18

SSIS Software Development Life Cycle

Software Development Life Cycles play an important role in any type of application development. Many SQL Server database administrators and DTS developers have little experience with Microsoft Source Control tools because the tools themselves have been less than "database project-friendly." Microsoft has responded with a more reliable version of Visual SourceSafe and a new source control architecture called Team System.

In addition, many SQL Server DBAs have not been involved with Software Development Life Cycles beyond executing scripts attached to change control documentation. Recent legislation in the United States has changed the role of the SQL Server DBA in the enterprise. Regarding Software Development Life Cycles, DBAs now must participate in ever-earlier phases of the project development.

In addition, SQL Server DBAs — especially SSIS developers — will realize greater productivity and development cycle fault tolerance as they employ source-controlled development practices. These practices produce code that is auditable, an added benefit in the current corporate climate.

This chapter provides an overview of some of the available features in Microsoft's new offerings. It includes a brief description of how to store a project in Visual SourceSafe and a detailed walk-through that describes creating a Team Project — using Visual Studio Team System — for SSIS. In practice, Team Projects will most likely be created by someone else in the software development enterprise.

> *A more detailed examination of Team System is beyond the scope of this book but may be found in Professional Visual Studio 2005 Team System by Jean-Luc David et al. (Wrox, 2006).*

Because the line between database administrator and software developer has blurred and blended over the years, the Team Project walk-through is built in Visual Studio 2005. In the Team Project walk-through, you are going to put together a project that uses the source control and collaboration functionality provided by Visual Studio Team System to demonstrate working with the tool and complying with your SDLC process.

This chapter also contains information about debugging and breakpoints — highlighting features new to database administrators and DTS developers in SSIS.

Included is a discussion regarding development and testing with an admitted bias toward the agile development methodology. In the author's humble opinion, there are two types of developers: those who use agile methodologies and those who will.

The chapter concludes with a discussion about managing package deployment.

Introduction to Software Development Life Cycles

Software Development Life Cycles (or *SDLCs*) are a systematic approach to each component of application development — from the initial idea to a functioning production application. A *step* (or *phase*) is a unit of related work in an SDLC. A *methodology* is a collection of SDLC steps in action, applied to a project. *Artifacts* are the recorded output from steps.

For example, the first step of an SDLC is Analysis. The methodology requires a requirements document as an Analysis artifact.

Software Development Life Cycles: A Brief History

Software Development Life Cycles have existed in some form or other since the first software applications were developed. The true beginning of what is now termed "software" is debatable. For your purposes, the topic is confined to binary operations based on Boolean algebra.

In 1854, mathematician George Boole published *An Investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities*. This work became the foundation of what is now called Boolean algebra. Some 80 years later, Claude Shannon applied Boole's theories to computing machines of Shannon's era. Shannon later went to work for Bell Labs.

Another Bell Labs employee, Dr. Walter Shewhart, was tasked with quality control. Perhaps the pinnacle of Dr. Shewhart's work is statistical process control (SPC). Most quality control and continuous improvement philosophies in practice today utilize SPC. Dr. Shewhart's work produced a precursor to Software Development Life Cycles, a methodology defined by four principles: Plan, Do, Study, and Act (PDSA).

Dr. Shewhart's ideas influenced many at Bell Labs, making an accurate and formal trace of the history difficult. Suffice it to say that Dr. Shewhart's ideas regarding quality spread throughout many industries; one industry influenced was the software industry.

Types of Software Development Life Cycles

SQL Server Integration Services provides integrated support for many SDLC methodologies. This chapter will touch on a few of them. In general, SDLCs can be placed into one of two categories: waterfall and iterative.

Waterfall SDLCs

The first formal Software Development Life Cycles are sequential or linear. That is, they begin with one step and proceed through subsequent steps until reaching a final step. A typical example of linear methodology steps is the following:

- **Analysis:** Review the business needs and develop requirements.
- **Design:** Develop a plan to meet the business requirements with a software solution.
- **Development:** Build the software solution.
- **Implementation:** Install and configure the software solution.
- **Maintenance:** Address software issues identified after implementation.

These methodologies are referred to as *waterfall* methodologies because information and software "fall" one-way from plateau to plateau (step to step).

Waterfall methodology has lots of appeal for project managers. It is easier to determine the status and completeness of a linear project: It's either in analysis, in development, in implementation, or in maintenance.

A potential downside to the waterfall methodology is that the analysis and design steps are traditionally completed in a single pass at the beginning of the project. This does not allow much flexibility should business needs change after the project starts. In addition, the development and implementation steps are expected to be defined prior to any coding.

Iterative SDLCs

Iterative methodology begins with the premise that it's impossible to know all requirements for a successful application before development starts. Conversely, iterative development holds that software is best developed within the context of knowledge gained during earlier development of the project. Development therefore consists of several small, limited-scope, feature-based iterations that deliver a product ever closer to the customer's vision.

The following are examples of iterative SDLCs:

- **Spiral:** Typified by ever-expanding scope in hopes of identifying large design flaws as soon as possible.
- **Agile:** A collection of methodologies fall into this category, including Scrum, Feature-Driven Development, Extreme Programming, Test-Driven Design, and others.
- **Microsoft Solutions Framework:** Microsoft's own practice gleaned from a sampling of best practices from different methodologies.

What happens if, hypothetically, an iteration fails to produce the desired functionality? The developer or DBA must remove the changes of the last iteration from the code and begin again. This is much easier to accomplish if the developer or DBA has stored a copy of the previous version someplace safe, hence the need for *source control*.

Source control is defined as preserving the software source code in a format that allows recovery to a previous state of development or version, and it is a basic tenet of all iterative Software Development Life Cycles.

Versioning and Source Code Control

SQL Server 2005 and SQL Server Integration Services (SSIS) integrate with source control products such as Microsoft Visual SourceSafe (VSS) and the new Team System. Visual SourceSafe is Microsoft's current source control product. Team Foundation Server is Microsoft's new suite of SDLC management tools — which includes a source control engine.

Microsoft Visual SourceSafe

Visual SourceSafe 2005, which ships with the 2005 developer product suites, is an upgrade to previous versions of the product. It boasts improved stability, performance, access, and capacity. In this section, you'll create a project in SQL Server Business Intelligence Development Studio (BIDS) and use it to demonstrate integrated source control with Microsoft Visual SourceSafe.

To configure SSIS source control integration with Microsoft Visual SourceSafe 2005, open the SQL Server Business Intelligence Development Studio. You don't need to connect to an instance of SQL Server to configure integrated source control.

To configure Visual SourceSafe as your SSIS source control, click Tools ➪ Options. Click Source Control and select Microsoft Visual SourceSafe. Expand the Source Control node and click Environment for detailed configuration, as shown in Figure 18-1.

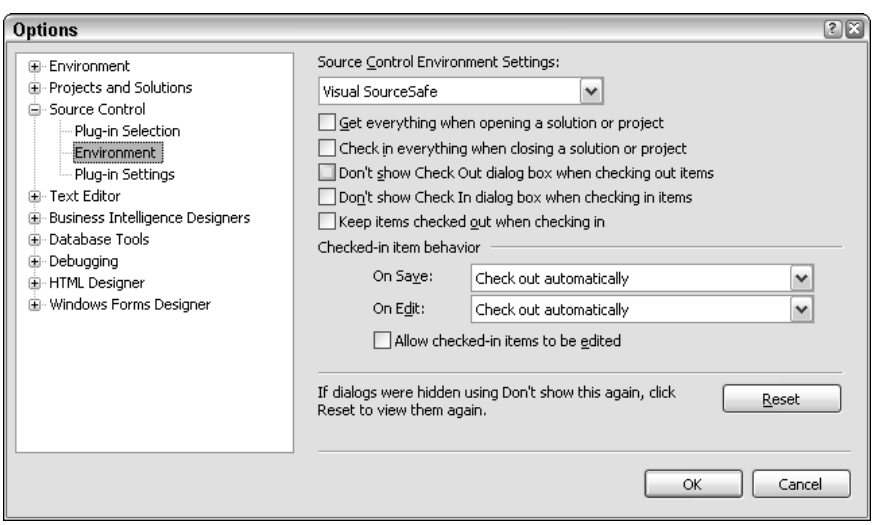

Figure 18-1

The Source Control Environment Settings drop-down list contains three options that represent source control environment roles: Visual SourceSafe, Independent Developer, and Custom.

The Custom role is automatically selected if you begin customizing the source control behaviors in the environment. The following options are available for customization:

- Get everything when opening a solution or project:
  - **Checked:** Retrieves all solution or project files from source control when a solution or project is opened.
  - **Not checked:** You must manually retrieve files from source control.
- Check in everything when closing a solution or project:
  - **Checked:** Automatically checks in all files related to a solution or project on close.
  - **Not checked:** Does not automatically check in all files related to a solution or project on close.
- Don't show Check Out dialog box when checking out items:
  - **Checked:** Hides Check Out dialog box when checking out items.
  - **Not checked:** Displays Check Out dialog box when checking out items.
- Don't show Check In dialog box when checking in items:
  - **Checked:** Hides Check In dialog box when checking in items.
  - **Not checked:** Displays Check In dialog box when checking in items.
- Keep items checked out when checking in:
  - **Checked:** Allows you to continue editing items that have been checked into source control.
  - **Not checked:** You must manually check out the file before editing it.
- Checked-in item behavior on Save:
  - **Prompt for checkout:** You are prompted to check out the files after each Save.
  - **Check out automatically:** Files are checked out automatically when you Save.
  - **Save as:** When Save is clicked, a Save As dialog box appears.
- Checked-in item behavior on Edit:
  - **Prompt for checkout:** You are prompted to check out the files when you begin editing.
  - **Prompt for exclusive checkouts:** You are prompted to exclusively check out the files when you begin editing.
  - **Check out automatically:** Files are checked out automatically when you begin editing.
  - **Do nothing:** When you begin editing, SQL Server Management Studio does nothing.

- Allow checked-in items to be edited:
    - **Checked:** When you begin editing a checked-in file, the Checkout on Edit dialog box appears. This option allows you to check out the file or continue editing without checking out the file.

This is not a best practice. The only situation where this has any useful application is if you intend to save the contents as a new file. If this is the case, it is recommended that you open the existing source-controlled version, save it as the other file, and then make your edits.

    - **Not checked:** Edits to checked-in items are not allowed.

The following predefined roles, and their settings, are available:

- **Visual SourceSafe —** A generic role with the following settings:
    - Keep items checked out when checking in: Not checked.
    - Checked-in item behavior on Save: Check out automatically.
    - Checked-in item behavior on Edit: Check out automatically.
    - Allow checked-in items to be edited: Not checked.
- **Independent Developer —** A role defined for stand-alone development with the following settings:
    - Keep items checked out when checking in: Checked.
    - Checked-in item behavior on Save: Check out automatically.
    - Checked-in item behavior on Edit: Check out automatically.
    - Allow checked-in items to be edited: Not checked.

Check out automatically is the default behavior for checked-in items when saving or editing a project. By not requiring developers to manually check out code, this feature alone saves hours of development time.

One of the options for Source Control (or the Plug-in Selection) is Microsoft Visual SourceSafe (Internet). You can configure Visual SourceSafe for remote access through and intranet or the Internet. This allows you to store source files off-site. A detailed description is beyond the scope of this book, but you can learn more by browsing the "How to: Enable the Internet Service for Remote Access" topic in the Microsoft Visual SourceSafe Documentation.

For the purposes of this demo, select Visual SourceSafe from the Source Control Environment Settings drop-down list and configure source control options as shown in Figure 18-1.

Open the SQL Server Business Intelligence Development Studio by clicking Start ➪ All Programs ➪ Microsoft SQL Server 2005 ➪ SQL Server Business Intelligence Development Studio (BIDS). Because BIDS uses the Visual Studio Integrated Development Environment (IDE), opening SQL Server Business Intelligence Development Studio will open Visual Studio 2005 if you have Visual Studio 2005 installed. When the BIDS IDE opens, click File ➪ New ➪ Project to start a new project. Enter a project name in the New Project dialog box. For now, do not check the Add to Source Control checkbox as shown in Figure 18-2.

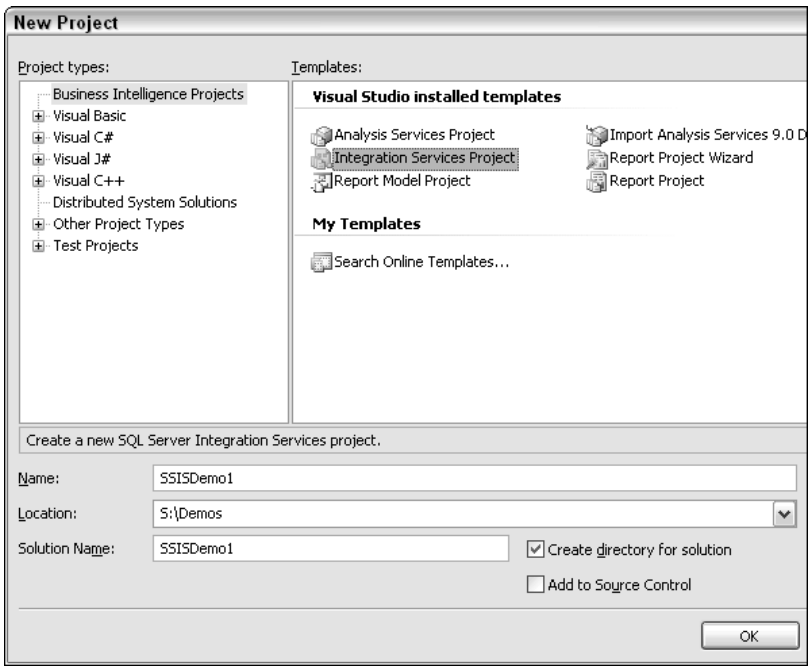

Figure 18-2

Click OK to proceed, and a new project is created in the BIDS IDE.

Add the project to Microsoft Visual SourceSafe by right-clicking the project name in the Solution Explorer and selecting Add to Source Control. You will be prompted to log in to Microsoft Visual SourceSafe. Enter your credentials and click OK as shown in Figure 18-3.

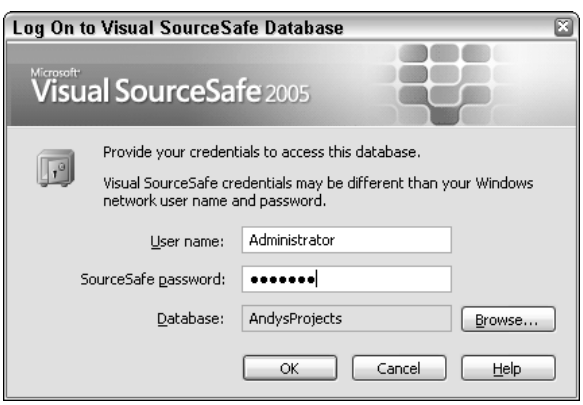

Figure 18-3

The Add to SourceSafe dialog box appears, as shown in Figure 18-4. SSISDemo1.root is the default VSS project name assigned to your project. Accept the default by clicking OK.

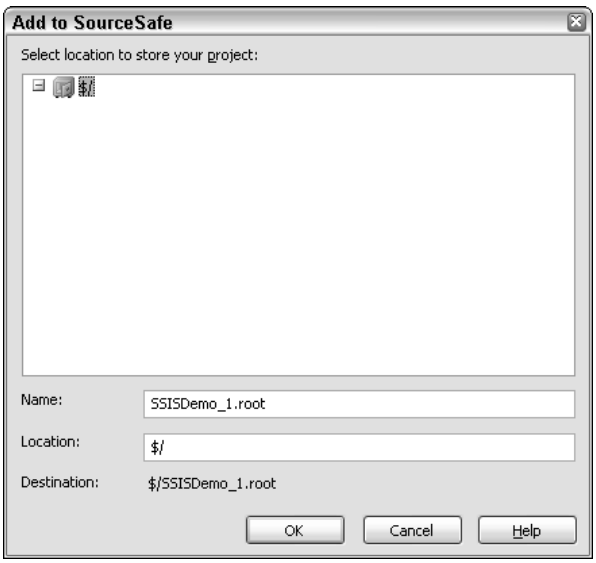

Figure 18-4

Since an SSISDemo1 project does not currently exist in your instance of Visual SourceSafe, you will be prompted to create a project. Click Yes on the dialog box.

After successfully creating a VSS project to maintain your source code, you are returned to the BIDS development environment. Notice the source control "lock" beside your project and Package file as shown in Figure 18-5. The lock icons indicate that the objects are checked in.

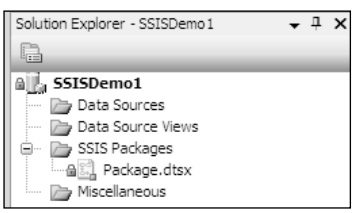

Figure 18-5

Manually check out Package.dtsx for editing by right-clicking Package.dtsx in the Solution Explorer and clicking Check Out for Edit. The Check Out for Edit dialog box appears as shown in Figure 18-6. You may enter a comment to identify why you are checking out the package. This is a good location for change control documentation references, or at a minimum, good notes.

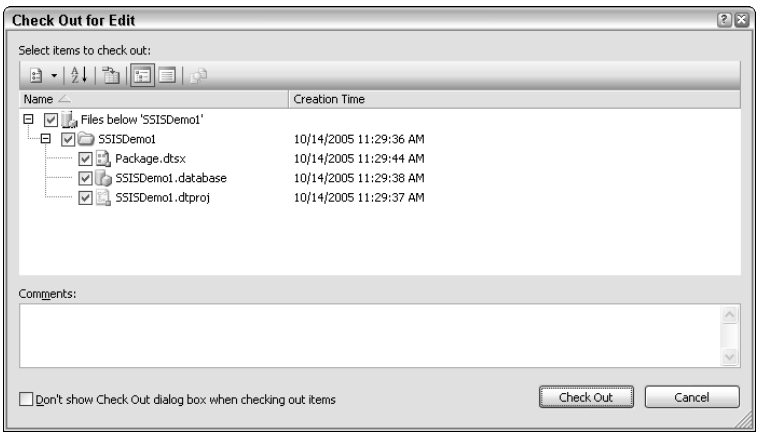

Figure 18-6

Click Check Out to start the checkout process. A Microsoft Visual SourceSafe dialog box will appear, prompting you overwrite your local file or keep your changes. Select the Replace Your Local File with this Version from SourceSafe? option and check the Apply to All Items checkbox. Click OK to begin editing. The Solution Explorer icon beside the Package.dtsx item will change to a red check mark to indicate that the item is checked out exclusively to you, as shown in Figure 18-7.

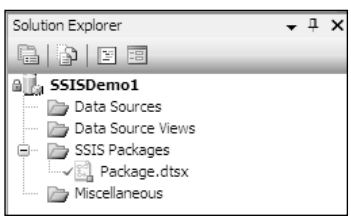

Figure 18-7

Click View ➪ Pending Checkins to open the Pending Checkins window. The Pending Checkins window displays checked-out files awaiting check-in, as shown in Figure 18-8.

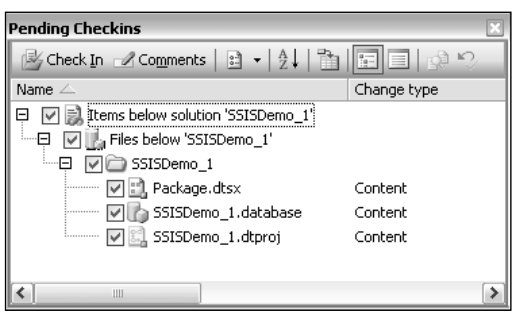

Figure 18-8

Click the Comments button to add any notes to your check-in operation. Again, this is an excellent place to add change control documentation references and bug fixes. Click the Check In button to check your code back into source control. The Source Control confirmation dialog box appears.

If you check the Don't Show this Dialog Box Again (Always Check In) checkbox, you will not see this dialog box on check-in operations. Click the Check In button to continue. Note that the Pending Checkins window is now empty, as no items are checked out for the project.

Observe the Package.dtsx item in the Solution Explorer as you drag a Data Source task onto the Control Flow workspace. A red check mark appears beside the Package.dtsx item. This is the "automatic check-out on edit" feature in action. The Pending Checkins window will now contain the Package.dtsx item, as well as its parent items.

Continue the package construction by right-clicking the Connection Managers workspace just below the Control Flow workspace. Click New ADO.Net Connection to launch the Configure ADO.NET Connection Manager. Click the New button to open the Connection Manager editor. Type or select a server name in the Server Name drop-down list. Select Use Windows Authentication to log on to the server, and select AdventureWorks in the Database Name drop-down list, as shown in Figure 18-9.

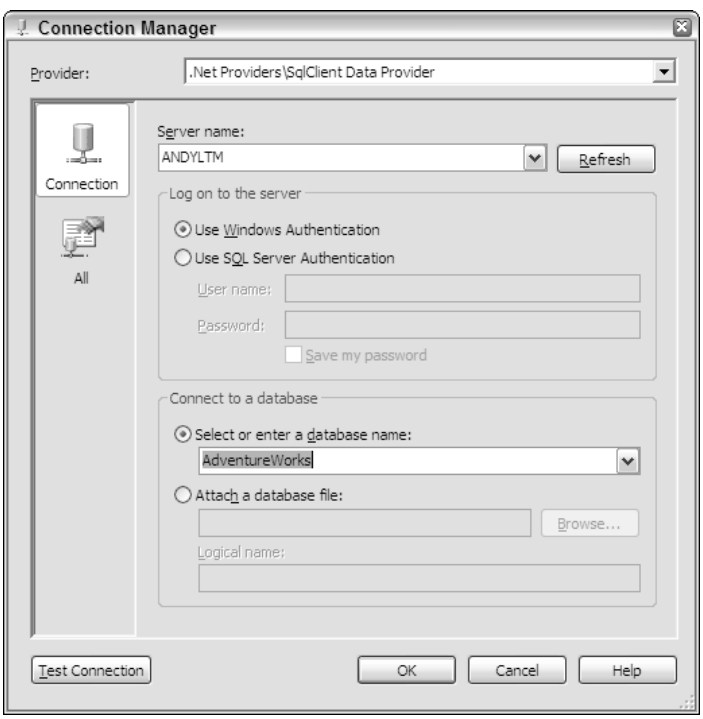

Figure 18-9

Click OK to continue, and OK again to close the ADO.Net Connection Manager.

Double-click the Data Flow task on the Control Flow workspace to edit it. Drag a DataReader Source onto the Data Flow workspace and double-click it to edit. On the Connection Managers tab, in the Connection Manager drop-down list, select the ADO.Net Connection Manager you just defined.

> *The ADO.Net Connection Manager I defined shows up as AndyLTM.AdventureWorks because my machine is named AndyLTM and the database is named AdventureWorks. Your ADO.Net Connection Manager will be named something different.*

Click the Component Properties tab and enter the following SQL query in the SQLCommand property:

```
SELECT * FROM Purchasing.vVendor
```

Your Component Properties tab will look like Figure 18-10.

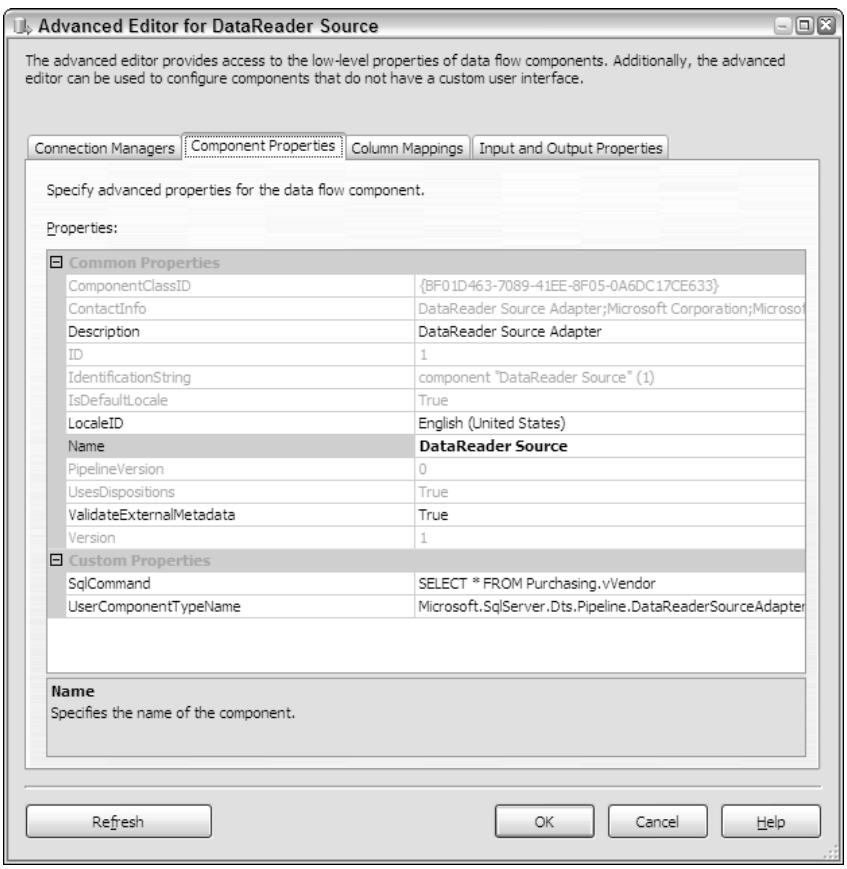

Figure 18-10

Click the Column Mappings tab, and then click OK to close the Advanced Editor for DataReader Source. Save your code, and then open the Pending Checkins window by clicking View ➪ Pending Checkins. Click the Comments button and enter **Added Connection and DataReader** in the Comment text box as shown in Figure 18-11.

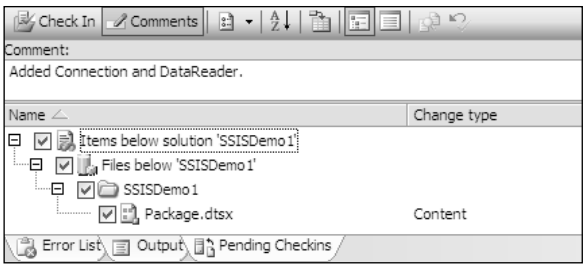

Figure 18-11

Click the Check In button to add current changes to source control. Continue editing by dragging a Flat File Destination onto the Data Flow workspace. Drag the DataReader Source output (represented by a green arrow) from the DataReader Source to the Flat File Destination as shown in Figure 18-12.

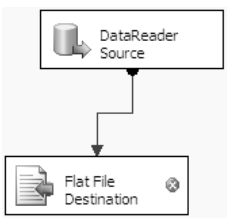

Figure 18-12

Double-click the Flat File Destination to edit. Click the New button beside the Flat File Connection Manager drop-down list. The Flat File Format dialog box will appear; select Delimited and click OK. The Flat File Connection Manager appears. Enter **File 1** in the Connection Manager Name text box. Click the Browse button beside the File Name text box and enter **C:\File1.txt** in the File Name text box. Click Open to continue. Check the Column Names in the First Data Row check box and accept the remaining defaults as shown in Figure 18-13.

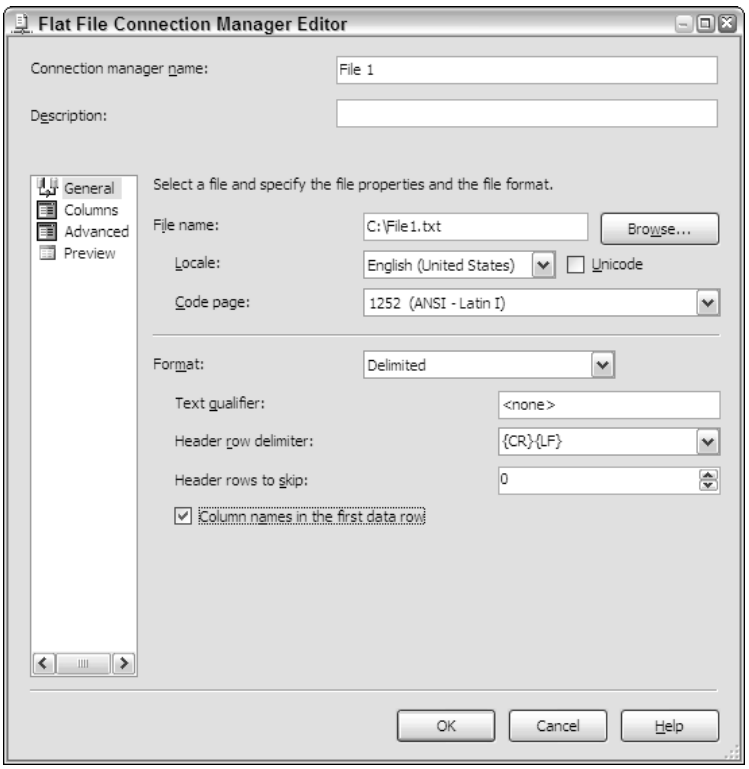

Figure 18-13

Click OK to close the Flat File Connection Manager Editor. This returns you to the Flat File Destination Editor. Click the Mappings item from the list on the left to configure column mappings for the connection, as shown in Figure 18-14.

Click OK to close the Flat File Destination Editor. Click View ➪ Pending Checkins to view the Pending Checkins window. Enter **Added File1.txt destination** in the Comments text box and click the Check In button.

You now have a functional version of a package in source control. Don't take my word for it — click the Play button (or press F5) to execute the package. After some validation completes, you should see the Data Flow items turn green, as shown in Figure 18-15.

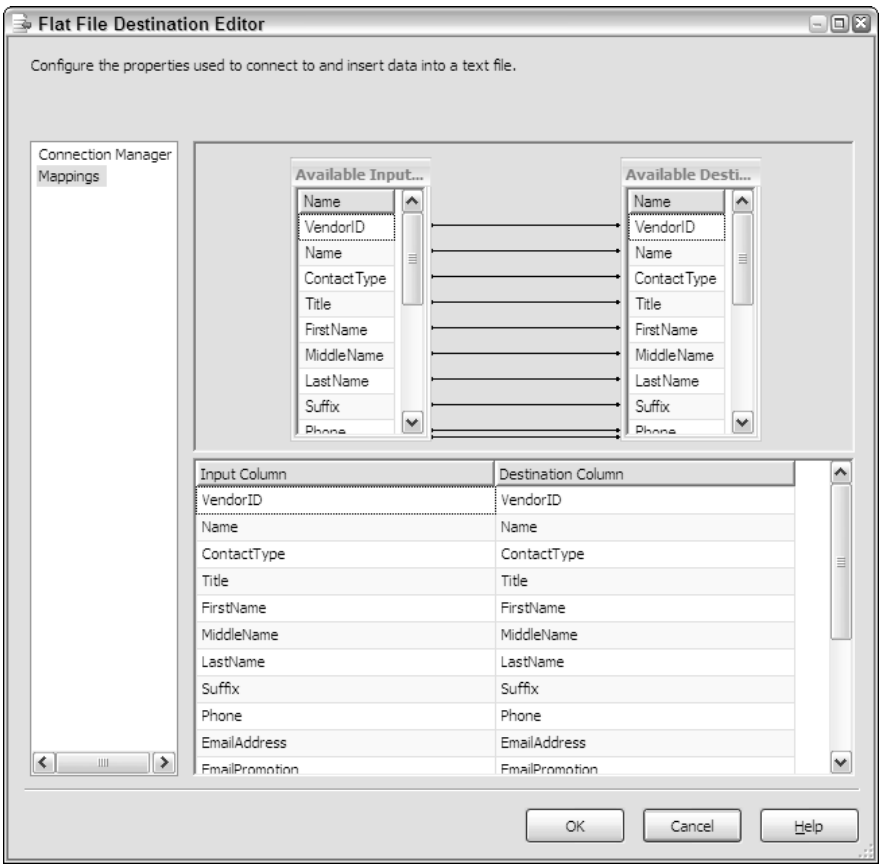

Figure 18-14

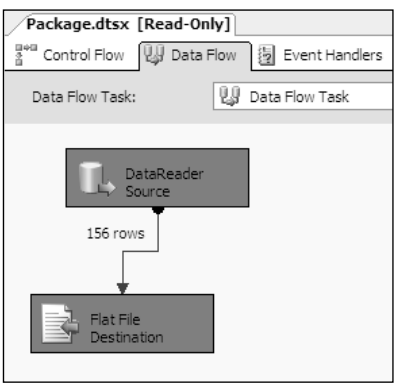

Figure 18-15

Note that the Package.dtsx item is read-only as it is now saved in VSS.

Click the Stop button (or press Shift+F5) to stop the debugger. You can view the resulting output by opening Windows Explorer and double-clicking the C:\File1.txt file, as shown in Figure 18-16.

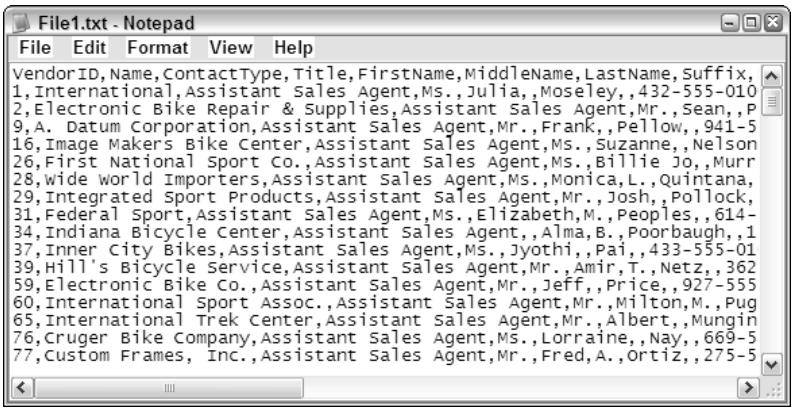

Figure 18-16

You will now roll back to an earlier version of the package. To begin, click File ➪ Source Control ➪ Launch Microsoft Visual SourceSafe. Navigate to the SSISDemo1 folder containing Package.dtsx as shown in Figure 18-17.

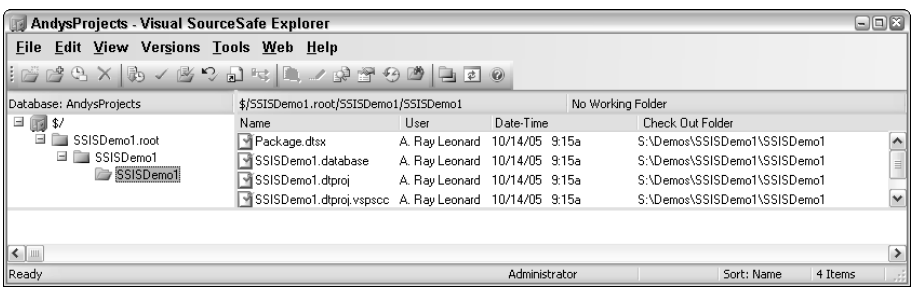

Figure 18-17

View the history of the project by clicking Tools ➪ Show History (or Ctrl+H). The Project History dialog box displays as shown in Figure 18-18.

Figure 18-18

For the purposes of this demo, click the OK button to accept the defaults. The History of Project dialog box appears, showing all source control activity and items, as shown in Figure 18-19.

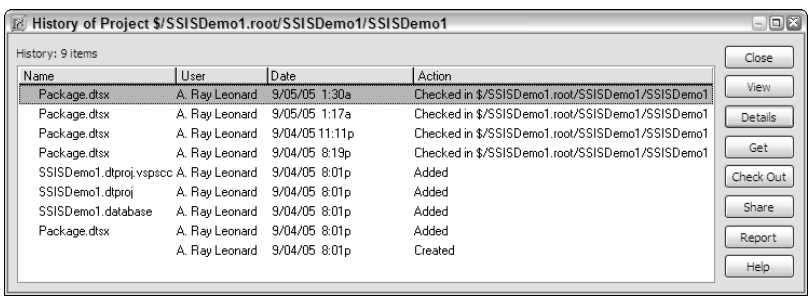

Figure 18-19

Click on the version of Package.dtsx that is the second newest and click the Get button. A dialog box asking if you wish to get the entire project with this version displays. Click the Yes button, and another dialog box prompts you for the location of the project files, as shown in Figure 18-20.

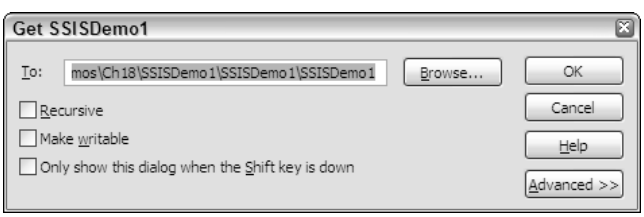

Figure 18-20

Clicking the OK button restores the previous version of code over your existing version. After clicking the OK button, return to the SQL Server Business Intelligence Development Studio environment. A prompt to reload displays as shown in Figure 18-21.

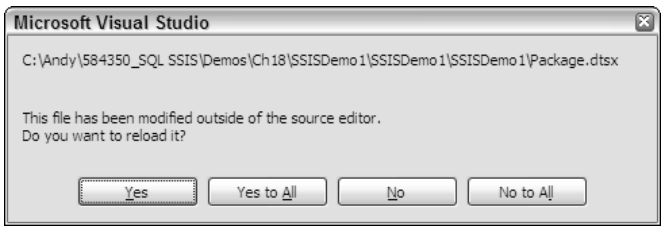

Figure 18-21

Click the Yes To All button to reload all files in the project. Click the Data Flow tab to observe that the Flat File Destination and File1 Connection Manager are no longer part of this project, as shown in Figure 18-22. They have been removed from the project due to your version rollback from source control.

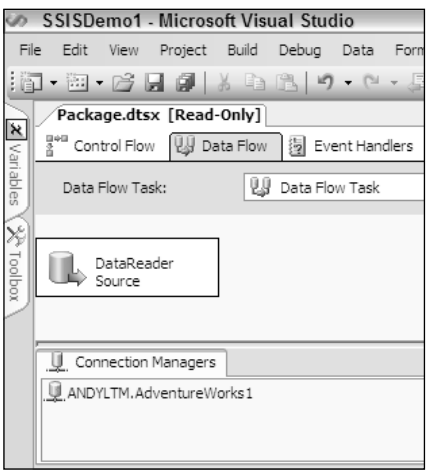

Figure 18-22

Add another Flat File Destination to the Data Flow workspace. Configure this Flat File Destination exactly like the first, except change the file and Connection Manager names from File1 to File2 as shown in Figure 18-23.

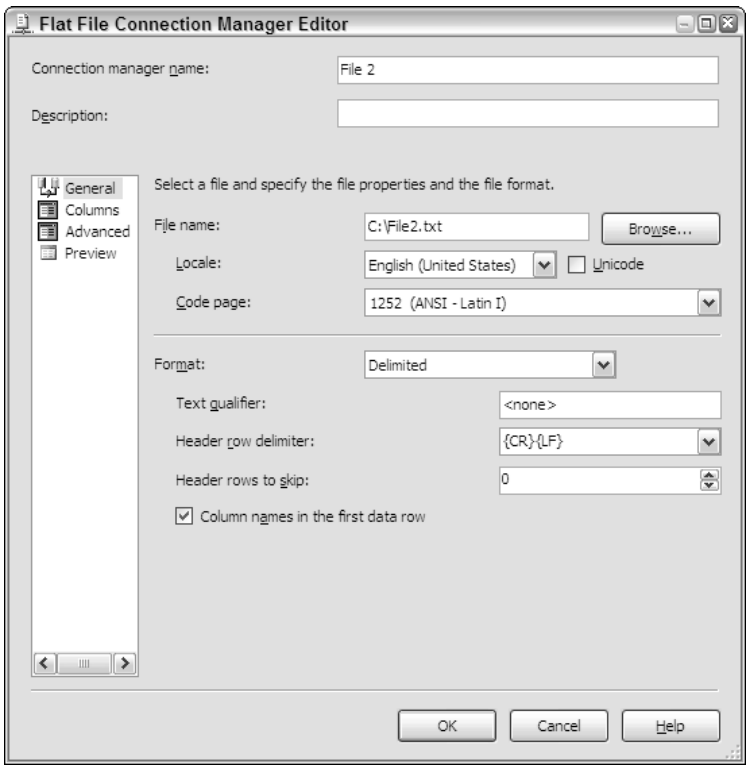

Figure 18-23

Open the Pending Checkins window and add the following comment: "Rolled back and added File2.txt destination." Click the Check In button to store this version in source control.

Execute the package by clicking the Play button. Verify C:\File2.txt is created and populated with Vendor data from the AdventureWorks database.

Return to Visual SourceSafe and click Tools ➪ Show History to view the project history. As before, select the second Package.dtsx in the history list and click the Get button as shown in Figure 18-24.

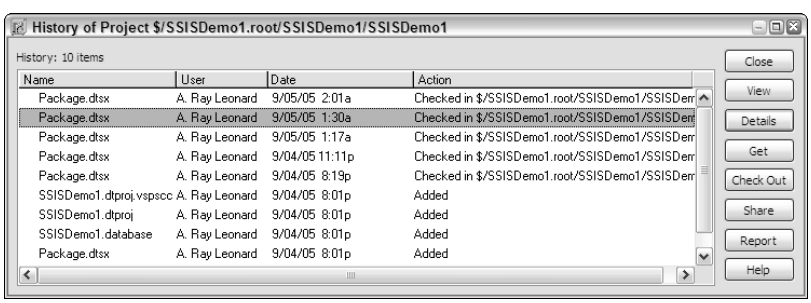

Figure 18-24

Click OK when the location confirmation dialog box displays and return to the BIDS environment. Click the Data Flow Task tab and confirm that you now see the original working version of the package. The File1 Connection Manager and Flat File Destination should now reflect this status, as shown in Figure 18-25.

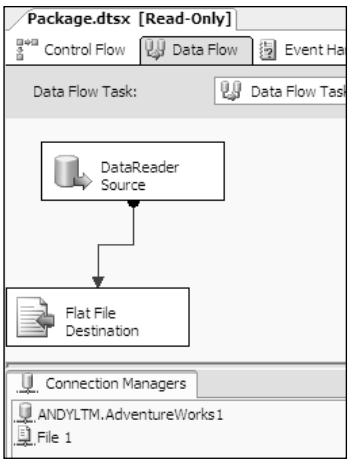

Figure 18-25

This example has provided a rudimentary procedure for manually accomplishing *branching* — a topic that will be covered in a section to come.

Visual SourceSafe is a familiar source control tool to many with application development experience. For this reason, it has been updated and integrated into the 2005 integrated development environments. The new version addresses many complaints and shortcomings of previous versions of the product that were not touched on in this section. One example of this is the native Internet connectivity functionality.

The next section provides a brief introduction to Microsoft's new source control (and so much more) server and client tools known collectively as Team System.

Team Foundation Server, Team System, and SSIS

With the coordinated release of SQL Server 2005 and Visual Studio 2005, Microsoft introduced Team System and Team Foundation Server — a powerful enterprise software development life cycle suite and project management repository consisting of collaborative services, integrated functionality, and an extensible application programming interface (API). Team System seamlessly integrates software development, project management, testing, and source control into the Visual Studio 2005 IDE.

To configure Team Foundation Server as your SSIS source control, click Tools ➪ Options. Click Source Control and select Visual Studio Team Foundation. Expand the Source Control node for detailed configuration, as shown in Figure 18-26.

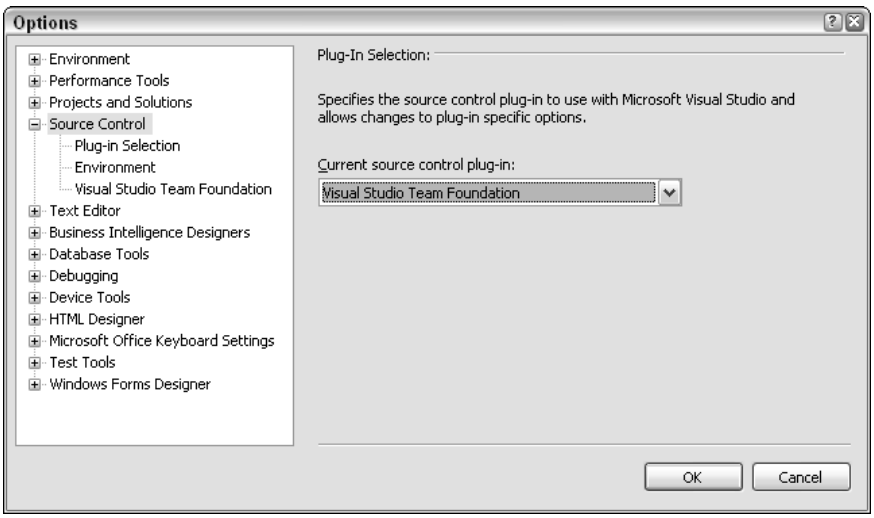

Figure 18-26

This section discusses the relationship between Team System and SQL Server Integration Services. The walk-through is shown using Visual Studio 2005, but it can be completed in SQL Server Business Intelligence Developer Studio (BIDS).

If Visual Studio 2005 is installed, opening BIDS will open Visual Studio 2005. If Team System is specified as the source controller for either environment, the environment, upon opening, will attempt to connect to a Team Foundation Server. Open Visual Studio 2005 to proceed.

Once Visual Studio 2005 is open, press Shift+Alt+T or click the Team Explorer tab to view the Team System properties. Click the Connect to the Team Foundation Server icon (as shown in Figure 18-27) to connect to the Team System server.

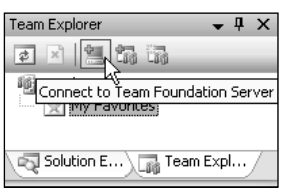

Figure 18-27

Click the Servers button to browse for a Team Foundation Server or select a TF Server from the drop-down list as shown in Figure 18-28.

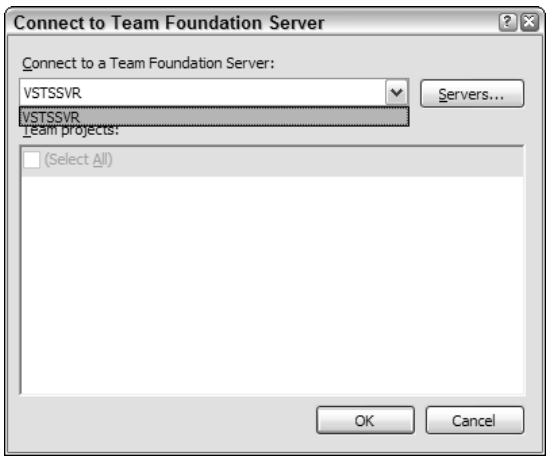

Figure 18-28

Once you've connected to the Team Foundation Server, open the Team Explorer and click the New Team Project icon, or right-click the Team Foundation Server and click New Team Project. The New Team Project wizard starts. Enter a name and optional description for the new team project, and click Next to continue. Select a Process Template on the next step of the New Team Project wizard, as shown in Figure 18-29.

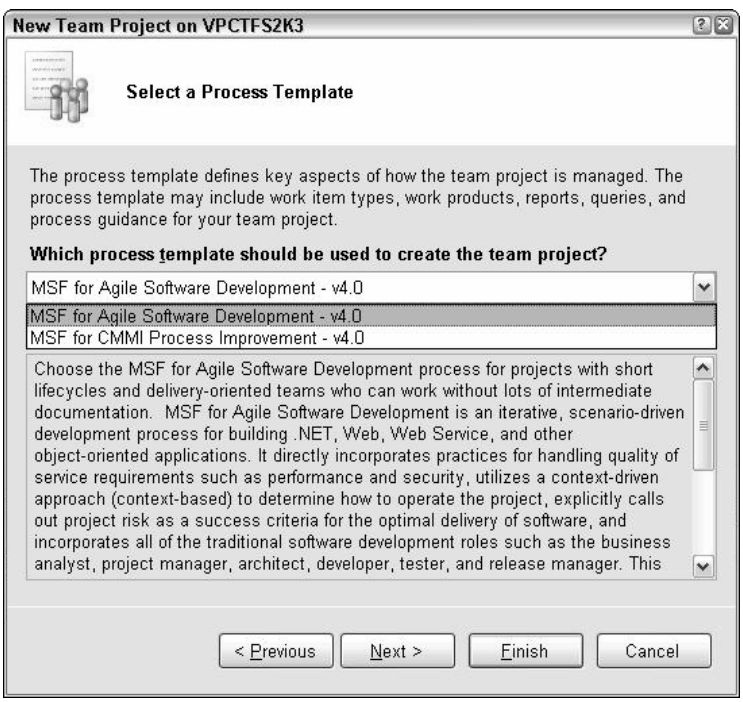

Figure 18-29

Click Next to continue. Here's where Team System gets fun: The good people on the Team System team at Microsoft automated the process of creating a project management Web site using Windows SharePoint Services and Reporting Services.

Click Next to proceed and enter a Team Project Portal title and description in the next step of the wizard as shown in Figure 18-30, and click Next to continue.

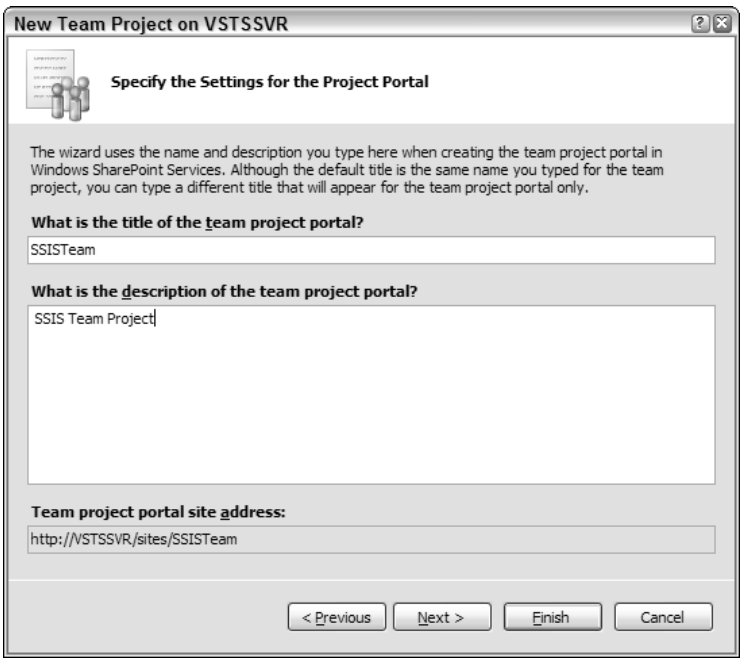

Figure 18-30

In the next step of the wizard, you'll initialize source control and click Next to continue. The confirmation dialog box displays a summary of selections made. Click Finish to set up the new Team Project. A new Team Project is defined according to the configuration you specified. Creation status is indicated by a progress bar as setup scripts execute. If all goes as expected, the wizard will display a Project Created Successfully dialog box as shown in Figure 18-31.

At this point, you have created a Team System container for your SSIS projects. A Team Project is similar to a Visual Studio solution, in that you can add several SSIS projects (or any other type of project) to it.

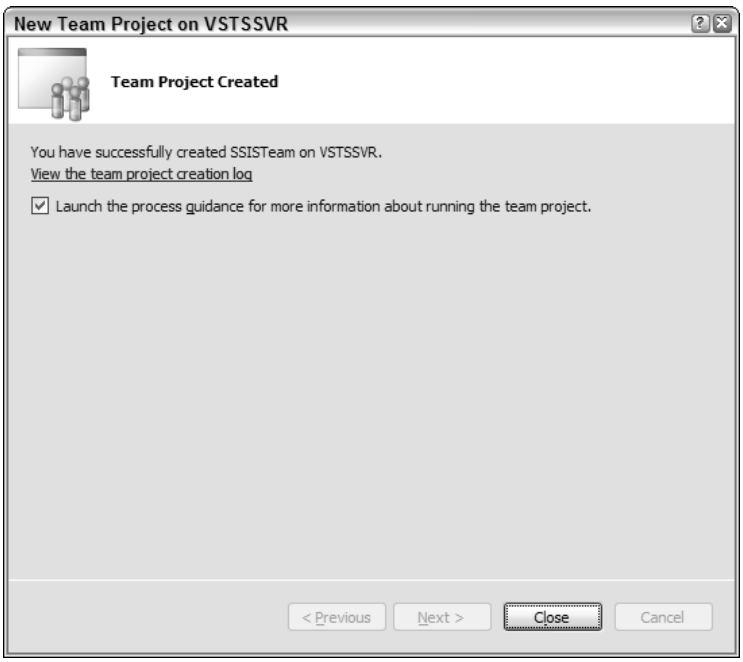

Figure 18-31

If this is your first Team Project, leave View Project Creation Log File checked, and click the Close button to complete the New Team Project wizard and view the log file. The project creation log file provides a lot of information that is helpful in troubleshooting should the project creation fail. If the project is created successfully, there is no need to view the project creation log.

View Process Guidance Page is checked by default. Team System provides a great overview of the process in the Process Guidance page as shown in Figure 18-32. These pages provide a wealth of information, useful to beginners and the experienced alike.

> *"Why create a Team Project?" you ask? The short answer is, "The practice of database development is changing." Team development is becoming practical, even required for DBAs, in software shops of all sizes. It is no longer confined to the enterprise with dozens or hundreds of developers.*

Team System provides a mechanism for DBAs to utilize team-based methodologies, perhaps for the first time. The Team Project is the heart of Team System's framework for the database developer.

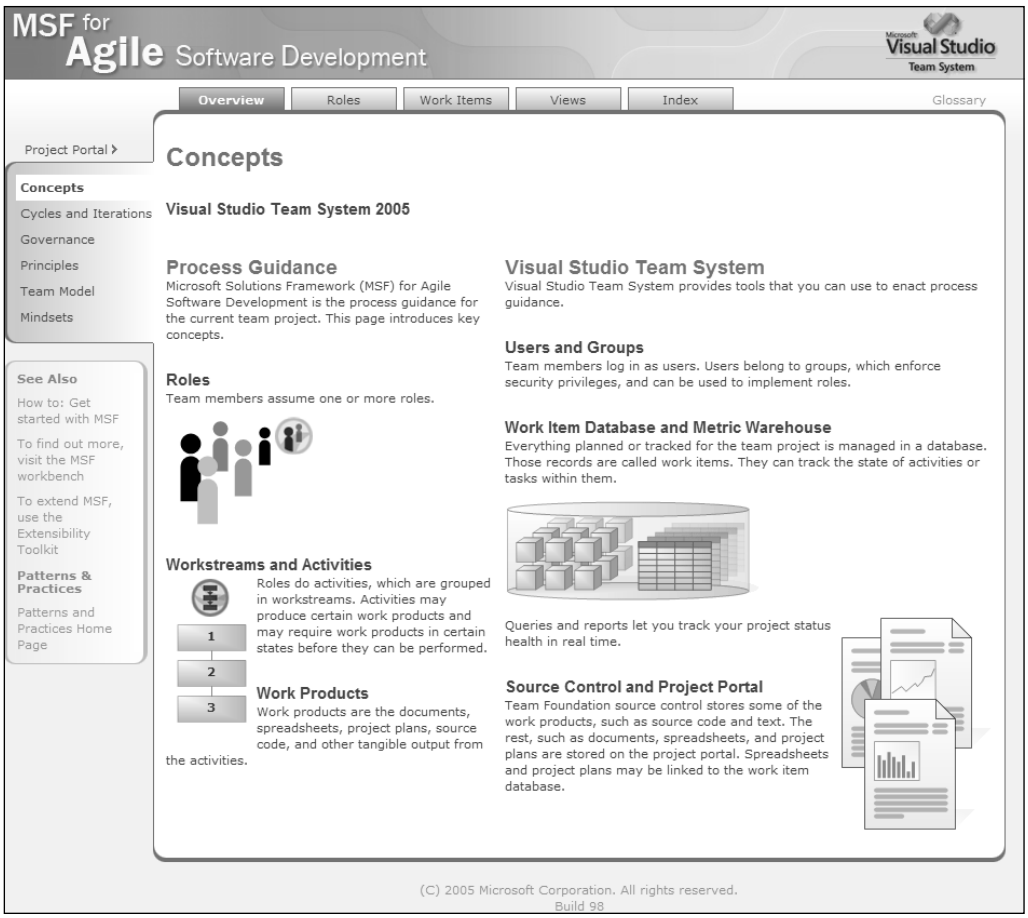

Figure 18-32

MSF Agile and SSIS

MSF Agile is an iterative methodology template included with Team System. In a typical agile software project, a time- and scope-limited project — called an *iteration* — is defined by collaboration with the customer. Deliverables are established, but they may be de-scoped in the interests of delivering a completed feature-set at the end of the iteration. An important aspect of agile iterations is that features slip, but timelines do not slip. In other words, if the team realizes that all features cannot be developed to completion during the time allotted, the time is not extended, and features that *cannot* be developed to completion are removed from the feature-set.

The author advocates agile methodologies.

No one uses a single methodology alone. There are facets of waterfall thinking in any iterative project. In practice, your methodology is a function of the constraints of the development environment imposed by regulatory concerns, personal style, and results.

Once an MSF Agile Team Project hierarchy has been successfully created, the following subitems are available under the project in Team Explorer (see Figure 18-33):

- Work Items
- Documents
- Reports
- Team Builds
- Source Control

Figure 18-33

Take a moment now to examine some subitems.

Work Items

In MSF Agile projects, work items consist of Tasks, Bugs, Scenarios, and Quality of Service Requirements. Bugs are self-explanatory — they are deficiencies or defects in the code or performance of the application. Scenarios map to requirements and are akin to Use Cases in practice. Quality of Service (QoS) Requirements include acceptable performance under attack or stress. QoS includes scalability and security. Tasks are a catchall category for work items that includes features yet to be developed.

Documents

The MSF Agile template includes several document templates to get you started with project documentation. Included are the following:

- Process Guidance — An HTML document that describes the MSF Agile process
- Development and Testing Project Plans — Microsoft Project templates for development and testing efforts
- Project Checklist — A template containing a project "to do" list

- Scenarios Spreadsheet — Listing requirements for validation scenarios
- Persona document — A template for listing all parties connected to the project
- QoS Requirements document — Defining the Quality of Service Requirements and conditions

Reports

The MSF Agile template contains several built-in Reporting Services project status reports. These reports are accessible directly from Reporting Services or from the Project Portal (SharePoint Portal Services) Web site.

The Reporting Services home page contains links to several reports as shown in Figure 18-34.

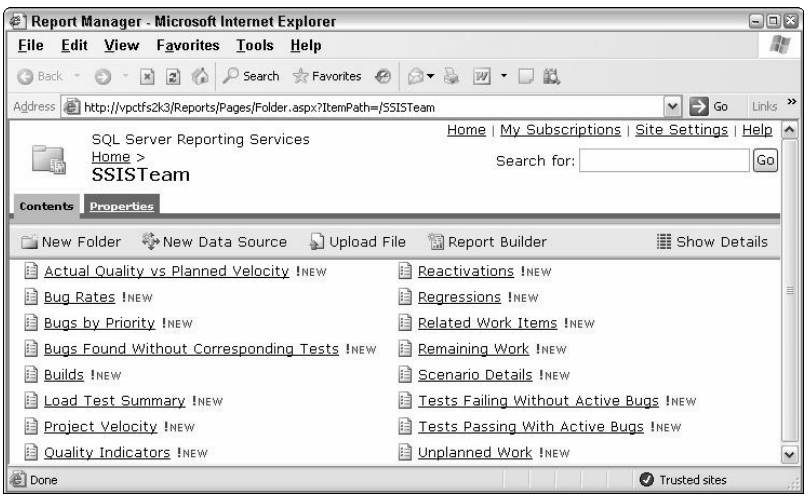

Figure 18-34

The reports are formatted in a style sheet that complements the SharePoint Portal Web site. The Remaining Work report is shown in Figure 18-35.

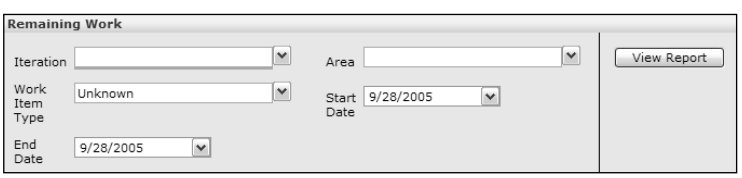

Figure 18-35

The Remaining Work report is part of the larger reporting solution provided by the Project Portal. The Project Portal provides a nice interface for the development team, but project managers are the target audience. The Project Portal can also serve to inform business stakeholders of project status.

To navigate to the Project Portal home page, right-click the Team Project in the Team Explorer and click Show Project Portal.

The Project Portal

The Project Portal (see Figure 18-36) is implemented in SharePoint Portal Services and contains several helpful portals, including the following:

- Main Menu
- Announcements
- Links
- Reports (Bug Rates, Builds, and Quality Indicators)

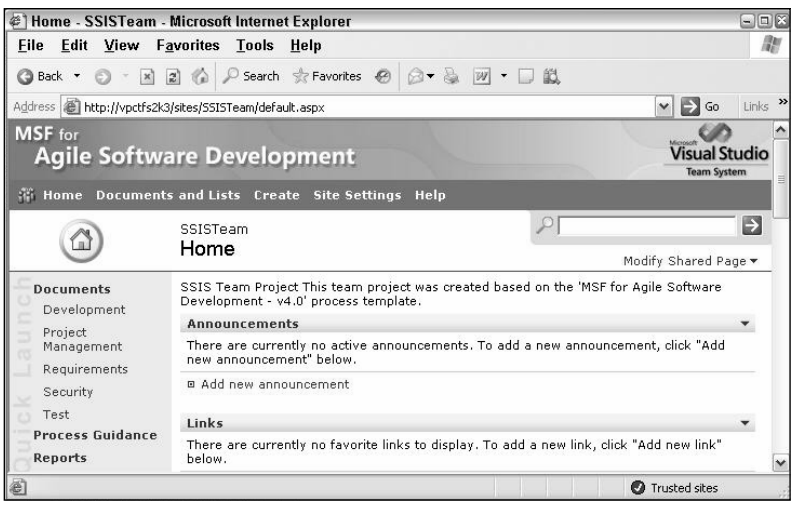

Figure 18-36

Putting It to Work

In this section, you'll create a small SSIS package to demonstrate some fundamental Team System features. To begin, create a new SSIS package in Visual Studio 2005 by clicking File ➪ New ➪ Project. From the Project Types treeview, select Business Intelligence Projects. From the Templates listview, select Integration Services Project. Do not check the Add to Source Control check box. Enter **SSISDemo** as the project name in the Name text box as shown in Figure 18-37.

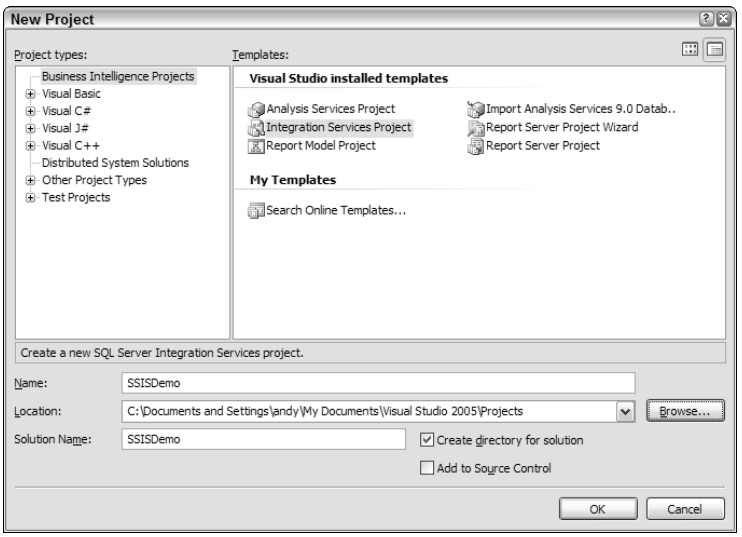

Figure 18-37

Click OK to create the new project. Drag a Data Flow task onto the Control Flow workspace as shown in Figure 18-38.

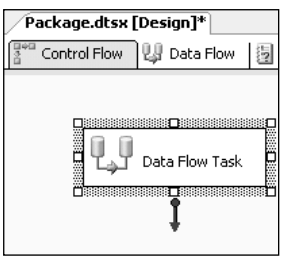

Figure 18-38

Right-click in the Connection Managers tab and select New OLE DB Connection to add a database connection. Click the New button to create a new OLE DB Connection and complete the configuration dialog box, as shown in Figure 18-39.

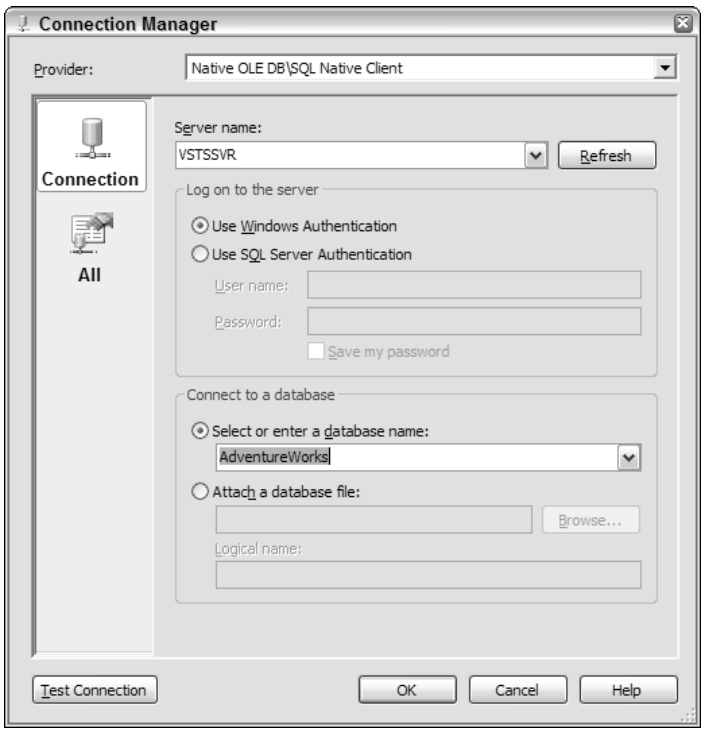

Figure 18-39

Select your local server from the Server Name drop-down list. Configure the connection for Windows or SQL Server authentication. Select AdventureWorks as the database name. You can click the Test Connection button to test connectivity configuration. Click OK to close the Connection Manager dialog, and OK again to continue.

Double-click the Data Flow task to edit. Drag an OLE DB Source onto the Data Flow workspace. Double-click the OLE DB Source to edit. Select the AdventureWorks connection in the OLE DB connection manager drop-down list. Select Table or View in the Data Access Mode drop-down list. Select [Sales].[vStoreWithDemographics] in the Name of Table or View drop-down list, as shown in Figure 18-40.

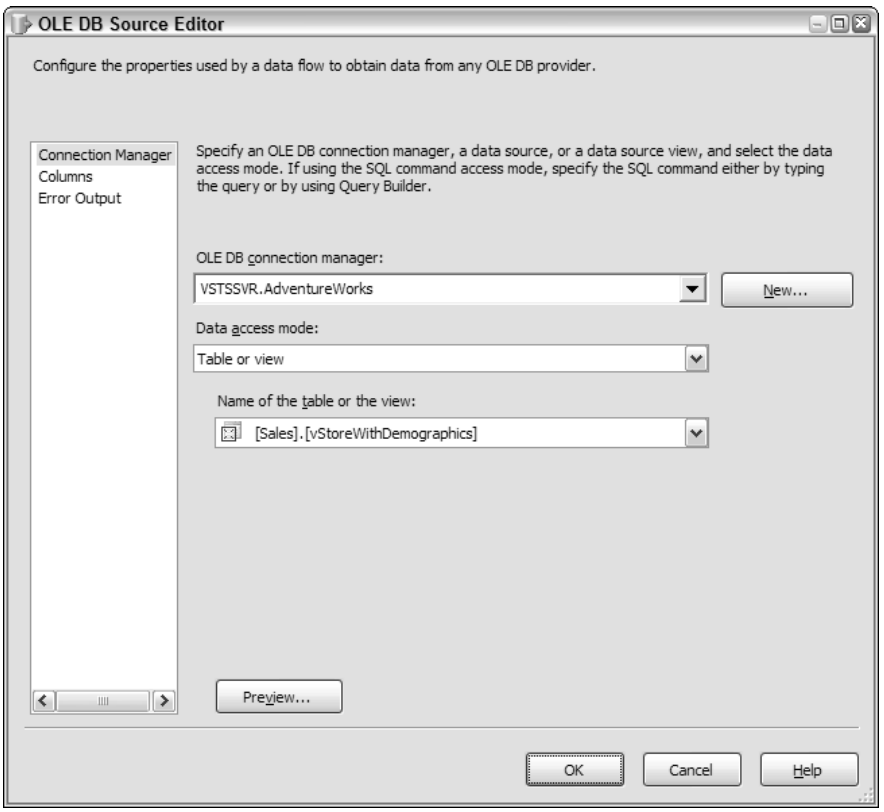

Figure 18-40

Click OK to continue. Drag an Aggregate transformation onto the Data Flow workspace. Connect the output of the OLE DB Source to the Aggregate transformation by dragging the green arrow from the source to the transformation. From the Available Input Columns table, select StateProvinceName, SquareFeet, and AnnualSales. In the grid below, ensure that the operation for StateProvinceName is Group by, the operation for SquareFeet is Average, and the operation for AnnualSales is Sum as shown in Figure 18-41.

Click OK to close the Aggregate editor, and drag an OLE DB Source Output (denoted by the green arrow) from the OLE DB Source to the Aggregate as shown in Figure 18-42.

Drag an Excel Destination onto the Data Flow workspace and connect the Aggregate output to it. Double-click the Excel Destination to open the Excel Destination Editor. Click the New button beside the OLE DB Connection Manager drop-down list to create a new Excel connection object. Enter or browse to the path of an Excel file as shown in Figure 18-43.

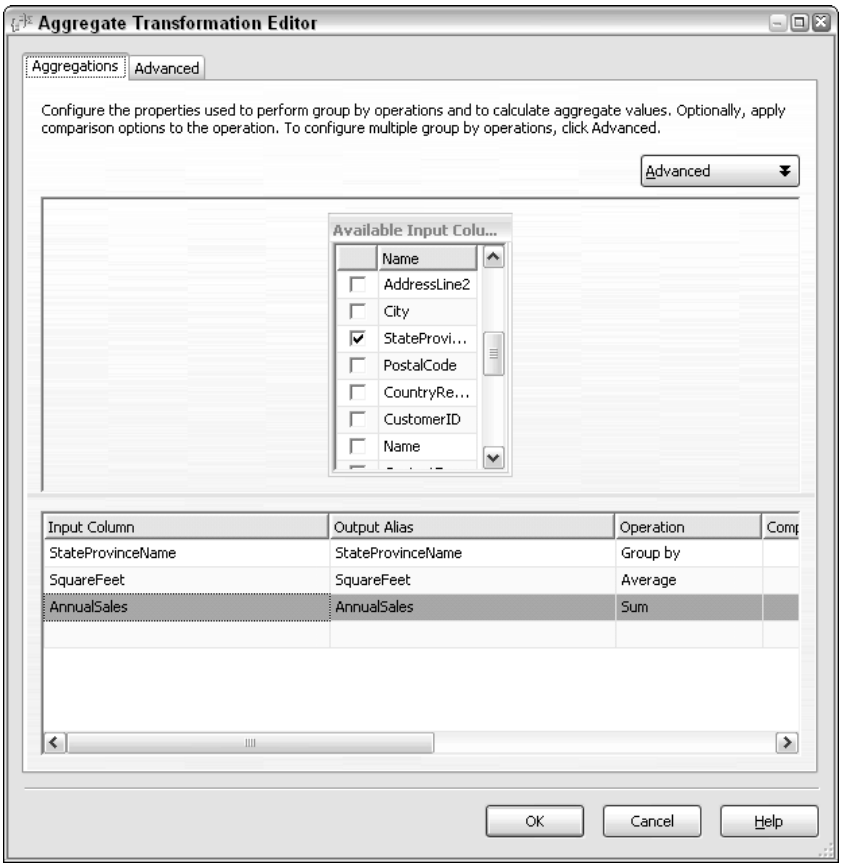

Figure 18-41

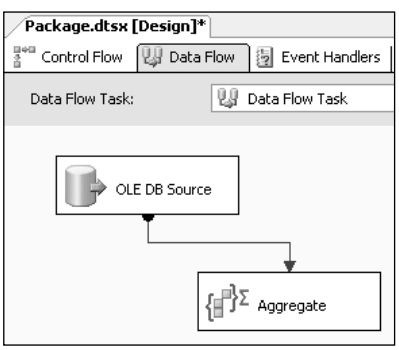

Figure 18-42

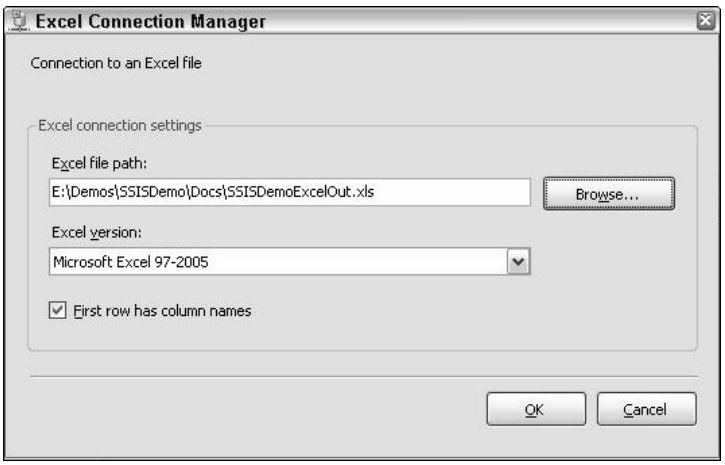

Figure 18-43

Click OK to continue.

You can create an Excel spreadsheet in this step. If you enter the desired name of a spreadsheet that does not yet exist, the Excel Destination Editor will not be able to locate a worksheet name. The "No tables or views could be loaded" message to this effect will appear in the Name of Excel Worksheet drop-down list as shown in Figure 18-44.

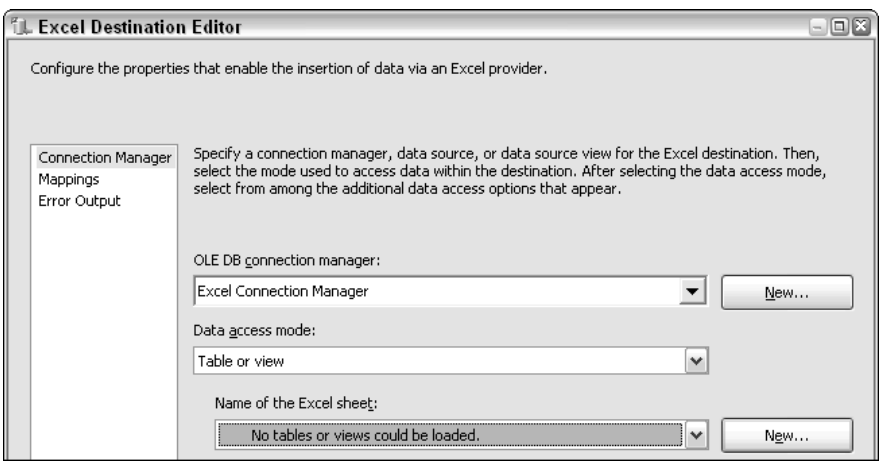

Figure 18-44

To create a worksheet, click the New button beside the Name of the Excel Sheet drop-down list. A Create Table dialog box will appear as shown in Figure 18-45. Click OK to accept the defaults and create the worksheet and Excel workbook.

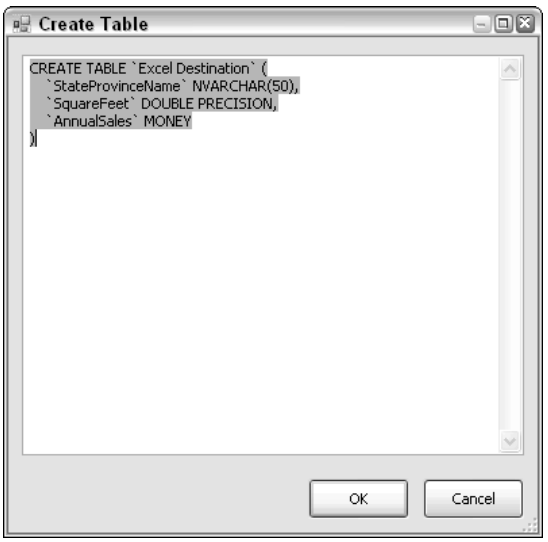

Figure 18-45

Click Mappings in the Excel Destination Editor to configure column-to-data mappings. Accept the defaults by clicking OK.

Click File ➪ Save All to save your work.

Version and Source Control with Team System

To add your SSIS project to the Team Project, open the Solution Explorer, right-click the project, and click Add to Source Control.

The Add Solution SSISDemo to Source Control dialog box appears containing a list of Team Projects. Select the SSISTeam Team Project you created earlier, as shown in Figure 18-46.

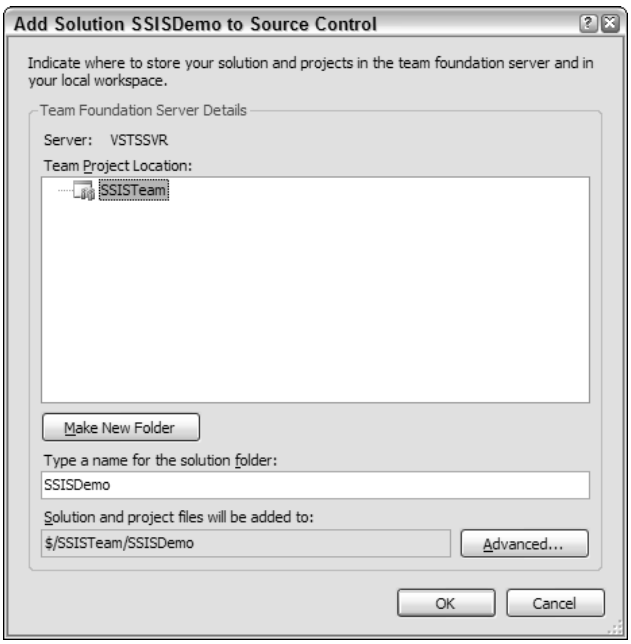

Figure 18-46

Click OK to continue.

You have successfully created a Team Project and a SSIS project. The Team Project contains version control information — even now.

> *Source control is loosely modeled after a library. Items are checked out for modification and checked in when modifications are complete.*

Click View ➪ Other Windows ➪ Pending Changes to view the current source control status for the SSIS project, as shown in Figure 18-47.

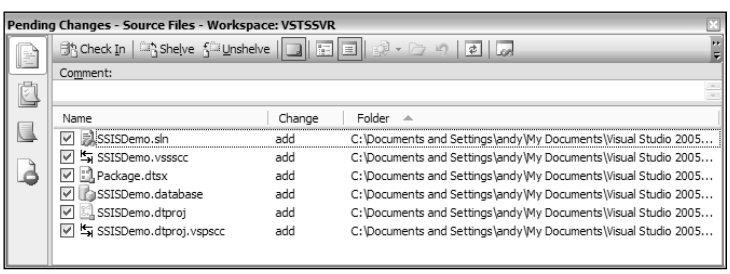

Figure 18-47

The Change column indicates that the files are currently in an Add status. This means the files are not yet source-controlled but are ready to be added to source control.

Click the Check In button to add the current SSISDemo project to the SSISTeam Team Project's source control. This clears the Pending Checkin list. Editing the SSISDemo SSIS project will cause the affected files to reappear in the Pending Checkin list.

> *Any change to the SSISDemo project is now tracked against the source-controlled version maintained by the SSISTeam Team Project. Seemingly insignificant changes count: For instance, moving any of the items in the Data Flow workspace is considered an edit to the package item and is tracked.*

The default behavior for source control in Visual Studio 2005 is that checked-in items are automatically checked out when edited. You can view the current status of all Team Projects on your Team Foundation Server in the Source Control Explorer. To access the Source Control Explorer, double-click Source Control in the Team Explorer or click View ➪ Other Windows ➪ Source Control Explorer as shown in Figure 18-48.

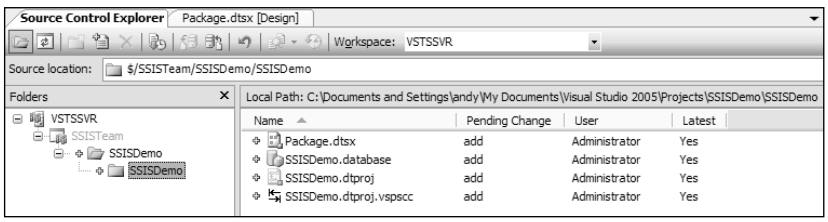

Figure 18-48

The example will now implement a larger change to demonstrate practical source control management before moving into some advanced source control functionality. In your SSIS project, add an Execute SQL task to the Control Flow workspace. Configure the task by setting the Connection Type to OLE DB, the Connection to your AdventureWorks connection, and the SQLSourceType to Direct input, as shown in Figure 18-49.

Set the SQLStatement to the following:

```
if not exists(select * from sysobjects where id = object_id('Log')
 and ObjectProperty(id, 'IsUserTable') = 1)
 begin
    CREATE TABLE Log (
      LogDateTime datetime NOT NULL,
      LogLocation VarChar(50) NOT NULL,
      LogEvent VarChar(50) NOT NULL,
      LogDetails VarChar(1000) NULL,
      LogCount Int NULL
    ) ON [Primary]
    ALTER TABLE Log ADD CONSTRAINT DF_Log_LogDateTime DEFAULT (getdate()) FOR
        LogDateTime
  end

INSERT INTO Log
(LogLocation, LogEvent, LogDetails, LogCount)
VALUES('SSISDemo', 'DataFlow', 'Completed', '1st Run')
```

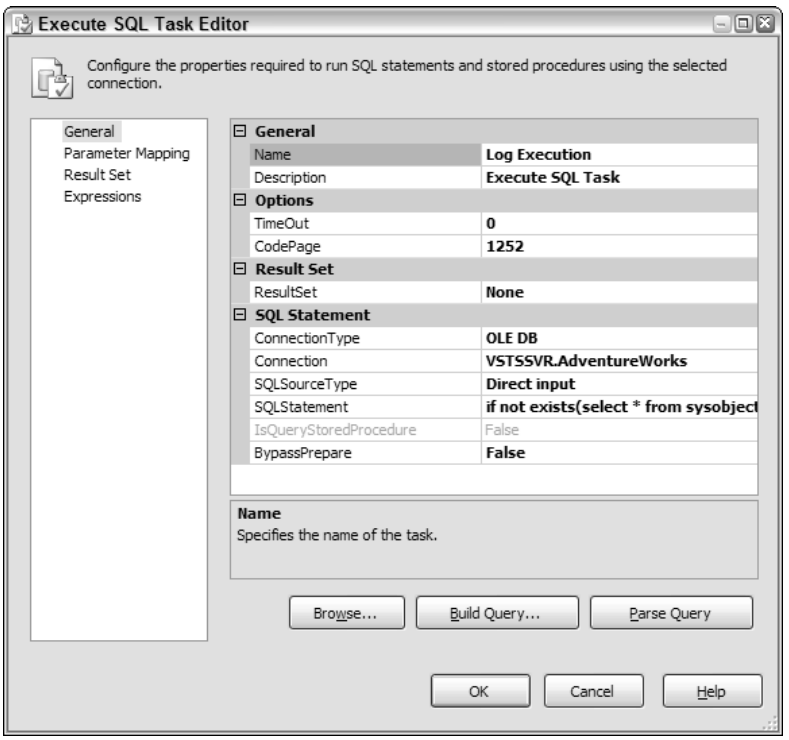

Figure 18-49

It is always a good practice to check your SQL before execution. Do so by clicking the Parse Query button and correct if necessary. Then click OK to continue.

Connect the Data Flow task to the Execute SQL task by dragging the output (green arrow) of the Data Flow task over the Execute SQL task as shown in Figure 18-50.

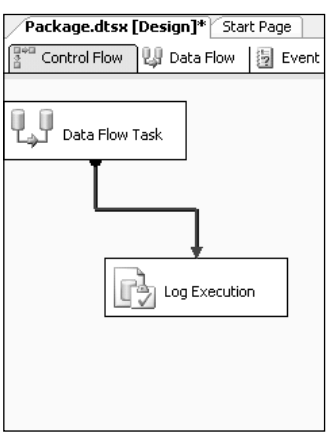

Figure 18-50

Save your changes by clicking the Save button on the toolbar. You now have updated your SSIS project and saved the changes to disk, but you have not committed the changes to source control. You can verify this in the Pending Changes window by clicking View ➪ Other Windows ➪ Pending Changes as shown in Figure 18-51.

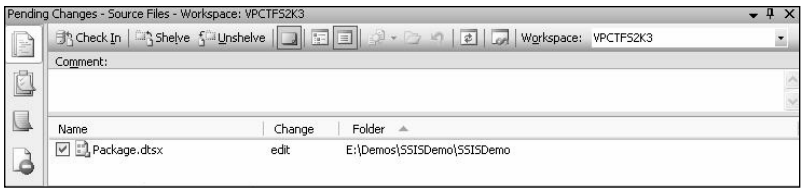

Figure 18-51

The Change column indicates that the Package.dtsx is in an Edit status. This means that changes to the existing source-controlled Package.dtsx file have been detected. Click the Check In button. The next section introduces shelving and unshelving changes, using the code in its current state.

Shelving and Unshelving

Shelving is a new concept in Microsoft source control technology. It allows you to preserve a snapshot of the current source state on the server for later retrieval and resumed development. You can also shelve code and pass it to another developer as part of a workload reassignment. In automated nightly build environments, shelving provides a means to preserve semi-complete code in a source control system without fully checking it into the build.

To shelve code, click the Shelve button on the Pending Checkin toolbar. The Shelve dialog box appears, as shown in Figure 18-52.

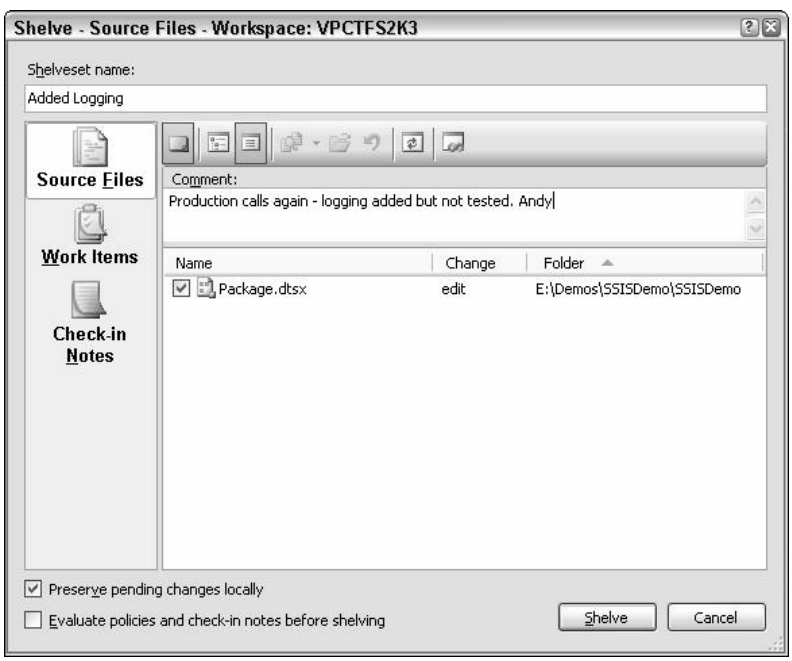

Figure 18-52

The "Preserve pending changes locally" checkbox allows you to choose between rolling back or keeping the edits since the last source code check-in. Checking the checkbox will keep the changes. Unchecking the checkbox will roll changes back to the last source-controlled version.

The rollback will effectively "undo" all changes — even changes saved to disk.

Leave the Preserve Pending Changes Locally checkbox checked and click Shelve to proceed.

To unshelve code, click the Unshelve button on the Pending Checkin toolbar. You'll see the dialog box shown in Figure 18-53.

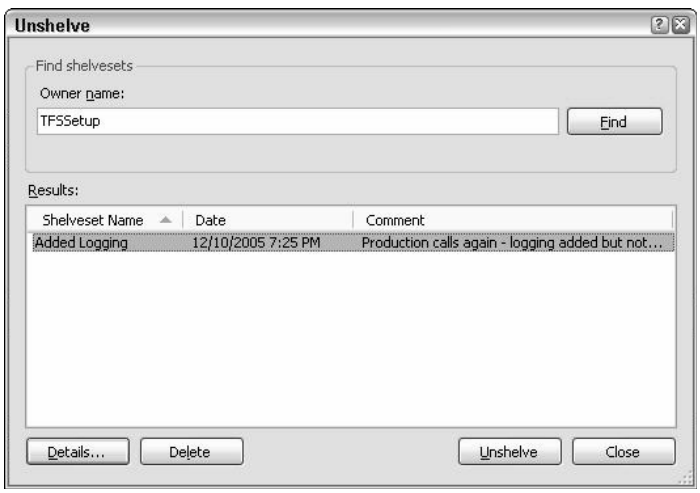

Figure 18-53

Click Unshelve to proceed with unshelving. The Unshelve Details wizard opens, providing options for unshelving metadata and preserving the shelve set on the server as shown in Figure 18-54.

Unshelving code with conflicts will roll the project back to its state at the time of shelving. For this reason, you may wish to consider shelving your current version of the code prior to unshelving a previous version.

If you see the dialog box similar to the one shown in Figure 18-55 and respond by clicking Yes or Yes to All, your current version will be rolled back to the shelve set version.

Branching

The ability to *branch* code provides a mechanism to preserve the current state of a SSIS project *and* modify it in some fashion. Think of it as driving a stake in the sod of project space marking the status of the current change set as "good."

To branch, open Source Control Explorer. Right-click the project name you wish to branch and click Branch from the context menu, which brings up the Branch dialog box shown in Figure 18-56. Select a name for the branched project and enter it into the To text box. Note the option to lock the new branch — thus preserving it indefinitely from accidental modification. You can further secure the branched code by including the option to not create local working copies for the new branch.

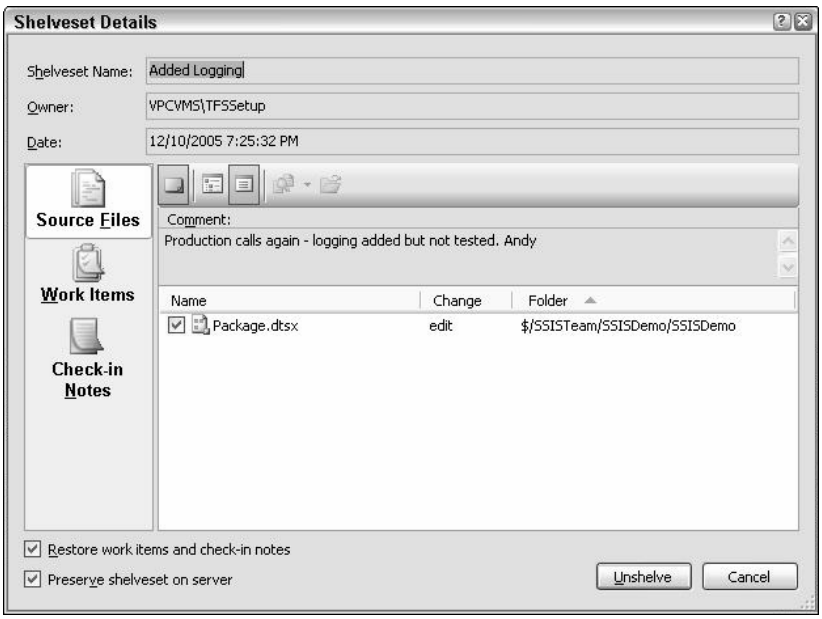

Figure 18-54

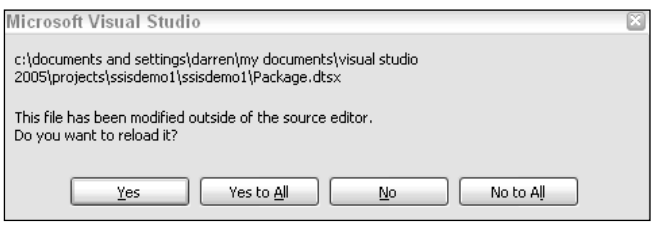

Figure 18-55

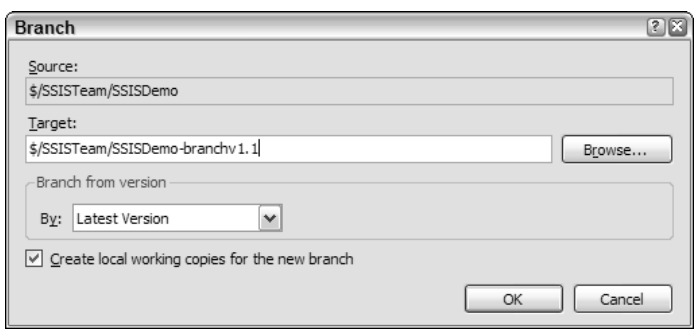

Figure 18-56

Merging

Merging is the inverse operation for branching. It involves recombining code that has been modified with a branch that has not been modified.

To merge projects, open Source Control Explorer. Right-click the name of the branched project containing the changes and click Merge. The project you right-clicked in the previous step should appear in the Source Branch text box of the Version Control Merge Wizard. Select the Target branch (the branch containing no changes) from the Target Branch drop-down, as shown in Figure 18-57. Note the options to merge all or selected changes from the Source branch into the Target branch. Click Next to proceed.

The Source Control Merge Wizard allows users to select the version criteria during merge. The options include Latest Version (default), workspace, label, date, and change set. Click Finish to proceed.

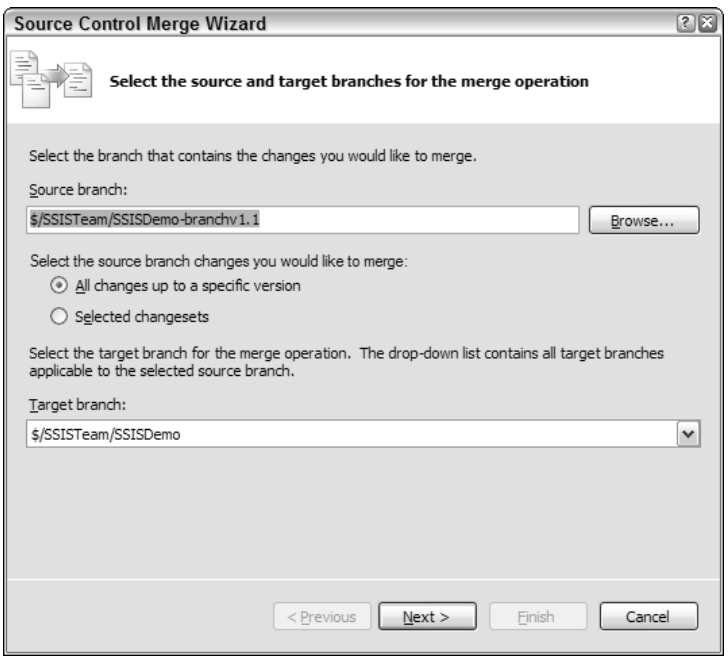

Figure 18-57

If the Version Control Merge Wizard encounters errors while attempting the merge, the Resolve Conflicts dialog box is displayed. Click Auto-Merge All to attempt an automatic merge. Click Resolve to manually merge branches.

When all conflicts have been resolved, the Resolve Conflicts dialog will reflect this condition.

Labeling (Striping) Source Versions

Labeling provides a means to mark (or "stripe") a version of the code. Generally, labeling is the last step performed in a source-controlled version of code — marking the version as complete. Additional changes require a branch.

To label a version, open Source Control Explorer. Right-click the project and click Apply Label. Enter a name for the Label and optional comment. Click the Add button to select files or project(s) to be labeled as shown in Figure 18-58.

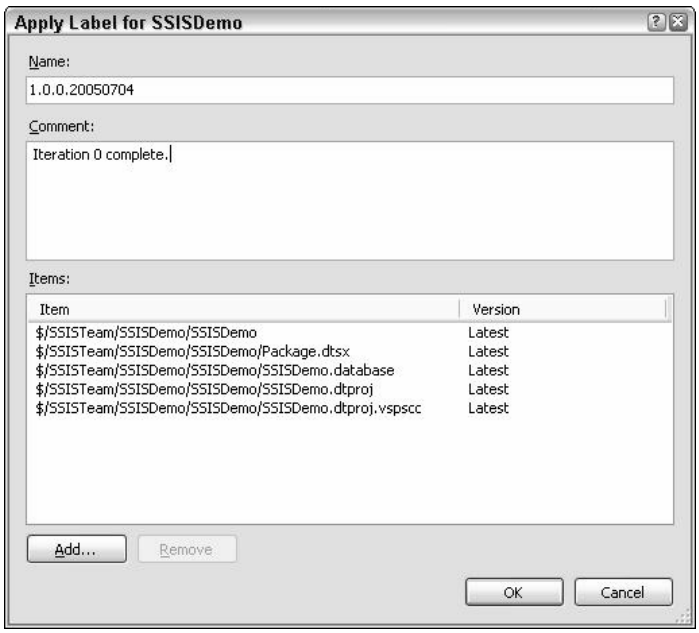

Figure 18-58

Click OK to complete labeling.

Much has been debated about when to Shelve, Branch, or Label. The following advice is based on years of utilizing many software source control products:

- **Shelve** — When your code is not code complete. In other words, if your code isn't ready for the nightly or weekly build, shelve it for now.
- **Branch** — When you need to add functionality and features to an application that can be considered complete in some form. Some shops will have you branch if the code can be successfully built; others will insist on no branching unless the code can be labeled.
- **Label** — When you wish to mark a version of the application as "complete." In practice, labels *are* the version; for instance "1.2.0.2406."

Code Deployment and Promotion from Development to Test to Production

In DTS, you had the option to store packages in SQL Server or persist them to disk as either a Visual Basic or Structured Storage file. But most DTS development occurred within the context of a SQL Server.

SQL Server Integration Services is decoupled from the SQL Server engine. Packages are developed in either Business Intelligence Development Studio or Visual Studio 2005. Because of this, code promotion is addressed in different ways.

For instance, in DTS you would likely develop a package on a local or development server. You would unit test the package to ensure proper functionality and desired results, and then you would click Package➪Save As to promote the package to a test server. This was by no means the only method available, but it was a popular method for accomplishing package migration through the SDLC hierarchy in many SQL Server 2000 shops.

Now that SSIS is decoupled from the SQL Server environment, developing packages is equivalent to external software development. Some DBAs have experience as developers. To them, this will pose no challenge or threat. To others, this may be outside their comfort zone. To the uncomfortable, I say, "Relax. You can do this!"

The Deployment Wizard

You will now look at one method for migrating a package created in Visual Studio 2005 into an instance of SQL Server 2005 Integration Services: using the Deployment wizard.

> *For the example, you'll use a project you built in Chapter 17. You can substitute the previous project or a project of your own.*

In Solution Explorer, right-click the project and click Properties to display the project Property Pages. Click Deployment Utility beneath Configuration Properties and set CreateDeploymentUtility to True as shown in Figure 18-59.

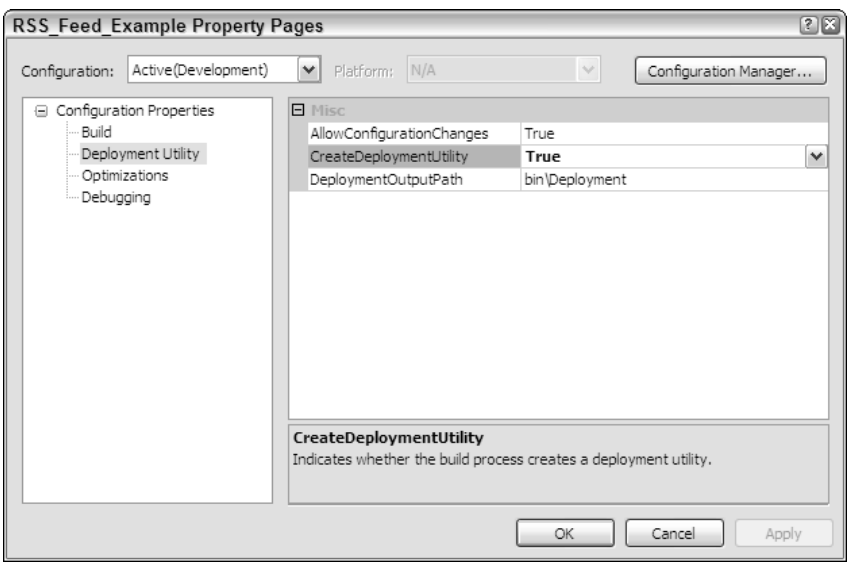

Figure 18-59

Click OK to close the Property Pages.

Build the solution in Visual Studio 2005 (or Business Intelligence Development Studio) by clicking Build ➪ the Build Solution (or Build [Solution Name]). A \Deployment folder is created in the project \bin directory if you accepted the Configuration Property defaults in a previous step. The Deployment folder contains the package dtsx files (one per package in the project) and a file of type SSISDeploymentManifest (one per project). To deploy the package, right-click the SSISDeploymentManifest file and click Deploy to start the Package Installation Wizard.

The Package Installation Wizard allows you to install a SSIS package to an instance of Integration Services or to a File System location. For SQL Server or File System installations, a folder is created (the default directory is in %Program Files%) to hold support files only or support and package dtsx files, respectively — as shown in Figure 18-60.

Figure 18-60

After you select the installation location, click Next to continue. A confirmation wizard screen displays; click Next to continue. A summary displays showing the location of the files installed; click Finish to complete the installation.

Import a Package

Another method for migrating a code-complete package is to import it directly into an instance of Integration Services on a target server, as follows:

First, build the solution. The Output window will display a message indicating the status of the build. If the Output window is not visible, click View ➪ Other Windows ➪ Output to view it.

Once the solution has been successfully built, open Microsoft SQL Server Management Studio and connect to an instance of Integration Services on the destination SQL Server. In the Integration Services treeview, expand the Stored Packages item. There are two subitems listed beneath Stored Packages: File System and MSDB. The package may be imported into either (or both — with the same name, if desired). Right-click File System or MSDB and click Import Package to begin the import.

Select a Package location (SQL Server, File System, or SSIS Package Store). Choosing File System disables the Server text box and Authentication controls. Select File System. Click the ellipsis beside the Package Path text box and navigate to the dtsx file of the package you desire to import as shown in Figure 18-61. Enter a Package name in the appropriate text box and click OK to import the package.

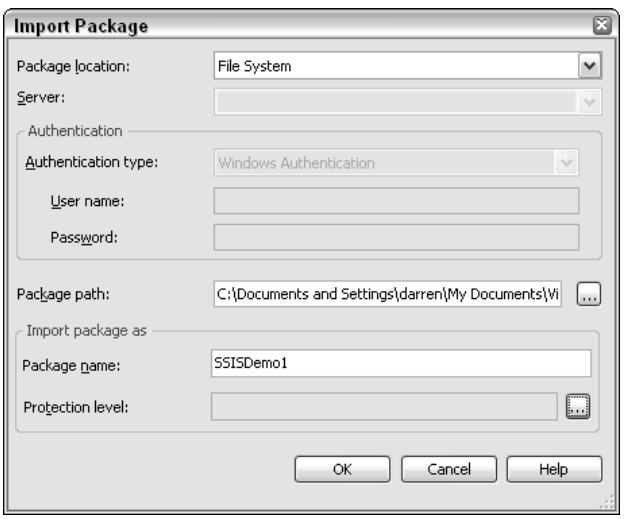

Figure 18-61

Once a package is imported into an instance of SQL Server Integration Services, it may be exported to another instance of SQL Server, File System, or SSIS Package Store via the Export Package functionality as shown in Figure 18-62. To start the export, right-click the Package name and click Export Package.

Export functionality can be used to promote SSIS packages from development to test to production environments.

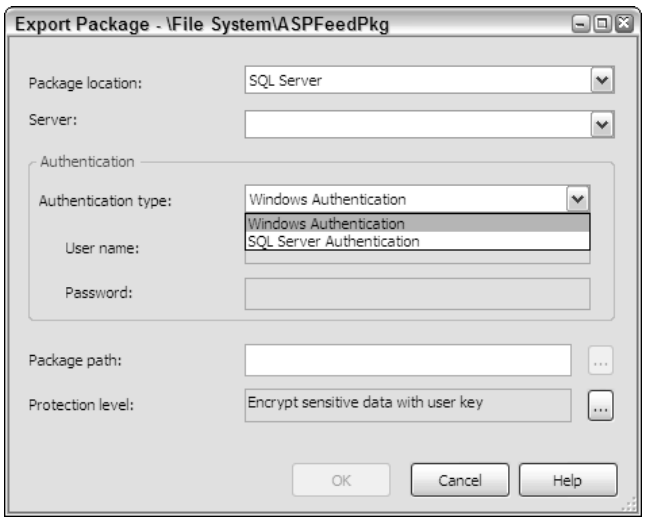

Figure 18-62

Summary

You now have a clearer picture of the Software Development Life Cycle of SSIS projects. In this chapter, you learned how to use the new Integrated Development Environment (IDE) to add SSIS projects to Microsoft Visual SourceSafe. You also learned how to do the following:

- Create a Team Project in Team System
- Add a SSIS project to the Team Project
- Manage and report project status
- Control the SSIS source code

Finally, you have more experience with code promotion — deploying an SSIS package from Development to an Integration Services server, as well as exporting a package to another Integration Services server.

You know more about software development methodologies and about how Team Foundation Server allows you to customize Team System to clearly reflect your methodology of choice. Team System is a fascinating framework! I encourage you to learn more about Team System and Team Foundation Server from the *Professional Visual Studio 2005 Team System*, by Jean-Luc David, et al. (Wiley, 2006), and the MSDN Team System Developer center (`http://lab.msdn.microsoft.com/teamsystem`).