment

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will explore the development environment and debug a simple application.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Use the Visual Studio .NET IDE.
- Create a simple Visual Basic .NET project.
- Set conditional breakpoints.
- Debug an application.
- Use the Task List and Command windows.

For trainer preparation purposes only

Prerequisites

Before working on this lab, you must have experience with developing applications in an earlier version of Visual Basic.

Scenario

In this lab, you will explore the Visual Studio .NET IDE and use its features to create a data connection and view event log information. You will create a simple Windows-based application and add a prewritten form to the project. Finally, you will debug the application by using the various debugging features of the IDE.

Starter and Solution Files

There are starter and solution files associated with this lab. The starter files are in the *install folder*\Labs\Lab021\Ex02\Starter folder, and the solution files are in the *install folder*\Labs\Lab021\Ex02\Solution folder.

Estimated time to complete this lab: 45 minutes

Exercise 1
Becoming Familiar with the Visual Studio .NET IDE

In this exercise, you will use Server Explorer to create a data connection for the Northwind SQL Server database. You will investigate Server Explorer, and the event logs in particular. You will then view the **Options** dialog box to become familiar with the default IDE settings.

The purpose of this exercise is for you to become familiar with the IDE, so take time to explore any parts of the IDE that are interesting to you.

↙ To add a data connection by using Server Explorer

1. Open Visual Studio .NET.
2. On the **View** menu, click **Server Explorer**.
3. On the toolbar, click **Connect to Database**. Use the following values to complete the **Data Link Properties** dialog box:

Property	Value
Server name	**localhost**
Logon information	**Windows NT Integrated security**
Database	**Northwind**

4. Click **Test Connection** to verify that you have successfully made the connection, and then click **OK**.
5. Click **OK** on the **Data Link Properties** dialog box.
6. If you are not familiar with the Data View window from previous versions of Visual Basic or Microsoft Visual InterDev®, explore the list of tables, views, and stored procedures by expanding the newly created ***servername*.Northwind.dbo** data connection.

For trainer preparation purposes only

↙ To explore the Application event log

1. Under the **Servers** node of Server Explorer, expand the name of your computer.
2. Expand the **Event Logs** node, and then expand the **Application** node.
3. Select an **EventLogEntry** node and view the application entry information in the Properties window.

↙ To explore the default IDE configuration options

1. On the **Tools** menu, click **Options**.
2. Spend several minutes becoming familiar with the default Environment settings.

Exercise 2
Creating a Visual Basic .NET Project

In this exercise, you will create a simple Visual Basic .NET project and remove the default form from the project. You will then add a prewritten form to the project and change the **Startup object** property of the project.

The prewritten form displays a text box and a command button. When you press the button, the value in the text box is sent to a subroutine. This subroutine verifies that the value is not empty and displays a message based on the value. If the value is empty, an error message appears.

↙ To create a new project

1. On the **File** menu, point to **New**, and then click **Project**.
2. In the **Project Types** box, click the **Visual Basic Projects** folder.
3. In the **Templates** box, click **Windows Application**.
4. Change the name of the project to **FirstApp**, set the location to *install folder*\Labs\Lab021\Ex02, and then click **OK**.

↙ To add the test form

1. In Solution Explorer, right-click **Form1.vb**, click **Delete**, and then confirm the deletion warning.
2. On the **Project** menu, click **Add Existing Item**.
3. Set the location to *install folder*\Labs\Lab021\Ex02\Starter, click **frmDebugging.vb**, and then click **Open**.
4. Using Solution Explorer, click **frmDebugging.vb**, and then click the **View Code** button.
5. Examine the code in the **btnDebug_Click** and **PerformValidation** procedures, and ensure that you understand the purpose of the code.

For trainer preparation purposes only

↙ To set the project startup property

1. In Solution Explorer, right-click **FirstApp**, and then click **Properties**.
2. In the **Startup object** list, click **frmDebugging**, and then click **OK**.
3. On the **File** menu, click **Save All** to save the project.

Exercise 3
Using the Debugger

In this exercise, you will use the Visual Studio .NET debugger to debug the simple application that you created in the previous exercise.

You will set a breakpoint to halt execution in the **btnDebug_Click** event handler and use the debugger to step through the subroutine. You will examine the parameter passed to the **PerformValidation** procedure and change the value by using the Locals window. You will then step through the rest of the code and verify that the correct message appears. Finally, you will modify the breakpoint so that it is conditional, and use the Command window to perform various IDE functions.

↙ To set a breakpoint

1. On the **View** menu, point to **Other Windows**, and then click **Task List**.
2. Right-click anywhere within the Task List window, point to **Show Tasks**, and then click **All**.
3. Double-click the single **TODO** task to navigate to the comment in the code.
4. Place the pointer on the line immediately after the TODO comment and press F9, the breakpoint shortcut key.

↙ To debug the project

1. On the **Debug** menu, click **Start**.
2. Enter any value into the text box and then click **Debug**.
3. When the program execution halts, on the **Debug** menu, click **Step Into**.
4. Continue to click **Step Into** until the **PerformValidation** procedure begins execution.
5. Examine the contents of each of the following windows: Locals, Breakpoints, and Call Stack.
6. In the Locals window, change the value of the **strValue** variable to a new value. Do not forget to include the quotation marks around the new value. Press ENTER.
7. Step through the remaining lines of code, closing any message boxes, until the form appears again.

For trainer preparation purposes only

↙ To modify the breakpoint

1. While the form is displayed, move to the Breakpoints window in the IDE.
2. Right-click the breakpoint, and then click **Properties**.
3. Click **Condition**, and then set the following condition value:

Condition	Break When
txtValue.Text	has changed

4. Click **OK** in the **Condition** dialog box, and then click **OK** in the **Breakpoint Properties** dialog box.
5. On the form, click **Debug**. This time your code should execute without debugging.
6. Change the value in the text box and click **Debug**. This will cause execution to halt because you have met the condition of the breakpoint.
7. On the **Debug** menu, click **Continue** to allow the execution to complete.
8. In the Breakpoints window, clear the breakpoint check box to disable the breakpoint. Verify that execution no longer halts, even if you change the value in the text box.

↙ To use the Command window

1. Display the Command window and enter the following command: **Debug.StopDebugging**. The debugging session will end and the IDE will return to the design state.
2. If the Command window is no longer displayed, on the **View** menu, point to **Other Windows**, and then click **Command Window**.
3. In the Command window, enter the **Exit** command to quit Visual Studio .NET.

For trainer preparation purposes only

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 the Integrated Development Environment
- Creating Visual Basic .NET Projects
- Using Development Environment Features
- Debugging Applications
- Compiling in Visual Basic .NET

---

1. List the file extensions for the following Visual Basic .NET files: Visual Basic .NET project files, classes, and modules.

   .vbproj, .vb, and .vb

For trainer preparation purposes only

2. Describe the purpose of namespaces and the **Imports** keyword.

   Namespaces organize the objects and items found in an assembly and prevent ambiguity when calling an object.

   The Imports keyword allows you to access an object from within a namespace without using the object's fully qualified name.

3. Describe the purpose of Server Explorer.

   Server Explorer allows you to view and manipulate databases and various server items, such as message queues, event logs, Windows services, and XML Web Services. You can also use Server Explorer to access these items from within your code.

4. The Object Browser is exactly the same as in previous versions of Visual Basic. True or false? If false, explain why.

   False. The Object Browser has been enhanced to include inheritance and interfaces in the object hierarchy.

5. Describe the purpose of a conditional breakpoint and how to create one.

   Conditional breakpoints halt execution when a particular condition is met, such as when a variable equals a certain value.

   To set a conditional breakpoint, you add a standard breakpoint, and then use the Breakpoint Properties dialog box to modify the conditions.

6. You can only build one version of an application at a time. True or false? If false, explain why.

   False. Using the Batch Build dialog box, you can specify which version of the application you want to build, such as Debug and Release.

For trainer preparation purposes only