G

Migrating Older ASP.NET Applications to ASP.NET 4.0

Migrating to ASP.NET 4.0 involves the migration of Web applications or projects developed in ASP.NET 1.x, ASP.NET 2.0, ASP.NET 3.0, or ASP.NET 3.5 to ASP.NET 4.0. The purpose behind the migration of earlier versions of ASP.NET to ASP.NET 4.0 lies in the requirement to utilize the wide range of new and enhanced features available with ASP.NET 4.0. You can develop powerful, extensible, and maintainable Web applications by exploring the new features of ASP.NET 4.0. It has an improved environment that simplifies Web development and provides more options for compiling and deploying Web projects. ASP.NET 4.0 comprises the following new features that help in managing Web application development:

Stores all the information related to a project in the solution (`.sln`) file or the `web.config` file when a Web application is developed in ASP.NET 4.0.

- Uses specific directories prefixed with `App_` to store components, such as resources, assemblies, and the source code of an application.
- Uses the code-behind (CB) page in the new CB model to store user-defined and system-generated code. Earlier, in ASP.NET 1.0, the CB page contained overlapped code that was not easy to understand. ASP.NET 4.0 uses system-generated partial classes that enable you to create more comprehensive and clean CB pages (lesser code in CB pages) so that the user-generated and system-generated code is identified easily.
- Comprises of the Web forms and stand-alone class files that are present in the `App_Code` folder and are compiled in different assemblies. As a result, the code of the Web application becomes more manageable and easy to debug.
- Enhances the deployment model by providing various options, such as pre-compile, full compile, and updateable sites, for deploying a Web application. Pre-compile refers to the compiling of all applications, and then deploying the complete compiled code. The new compilation model also allows you to dynamically compile a Web application. This allows you to modify the content and CB pages of a Web application on a Web server.
- Provides built-in support for Language Integrated Query (LINQ), which allows developers to query data without any need of writing the specific code to store the data. This feature was not present in the previous versions of ASP.NET, such as ASP.NET 1.0 and ASP.NET 2.0.
- Supports multiple versions of .NET Framework that include versions 2.0, 3.0, 3.5, and 4.0. You can write code, targeting any one of these .NET Frameworks as per your requirements.
- Provides an option to show all the Cascading Style Sheets (CSS) file for the Web pages of a Web application by introducing a new tool in the Integrated Development Environment (IDE) called Manage Styles.

This appendix describes the various issues that may arise while migrating the ASP.NET 1.0, ASP.NET 2.0, or ASP.NET 3.5 Web application or project to ASP.NET 4.0. After focusing on migration-related issues, this appendix discusses how these issues can be resolved and how the ASP.NET 3.5 Web project can be converted to ASP.NET 4.0.

Migration: Issues and Solutions

During the migration of a Web application (that contains several Web projects and class libraries) developed in the earlier version of ASP.NET to ASP.NET 4.0, you may encounter various issues, which are as follows:

- Circular references
- Extra types in the CB class file
- Switching to Design view
- Resource Manager
- Orphaned resource (`.resx`) files
- Errors that occur during file parsing
- Code-behind class file (CB–CB) references
- Stand-alone class file (SA–CB) references

Now, let's discuss each of these issues and their solutions.

Circular References

A circular reference is a chain of references where one object in the chain refers to the next object, and the last object refers to the first object again. The following two scenarios may lead to circular references during migration of Web projects:

- **Scenario I**—A CB file refers another CB file, which, in turn, refers the first file again. For example, there are four files in a sequence: `File1.aspx.cs`, `File2.aspx.cs` `File3.aspx.cs`, and `File4.aspx.cs`. The `File1.aspx.cs` file refers the `File2.aspx.cs` file, which refers the `File3.aspx.cs` file, and so on. The circular reference occurs when the last file in a sequence, `File4.aspx.cs`, refers the first file, `File1.aspx.cs`, as shown in Figure G.1:

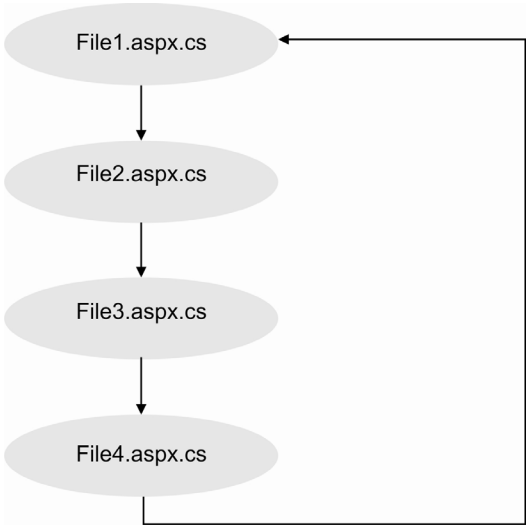

Figure G.1: Displaying Circular Reference in the CB Files

You can break the circular reference in the first scenario by performing the following two steps:

1. Create an abstract base class for any one of the referenced files in the `App_Code` folder.

2. Remove the reference directive of the referenced file from the Web form or user control, which uses the referenced file.

- **Scenario II**—This scenario consists of two assemblies. The first assembly refers the second assembly, which, in turn, refers the first assembly. For example, there are two assemblies, `Asmb1` and `Asmb2`. The `Asmb1` assembly stores the `File1.aspx.cs` and `File2.aspx.cs` files. The `Asmb2` assembly stores the `File3.aspx.cs` and `File4.aspx.cs` files. The `File1.aspx.cs` file of the `Asmb1` assembly refers the `File3.aspx.cs` file in the `Asmb2` assembly. The `File4.aspx.cs` file in the `Asmb2` assembly, in turn, refers the `File2.aspx.cs` file in the `Asmb1` assembly. Such a combination of files also leads to the circular reference, as shown in Figure G.2:

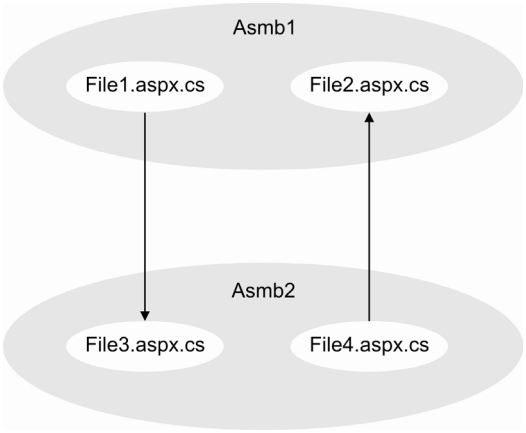

Figure G.2: Displaying Circular Reference in Assemblies

You can break the second scenario by shifting the referenced files, `File2.aspx.cs` and `File3.aspx.cs`, to a separate folder. At the time of compilation, the ASP.NET compiler, by default, creates a separate assembly for the folder containing these files. This way the circular reference between the `Asmb1` and `Asmb2` assemblies is removed, as shown in Figure G.3:

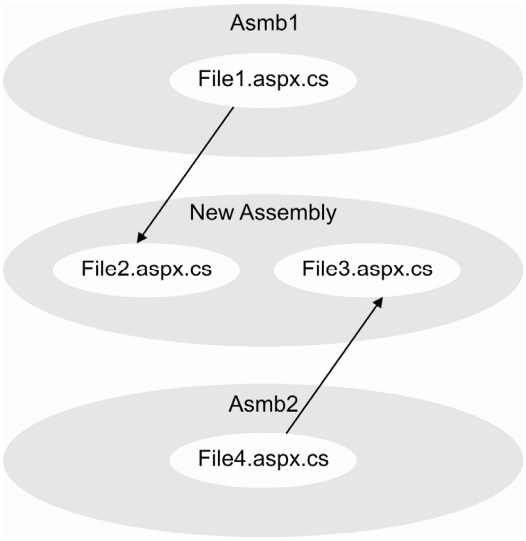

Figure G.3: Removing Circular Reference in Assemblies

Extra Types in the CB Class File

ASP.NET 3.5 enables you to store shared data types, such as structs, enums, and interfaces, which, in turn, helps you to share these data types across different Web projects. These shared data types are also called extra types in a `CB` class file related to a Web page or user control. However, in ASP.NET 4.0, a Web page or user control is compiled within its own assembly that is different from the assembly in which the shared types are compiled. As a result, the Web page encounters a problem, such as compilation error, when it tries to access the shared type. The Web page fails to locate the shared type, since the shared type is stored in a separate assembly.

To ensure that the shared types can easily be accessed, you need to make the following changes in the ASP.NET 4.0 code of the converted Web application:

- Move the shared types to a separate stand-alone class file in the `App_Code` folder
- Specify the access modifier of the shared type as public

At compilation time, the compiler compiles the stand-alone files containing the shared types and makes these types available to all assemblies existing in the Web application.

NOTE

Stand-alone files are discussed later in this appendix.

Switch to Design View

After migrating the ASP.NET 2.0/3.5 Web application to ASP.NET 4.0, you may find it difficult to switch from the Code view to the Design view of the Web application. This may be because either the HTML code of the Web application is not well formed or it contains mismatched tags. ASP.NET 4.0 uses the source code preservation and validation functions to confine the end-user to the Code view till all the problems are fixed. The only remedy is to ensure that all the tags in the Web form (`.aspx` file) occur in pairs and that the HTML code is well formed.

Resource Manager

ASP.NET 2.0 manages resources, such as bitmaps and strings, of a Web application by using Resource Manager, which maintains resources in different resource files. In ASP.NET 2.0, Resource Manager is created by using a fixed assembly name and you need to remember the assembly name before creating Resource Manager. To explain this concept better, let's look at an example. Suppose your Web application contains a resource file, `MyResources.resx`, which uses a string resource `hello`. This resource is managed by using an instance of Resource Manager, `rm`, which is created by using the assembly name `asmb`, as illustrated in the following ASP.NET 1.x code:

```
Assembly asmb = Assembly.Load("ABC");
ResourceManager rm = new ResourceManager("ABC.MyResources", asmb);
String str = rm.GetString("hello");
```

The preceding code works well in ASP.NET 2.0, but generates an error in ASP.NET 4.0 after the Web application containing the code is converted. This is because ASP.NET 4.0 does not support predetermined assembly names. ASP.NET 4.0 uses the non-deterministic assembly naming method wherein the name of the assembly used to create Resource Manager cannot be determined. Therefore, a lot of effort is saved, as one is not required to remember several assembly names to manage resources while working in ASP.NET 4.0.

To fix this problem of predetermined assembly names, move the `MyResources.resx` resource file to the `App_GlobalResources` folder. This folder stores all the resources that are used globally in a Web application and is present in the root directory of the Web application. The resources defined in the `App_GlobalResources` folder are automatically made available to the Web application. In addition, modify the ASP.NET 4.0 code of the converted Web application to define the `hello` resource, as shown in the following line of code:

```
String str = Resources.MyResources.hello;
```

Orphaned Resource (.resx) Files

Whenever you develop a Web project in ASP.NET 2.0, a resource file (with the `.resx` extension) is generated automatically to store the resources related to the Web project. These resource files are rarely used by developers as these files may overwrite the source code of Web pages. Therefore, these files may not contain any user-defined resource code. However, during migration, it is not possible to identify resource files that do not contain the user-defined resources. As a result, these resource files are also converted to the project.

In ASP.NET 2.0, the resource files belong to the assemblies, but no such assemblies exist in ASP.NET 4.0. As a result, after migration, these resource files are introduced to a new environment wherein there are no folders to contain these resource files. These files are termed as orphaned resource files. Perform the following steps to ensure the non-existence of orphaned resource files:

1. Review the resource files in the converted project to identify the existence of any code for user-defined resources.
2. Store the identified code for the user-defined resources in a new resource file.
3. Move the new resource file to the `App_GlobalResources` or `App_LocalResources` folder. As a result, the data is available to all Web applications referencing these resource files.
4. Delete the converted resource files (`.resx` files) from the Web project.

You can ensure that no orphaned resource files exist in the converted Web project by performing the preceding steps.

Error During File Parsing

During the process of migration, the Visual Studio Conversion Wizard may flash an error message indicating that a file cannot be parsed. The error occurs when the Web project contains one or more pages where the `Codebehind` or `Src` attribute is not placed before the HTML tag. When the `Src` or `Codebehind` attribute is not present before the HTML Tag, the Visual Studio Conversion Wizard fails to convert the Web page during the migration process.

The `Src` attribute contains a reference to the file containing the source code of a Web page. The `Codebehind` attribute helps in locating the `CB` class file associated with the Web page.

If the error message arises due to a pure HTML page stored with the `.aspx` extension during the process of migration, then ignore the error message. Moreover, to avoid this issue from arising, ensure the following:

- The HTML pages are stored with proper extensions
- The `Codebehind` and `Src` attributes in Web forms and user controls are placed before the HTML tags

Code-Behind Class File (CB–CB) Reference

Every Web project contains a `CB` class file. In certain situations, you can use the `CB` class file of either your own Web project or even different Web projects. This can be done easily while creating the Web project. However, the problem arises when such Web projects are converted. The major reason behind the problem is that the `CB` class files of both the projects are stored in different assemblies; and therefore, cannot be linked directly. This problem of referencing the `CB` class files of different Web projects is termed as the CB–CB reference problem.

Now, let's look at some situations that require you to reference a `CB` class file of a separate Web project in your Web project:

- A Web form or user control is used as a base class for creating another Web form or user control
- A user control is created in one Web page and is used in another Web page
- The declaration and the instance of the class that implements a Web form are not placed in the same Web form

Now, let's take an example to understand the concept. Suppose the `CB` class file, `mypage.ascx.cs`, contains the following code to load the `CustomCtrl` server control:

```
CustomCtrl c = (CustomCtrl)LoadControl("~/CustomCtrl.ascx");
```

The mypage.ascx file refers the CustomCtrl server control, which is defined in the CB class file, mypage.ascx.cs. In ASP.NET 4.0, as both the CB class file and content page are contained in different assemblies, the problem of the CB–CB reference may arise during the process of migration. To fix this problem, add the reference directive of the server control in the mypage.aspx file, which refers the server control. The following code shows the reference directive for the CustomCtrl server control:

```
<%@ Reference Control="~/CustomCtrl.ascx" %>
```

The reference directive shown in the preceding code enables the compiler to locate the server control or the Web page used in the mypage.aspx page without generating any errors.

NOTE

The Visual Studio Conversion Wizard automatically locates the server control or the Web page used in the mypage.aspx page.

Stand-Alone Class File (SA–CB) Reference

The SA–CB reference problem arises when a stand-alone (SA) class file refers the code of a CB class file. A common instance of such a reference is when the SA file refers a class variable defined in the CB class file. Now, let's understand it with an example.

Suppose a Web page uses a control with the name, CustomCtrl. This control has been defined in the CustomCtrl.ascx.cs file by using the following code:

```
class CustomCtrl : System.Web.UI.UserControl
{
  public static string MyID = "Custom Control";
  public void f()
  {
        .........
        .........
  }
}
```

Now, create an SA file, Try.cs, in the same project that uses the MyID variable of the CustomCtrl class, as shown in the following line of code:

```
String MyID = "Class + " + CustomCtrl.MyID;
```

When the project containing these files is converted to ASP.NET 4.0, the SA class file is moved to the folder named App_Code. Now, this file can access only the App_Code assembly within which it has been compiled. At compilation time, when the Try.cs file tries to reference the MyID class variable stored in a different page assembly, the SA–CB reference problem occurs. Moreover, as it is an SA file, you cannot solve the problem by using a reference directive for locating the CustomCtrl class.

To resolve this issue, you need to use the CustomCtrl abstract base class in the App_Code folder. The code of the class file, App_Code/Stub_CustomCtrl.cs, which contains the CustomCtrl base class, is as follows:

```
abstract class CustomCtrl : System.Web.UI.UserControl
{
  public static string MyID = "CustomCtrl";
  abstract public void f();
}
```

The CustomCtrl abstract base class is used in the CB class file CustomCtrl.ascx.cs, as shown in the following code:

```
class migrated_Control : CustomCtrl
{
  override public void f()
  {
        .........
        .........
  }
}
```

Defining the CustomCtrl abstract base class in a file stored in the App_Code folder enables both the SA and CB class files to reference the CustomCtrl class during the compilation of the Web project. This abstract base class also causes the SA file to load the actual class, migrated_control, during runtime. Therefore, the SA–CB

reference problem is resolved by declaring the abstract base class in the Web project. The original class is loaded by using late binding.

NOTE

You need to first run the Visual Studio Conversion Wizard on the Web projects for implementing the solutions.

Now, you have a fair idea of the various issues that can arise while migrating Web projects developed in ASP.NET 2.0 or ASP.NET 3.5 to ASP.NET 4.0.

Now, let's learn how to convert the ASP.NET 2.0/3.5 Web project into the ASP.NET 4.0 Web project.

Conversion of the ASP.NET 2.0/3.5 Based Web Project to the ASP.NET 4.0 Web Project

ASP.NET 4.0 provides with the Visual Studio Conversion Wizard, which automates the conversion of the ASP.NET 3.5 based Web project to the ASP.NET 4.0 Web project. Certain preliminary steps, often called the best practices, are recommended before running the Visual Studio Conversion Wizard. These best practices help minimize the occurrences of migration issues that are often encountered, while running the Visual Studio Conversion Wizard. The best practices to be followed before migrating the Web projects developed in one platform to another are as follows:

- The Web project is configured properly in the ASP.NET 2.0/3.5 environment before migration. To ensure this, open the solution file (with the .sln extension) of the Web project in ASP.NET 2.0/3.5 and then build and run the Web project to verify whether it runs properly without any errors.
- All excluded files, which are not referred to by any of the files from your Web project, should be removed during the conversion. You can move these files to a location different from the folder where the solution file of your Web project is stored. If you do not remember names of the excluded files, you can use the Show All Files button in the Solution Explorer window of ASP.NET 2.0/3.5 to view all the existing files in the Web project. You can then identify, and either delete or move the files, which are marked excluded from your Web project.
- The root folder of multiple Web projects exists in the same Internet Information Services (IIS) Web application. The problem arises when two Web projects within the same Web application refer common files. These common files are a part of multiple projects because of which they are migrated twice when you convert the Web application containing these projects. You should remove these common or duplicate project files from your project to ensure that these are not migrated twice.

NOTE

Visual Studio 2010 automatically converts the applications that were developed in .NET Framework 2.0, 3.0, and3.5 by making changes in the Web.config file. Thus, the application can execute in .NET framework 4.0.

You are bound to encounter errors if you try to convert the solution files containing the same Web projects. In such a case, the first solution file is converted successfully, but errors are encountered while converting the second solution file. To remove these errors, before converting a solution file, you should check if it contains any Web projects that have already been converted before and, if it is, then remove them. Removing the already converted Web projects ensures that the solution files can be converted without any problem.

Now, let's see how to convert the ASP.NET 2.0 Web project to ASP.NET 4.0 using the Visual Studio Conversion Wizard. Perform the following steps to migrate the Visual Basic or C# project using the Visual Studio Conversion Wizard:

1. Open the ASP.NET 2.0 Web project that you want to convert to ASP.NET 4.0. The Visual Studio Conversion Wizard starts automatically when you open the solution file. The welcome page of the Visual Studio Conversion Wizard appears, as shown in Figure G.4:

Figure G.4: Displaying the Welcome Page

2. Click the Next button on the welcome page to display the Choose Whether to Create a Backup page of the Visual Studio Conversion Wizard, as shown in Figure G.5:

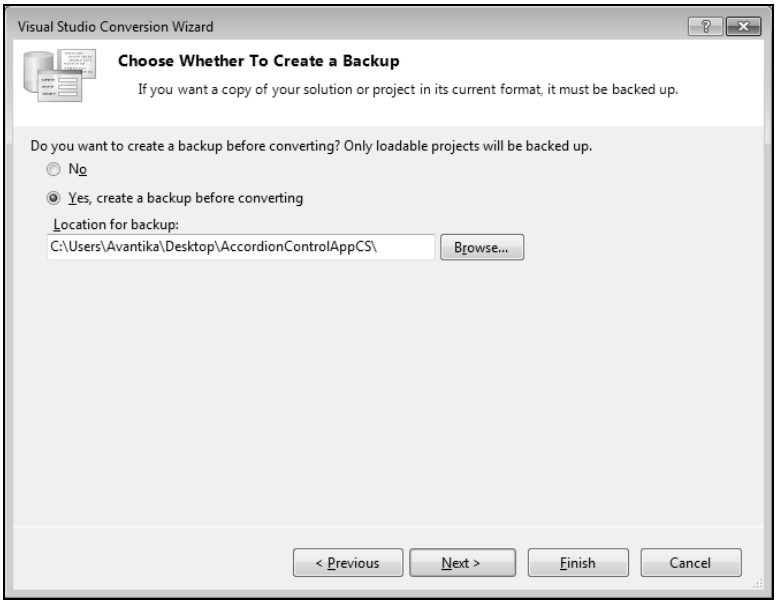

Figure G.5: Displaying the Choose Whether to Create a Backup Page

The Choose Whether to Create a Backup page helps you to specify whether or not you wish to take a backup of the Web project you are converting. It is recommended that you take a backup of a Web project before migrating it. Select the Yes radio button if you want to create a backup of your original Web project.

3. Click the Next button. The Ready to Convert page appears, as shown in Figure G.6:

Figure G.6: Displaying the Ready to Convert Page

Figure G.6 displays instructions on how to handle the Web project after conversion and also presents the summarized information, such as the name of the Web project and location of the backup.

4. Click the Finish button to start the conversion (Figure G.6). After the conversion process is complete, the Conversion Complete page appears, as shown in Figure G.7:

Figure G.7: Displaying the Conversion Complete Page

Figure G.7 contains the Show the conversion log when the wizard is closed option, which helps in viewing the Conversion Report page related to migration.

5. Select the Show the conversion log when the wizard is closed check box, and then click the Close button. The Conversion Report page appears, as shown in Figure G.8:

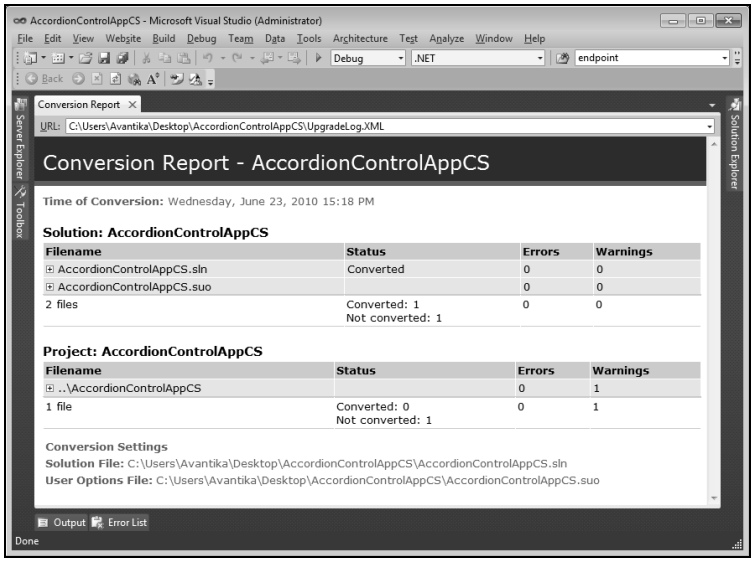

Figure G.8: Displaying the Conversion Report Web Page

To convert a Web project developed in ASP.NET 3.5 to ASP.NET 4.0, perform the following steps:

1. Open the Web project in ASP.NET 3.5. After the Web project is opened, you will be prompted to upgrade the application to ASP.NET 4.0, as shown in Figure G.9:

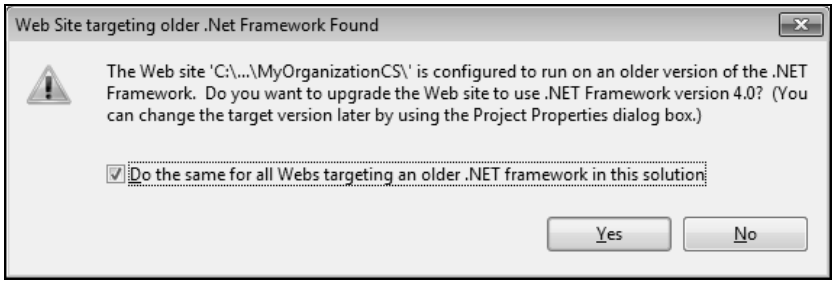

Figure G.9: Displaying the Message Box to Upgrade the Web Project

2. Click the Yes button on the message box (Figure G.9) if you want to upgrade your Web project to ASP.NET 4.0. The message box (Figure G.9) also provides an option Do the same for all Webs targeting an older .NET framework in this solution. You can manually upgrade the .NET Framework Web applications of your solution by unselecting this option.

You can also run the Web project developed in ASP.NET 2.0 in ASP.NET 4.0 without upgrading the Web project. You can later change the target framework of the Web project by right-clicking the solution in the Solution Explorer window and selecting the Property Pages option from the context menu. In the Property Pages window, change the target framework according to your requirements, as shown in Figure G.10:

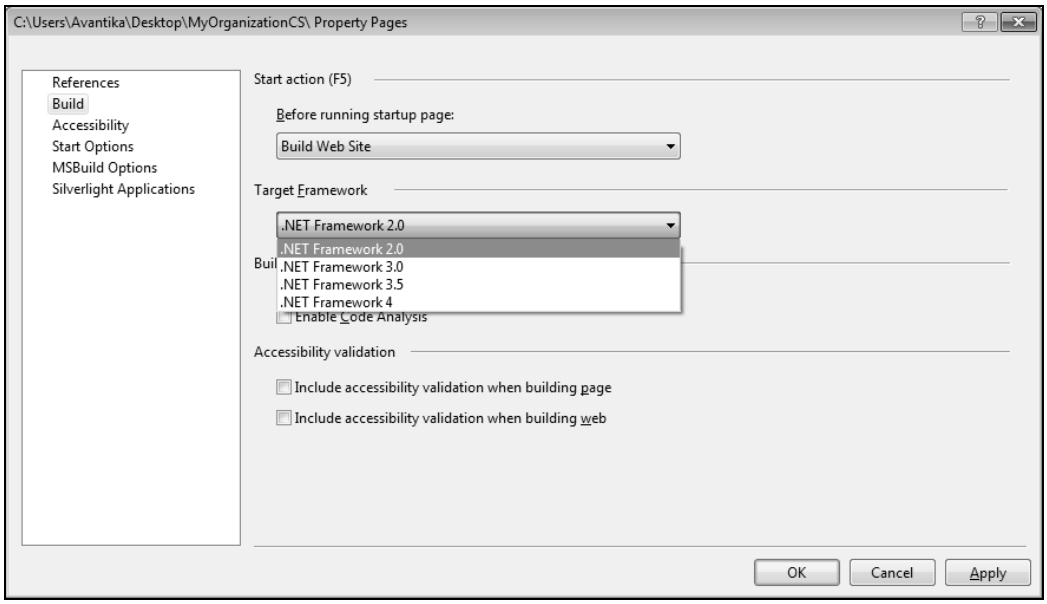

Figure G.10: Displaying the Property Pages Window

3. Select the target framework--2.0, 3.0, 3.5, or 4—in the Target Framework drop-down list and click the Apply button to apply the properties you have changed.

With this, you come to the end of this appendix. In this appendix, you have learned about migrating older applications to ASP.NET 4.0.