msdn® training

Module 11: Upgrading to Visual Basic .NET

Contents

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

Microsoft®

Instructor Notes

Presentation:
60 Minutes

Lab:
00 Minutes

This module provides students with the knowledge needed to decide whether to upgrade existing Microsoft® Visual Basic® 6.0–based applications and describes the options that are available for upgrading. Students will receive recommendations for which types of applications to upgrade and which to leave alone. Finally, they will learn the tasks that are necessary before and after using the Visual Basic Upgrade Wizard and view a demonstration of the wizard.

After completing this module, students will be able to:

- Make an informed decision about whether to upgrade an application.
- Describe the various upgrade options available.
- Use the Upgrade Wizard.

Materials and Preparation

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

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2373A_11.ppt
- Module 11, " Upgrading to Visual Basic .NET"

Preparation Tasks

To prepare for this module, you should:

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

Demonstration

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

Using the Upgrade Wizard

↙ To review the application

1. Open Visual Basic version 6.0, and open the file Invoices.vbg from the *install folder*\DemoCode\Mod11 folder.
2. Describe the two projects in the project group, and give a synopsis of each of the classes and its purpose.
3. Step through the running application, explaining each of the main features.
4. Close the running application, and close Visual Basic version 6.0.

↙ To upgrade the user interface

1. Open Visual Basic .NET, and open the file InvoicesUI.vbp from the *install folder*\DemoCode\Mod11 folder.
2. This will invoke the Upgrade Wizard. Accept the defaults on all pages, explaining the options that are available.

↙ To upgrade the data access tier

1. On the **File** menu, point to **Add Project**, and then click **Existing Project**. Select Invoices.vbp from the *install folder*\DemoCode\Mod11 folder, and click **Open**.
2. This will invoke the Upgrade Wizard. Accept the defaults on all pages, explaining the options that are available.
3. Save the solution.

↙ To update references

1. Remove the **Interop.Invoices_1_0** reference from the **InvoicesUI** project.
2. Right-click **References**, and then click **Add Reference**. On the **Projects** tab click **Invoices**. Click **Select**, and then click **OK**.

↙ To review the upgraded application

- Run the application, demonstrating that the code is now functioning as expected.

↙ To review the issues identified by the Upgrade Wizard

1. Review the comments added to each of the classes in the Invoices project. Note the hyperlinks to Help topics.
2. Review comments in the modules of the InvoicesUI project.
3. If time allows, go through and make appropriate changes to the code, using the hyperlinks for further information about the issues.

↙ To discuss further modifications

- If time allows, discuss the other options available for further enhancements. These are mentioned in the student notes and can lead to a class discussion.

Module Strategy

Use the following strategy to present this module:

- Deciding Whether to Upgrade

  In this lesson, you will discuss the advantages and disadvantages of upgrading an application created in Visual Basic 6.0 to Visual Basic .NET. You will talk about the advantages—such as enhanced scalability, improved performance, and ease of deployment—and compare these to the disadvantages of time and money. This is obviously a subjective issue because no two projects will ever have the same advantages and costs, but the goal of the lesson is to make students aware of the factors that they need to consider.

- Options for Upgrading

  This lesson describes the three options for upgrading: full upgrade, partial upgrade, and full rewrite.

- Recommendations

  This is a single slide that outlines Microsoft's recommendations for the upgrading of different types of applications.

- Performing the Upgrade

  This lesson covers the details of the actual upgrade process. It begins by discussing what you need to do to the Visual Basic 6.0–based application to prepare it for the upgrade. These changes ensure that the Upgrade Wizard can maximize its output.

  You will then discuss how to use the Upgrade Wizard and what the resulting code and project will look like. Remember that you have spent the whole course discussing Visual Basic .NET and that the goal of this topic is to point out the upgrade-specific issues, not the general Visual Basic .NET code. You will also discuss the lists of problematic pieces of code that the wizard creates, including the Upgrade Report and Task List.

  Finally, you will demonstrate the Upgrade Wizard. If time allows, this demonstration can lead into a class discussion about further possible ways to enhance the project.

Overview

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

Lead-in
In this module, you will learn about upgrading applications created in Visual Basic 6.0 to Visual Basic .NET.

- **Deciding Whether to Upgrade**
- **Options for Upgrading**
- **Recommendations**
- **Performing the Upgrade**

As you have seen throughout this course, there are some fundamental changes in Microsoft® Visual Basic® .NET version 7.0. These changes are necessary because Visual Basic .NET is a significant upgrade that takes full advantage of the Microsoft .NET Framework.

Because of these changes, you will find that upgrading applications to Visual Basic .NET might take time and effort, but it does allow you to take advantage of the new features in the .NET Framework. The Visual Basic Upgrade Wizard has been provided as a step in the upgrade process to help you upgrade, but there are tasks that you should complete both before and after its use.

In this module, you will learn the factors you must consider when deciding whether to upgrade an existing application, the options you have for upgrading, and how to use the Upgrade Wizard.

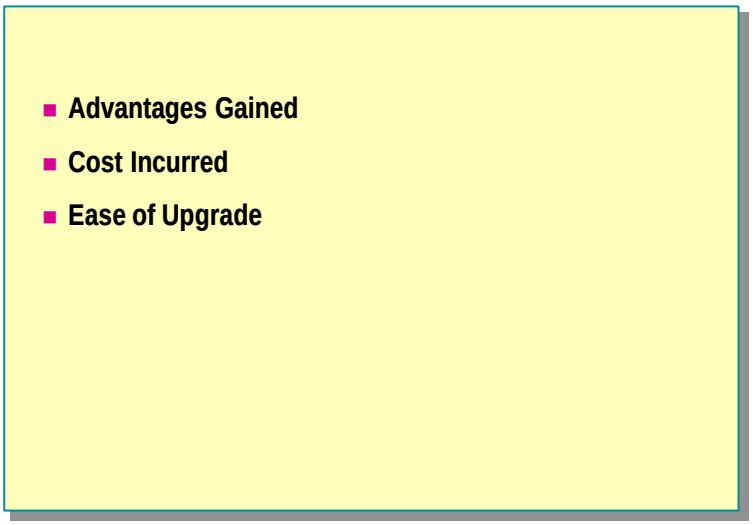
After completing this module, you will be able to:

- Make an informed decision about whether to upgrade an application.
- Describe the various upgrade options available to you.
- Use the Upgrade Wizard.

Deciding Whether to Upgrade

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

Lead-in
Upgrading an application is not a necessity; you must examine the advantages and disadvantages of doing so before deciding to upgrade.



You must consider various factors when deciding whether to upgrade an application. In some situations, the advantages gained from porting the application to the .NET Framework will greatly outweigh the costs involved. In other situations, you might decide that the advantages are not worth the investment. Upgrading is not a necessity, and you should carefully examine the advantages and disadvantages before starting the process.

After completing this lesson, you will be able to:

- Evaluate the advantages and disadvantages of the upgrade process.
- Identify how to decide when to upgrade your applications to Visual Basic .NET.

Advantages Gained

Topic Objective
To discuss the advantages that can be gained from upgrading Visual Basic 6.0-based applications to Visual Basic .NET.

Lead-in
Upgrading your Visual Basic 6.0- based applications to Visual Basic .NET may result in improved performance and scalability.

- **Scalability**
- **Performance**
- **Deployment**
- **Access to Rich Set of Base Classes**
- **Better Debugging**
- **Solves DLL Conflicts**
- **Maintenance**

Delivery Tip
Remind students that there are many advantages to using the .NET Framework, but here we are considering the main advantages to upgrading an existing application.

The .NET Framework provides many benefits to the application developer that may enhance applications created in Visual Basic version 6.0.

Advantages

Three major advantages upgrading your application to Visual Basic .NET provides are:

- Scalability

  ADO .NET enhances scalability by means of the disconnected data architecture, which reduces the number of concurrent database connections necessary, thereby reducing the overhead needed to run the application.

  ASP .NET's state management system improves upon that of Active Server Pages (ASP). Session state can be shared among many servers in a Web farm, allowing for greater scalability.

- Performance

  ADO .NET is a simplified version of Microsoft ActiveX® Data Objects (ADO). It is designed around Extensible Markup Language (XML) to work seamlessly with disconnected data. The **DataReader** object is designed for speed and greatly increases the performance of data intensive applications.

  ASP .NET has improved performance over ASP and other Web development technologies. ASP .NET is a compiled .NET-based environment, which will run faster than existing applications, and allows you to use early binding throughout your applications.

- Deployment

  Deployment is greatly simplified in the .NET Framework. Depending on the complexity of your application, deployment can entail running an application directly from a server, using XCOPY to copy the application to a workstation or Web server, or installing by using Microsoft Windows® Installer.

Other advantages include:

- Maintenance
- Access to Rich Set of Base Classes
- Better Debugging
- Solves DLL Conflicts

Cost Incurred

Topic Objective
To discuss the potential costs of upgrading an application to Visual Basic .NET.

Lead-in
Costs can be measured in a number of ways.

- **Time to Upgrade May Trade-Off Against Future Maintenance Time**
- **May Require Redesign, As Well As Upgrading and Recoding**
- **Financial Costs Can Be Spread by Upgrading an Application Section by Section**

The costs you may incur in upgrading an application can be measured in terms of time, effort, and ultimately finance.

It will take you time to upgrade your applications from Visual Basic 6.0 to Visual Basic .NET; however, that time may actually recoup itself in the reduced maintenance time associated with the upgraded application or be outweighed by the benefits obtained by the upgrade. Visual Basic .NET–based applications can require less maintenance because of the improvements associated with the .NET Framework. XCOPY deployment ends DLL conflicts.

Some applications will gain little benefit from simply upgrading the existing application to Visual Basic .NET. These include applications that may have been upgraded through various versions of Visual Basic and never redesigned to take full advantage of the current systems architecture. The costs of redesigning an application will greatly increase the overall cost, and may be a deciding factor in your choice.

Some application architectures lend themselves to a gradual upgrade process over a period of time. For example, an application using a number of classes that contain data access code can be upgraded in a number of steps. First you can upgrade the user interface, then you can upgrade the middle-tier components, and then you can recode the existing ADO code to ADO .NET in the data tier.

Ease of Upgrade

Topic Objective
To discuss the factors that affect the ease of upgrading.

Lead-in
The ease with which an application can be upgraded is affected by various factors, including the structure of the code, the project types involved, and the language features used.

- Modularity of Code
- Project Types
- Control Types
- Language Constructs

There are a variety of factors that will affect how easy it is to upgrade an application. These include the original application architecture, the modularity of the code in the application, the types of projects and controls used in application, and the language constructs used.

Modularity of Code

Because Visual Basic .NET supports object-oriented features not available in Visual Basic 6.0, it is easier to upgrade modular code than non-modular code. If an application has been designed in a modular fashion, changes to one component should not adversely affect another, and this results in a simpler upgrade path.

Project Types

Visual Basic .NET does not support some of the Visual Basic 6.0 project types, such as dynamic HTML (DHTML) applications and ActiveX Documents. These applications cannot be upgraded and should be left in Visual Basic 6.0 or rewritten in Visual Basic .NET by using Web Forms.

Control Types

Some Visual Basic 6.0 controls are not supported in Visual Basic .NET and will upgrade to substitute controls. For example, the **Shape**, **Line**, and **OLE Container** controls are all unsupported and will upgrade to **Label** controls in Visual Basic .NET. If your application makes extensive use of these types of control, it may require more work to upgrade the application to a working solution.

Language Constructs

Some Visual Basic 6.0 keywords are not supported in Visual Basic .NET. For example, **Option Base**, **LSet**, and **GoSub** are not supported. Extensive use of these keywords in your projects will require manual work after the upgrade process.

◆ Options for Upgrading

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

Lead-in
Once you have decided to migrate your application to Visual Basic .NET, you will then need to determine the upgrade path to follow.

- **Complete Rewrite**
- **Complete Upgrade**
- **Partial Upgrade**

Delivery Tip
Remind students that the advantages and disadvantages listed here are general guidelines, and that each project must be assessed individually.

There are three options available if you decide to upgrade an existing application: You can completely rewrite the application, gaining all the benefits of the .NET Framework. You can use the Upgrade Wizard on all sections of the application, gaining some of the benefits. Finally, you can do a partial upgrade, leaving legacy sections in Visual Basic 6.0.

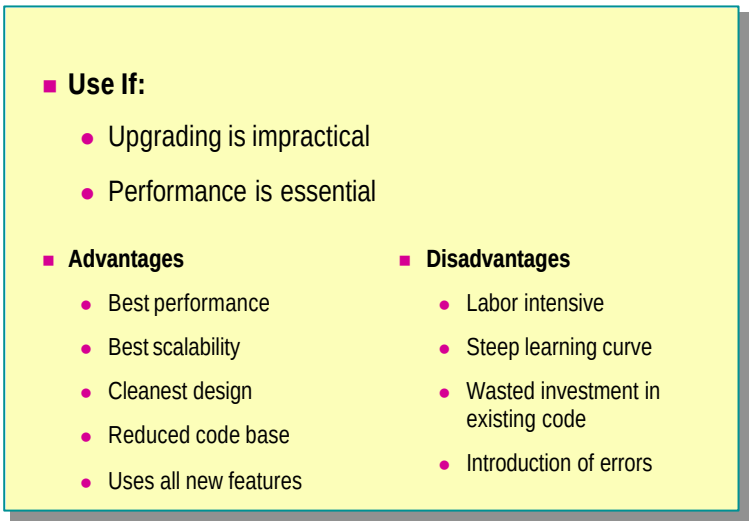
After completing this lesson, you will be able to:

- Identify the three upgrade options.
- Describe the advantages and disadvantages of each upgrade approach.

Complete Rewrite

Topic Objective
To discuss the advantages and disadvantages of a complete rewrite of an existing application.

Lead-in
Completely rewriting an application is not the easiest way to convert an application to Visual Basic .NET.



A complete rewrite of the application is the best way to gain all of the benefits of the .NET Framework, but this can also be the most costly solution. It is most commonly used when the application contains sections that are not upgradeable but that need to take advantage of the .NET Framework.

Advantages

- Performance can be improved through the use of new technologies such as ASP .NET and ADO .NET.
- Scalability is increased when using ASP .NET rather than ASP or other Visual Basic 6.0 Web project types.
- If you rewrite your application from the very beginning, you will have the chance to redesign it to take advantage of the object-oriented features of Visual Basic .NET.
- Your code base will be reduced due to some of the new features in Visual Basic .NET; for example, resizing code can be replaced by using the Anchor properties of a control.

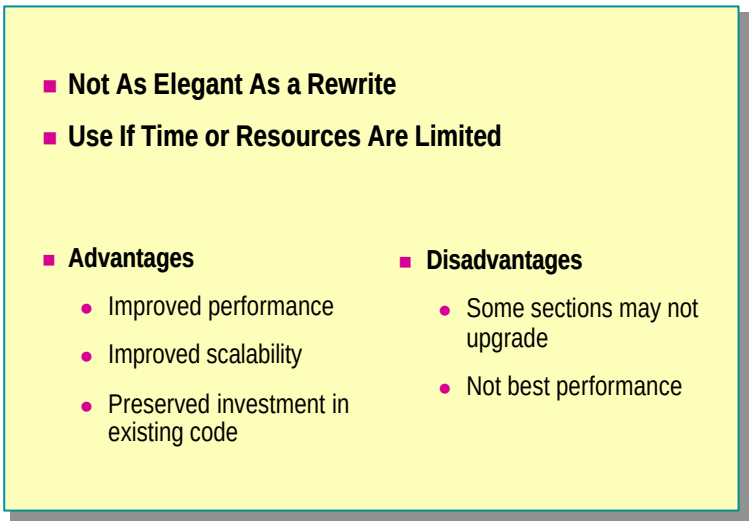
Disadvantages

- Rewriting an application can be labor intensive, as it will potentially involve software analysts as well as developers.
- Learning Visual Basic .NET by rewriting an application can be very difficult for those involved. It may be better to start by upgrading applications and use the knowledge gained from this to learn the product.
- Any existing code that has been written will not be reused, and this results in wasted investment of the existing code.

Complete Upgrade

Topic Objective
To discuss the advantages and disadvantages of a complete upgrade of an existing application.

Lead-in
A complete upgrade is generally easiest, but it is the least likely of all the scenarios.



You will probably find that this is the easiest option for upgrading, but it will not be a common occurrence. Even if the application is completely upgradeable, it may not result in the most efficient code, so you are likely to need to revisit sections anyway.

Advantages

- You will gain performance and scalability from the upgraded sections of the code.
- You will preserve the earlier investment made in the existing code by reusing the code.

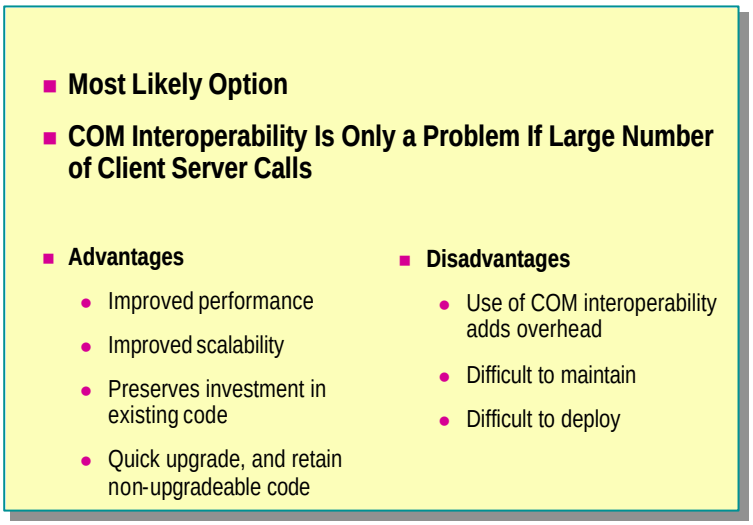
Disadvantages

- Some sections of the application may not be upgradeable (for example, ADO code), so this will not take advantage of .NET.
- Some upgraded sections may use COM interoperability to communicate with the .NET components, resulting in lower performance. Other sections may use the Visual Basic compatibility library, again introducing overhead into the system.

Partial Upgrade

Topic Objective
To discuss the advantages and disadvantages of a partial upgrade of an existing application.

Lead-in
A partial upgrade is the most likely option for migrating your application to Visual Basic .NET.



A partial upgrade is the most likely option for migrating your application to Visual Basic .NET. This allows you to upgrade the sections of your code that will make the most benefit of using the .NET Framework while continuing to use the ones that will be difficult to upgrade. Sometimes you will use this method as a progressive upgrade option, allowing you to focus the upgrade process on small sections of the application at a time.

Advantages

- Performing a partial upgrade can allow you to take advantage of the particular performance and scalability enhancements in the .NET Framework that are appropriate to your application.
- It preserves the investment made in your existing code, and allows reuse of as much or as little as you want.

Disadvantages

- A partial upgrade may result in using COM interoperability to communicate between COM and .NET components. This may degrade performance.
- Applications that mix Visual Basic 6.0 and Visual Basic .NET are harder to deploy and maintain than single-language applications.

Recommendations

Topic Objective
To provide some general guidelines about which type of upgrade to apply to various types of applications.

Lead-in
You have seen the upgrade options available. Now let's look at applying these to different application architectures.

- **Web Client Server**
  - Complete upgrade
  - ASP to ASP .NET and Web Forms, COM components to .NET components, and ADO to ADO .NET
- **Traditional N-Tier Applications**
  - Partial upgrade
  - Leave client in Visual Basic 6.0
- **Enterprise Legacy Applications**
  - Complete rewrite
  - Encapsulate legacy system in Web Service
- **Stand-Alone Windows-based Applications**
  - Little benefit to upgrading

You have seen that the various upgrade options will lend themselves to particular application architectures. The following recommendations can be used as general guidelines to help you decide on the best upgrade process for your particular needs.

Web Client Server

This type of application will typically use ASP as a front end, business logic in COM components in the middle tier, and ADO code for the data access layer. It will gain the greatest benefits from upgrading to Visual Basic .NET because it will be performance driven and need to be extremely scalable. All the technologies used in Visual Basic 6.0 will correspond directly to technologies available in Visual Basic .NET, making the upgrade path relatively simple.

Traditional N-Tier Applications

This type of application will be fairly similar to the thin client server, but it will include some of the logic on the client side. The middle tier and data tier will benefit from upgrading, but unless the front end is extremely simple, it is best left as a Visual Basic 6.0–based application.

If the main goal of the upgrade is to improve performance, then you should rewrite any ADO code to ADO .NET.

Enterprise Legacy Applications

This type of application will typically be client/server with a legacy data source at the back end. They are perfect examples of applications that will benefit from being rewritten in Visual Basic .NET. You can encapsulate the legacy system in a managed component and expose it to many clients by using a Web Service.

Rewriting sections of these applications will provide the greatest improvements in scalability and performance.

Stand-Alone Windows-based Applications

These applications might benefit from the Windows Forms package and they can provide excellent environments to learn the upgrade process.

◆ Performing the Upgrade

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

Lead-in
The Upgrade Wizard is provided to aid you in the upgrade process.

- **Preparing for the Upgrade**
- **Using the Upgrade Wizard**
- **Results of the Upgrade Wizard**
- **Completing the Upgrade**

Delivery Tip
Mention GIGO - garbage in, garbage out. Modifying Visual Basic 6.0 code will optimize the output of the wizard. Unmodified code will result in potentially more complex post-wizard tasks.

You can use the Upgrade Wizard to assist you in upgrading your Visual Basic 6.0–based applications to Visual Basic .NET. Opening a Visual Basic 6.0–based application in Visual Basic .NET will create a new application and leave the existing application as it is.

Because of the differences between the two products, the Upgrade Wizard cannot perform the entire process, but it can simplify some of the tasks involved.

After completing this lesson, you will be able to:

- Identify tasks you need to perform before, during, and after using the Upgrade Wizard.

Note The Upgrade Wizard should only be used to upgrade applications created in Visual Basic 6.0. If you want to upgrade projects created in earlier versions of Visual Basic, open and compile them in Visual Basic 6.0 before using the wizard.

Preparing for the Upgrade

Topic Objective
To discuss the tasks that should be performed before using the Upgrade Wizard.

Lead-in
There are certain changes you can make to your Visual Basic 6.0 code to maximize the usefulness of the output of the Upgrade Wizard.

- Early Binding
- Null Propagation
- Date Variables
- Constants
- Data Access

There are certain tasks that you can perform before using the Upgrade Wizard to maximize the usefulness of its output. These are tasks that make the wizard able to upgrade what would otherwise be ambiguous code.

It is easier to modify your Visual Basic 6.0 code and allow the wizard to upgrade it than to need to address the issues after the upgrading has occurred. You can identify these issues by upgrading your application, reading the comments added by the wizard, and then modifying your Visual Basic 6.0–based project before beginning the upgrade process again.

Early Binding

Late-bound objects can cause problems during the upgrade process when default properties are resolved, and when calls to updated properties, methods, and events are upgraded.

Example

In the following example, a **Label** object is declared as type **Object**, meaning that the Upgrade Wizard is unable to upgrade the **Caption** property to the Visual Basic .NET equivalent of the **Text** property:

```
Dim objLabel as Object
Set objLabel = Form1.Label1
objLabel.Caption = "Enter your password" 'Cannot be upgraded
```

To avoid this, you should declare all of your variables as the explicit type.

```
Dim objLabel as Label
Set objLabel = Form1.Label1
objLabel.Caption = "Enter your password" 'Can be upgraded
```

Null Propagation

Null propagation is the behavior in Visual Basic 6.0 that dictates that if you add **Null** to another data type, the result will always be **Null**.

Example

```
a = Null
b = 5
c = a + b
```

For Your Information
Using the concatenation operator (&) in Visual Basic 6.0 does not result in **Null** propagation.

In Visual Basic 6.0, the above code will result in variable *c* evaluating to **Null**. In Visual Basic .NET, this code will return an invalid cast exception. To avoid this error, you should always check the contents of a variable that could potentially be **Null** prior to performing operations on it.

The following code shows how to write your Visual Basic 6.0 code to ensure compatibility with Visual Basic .NET:

```
a = Null
b = 5
If IsNull (a) Then
  ' Take appropriate action
Else
  c = a + b
End If
```

Note The Upgrade Wizard will upgrade the Visual Basic 6.0 **Null** constant to **System.DBNull.Value** and the **IsNull** function used in Visual Basic 6.0 to **IsDbNull**.

Date Variables

In Visual Basic 6.0, dates are stored internally as **Double**, so you can declare them as either **Date** or **Double**, and implicit type conversion will occur. In Visual Basic .NET, dates are not stored internally as **Double**, so they need to be declared as explicit **Date** types.

Because there is no way for the Upgrade Wizard to determine which **Doubles** were intended as **Dates**, it cannot upgrade dates declared as **Doubles**. To avoid this problem when upgrading, declare all dates explicitly in Visual Basic 6.0 as the **Date** data type.

Constants

Some of the underlying values of intrinsic Visual Basic constants have changed. If your Visual Basic 6.0 code uses the constants, the Upgrade Wizard will automatically upgrade them to the new constants storing the new underlying values; however, if you have used explicit values in your code, the Upgrade Wizard will leave these unaltered, and errors may occur.

The following example shows how to correctly use Visual Basic constants in your Visual Basic 6.0 code:

Example

```
'Incorrect use of underlying values
Response = MsgBox("Do you want to continue?", 4)
If Response = 6 Then
  'Do something
End If

'Correct use of predefined constants
Response = MsgBox("Do you want to continue?", vbYesNo)
If Response = vbYes Then
  'Do something
End If
```

Data Access

The only forms of supported data binding in Visual Basic .NET are ADO and ADO .NET. For this reason, it is recommended that you upgrade all Data Access Object (DAO) or Remote Data Objects (RDO) data binding to ADO in your Visual Basic 6.0 applications before upgrading to Visual Basic .NET.

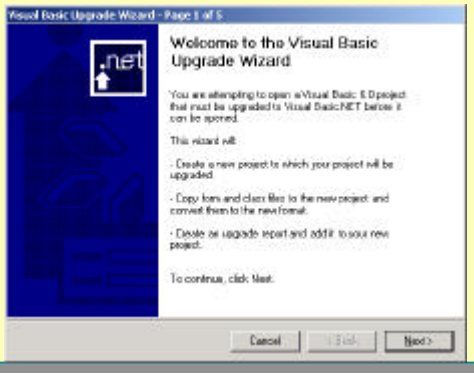
DAO, RDO, ADO and ADO .NET code are all supported in Visual Basic .NET, so you do not need to change these before upgrading. However, you may decide to take advantage of the disconnected ADO .NET architecture and upgrade your code to ADO .NET after the upgrade process has been completed.

Using the Upgrade Wizard

Topic Objective
To discuss how to use the Upgrade Wizard.

Lead-in
The Upgrade Wizard will gather the information necessary to upgrade your application.

- **Open a Visual Basic 6.0- based Project in Visual Basic .NET**



Once you have prepared your application for upgrade, use the Upgrade Wizard to perform the process. It is recommended that you begin by upgrading the user interface tier of your application, and work back through the other tiers.

You can launch the Upgrade Wizard by opening a Visual Basic 6.0–based application in Visual Basic .NET. This will gather the information necessary to upgrade your application.

The Upgrade Wizard will not modify your original application; it will create an upgraded copy at the location you specify.

Results of the Upgrade Wizard

Topic Objective
To discuss the results of the Upgrade Wizard.

Lead-in
The Upgrade Wizard does not change the basic structure of your projects.

- **Language Changes**
  - Code upgraded to be syntactically correct in Visual Basic .NET
- **Form Changes**
  - Most controls will upgrade
- **Other Changes**
  - Other functionality will be upgraded to similar objects

When you have upgraded your application, the resulting project will still be very similar to the original.

Delivery Tip
Remember that the Upgrade Wizard creates a new project and leaves the original project alone.

Anything that can be upgraded is upgraded, and anything that cannot be upgraded, or anything that is ambiguous, will be marked with comments and entered in the Upgrade Report. Links are created to relevant topics in the documentation files to help you resolve any outstanding issues.

Some Visual Basic 6.0 functions do not have equivalents in Visual Basic .NET, and these will be retained through use of compatibility functions.

Language Changes

The Upgrade Wizard modifies the code where possible to take into account the syntax changes in Visual Basic .NET. This includes:

- Resolving parameterless default properties.
- Adding the **ByRef** keyword to procedure parameters.
- Changing property procedures to the new syntax.
- Adding parentheses to all function calls.
- Changing all data types to their new equivalents.

Form Changes

Visual Basic forms will be upgraded to Windows Forms, although a few controls cannot be upgraded because they have no counterpart in Visual Basic .NET. These include the following, which all upgrade to Visual Studio® .NET **Label** controls:

- **OLE Container** control
- **Shape** controls
- **Line** controls

Other Changes

Other functionality in applications created in Visual Basic 6.0 may not have a direct counterpart in Visual Basic .NET but will be left as is or upgraded to similar objects. For example:

- Resource files will upgrade to .resx files that can store any .NET data type.
- Web classes will not upgrade.
- ADO data environments will not upgrade.
- ADO code and ADO data binding will remain unchanged.
- Property pages are no longer used in Visual Basic .NET.

Completing the Upgrade

Topic Objective
To discuss the manual tasks required to complete the upgrade process.

Lead-in
Running the wizard is not the end of the upgrade process.

- Upgrade Report
- Upgrade Comments
- Task List Entries
- Testing
- Other Tasks

The Upgrade Wizard also identifies any potential issues in the upgraded project. It creates an Upgrade Report that lists all potential problems, and adds tasks to the Task List for changes you need to make. These changes are also marked with comments in the code.

Upgrade Report

The Upgrade Report lists all upgrade errors and warnings, grouped by the file in which they occur. It contains details of the issue, the location, and the Help topic associated with the issue. This topic will explain why there is a problem with the code and what you should do to correct the code.

Upgrade Comments

The Upgrade Wizard adds fours types of comments to your code:

- UPGRADE_ISSUE

  These mark any lines of code that will prevent your code from compiling.
- UPGRADE_TODO

  These mark any code that will compile but that will still cause a run-time error.
- UPGRADE_WARNING

  These mark code that will compile but that may cause run-time errors.
- UPGRADE_NOTE

  These mark code that will compile and run but for which the changes in Visual Basic .NET may cause unwanted side effects.

The comments also include a hyperlink to the Help topic associated with the issue.

Task List Entries

The Task List shows all upgrade comments that you must resolve to ensure the correct running of your application. This includes Issues, ToDos, and Warnings. It will also list any other issues in your code that have not been introduced by the Upgrade Wizard. You can use this list to quickly find all the comments in the code.

Testing

You must ensure that you complete a full test cycle after the upgrade process, to check that the application is still functioning as you would expect.

Other Tasks

There are a number of other modifications that you can make to your code to improve it. The first of these should be done immediately; the rest can be done in the next phase of the upgrade.

- Replace compatibility functions and controls with .NET code.

  These are only provided for convenience during the upgrade process and should not be used in deployed applications.

- Upgrade ADO code to ADO .NET.

  This will take advantage of the benefits of ADO .NET.

- Replace COM components with NET components.

  This will reduce the number of managed to unmanaged calls, which will improve the performance of your application.

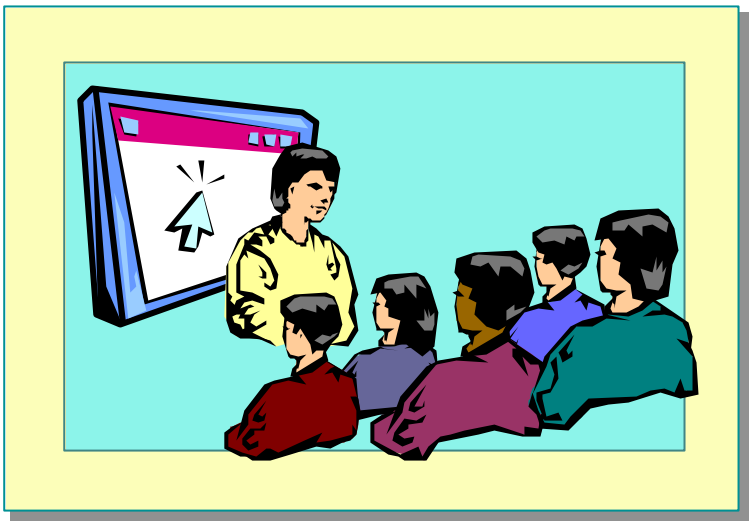
- Replace error handling code.

  You should replace any existing Visual Basic 6.0 error handling code with Visual Basic .NET exception handling using Try..Catch blocks to ensure a more structured approach to your error handling.

Demonstration: Using the Upgrade Wizard

Topic Objective
To demonstrate how to use the Upgrade Wizard.

Lead-in
Now that you have learned how to upgrade an application, let's watch the process as it happens.



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

In this demonstration, you will see how to upgrade a Visual Basic 6.0 application to Visual Basic .NET. You will see the original Visual Basic 6.0 application, how to use the Upgrade Wizard, and some of the tasks that could be completed afterwards.

This application is a simple invoice viewing system for the Cargo system. It is currently running as a Visual Basic 6.0 form-based application, interacting with class modules providing the data access code.

Delivery Tip
Ensure that students understand that this is a simple application created for demonstration purposes. Be sure to discuss with them the options that are available to further enhance an application after they have used the Upgrade Wizard and addressed the major issues.

You will see the projects being upgraded by the Upgrade Wizard and review the issues identified in the comments and Upgrade Report.

Once the critical issues have been solved, there are other considerations for the project. All the data access code is ADO and could be upgraded to ADO .NET to take advantage of the disconnected architecture.

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.

- Deciding Whether to Upgrade
- Options for Upgrading
- Recommendations
- Performing the Upgrade

1. List two benefits of upgrading an application and how those benefits are gained.

   Scalability can be gained through the use of ASP .NET. Performance can be improved through the use of ASP .NET and ADO .NET.

2. What is the most commonly followed upgrade path? Why?

   Partial upgrade, because a complete rewrite is generally too expensive and a complete upgrade too impractical.

3. Which upgrade comments are not listed in the Task List? Why?

   UPGRADE_NOTE comments are not listed because they only highlight potential problems, and the code will run without any modifications.