Real World Software Configuration Management

SEAN KENEFICK

Apress™

Real World Software Configuration Management
Copyright ©2003 by Sean Kenefick

ISBN (pbk): 1-59059-065-1

Printed and bound in the United States of America
12345678910

Technical Reviewer: David Birmingham

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Tracy Brown Collins

Copy Editor: Kim Wimpsett

Production Editor: Janet Vail

Compositor: Diana Van Winkle, Van Winkle Design Group

Illustrator: Cara Brunk, Blue Mud Productions

Proofreader: Nancy Sixsmith

Indexer: Kevin Broccoli, Broccoli Information Managment

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 10

Installations

ONCE YOUR DEVELOPERS CREATE a product, the resulting software must somehow end up on a customer's machine. Larger corporations used to divvy out these tasks to install teams—but this luxury is disappearing in these lean days. Many organizations today, especially small shops, choose to task the Software Configuration Manager (SCM) with installing and deploying the goods.

Creating installations for end users has many issues. Of course, instruction manuals for using the tools discussed in this chapter can (and do!) fill up entire books, so this chapter instead briefly describes some of the installation tools on the market and then presents a quick use case for each of them so you can make informed purchase decisions.

Installations are tricky—don't let anyone tell you differently. They seem like they should be simple—you merely ask your customers where they want the software installed and then copy the files over, right? And then come the other questions.... How is the product uninstalled? What if a previous version of the software exists on the machine? Will there be future enhancements to the product that will affect installing the product now? What about allowing the user to customize the application during the installation? You get the picture. Believe it or not, installations can be overwhelmingly complicated.

Worse, many installations are created by people who have never written code before. This can be a problem because a lot of logic is required when writing installations—and programming logic is often learned through experience. Unfortunately, as it is with the SCM, installation developers often get no respect. Other developers may thumb their noses at the install—and be ready, the install is *always* the first "fall guy" when bugs are found—but those same developers are at the mercy of the installation developer. Always remind the programming snobs that there's only one way for the end user to get the code—and that's through the install.

Thinking It Through

Before I discuss the tools used to create installations, let's first spend some time plotting them out. Why? Frankly, most of the large problems you'll run into when writing the installation come from poor preparation. Like any piece of code, an installation should be well architected with scalability in mind before any code is written.

NOTE *Building an installation is a team effort—even if you end up being the only one writing any code. Feel free to have the developers try to install their components to a common location early in the development cycle—even if it means simply copying them to a shared directory—so you have a good starting point from which to work. In addition to helping you out, this sort of exercise can weed out architectural issues and flaws without having to wait for the first draft of the installation project.*

Step 1: Break It Down

Depending on the type of product you support, you may need to provide separately installable features. To ensure that your customers receive the features complete with all of the proper components—regardless of whatever install choices the user may have made—it's important to break down your product into well-documented parts—much like I discussed in Chapter 1, "Getting to Know the SCM Role." Use a spreadsheet program to list all the separate features of the product and what they contain.

TIP *If supporting a large enterprise product, it might be worth your while to instead use a friendly database program such as Microsoft Access.*

When coming up with your list, don't be afraid to break it down into the smallest possible parts. Start by creating a column in your spreadsheet for each large feature of your product. Underneath the appropriate column heading, list the files to be copied and any database or system changes (for instance, the Registry in Windows) that might need to be performed for each feature. The first column should list the "default"—the shared objects that are always installed regardless of the options presented to the end user.

TIP *Don't fall into the shared features trap! It's tempting to group a shared set of features into every column. For instance, if you were to write an installation for an Office-style program, the spell checker might seem like it needs to be part of both the word processor and spreadsheet programs. Instead of simply adding the checker to both columns, try to imagine the spell checker as a separate and worthy feature in its own respect. Once it's been mapped out, you can group the features together as separate installable parts. This allows a great deal of flexibility should the requirements of your product change in the future.*

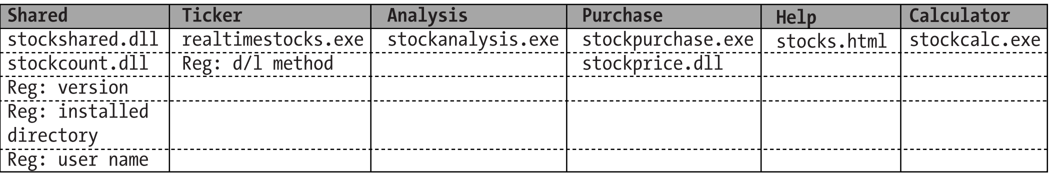
In Chapter 9, "Builds for Windows .NET," I used a stock market tool as an example. I'll use that same example for the installation preparation. Look at the product that was created, and you'll probably come up with six features:

- Shared objects and settings that are always installed.
- The real-time stock ticker. It's optionally installed at the request of the user.
- The stock analysis tool. It's optionally installed at the request of the user.
- The stock purchaser program. It's optionally installed at the request of the user.
- Help. It's a shared set of components that's always installed.
- The calculator. It's a shared set of components that's installed with both the stock analysis and the stock purchaser features.

TIP *Even though you'll always install the Help objects, you may come across a time when you want to optionally install it or split it into smaller Help objects. To facilitate this possibility and to ensure future scalability, separate Help objects into their own components. Think about the other components—a little consideration now can save you from huge headaches later.*

When you're finished, your spreadsheet might look something like Figure 10-1. Keep in mind that this is a simple example—the spreadsheet for your real-life product will probably be much more detailed (and you're the lucky one if it isn't).

TIP *List Windows Registry and initialization file (.INI file) changes in your spreadsheet as if they were files. By treating them as installation "objects," you ensure that they're only changed as necessary.*



Shared	Ticker	Analysis	Purchase	Help	Calculator
stockshared.dll	realtimestocks.exe	stockanalysis.exe	stockpurchase.exe	stocks.html	stockcalc.exe
stockcount.dll	Reg: d/l method		stockprice.dll		
Reg: version					
Reg: installed directory					
Reg: user name					

Figure 10-1. Divvying up the Stock Suite features

Step 2: Map the Features Together into Installable Parts

Once you create your feature chart, "map" the features together into installable parts. This allows users to decide which features they might want to put on their computers. Why install an entire office suite if you only need the word processor?

NOTE *Your product might be small with limited functionality. In this case, there may be no need to provide conditional installations of features. Feel free to skip this section.*

For instance, in the previous example, the Stock Suite has four installable parts:

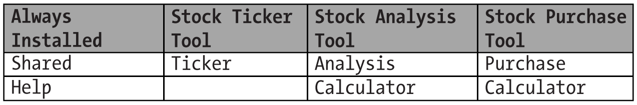
- Shared objects, their settings, and Help
- The real-time stock ticker

- The stock analysis tool and calculator
- The stock purchasing tool and calculator

Create a new spreadsheet that details the relationship of these features. Most installable parts will repeat features—for instance, all of the parts will include the shared objects feature. However, only add features from the first spreadsheet to any installable part.

As you do this, think about how the features are packaged from a customer perspective. If a user wants to install just the real-time ticker portion, which objects would need to be installed? Most of the installable parts, though probably not all of them, should have a "conditional" aspect to their installation—they're only installed if the end user so chooses.

Look at the Stock Suite example—again, it's a simple one—displayed in Figure 10-2.



Always Installed	Stock Ticker Tool	Stock Analysis Tool	Stock Purchase Tool
Shared	Ticker	Analysis	Purchase
Help		Calculator	Calculator

Figure 10- 2. Listing installable components

Step 3: Architect the User Interface

Once you've determined the different parts in which you'll divvy up your product, now you must design how you'll graphically present these choices to the end users. When designing this interface, keep in mind that you want to make the installation experience as pleasant and easy as possible. Remember that a computer is a personal item, such as a house or a car, and you should balance the convenience of automation with the customers' desire to choose what's installed on their respective machines.

Ask yourself questions about the choices you might like to have when installing another company's software on your computer. In the case of the Stock Suite product, some of these questions might be as follows:

- In which directory would you choose to install the product?
- Which portions of the product would you like to install? The entire product made up of the ticker, analyzer, and purchaser? A combination of two of the components? One of the components?
- Should a program group be created? If so, should it be shared between all users on the machine or available to just the person installing the product?
- Should a desktop shortcut icon be created? Should a quick-launch shortcut icon be created?
- If this product is a client for an enterprise server product, what is the name or Internet Protocol (IP) address of the server?

Although this is by no means a comprehensive list of the questions you might come up with, it's a good starting point. Keep trying for the "Goldilocks" of installation User Interfaces (UIs): Ask only the questions necessary to ensure both flexibility and a correctly installed product. Too many questions will annoy the end user and create more work for yourself. Too few questions, and the complaints will roll in. In other words, try to be "just right."

TIP *Consider a UI approach that presents both an express installation of your product's most common configuration and a customizable installation that allows power users to make detailed choices.*

Exploring the Windows Tools

Windows has a variety of tools designed for installation—some of which are now built into the operating system. The following sections explore the three most popular methods:

- The Microsoft Windows Installer (MSI) and Visual Studio's included MSI plug-in
- InstallShield and its line of tools, a third-party alternative to MSI
- Wise Systems and its line of tools, a third-party alternative to MSI

Using the Windows Installer

When Microsoft introduced Windows 2000, it included a new installation tool, the Windows Installer. In theory, this tool fixed many of the perceived problems presented by third-party installation engines and is still supported in all post–Windows 95 Microsoft operating systems.

Instead of using procedural scripts like previous installation tools, MSI is database driven. Installation writers create an MSI database object that details to the operating system how to install files, Registry settings, and other settings. Using this methodology, MSI mitigated the largest problem known to both developers and end users—system "DLL hell"—and ensured the easy, complete removal of products.

NOTE *To use MSI installations, the installer must be installed on your user's system. By default, it's included in all versions of Windows created after 1999.*

Surprisingly enough, the development community didn't open its arms to MSI when it was released. Much of this was because Microsoft insisted that developers use the new installation tool before they could receive the much-coveted "Designed to Work with Windows 2000" logo that gets pasted on software boxes. Developers found the new MSI Application Programming Interface (API) difficult to learn and complained about having to completely rewrite older legacy installations. Microsoft listened and summarily dropped the requirement.

There's truth that using MSI in its native state is an arduous task. Microsoft didn't provide a feature-rich graphical interface for the API—perhaps in fear that installation tool manufacturers would cry foul in the way Netscape did when Internet navigation was built into Windows. But with the release of Visual Studio .NET, Microsoft now provides a simple graphical version of MSI and leaves the "hard stuff" for the third-party installation tool developers.

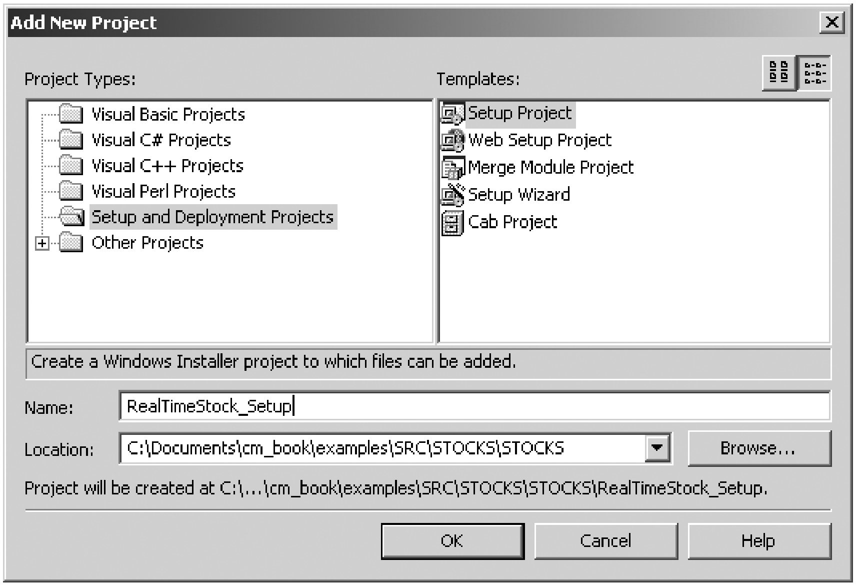
The MSI tool provided by Visual Studio may not be feature rich enough for you. One of its largest limitations is that it can't automatically install the .NET Framework—a must for .NET applications. On the other hand, the default Visual Studio MSI integration is quick and easy for one-trick-pony software applications.

Creating an MSI Installation Using Visual Studio .NET

Let's build an MSI installation for the Stock Suite of products using Visual Studio .NET. If your team already has a solution containing your product's applications, you can easily create an installation project for Stock Suite in the same solution. On the other hand, you may want to create the installation in a different location, but as you'll see later in this chapter, it's much more convenient to have the installation in the same solution. Any number of MSI installation projects can co-exist in the same solution, and, if you chose to do so, you could create a separate installation for each product in the Stock Suite.

TIP *If the developers are wary about having the installation in the same solution, create a copy of the solution for yourself and then rename it. Because solutions simply point to included projects, any changes your developers might make will still be accessible to you when you use your solution.*

The first step to creating your installation project is to right-click the solution in the Solution Explorer window and choose the Add ➤ New Project menu item. The Add New Project dialog box appears. By choosing the Setup and Deployment Projects folder in the Project Types window, you'll see the five installation/setup project templates available for use, as displayed in Figure 10-3. The five types are described with detail in the "Determining Which Setup Template to Use" sidebar. Regardless of whether you install other third-party installation tools on the same machine, only MSI projects appear in the Setup and Deployment Projects folder.


Add New Project

Project Types:
- Visual Basic Projects
- Visual C# Projects
- Visual C++ Projects
- Visual Perl Projects
- Setup and Deployment Projects
- Other Projects

Templates:
- Setup Project
- Web Setup Project
- Merge Module Project
- Setup Wizard
- Cab Project

Create a Windows Installer project to which files can be added.

Name: RealTimeStock_Setup

Location: C:\Documents\cm_book\examples\SRC\STOCKS\STOCKS

Project will be created at C:\...\cm_book\examples\SRC\STOCKS\STOCKS\RealTimeStock_Setup.

Figure 10-3. Adding a new MSI project

Determining Which Setup Template to Use

Five setup and deployment project templates are available through Visual Studio's default MSI offering.

Setup Project template: The Setup Project template is the standard installation project used for Windows applications. It has all of the standard installation features:

- **The ability to install and uninstall the application**: The application's icon is added to the Control Panel's Add/Remove Programs feature, and the user can, in most cases, uninstall it without any extra logic provided by the installation creator. It should be noted, however, that any custom actions provided by Dynamic Link Libraries (DLLs) or other scripts must have an action provided for both rollbacks and uninstalls to complete.
- **The ability to repair an existing installed application**: Should parts of the application become corrupted or deleted, the application can be reinstalled without the loss of user customization.
- **The ability to roll back all changes should any part of the installation fail or if the user chooses to cancel before it has completed**: Again, it should be noted, however, that any custom actions provided by DLLs or other scripts must have an action provided for both rollbacks and uninstalls to complete.
- **The ability to add or change the Windows Registry and product file associations**: If an application has a specific file type on which it can perform, these file associations can easily be made.
- **The ability to add shared assemblies to the Global Assembly Cache (GAC)**: As referenced in the previous chapters, any .NET assemblies that are shared must be added to the GAC.
- **The ability to confirm the existence of prerequisite software and to cancel gracefully should it not be found**: This is especially important with .NET installations because earlier Windows systems may not have the Framework installed.

Web Setup Project template: A Web Setup Project template has similar functionality to the Setup Project template but differs in that it installs itself into an Internet Information Services (IIS) virtual Web directory instead of the Program Files directory.

Merge Module Project template: You use merge modules for deploying separate "installation parts" instead of complete applications. An example of the proper use of a merge module is the installation of a shared utility. Merge modules have their own extension—.MSM—that can be added to a standard setup project.

(continued)

CAB Project template: Microsoft cabinet projects—often shortened to *CAB projects*—allow for the deployment of redistributable ActiveX controls. Once controls are added to these CAB files, they can then be signed as "authenticated" by a representative of the development house and placed on a Web site. When users access the page, browsers display a dialog box informing them of the ActiveX installation.

Setup Wizard: The Setup Wizard isn't actually a project template so much as a tool—it simply provides a friendlier face to the four installation templates mentioned previously in this sidebar.

NOTE *Discussing the entire feature sets of MSI, InstallShield, and Wise would be a book unto itself. This chapter provides only examples of basic installations.*

The Stock Suite is a simple application, so choose the Setup Project option. Solution Explorer uses the name you designate now for saving the installation project files; however, regardless of what name you choose, the default name for the installation output will be Setup. After typing a descriptive name and clicking OK, the system displays a window that looks similar to Figure 10-4. Keep in mind that Visual Studio is extremely customizable—the appearance of your environment may vary. Solution Explorer now highlights your new setup project, and the work area of Visual Studio lists three folders in an Explorer-like view.

The Properties window in the Visual Studio environment, as displayed in Figure 10-5, allows you to "personalize" the setup program with information about your company and the current project. If the Properties window isn't currently displayed, choose View ➤ Properties Window to bring it to the forefront.

NOTE *Unless you're working with a huge monitor, you'll quickly find that your screen doesn't have enough real estate for all the windows you want to open! In these cases, simply open and close the windows you need by using the View menu.*

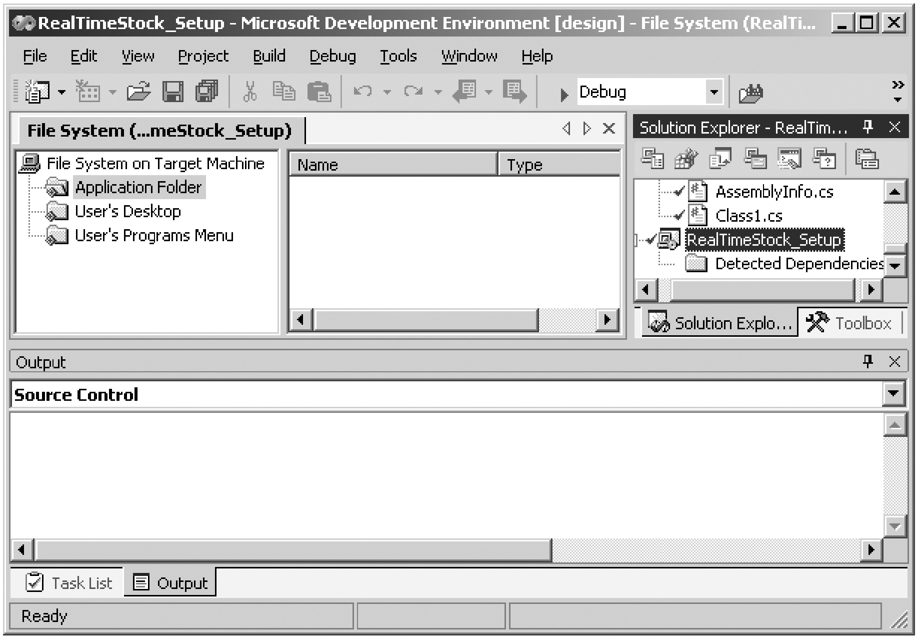
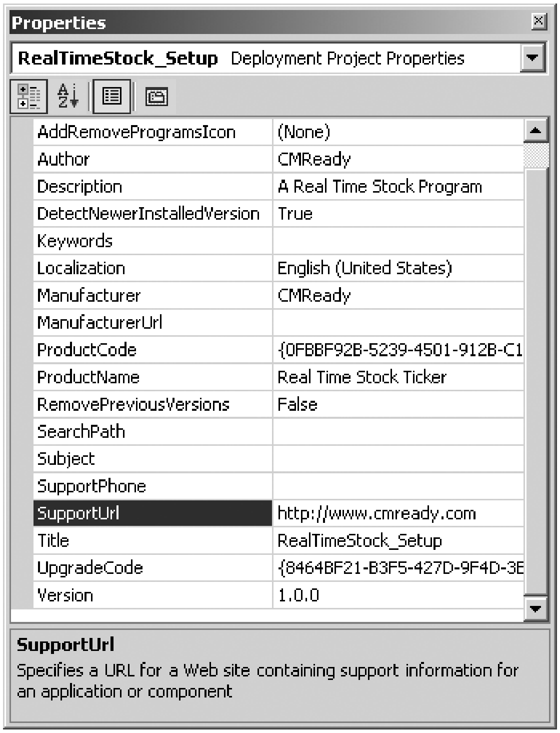

RealTimeStock_Setup - Microsoft Development Environment [design] - File System (RealTi...
File Edit View Project Build Debug Tools Window Help
Debug
File System (...meStock_Setup)
File System on Target Machine
Application Folder
User's Desktop
User's Programs Menu
Name
Type
Solution Explorer - RealTim...
AssemblyInfo.cs
Class1.cs
RealTimeStock_Setup
Detected Dependencies
Solution Explo...
Toolbox
Output
Source Control
Task List
Output
Ready

Figure 10-4. The newly created RealTimeStock setup project in Visual Studio



Properties

RealTimeStock_Setup Deployment Project Properties

AddRemoveProgramsIcon	(None)
Author	CMReady
Description	A Real Time Stock Program
DetectNewerInstalledVersion	True
Keywords	
Localization	English (United States)
Manufacturer	CMReady
ManufacturerUrl	
ProductCode	{0FBBF92B-5239-4501-912B-C1
ProductName	Real Time Stock Ticker
RemovePreviousVersions	False
SearchPath	
Subject	
SupportPhone	
SupportUrl	http://www.cmready.com
Title	RealTimeStock_Setup
UpgradeCode	{8464BF21-B3F5-427D-9F4D-3E
Version	1.0.0

SupportUrl
Specifies a URL for a Web site containing support information for an application or component

Figure 10-5. The setup's Properties window

The following are a couple of quick notes regarding the important properties for installation projects:

The Manufacturer property: This sets the "company" name used for the creation of the application's installation directory and Registry keys.

The ProductCode and UpgradeCode properties: MSI uses these to determine whether previous versions of the application have been installed on the end user's machine. Visual Studio automatically provides these values, and only the product code should be changed when a new version of the program is created. Don't try to come up with your own value for this property—instead, double-click the property and click the New Code button to create new IDs as necessary. Also, don't change the UpgradeCode property once an installation has been created and deployed—even to support different language versions of the product.

The ProductName property: This reflects how the title pane of the installation will describe your application—feel free to use spaces and capitalization as desired.

The Version property: This reflects the version of the resulting MSI or CAB file—it's in no way connected with the version of the separate executables included in the installation. For consistency, you may want to set this version number to match your product's release number.

Adding Files and Other Objects to the Installation

Once you've set the properties for your setup project, it's time to add files and Registry items. You may want to architect the user interface first, but that isn't advisable. Even with your detailed spreadsheet, you might have changes as you build your project that affect the user interface.

To add files, navigate to the File System window for your setup project in Visual Studio. If it's not already displayed, open it by right-clicking the setup project in the Solution Explorer and choosing View ➤ File System, as displayed in Figure 10-6. You can open all setup project windows in this manner.

The left side of the Explorer window lists three folders: the Application folder, the User's Desktop folder, and the User's Program Menu folder. These three folders simulate where you might place application files and shortcuts on your user's hard drive.

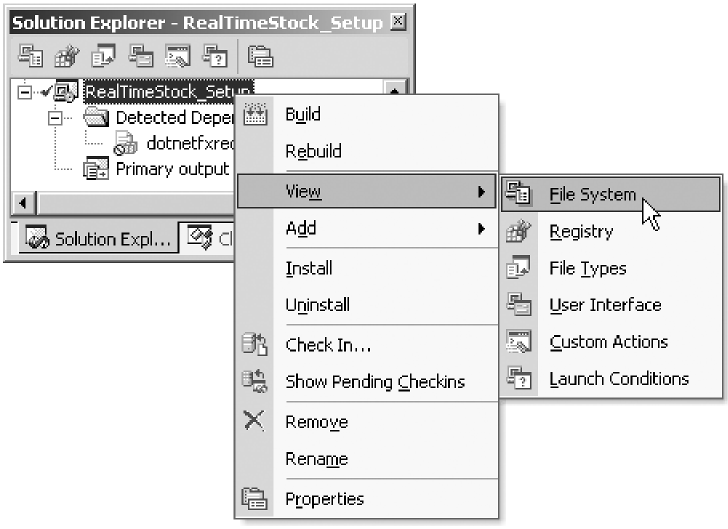


Figure 10-6. Viewing other setup options

Use the Application folder for the files that belong in the default directory of your application. The Setup Project Wizard creates this directory string automatically if you've set the Manufacturer and ProductName properties as previously instructed. Change the default installation location of the application by right-clicking the Application folder and choosing Properties Window.

NOTE *Some properties can use variable names for common directories or other project properties. For instance, the value for the DefaultLocation property of the Application folder object is [ProgramFilesFolder][Manufacturer]\[ProductName]. [ProgramFilesFolder] is a variable that stands for the user's default Program Files directory. You set the [Manufacturer] and [ProductName] variables manually in the project's properties. Other variables that can be used for the DefaultLocation property include [CommonFilesFolder], [FontsFolder], [GAC], [SystemFolder], [WindowsFolder], and [TargetDir]. Only [TargetDir] isn't self-explanatory—this variable is used with merge modules to indicate the directory of the parent MSI file that installs it.*

TIP *If installing to a directory other than Program Files, such as a user's Favorites or Windows directory, don't try to guess the path. Instead, right-click in the left window to add the appropriate "special folder." There are 20 or so special folders into which you can install an application.*

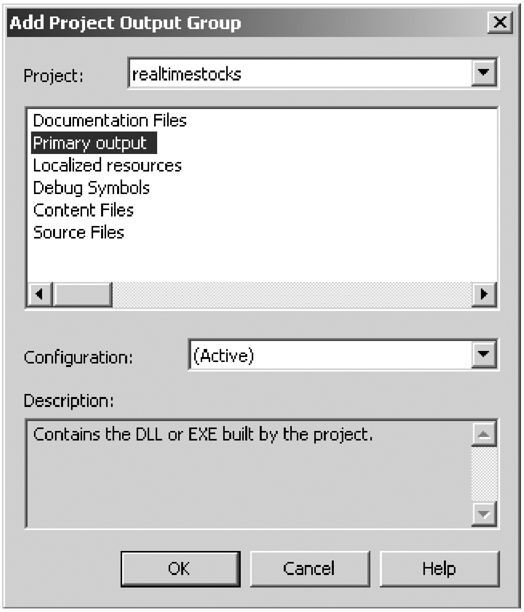
Once you've set where the application will install, it's time to add files to the project. First, right-click in the file pane of the File System window and choose Add. Then choose the type of object to be installed:

File: Choosing this option opens the standard Windows Add File dialog box. For instance, you might choose this option to provide a README file by navigating to the file on the development machine. Though listed first, this isn't the choice you make for the application files listed in your solution—see the upcoming Project Output option.

Folder: Create a directory structure inside your default folder by using this command. For example, use this command if you want to keep your help and program files in separate child directories beneath the main application folder. Once created, navigate to the new folders and then use the Add menu to populate them.

Assembly: Use this command to install Windows system assemblies. This item displays the shared assemblies installed on the current machine. Use this only for third-party components—and not for shared assemblies you plan to put in the GAC. Add your own assemblies by using the Add ➤ Project Output or Add ➤ File command.

Project Output: This is the best way to add the applications you build. Choose a product in the solution using this option, and the installation automatically includes its output executables in the created MSI file. Visual Studio automatically checks the project's output files for dependencies and includes them in the installation. On the off-chance that a dependency can't be included for some reason—for instance, because of a licensing conflict—Visual Studio's Output window tells you as you build the installation. After choosing this command, the Add Project Output Group dialog box appears, as displayed in Figure 10-7. By using the Project drop-down list, choose the project whose output you want to include in the installation. Once the project is displayed, use the mouse to click any desired combination of project or source output. Choose multiple items by holding down the Control key during a mouse click. Lastly, choose from which configuration the items should come. Pick Active, and Visual Studio dynamically chooses which output to include based upon the current setup configuration. You can use the Project Output command only for projects included in the solution. Otherwise, use the Add ➤ File command.



Project: realtimestocks

Documentation Files
Primary output
Localized resources
Debug Symbols
Content Files
Source Files

Configuration: (Active)

Description:
Contains the DLL or EXE built by the project.

Figure 10-7. Adding executables to be installed by MSI

Once you've added all the application's files to the installation, it's time to include Registry items. Choose View ➤ Registry in the Solution Explorer to open a window that allows you to add Registry items in much the same way you did files. Simply navigate to the proper key in the left pane and add or edit Registry items as necessary. The [Manufacturer] variable indicates a default Registry key based upon the application's name and manufacturer—in most cases, it's what you'll use.

Add file type associations to the installation by choosing View ➤ File Types in the Solution Explorer. Once the window opens, right-click in the left pane and choose Add Type to add a new object entitled New Document Type. Use the Properties window to change the (Name) and Extensions property of the object to reflect the proper file type.

NOTE *Be careful when changing the default actions for common extensions such as .DOC, .HTML, and .JPG. Users may not appreciate your installation changing these default associations.*

Fill in the Extensions property with at least one file type; otherwise, you'll get a build error when trying to compile. Type extensions without the preceding period and add more of them by using a semicolon as a separator. For instance, setting the Extensions property to *YTG;EER* means that BOB.YTG and BOB.EER are both associated with your application once it's installed.

After adding the extensions, dictate the type of action you want to associate with your application. The Open action is provided by default and indicates that your application should start when users double-click the file type. You may want to add an action other than the default Open command. For instance, if your company provides a Hypertext Markup Language (HTML) editor, you might want the Open action to remain with the user's chosen browser. Instead, you might associate the Edit action with your application instead. Then when the user right-clicks an HTML file, he or she can choose either Open or Edit depending on the desired action. If you choose to use another action, be sure to remove Open.

NOTE *Every file type grouped together in the Extensions property setting must have the same action. To have two different Open actions for two different extensions, you must create two different file type objects.*

Changing the User Interface

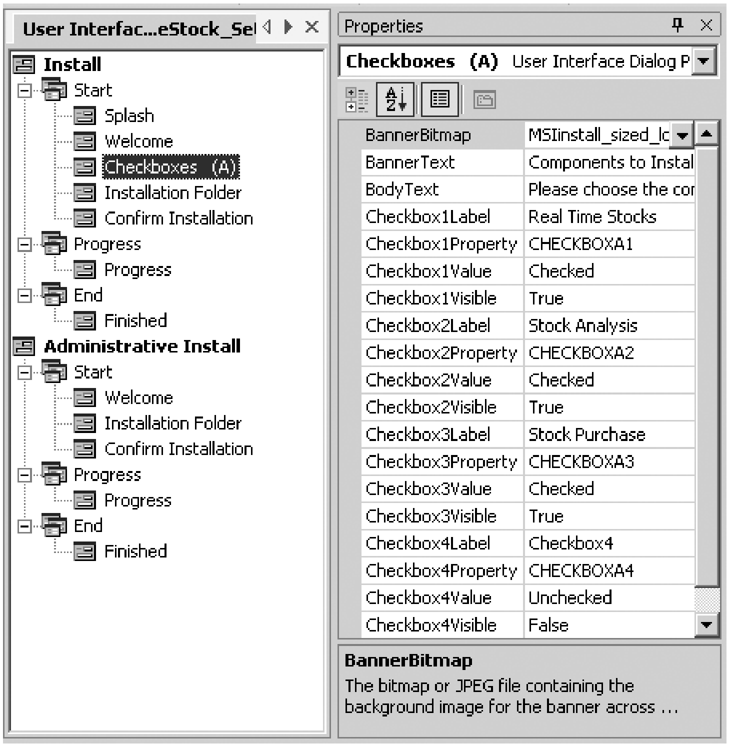
Now that you've got a good idea of how your setup will change the end user's system, it's time to create the user interface. Surprisingly, the default Visual Studio MSI integration provides a fairly robust tool for creating this interface.

The default interface that's created for new installations includes the following screens:

- The Welcome screen introduces the user to the installation and supplies the usual copyright notices.
- The Installation Folder screen asks the user to confirm the directory into which the application will be installed.
- The Confirm Install screen allows the user to cancel the installation before any changes have been made to the system.
- The Progress screen displays the progress of file copies and system changes.
- The Finished screen indicates to the user that the setup has completed the installation successfully.

NOTE *If for some reason the user cancels the installation, MSI displays what appears to be a different Finished dialog box and rolls back all changes made to the system. The new Finished screen is simply a different view of the regular one. Several screens in the user interface have double duty when installations are canceled, repaired, or removed. Be sure to test the installation thoroughly to ensure that the customizations you've added to dialog boxes are appropriate for all uses.*

To change the user interface, right-click the setup project and choose View ➤ User Interface. Your main work area then contains a user interface tree similar to the one displayed in Figure 10-8. The Properties window also reflects the currently highlighted dialog box.



User Interfac...eStock_Se

- Install
  - Start
    - Splash
    - Welcome
    - Checkboxes (A)
    - Installation Folder
    - Confirm Installation
  - Progress
    - Progress
  - End
    - Finished
- Administrative Install
  - Start
    - Welcome
    - Installation Folder
    - Confirm Installation
  - Progress
    - Progress
  - End
    - Finished

Properties

Checkboxes (A) User Interface Dialog P

BannerBitmap	MSIinstall_sized_lc
BannerText	Components to Instal
BodyText	Please choose the co
Checkbox1Label	Real Time Stocks
Checkbox1Property	CHECKBOXA1
Checkbox1Value	Checked
Checkbox1Visible	True
Checkbox2Label	Stock Analysis
Checkbox2Property	CHECKBOXA2
Checkbox2Value	Checked
Checkbox2Visible	True
Checkbox3Label	Stock Purchase
Checkbox3Property	CHECKBOXA3
Checkbox3Value	Checked
Checkbox3Visible	True
Checkbox4Label	Checkbox4
Checkbox4Property	CHECKBOXA4
Checkbox4Value	Unchecked
Checkbox4Visible	False

BannerBitmap
The bitmap or JPEG file containing the background image for the banner across ...

Figure 10-8. Listing the properties of your setup's user interface

NOTE *There's no way from within Visual Studio itself to preview customized user interface dialog boxes before adding them to the installation. To preview all the available MSI dialog boxes, hit F1 in the User Interface window and click the Deployment Dialog Boxes hyperlink.*

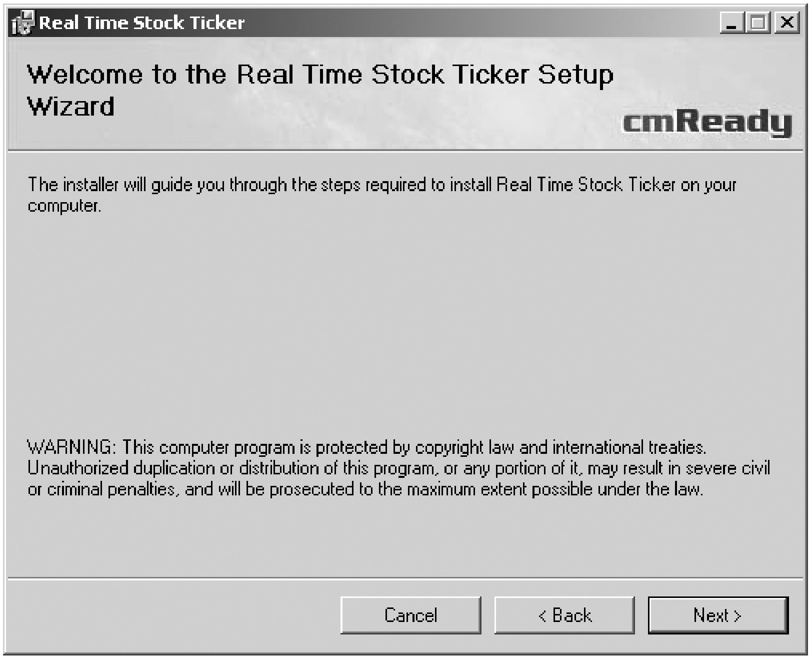
The User Interface window hierarchy is broken down into two sections—Install and Administrative Install. To keep things basic, this chapter only discusses the Install section—the part of the tree applicable to end users. The Administrative Install section is used when network administrators might distribute the product in bulk from network locations.

Each separate install type has three action nodes that reflect positions within the installation—Start, Progress, and End. You can drop about 20 nondefault dialog boxes into the installation in almost any position or order. It's important, however, to place dialog boxes in positions logical to their functions. For instance, the Start node is for gathering information from the user before the installation proceeds. With that understanding, it wouldn't make sense to include the Finished dialog box in that node. But no worries—99 percent of the added dialog boxes go in the Start node because they ask the user questions. Progress and End are likely to only have a dialog box that reflects the node itself.

To demonstrate the use of dialog boxes, you'll add and customize four of them in the installation for this example.

NOTE *Throughout this chapter, I use the name of my own company, cmReady, whenever a company name is required.*

First, add a corporate logo to each dialog box in the installation. Using a graphics program such as Adobe Photoshop, create a picture that is 500 pixels long by 70 pixels high. The file type can be a bitmap or compressed JPEG file. When designing the logo, remember that MSI displays text over the left three-quarters of the picture. For example, MSI displays the *Welcome to the Real Time Stock Ticker Setup Wizard* text displayed in Figure 10-9. There's no need (nor is it desirable) to write the text on the logo yourself.


Real Time Stock Ticker

Welcome to the Real Time Stock Ticker Setup Wizard

cmReady

The installer will guide you through the steps required to install Real Time Stock Ticker on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Figure 10-9. How a logo might appear in an MSI installation

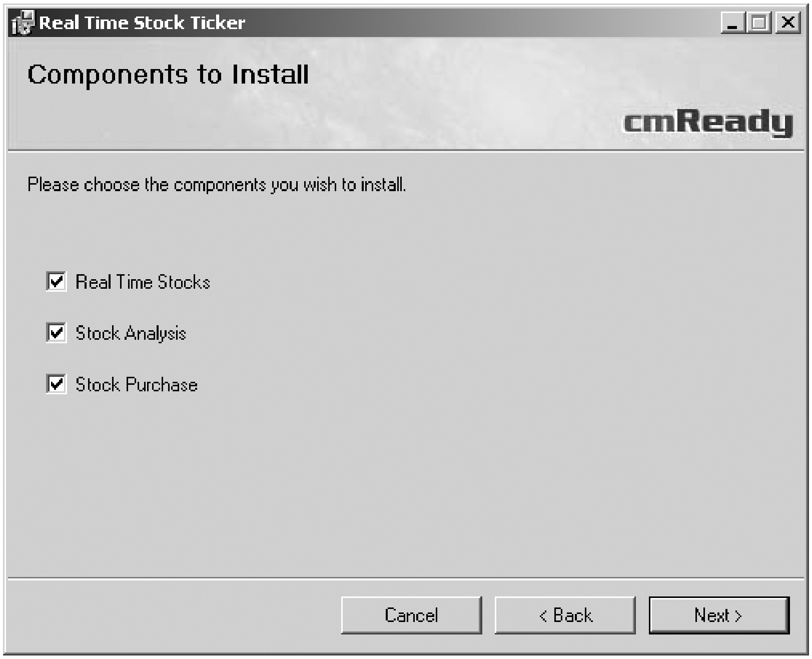
Once created, add the bitmap to one of the install folders by right-clicking it and choosing the Add ➤ File menu option. Afterward, highlight each dialog box that should display the banner and set the BannerBitmap property as necessary. Unfortunately, you must add the banner to every dialog box. This does allow you, however, to use different logos on all of the dialog boxes if desired. You might want to use banners in such a manner to denote the user's current position in the installation—for instance, you may label the first dialog box as *Step 1*.

TIP *Using the Add ➤ File command to add banners causes the setup to copy them to the end user's file system. To get around this issue, set the logo's Condition property to zero (0). It will still display during installation but will not be copied to the end user's machine.*

After adding the banner to the Welcome dialog box, note it also has two customizable text properties. Customize the text as desired. Although you can customize all MSI dialog boxes in such a manner, it's likely that you'll accept the default settings. Multiple language installations require you to use separate setup projects so that you can customize these text settings for each language.

To add other dialog boxes, right-click the appropriate node and choose Add Dialog. MSI displays a list of available dialog boxes—choose to add one, some, or all of them. The names pretty much designate their functionality—but refer to the Help if you have further questions. Now, go ahead and add the Splash, License Agreement, and Customer Information dialog boxes to the installation and customize them appropriately. Splash logos must be bitmaps or JPEG pictures that are sized exactly 480 pixels long and 320 pixels high.

The last dialog box you'll add to the installation allows users to conditionally install your three main features. To do this, use one of the several "choice" dialog boxes. These allow you to display checkbox or radio button group choices to the end users. For instance, you might present a checkbox dialog box as displayed in Figure 10-10 to allow users to install any or all of the Stock Suite.


Real Time Stock Ticker

Components to Install

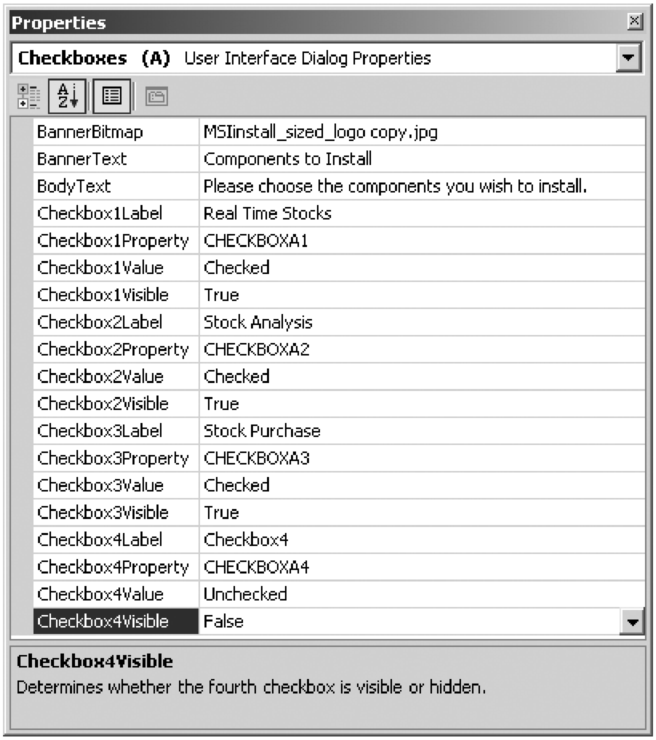
Please choose the components you wish to install.

- [✓] Real Time Stocks
- [✓] Stock Analysis
- [✓] Stock Purchase

Figure 10-10. Allowing the end user to install all or part of the Stock Suite product

Because the “choice” dialog boxes are generic by default, create text for both the BannerText and BodyText properties that indicates to the user just what they’re choosing. In this case, for instance, you might set the BannerText property to be *Components to Install* and the BodyText property to *Please choose the components you wish to install.*

To use choice dialog boxes, you set variables that indicate to the installer which choice the user made. As displayed in Figure 10-11, each checkbox on the dialog box has four properties attached to it.


Properties

Checkboxes (A) User Interface Dialog Properties

BannerBitmap	MSIinstall_sized_logo copy.jpg
BannerText	Components to Install
BodyText	Please choose the components you wish to install.
Checkbox1Label	Real Time Stocks
Checkbox1Property	CHECKBOXA1
Checkbox1Value	Checked
Checkbox1Visible	True
Checkbox2Label	Stock Analysis
Checkbox2Property	CHECKBOXA2
Checkbox2Value	Checked
Checkbox2Visible	True
Checkbox3Label	Stock Purchase
Checkbox3Property	CHECKBOXA3
Checkbox3Value	Checked
Checkbox3Visible	True
Checkbox4Label	Checkbox4
Checkbox4Property	CHECKBOXA4
Checkbox4Value	Unchecked
Checkbox4Visible	False

Checkbox4Visible
Determines whether the fourth checkbox is visible or hidden.

Figure 10-11. The Properties window reflecting the checkbox properties

The properties are as follows:

The Label property: This contains the phrase that indicates choices to the end user. For instance, the screen previously displayed in Figure 10-10 allows the end user to choose to install any or all of the three main Stock Suite features.

The Property property: This holds the variable that links to installable files and objects. This variable is a boolean value—if the user checks it, the variable value during file installation is true. If the user removes the check, the value is false. I'll discuss this property with more detail in a moment.

The Value property: This allows you to designate whether the checkbox is checked or unchecked by default when the dialog box first displays.

The Visible property: This indicates to MSI that the checkbox should be visible to the end user when the dialog box is displayed. You might ask: Shouldn't they always be visible? Not necessarily! The Checkbox dialog box always has four checkboxes on it by default—if you choose to use only three of the checkboxes, as previously displayed in Figure 10-10, the fourth checkbox's Visible property should be set to false as displayed in Figure 10-11. Leaving an unused checkbox's Visible property to true displays it with whatever default values it might have. No worries about invisible checkboxes—their values are unimportant unless you choose to link them to installable files or other objects.

NOTE *You may notice that there are several checkbox dialog boxes available when adding to your installation's user interface. This is because MSI is limited in that each dialog box can only be used once. Microsoft provides the ability to retrieve information from multiple dialog boxes by including several of them. It doesn't matter which dialog box you choose to use first though the anal-retentive folks will want to use them in alphabetical order. The radio button and textbox dialog boxes both work in the same manner.*

After setting the values for each of the checkboxes, navigate to the File System window by right-clicking the setup project and choosing View ➤ File System. In the application folder, choose a file that's conditionally installed and set its Condition property to the same variable as the one set for the applicable checkbox. For instance, the Stock Analysis checkbox's property is CHECKBOXA2. Therefore, set the Condition property for every file object that should be installed when this checkbox is selected to CHECKBOXA2.

This is where the product spreadsheets you created at the beginning of this chapter come in handy. Use the mappings you created to determine what makes up each of the three features. For instance, the Stock Analysis feature consists of the Analysis objects and the Calculator objects. You must set the Condition property for every object listed under either the Analysis or Calculator headings to CHECKBOXA2.

Follow this same methodology for Registry objects. Simply navigate to the Registry window and set conditions for any objects as applicable.

NOTE *Only files and Registry objects can be installed conditionally.*

What about objects that must be installed in more than one conditional feature? In this case, use bitwise operators in the Condition property. For instance, the Calculator feature should be installed with both the Stock Analysis and Stock Purchase tools. To ensure that this occurs, use the OR keyword in the Condition property for all Calculator objects to indicate that they should be installed if either the CHECKBOXA2 or CHECKBOXA3 variable is set to true when a file transfer occurs. In this case, the condition reads as follows:

```
CHECKBOXA3 OR CHECKBOXA2
```

You can also use the AND keyword in your conditions. If you used the AND keyword instead of OR in the previous statement, it would mean that the Calculator object should be installed only if both the Stock Purchase *and* the Stock Analysis checkmarks are ticked. Now that would be an unusual scenario!

Conditional Love

Using conditions in MSI occurs in more than just user interface dialog boxes. For instance, you might want to install files only on certain versions of Windows or based upon the end user's language settings. The MSI installer gives you a nifty way of making some of these checks.

In the choice dialog box example used earlier in this chapter, ticking a checkbox installs the items linked to the same variable in its Property property. With other available conditions, however, you have to give MSI a full phrase to evaluate. Table 10-1 displays several comparison operators that you can use to create those phrases.

(continued)

Table 10-1. MSI Comparison Operators

Operator	Description
>	The value on the left side *is more than* the value on the right side of the equation in order for the phrase to evaluate to true.
<	The value on the left side *is less than* the value on the right side of the equation in order for the phrase to evaluate to true.
>=	The value on the left side *is more than or equal to* the value on the right side of the equation in order for the phrase to evaluate to true.
<=	The value on the left side *is less than or equal to* the value on the right side of the equation in order for the phrase to evaluate to true.
=	The value on the left side *is equal to* the value on the right side of the equation in order for the phrase to evaluate to true.
<>	The value on the left side *is not equal to* the value on the right side of the equation in order for the phrase to evaluate to true.
AND	Logical operator indicating that *values on both sides must be true* in order for the phrase to evaluate to true.
OR	Logical operator indicating that *one of the values on either side must be true* in order for the phrase to evaluate to true.

Table 10-2 lists several keywords that can be combined with the previously mentioned operators to create phrases.

Table 10-2. Common MSI Keywords

Keyword	Description
Version9X	Windows operating system version number for 95, 98, and ME
VersionNT	Windows operating system version number for NT, 2000, and XP
ServicePackLevel	The level of operating system service pack installed
WindowsBuild	The build number of the Windows operating system
SystemLanguageID	The ID for the language of the installed operating system
PhysicalMemory	The amount of Random Access Memory (RAM) in megabytes on the machine
Intel	The version of the processor on the machine

This is by no means an exhaustive list—refer to Visual Studio's Help or `http://msdn.microsoft.com` for a slew of conditional keywords.

Here's an example of how you might use these keywords and operators together: Perhaps you have three DLLs that perform specific functions depending on the end user's operating system. The first DLL, called THEWIN9X.DLL, should be installed only on Windows 95, 98, and ME. THEWINNT.DLL is for Windows NT version 4.0, and THEWIN2000.DLL is for Windows 2000 and XP machines.

After taking a look at Table 10-2, you can see that it's possible to use the Version9X and VersionNT property values to create conditional phrases. The Microsoft MSDN site, `http://msdn.microsoft.com`, details the values you use for these properties—find them by searching the MSDN Knowledge Base for *"Operating System Property Values."* For example, the MSDN site tells you that the Version9X property has a value of 400 to indicate Windows 95, 410 for Windows 98, and 490 for Windows ME.

NOTE *When searching MSDN, be sure to put quotes around the* "Operating System Property Values" *phrase or the site searches for all combination of the words—it can make it much harder to find the proper page.*

After evaluating the chart, you might set your conditions for each of the previously mentioned DLLs to look something like Table 10-3.

Table 10-3. Conditions for Operating System

Object	**Condition**
THEWIN9X.DLL	Version9X = 400 OR Version9X = 410 OR Version9X = 490
THEWINNT.DLL	VersionNT>=400 AND VersionNT<500
THEWIN2000.DLL	VersionNT>=500

Setting Launch Conditions

You can kill an installation before it ever begins by using launch conditions. This can be important when your product depends on a certain operating system or installed component. By setting a launch condition, MSI gracefully exits from the installation with an informative message box before making any changes to the end user's system.

To set a launch condition, right-click the setup project and choose View ➤ Launch Conditions. An Explorer window tree appears with two folders—Launch Conditions and Search Target Machine.

NOTE *By default, all .NET setup programs include a launch condition using the MsiNetAssemblySupport variable. By including this launch condition, installations don't continue should the .NET Framework not be installed. See the "Redistributing Microsoft .NET Framework Files" section for more information.*

To use the Launch Conditions folder, simply right-click it and choose Add Launch Condition. The Properties window will reflect three new values: (Name), Condition, and Message. Use the (Name) property to identify the new condition. The (Name) property is for description purposes and has nothing to do with the condition itself—feel free to use spaces or any naming pattern desired when naming launch conditions. Use the Condition property to add a conditional phrase—see the previous "Conditional Love" sidebar for more information about creating phrases. Finally, set the Message property to the text that should be communicated to the end user should the condition not be met.

As an example, set your Stock Suite installation to fail if the end user's operating system isn't running Windows 2000 or XP. You can do so by including the launch condition listed in Table 10-4.

Table 10-4. Example Launch Condition

Property	**Value**
Name	Not_Windows_NT
Condition	VersionNT>=500
Message	You must be running Windows 2000 or Windows XP in order to continue this installation.

Of course, it might be that properties aren't enough to ensure a smooth installation. If the Stock Suite product depends on a certain video driver to be installed, for example, there isn't likely to be a variable that you can use to create a conditional phrase. Instead, you can search for the driver on the end user's machine by right-clicking Search Target Machine and choosing Add File Search. If necessary, you can also search the Registry for particular entries before continuing the installation.

There are quite a few more properties for new objects in the Search Target Machine folder. Use (Name), FileName, Folder, and Property for basic searches. You can fine-tine searches by setting other properties for file versions, sizes, or versions.

Again, the (Name) property is important only as a descriptive title. Set the FileName and Folder properties to indicate which file should exist on the end user's machine in order for the installation to continue. Use [brackets] variables to specify special folders, such as [ProgramFilesFolder] or [SystemFolder].

Finally (and here's the hard part), you have to set an associated launch condition that cancels the installation should the search not be successful. Do this by creating a new launch condition and making up a new association variable—much as you did with the checkbox dialog box earlier in the chapter—and placing it in the launch condition's Condition[1] property. You can call the variable whatever you like—just make sure that no other variables—including built-in Microsoft variables—use the same name.

TIP *Use your company or product name as part of any variable name to ensure its uniqueness. For instance, in the previously described scenario, you might use a variable called STOCKSUITE_VIDEODRIVERCHECK.*

Use the same variable name in the search's Property property.[2] If MSI finds the search file during installation, it sets the variable listed in Property to be true. The launch condition checks the same variable and, if it finds that it's been set to true, allows the installation to continue. Table 10-5 and Table 10-6 demonstrate how you might set up this type or associative search/launch condition.

Table 10-5. Part 1: Using the Search Target Machine Folder

Property	Value
Name	Video_Driver_Check
FileName	VIDEO.DRV
Folder	[SystemFolder]
MinVersion	3.42
Property	STOCKSUITE_VIDEODRIVERCHECK

Table 10-6. Part 2: Creating a Launch Condition

Property	Value
Name	Video_Driver_Check_LaunchCond
Condition	STOCKSUITE_VIDEODRIVERCHECK
Message	You must have the cmReady video driver installed in order for the Stock Suite to work properly. Please install the driver and then run this setup again.

1. Could they have named these to be any more confusing?
2. Ibid.

Using Custom Actions

Finally, I'll talk about the ability to add custom actions to your installation. MSI is pretty much limited to adding files or Registry items. By default, you can't use it to run a script on a database or update an installation log file. You can fulfill these tasks, however, by using custom actions to run code from a DLL or other executable.

NOTE *This section details how to create and call code from an executable outside of the MSI database and, as such, is designed for more advanced users. The section presents example code using Microsoft's proprietary C# language.*

Right-click the setup project and choose View ➤ Custom Actions to find the four available custom actions: Install, Commit, Rollback, and Uninstall. Each of these folders represent separate "moments in time." When the setup program reaches an appropriate point in the installation, MSI runs the actions listed in the applicable custom action folder.

MSI calls the Install node actions just before it copies files and other objects to the end user's machine. After all objects have been successfully installed, it calls the Commit node actions. If the installation is cancelled for any reason before it has finished, MSI runs the actions listed in the Rollback folder. Finally, the Uninstall node actions are called—surprise!—during the uninstallation of the program.

Before you use the custom actions folders, however, you have to create an application or DLL that can be called by MSI. To learn about custom actions, in this section you'll create a console application that creates a basic installation log file on the end user's system.

Start by adding a new project to your Stock Suite solution. Use the Console Application template under the Visual C# Projects folder and call the newly created project SETUP_SS.[3]

Once you've created the project, change the code in the CLASS1.CS file so that it looks like Listing 10-1.

3. The SS listed in the project name refers to *Stock Suite.*

TIP *Use a console application configuration as you design and code custom actions so that a DOS-style box appears when MSI launches the action. Then, should the code not work as expected, you can at least be sure that the custom actions have actually run. After you've finished debugging the custom actions, copy the code to a new windowless application project so that custom actions are seamless to the end user.*

Listing 10-1. The SETUP_SS Project CLASS1.CS Example File

```
using System;
using System.IO;

namespace SETUP_SS
{
     /// <summary>
     /// Summary description for Class1.
     /// </summary>
     class Class1
     {
          /// <summary>
          /// The main entry point for the application.
          /// </summary>
          [STAThread]
          static void Main(string[] args)
          {
               foreach (string s in args)
               {
                    switch (s)
                    {
                         case "install":
                              install();
                              break;

                         case "commit":
                              commit();
                              break;

                         case "rollback":
                              rollback();
                              break;
                    case "uninstall":
                        uninstall();
                        break;
                }
            }
        }

        static void install()
        {
            TextWriter tx = new StreamWriter("C:\\SETUP_SS.TXT", true,  ↵
                System.Text.Encoding.Default);
            tx.WriteLine("The \"install\" custom action has been called.");
            tx.Close();
        }

        static void commit()
        {
            TextWriter tx = new StreamWriter("C:\\SETUP_SS.TXT", true,  ↵
                System.Text.Encoding.Default);
            tx.WriteLine("The \"commit\" custom action has been called.");
            tx.Close();
        }

        static void rollback()
        {
            TextWriter tx = new StreamWriter("C:\\SETUP_SS.TXT", true, ↵
                System.Text.Encoding.Default);
            tx.WriteLine("The \"rollback\" custom action has been called.");
            tx.Close();
        }

        static void uninstall()
        {
            TextWriter tx = new StreamWriter("C:\\SETUP_SS.TXT", true, ↵
                System.Text.Encoding.Default);
            tx.WriteLine("The \"uninstall\" custom action has been called.");
            tx.Close();
        }
    }
}
```

This is a pretty simple program. First, it checks for the existence of install, commit, rollback, or uninstall command-line parameters. If found, the DLL updates a log file located at C:\SETUP_SS.TXT to indicate that the desired action took place.

Once you've written the code, right-click the SETUP_SS project and choose Build to ensure that it has no problems that need to be addressed. Now that you've built the executable, you need to associate it with a custom action.

First, go back to the setup project and the Custom Action pane. In this particular case, you'll add the same action to each node with slightly differing parameters. Right-click the top-level tree item entitled Custom Action and choose Add Custom Action from the pop-up menu. Now add the primary output from the SETUP_SS project to the Application Folder item. This will populate each of the separate custom actions. Alternatively, you can right-click each custom action separately and choose Add Custom Action.

Second, you need to set two properties for each custom action. The first property is Arguments. This property contains a string that's passed to your executable as a command-line parameter during the installation. Use the respective arguments in Listing 10-1 for each action. For instance, the Argument property for the Install action should be install, the Commit action's argument should be commit, and so on.

Lastly, change the InstallerClass property to false. Installer classes are useful because they're written with rollback functionality built into them. Unfortunately, they require too much detail for this chapter so your SETUP_SS.EXE program will not allow automatic rollback. In this case, the InstallerClass property must be set to false, or your custom actions will not work correctly.

As you test your install, notice that a DOS-style box flashes twice. The log file created by your application, located at C:\SETUP_SS.TXT, indicates that the custom actions were called correctly.

Redistributing the Microsoft .NET Framework Files

Even though much has improved in new versions of Visual Studio, there's still a bit of a sting when it comes to deploying .NET Framework applications. You must make sure that the .NET Framework common language runtime exists on the end user's machine before installing your application. If you find that the Framework hasn't yet been installed, you need to either do so yourself or guide your end users in its setup.

NOTE *If using a third-party installation tool such as InstallShield or Wise, the Framework is automatically installed. Feel free to skip this section.*

For the poor SCM in the past, this meant picking and choosing which DLLs needed to be installed into the SYSTEM directory. Nowadays—whew!—you simply need to point your users to a single MSI installable object. The entire .NET Framework runtime is available through a file called DOTNETFX.EXE.

NOTE *Depending on the third-party applications that your developers have chosen to integrate with applications, it may still be necessary to install DLLs. In these cases, however, Microsoft has done a good job of providing MSI packages for groups of important system files. If forced with a choice between copying separate DLLs to a user's system and forcing a prerequisite installation of a service pack or framework package, it's much preferable to force the installation of the service pack.*

The .NET Framework is installed by default on the professional and server versions of Windows 2000 and XP. For earlier versions of Windows, however, it's necessary for the installation to check for the Framework's existence. If your installation determines that the Framework isn't installed on the user's system, it should halt and inform the user how to do so. The default MSI integration in Visual Studio automatically halts the installation for you.

You have several ways to install the .NET Framework package on an end user's system. The best way is to use a third-party installation tool—they've done all the work for you. If that method isn't available as an option, the following are a few other methods:

- You can instruct the user to utilize the Windows Update feature, located at `http://v4.windowsupdate.microsoft.com`, to download the installation package.
- Depending on your application's use, IT administrators can push the Framework onto client machines using a software deployment tool before they install the application.
- You can provide the DOTNETFX.EXE redistributable file on your application's installation media CD.

- You can create a program that adds the Framework to the end user's machine before running your application's setup. This is by far the most complicated scenario—but the steps to do so are detailed at Microsoft's MSDN website; use your browser to navigate to `http://msdn.microsoft.com` and search for the words *.NET Framework bootstrap*.

NOTE *End users must have administrative privileges to install the .NET Framework redistributable package.*

If you choose to include the Framework as a redistributable package on your CD media, keep in mind that it's possible Microsoft has updated the Framework with bug fixes or improvements since the CD was burned. Your README file should indicate to the user that a check for updates is in order.

To download the redistributable version of the .NET Framework for placement on a CD, navigate to `http://msdn.microsoft.com` and search for DOTNETREDIST.EXE. Once downloaded, however, *don't* redistribute the package in its current state. Instead, double-click the DOTNETREDIST.EXE executable on an in-house computer and save the output file, DOTNETFX.EXE, to a directory that mimics your media CD. Alternatively, the package is also available on the Visual Studio .NET Framework Software Development Kit (SDK) CD in a directory called DOTNETREDIST located at the CD's root—however, you can be assured that this won't be the latest version of the Framework.

NOTE *Microsoft makes it clear that you must own a valid Microsoft .NET Framework SDK license, available with your purchased copy of Visual Studio, in order to distribute the Framework package.*

You can install the Framework redistributable package on the following operating systems as long as Internet Explorer 5.01 or higher and MSI 2.0 or higher exists on the machine:

- Windows 98 and Windows 98 Second Edition
- Windows ME
- Windows NT 4.0 with Service Pack 6A (Workstation and Server)

- Windows 2000 (Professional, Server, and Advanced Server)
- Windows XP (Home, Professional, .NET Server)

Depending on the application being deployed, servers may also require the installation of MDAC 2.7 and Internet Information Services (IIS).

Using InstallShield

So you've looked at the "free" tool that comes with Visual Studio and decided for whatever reason that it just doesn't do it for you. If you've got some cash, you have several other choices for creating installations on the Windows operating system.

InstallShield Corporation, founded in 1987, is generally considered the industry leader for installation tools. It has recently branched out into providing system administration tools that can repackage applications and resolve potential conflicts prior to deployment.

At the time of this writing, InstallShield has several product offerings for installations. These products vary in sophistication, and, as you may expect, the products with the most flexibility tend to be the most expensive:

Developer: This product provides a frontend to the MSI API. In addition to the basic MSI features mentioned earlier in this chapter, Developer also provides the ability to redistribute the MSI installer, the .NET Framework, ActiveX Data Object (ADO) components, and other popular Microsoft system objects. In addition to creating typical MSI installations, Developer optionally allows install writers to script installations using InstallScript, a proprietary language that users may know from previous incarnations of InstallShield products. Developer installations that use InstallScript still exploit MSI for much of their functionality.

Express: This is the entry product to the InstallShield line—a product that might be called *Developer, Jr*. Its key benefits are its ease of use and relatively low cost. Because it doesn't provide the scripting capabilities of Developer, Express is a completely MSI-based product. In other words, most of what you can do with Express, you can do with the free version of MSI that comes with Visual Studio—albeit sometimes with a great deal of difficulty. In addition to being able to quickly package the .NET Framework and other redistributables, Express provides a wizard-based creation tool that allows installation writers to quickly create a usable installation. With a price point that's almost 75-percent off the Developer or Professional tools, Express is worth checking out for less-complicated installation procedures.

Professional: This product is the update to the script-based InstallShield product that has been around for the last decade. The bright side to using this tool is that legacy installations can easily be updated, and there's no need to start from scratch. The downside to using this product is its incompatibility with the feature-rich MSI functionality. Because the Manufacturer Suggested Retail Price (MSRP) of this product is more than $1,000 (though upgrades are available for less), I recommend using an MSI-based tool in its stead if starting an installation from scratch.

NOTE *In order to be eligible for the "Optimized for Enterprise" logo provided by Microsoft, you must use an MSI-based installation tool. Professional and Developer installations using InstallScript might not be the tools to use to meet this requirement. Eligibility for the basic Windows XP logo is still available to script-based installation programs.*

MultiPlatform: If creating installations for a product that spans several operating systems, MultiPlatform might be the right tool for the job. Its exploitation of the Java language allows an installation to have a consistent look and feel regardless of the platform on which it's being installed. Supported platforms include many flavors of Unix and Linux as well as Windows. This type of power is expensive, however, both in terms of financial cost and the possible loss of operating-specific installation features included with MSI, the Sun package technology, and Linux RPM.[4]

Creating a Setup in Four Hours (or Less) Using InstallShield Express

All of InstallShield's products are available as evaluation downloads from its Web site at `http://www.installshield.com`. Any installations created with the evaluations are fully featured but hobbled by a message indicating that the install was created with a free version of the software. Let's take a moment to try out these products by creating installations for the Stock Suite product using Express and Developer—the MSI-based installation tools.

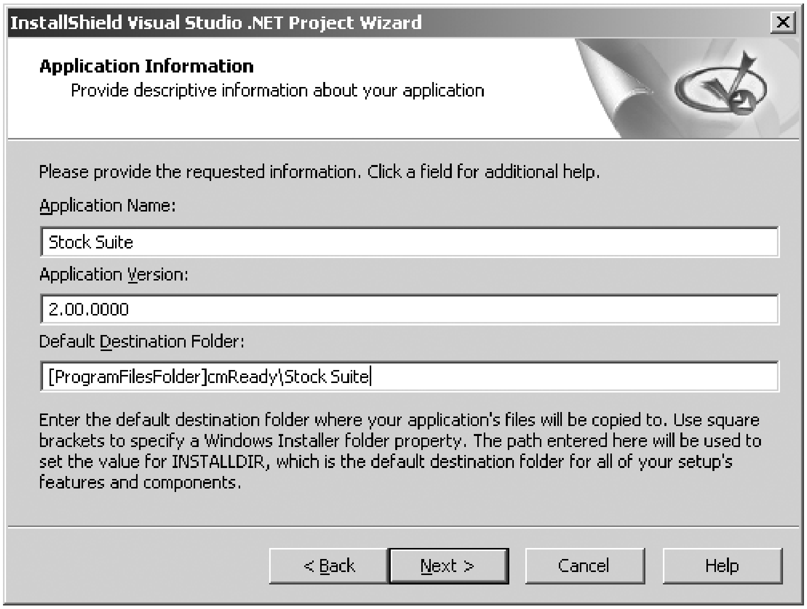
After downloading and installing the last version of Express,[5] try building a new setup by using its Visual Studio integration. After loading the Stock Suite

4. The folks at InstallShield have assured me that there's no loss of any operating-system-specific functionality when using the MultiPlatform product.
5. The example included in this section was created with InstallShield Express 4.0.

solution in Visual Studio, right-click it in Solution Explorer and choose Add ➤ New Project. Navigate to the InstallShield Express Projects folder and choose the Project Wizard. Before clicking OK, be sure to name the project appropriately and choose a proper destination folder. In this case, name the project *IS_Express* and use the default destination folder.

Configure the installation's functionality by using the InstallShield Project Wizard. Click Next at the splash screen, and the Application Information screen appears, as displayed in Figure 10-12. Change the application name, version, and default install location as directed by the edit boxes. The default install location refers to where the application should be installed on an end user's machine.

NOTE *Be aware that some of the InstallShield Wizard dialog boxes aren't as intuitive as they could be. For instance, in the Default Destination Folder edit box, it could be assumed that the Program Files Folder, Your Company Name, and Your Product Name are variables that will be filled in automatically by the wizard. Although Program Files Folder is indeed a variable (as indicated by the brackets surrounding it), the others aren't. Be sure to change these values to reflect your company and product names as necessary. If you do make a mistake, however, don't worry about it. Express is forgiving and will let you change any properties using the Visual Studio integration interface at a later time.*



Application Information
Provide descriptive information about your application

Please provide the requested information. Click a field for additional help.

Application Name: Stock Suite

Application Version: 2.00.0000

Default Destination Folder: [ProgramFilesFolder]cmReady\Stock Suite

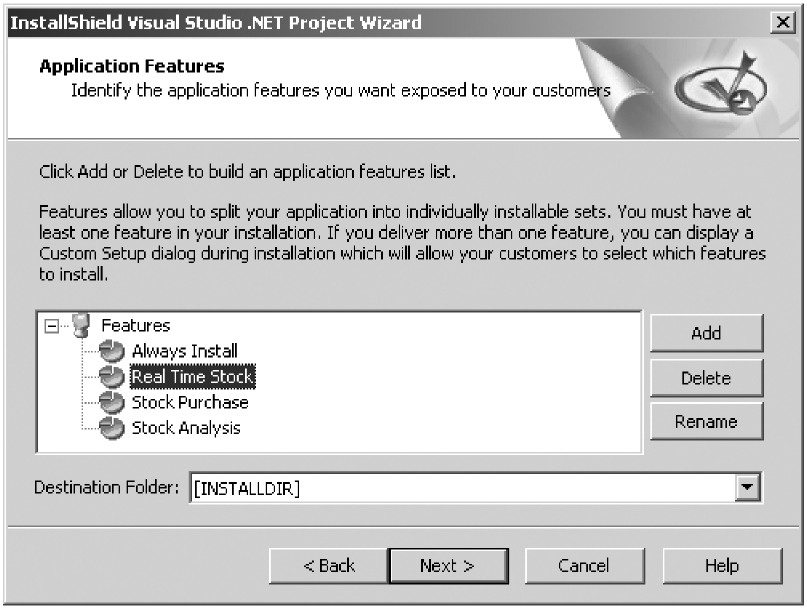
Enter the default destination folder where your application's files will be copied to. Use square brackets to specify a Windows Installer folder property. The path entered here will be used to set the value for INSTALLDIR, which is the default destination folder for all of your setup's features and components.

Figure 10-12. Adding application information using the InstallShield Express Project Wizard

Click Next, and a screen will ask if you'd like to enable your product to receive updates via the InstallShield update service. This is a pay-per-customer service provided by InstallShield to ease the pain of providing patches and service packs to its customers. This service is both set up and paid for separately. You won't be using this functionality for the example project in this chapter, so uncheck the Enable checkbox and click Next.

The next screen asks you for your company name, phone number, and Web address. This information is displayed to customers via the Add/Remove Programs Control Panel. The company name is also used as the base point for your product's Registry tree of information. This name should be consistent with any other products you may have created and distributed.

After you've clicked Next on the Company Information screen, the Application Features screen appears. As displayed in Figure 10-13, use this screen to divvy up the product into installable parts or, as InstallShield calls them, features. Use the spreadsheets you created at the beginning of this chapter and remember to approach this task from the customer's perspective.


InstallShield Visual Studio .NET Project Wizard

Application Features
Identify the application features you want exposed to your customers

Click Add or Delete to build an application features list.

Features allow you to split your application into individually installable sets. You must have at least one feature in your installation. If you deliver more than one feature, you can display a Custom Setup dialog during installation which will allow your customers to select which features to install.

- Features
  - Always Install
  - Real Time Stock
  - Stock Purchase
  - Stock Analysis

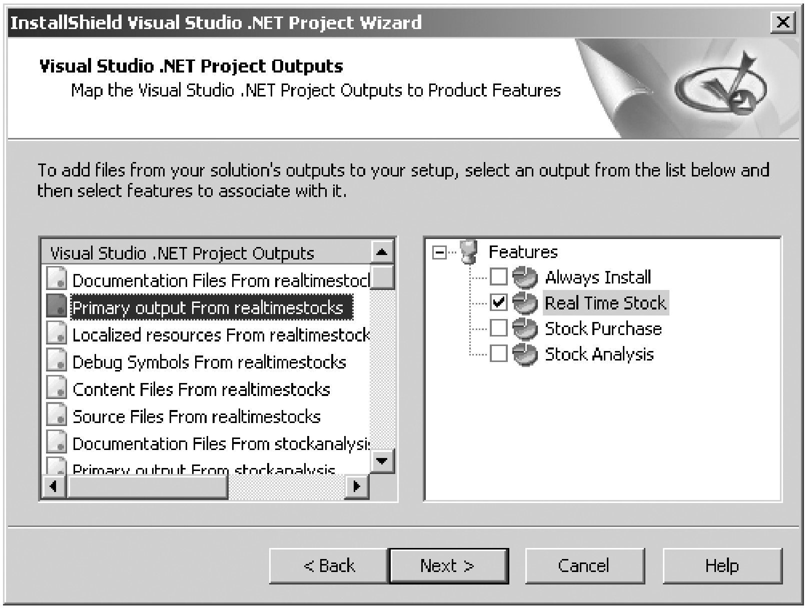
Destination Folder: [INSTALLDIR]

Figure 10-13. Including installable application features using the InstallShield Express Project Wizard

TIP *For all top-level features, make sure that the Features item in the tree is highlighted before selecting the Add button. Otherwise, features will be added as children beneath the currently selected object.*

Other than optionally specifying different destination directories for each of your installable parts, there's no need for any other customization at this time—you're simply informing the wizard of the installable parts' existence. Click Next to continue.

On the Visual Studio Project Outputs screen, as displayed in Figure 10-14, specify which files make up each feature. Designate these files as outputs as you did with the default MSI integration discussed earlier in this chapter.


InstallShield Visual Studio .NET Project Wizard

Visual Studio .NET Project Outputs

Map the Visual Studio .NET Project Outputs to Product Features

To add files from your solution's outputs to your setup, select an output from the list below and then select features to associate with it.

Visual Studio .NET Project Outputs
- Documentation Files From realtimestocl
- Primary output From realtimestocks
- Localized resources From realtimestock
- Debug Symbols From realtimestocks
- Content Files From realtimestocks
- Source Files From realtimestocks
- Documentation Files From stockanalysi:
- Primary output From stockanalysis

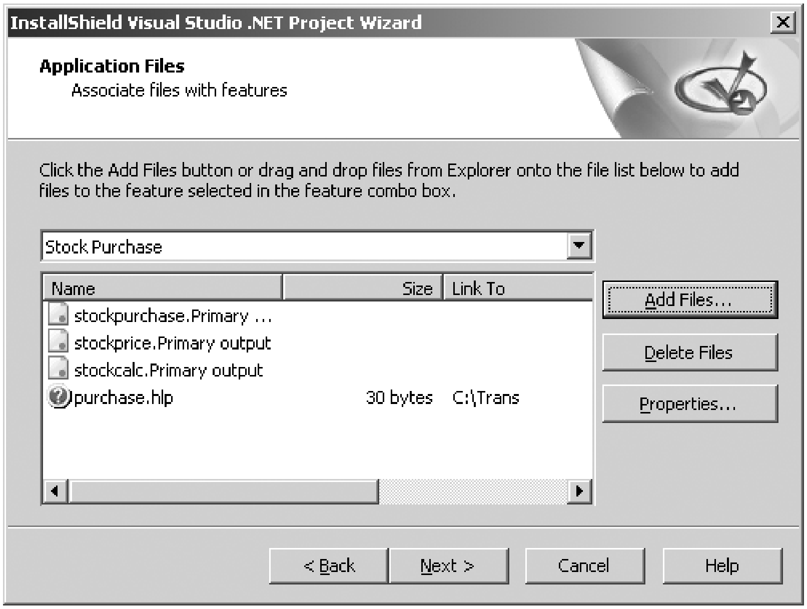
Features
- [ ] Always Install
- [✓] Real Time Stock
- [ ] Stock Purchase
- [ ] Stock Analysis

Figure 10-14. Choosing executable files to install

NOTE *If you're asked to navigate the file system to add objects to the setup, you're probably running the wizard in a solution that doesn't contain any other projects. Although this is an acceptable way to create an installation, it means that the steps in this example may not always be applicable to your solution.*

While navigating down the list of Visual Studio .NET Project Outputs on the left side of the screen, check the features with which that particular object should be installed. It's likely that many of the outputs—such as the debug symbols—won't be included in the installation. Simply keep all the features unchecked for those outputs. For outputs that are needed by multiple components, check all of the features for which the output should be installed. For outputs that should always be installed regardless of which features the user has chosen, check the Always Install checkbox. You can leave the other features unchecked if the Always Install checkbox is chosen—the object will still always be installed.

Use the Application Files screen to add non-project-related files from the file system to the project. For instance, if you were writing an office suite program, your word processor might have an associated set of templates that should be installed on the user's system but wouldn't be included in any of the projects of the solution. Keep in mind that files should be added to each feature as listed in the drop-down combo box. For instance, if the Stock Purchase component has an associated Help file called PURCHASE.HLP, use the drop-down combo box to highlight the Stock Purchase feature and then use the Add Files button to search the hard drive for the file. Figure 10-15 displays this example.


InstallShield Visual Studio .NET Project Wizard

Application Files
Associate files with features

Click the Add Files button or drag and drop files from Explorer onto the file list below to add files to the feature selected in the feature combo box.

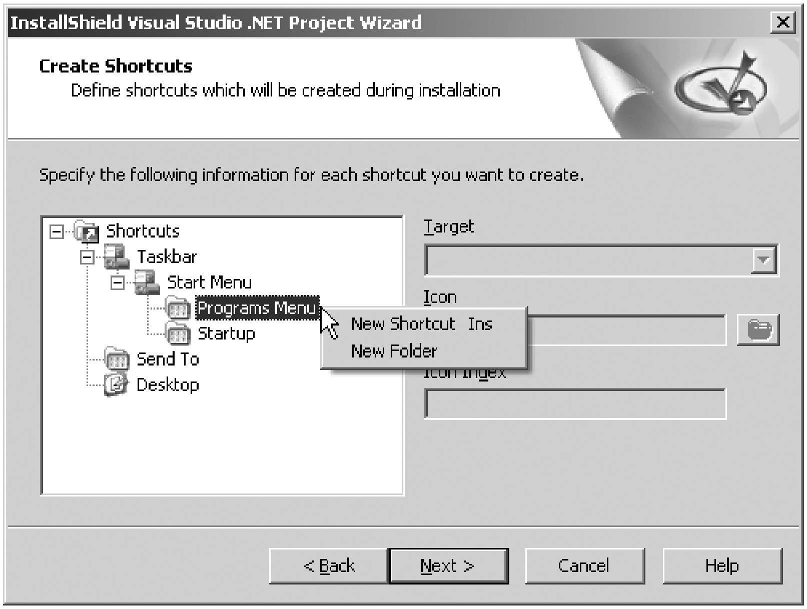
Stock Purchase

Name	Size	Link To
stockpurchase.Primary ...		
stockprice.Primary output		
stockcalc.Primary output		
purchase.hlp	30 bytes	C:\Trans

Figure 10-15. Choosing nonexecutable files to install

If you choose to set specific attributes for any of your outputs or files, highlight them now and click the Properties button. The screen that appears allows you to set system attributes (such as making a file read-only or hidden) or allow for applicable Component Object Model (COM) registration. Be sure to consult your engineers whenever changing these specifications (or, in some cases, not changing them)—this can sometimes severely limit your project's functionality.

After clicking Next, your next step is to create shortcuts for objects as necessary. Again, the user interface of this dialog box isn't always the most intuitive—to add a shortcut, right-click a folder in the left window and choose New Shortcut from the pop-up menu, as displayed in Figure 10-16.



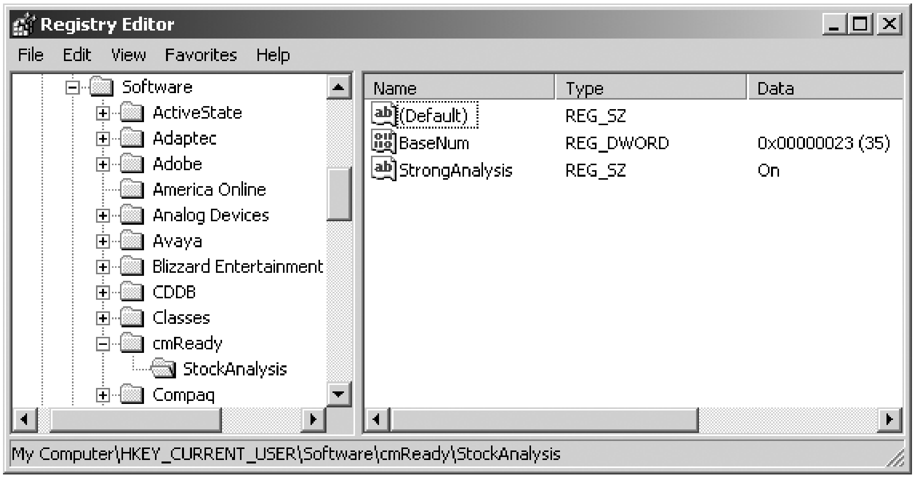
InstallShield Visual Studio .NET Project Wizard
Create Shortcuts
Define shortcuts which will be created during installation
Specify the following information for each shortcut you want to create.
Shortcuts
Taskbar
Start Menu
Programs Menu
Startup
Send To
Desktop
Target
Icon
Icon Index
New Shortcut Ins
New Folder
< Back
Next >
Cancel
Help

Figure 10-16. Creating shortcuts to install with the pop-up menu

Once created, use the Target and Icon edit boxes on the right side of the screen to specify the shortcut's name and look. If you choose an application file that has multiple icons embedded in it, use the Icon Index edit box to specify which icon should be used. The index will be one based (for you zero-using programmer types out there), so use 1 for the first icon in the file, 2 for the second, and so on.

You're almost done! Just a couple more steps. Use the next dialog box to add Registry information. For instance, you might want to set a couple of default values for your Stock Analysis tool. If you did this by hand using the REGEDIT tool, these default values might look like those displayed in Figure 10-17.

When you worked with the Visual Studio MSI integration, you set these values by hand using a Registry-like tree. You'll have that same ability in InstallShield Express after the wizard completes. If you like, however, you can specify Registry values now by writing what are known as *REG files*. You must write these files in a specific format, as displayed in Listing 10-2.



Name	Type	Data
(Default)	REG_SZ	
BaseNum	REG_DWORD	0x00000023 (35)
StrongAnalysis	REG_SZ	On

My Computer\HKEY_CURRENT_USER\Software\cmReady\StockAnalysis

Figure 10-17. Using RegEdit to display Registry values

Listing 10-2. An Example of a REG File

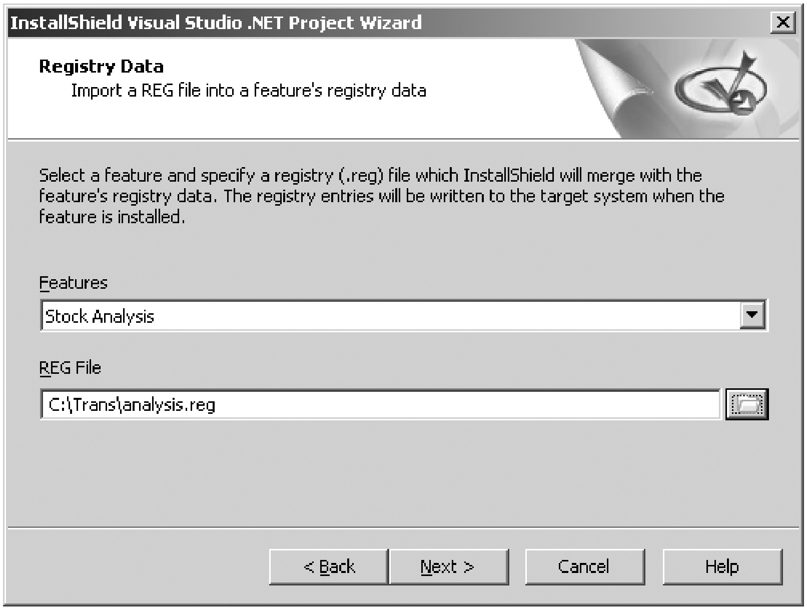
```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\cmReady\StockAnalysis]
@=""
"StrongAnalysis"="On"
"BaseNum"=dword:00000023
;
```

TIP *Typing this information yourself is the hard way to create this file. Instead, try using the REGEDIT default Registry editing tool to manually add whatever default values you'd like to include in the Registry and then export them back out to a text file. Choose Start ➤ Run and type* REGEDIT *followed by a Return. When the tool opens, navigate to the proper location for your values and add them. When finished, highlight the key that contains those values and choose File ➤ Export to create a Registry file on your file system. Make sure you only export the information you need by clicking specific keys—any child keys that fall under the highlighted key will also be added to the new REG file.*

CAUTION *The Windows Registry isn't a toy with which to trifle. You can severely damage your Windows operating system by making a mistake while editing the Registry. Worse, by making a mistake with REG files and installations, you can severely damage your end user's machine. If you don't understand how the Registry works and how to use it, consult an experienced Windows programmer before changing Registry information in installations.*

Once you've created a REG file on your hard drive, use the Registry Data page to associate it with a feature, as displayed in Figure 10-18. As with earlier dialog boxes, use the drop-down combo box to choose the appropriate feature before filling in the REG File edit box.


InstallShield Visual Studio .NET Project Wizard

Registry Data

Import a REG file into a feature's registry data

Select a feature and specify a registry (.reg) file which InstallShield will merge with the feature's registry data. The registry entries will be written to the target system when the feature is installed.

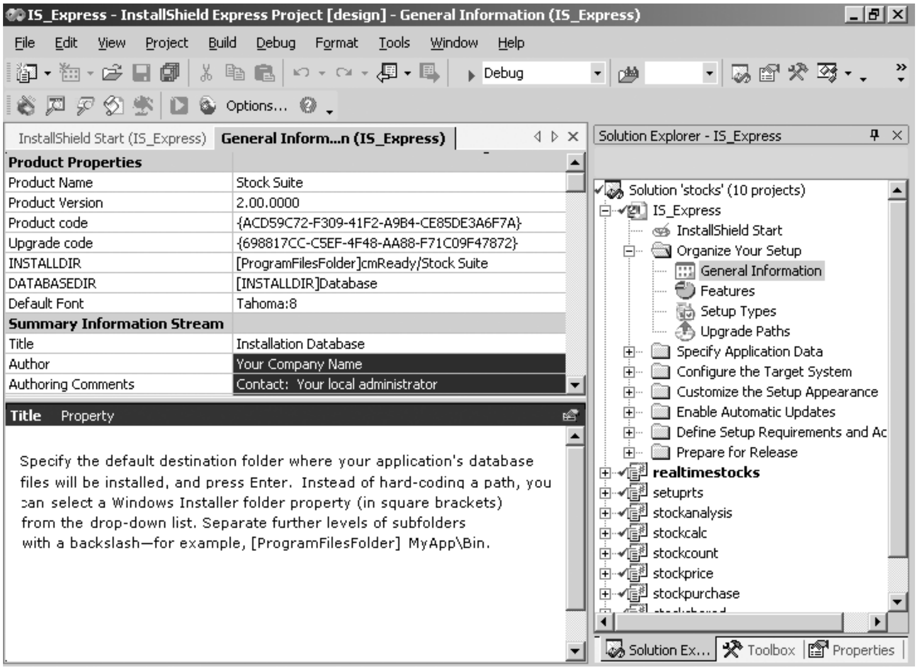
Features: Stock Analysis

REG File: C:\Trans\analysis.reg

Figure 10-18. Specifying REG files to be installed

Use the final wizard step to add optional dialog boxes to the installation. For instance, include the dialog boxes that display a license agreement, ask for specific customer information, and show a "ready" dialog box by checking their appropriate checkboxes. You'll configure these more completely after the wizard finishes because you can't set several properties here.

After a summary screen, the InstallShield wizard exits, and you'll have a project named IS_Express in the Solution Explorer. This new project is quite different from that of a C# or other project. The objects listed aren't source files as you know them. Instead, the project virtualizes sets of properties together into logical groups—somewhat akin to the windows you used with the Visual Studio MSI setup tools. Figure 10-19 displays an example of the IS_Express window. The right side of the window is the familiar Solution Explorer. But now the top section of the left pane displays a set of properties relevant to the highlighted InstallShield listing in the Solution Explorer. Express always displays the help in the bottom-left window pane.


IS_Express - InstallShield Express Project [design] - General Information (IS_Express)
File Edit View Project Build Debug Format Tools Window Help
Debug
Options...
InstallShield Start (IS_Express)
General Inform...n (IS_Express)
Solution Explorer - IS_Express
Solution 'stocks' (10 projects)
IS_Express
InstallShield Start
Organize Your Setup
General Information
Features
Setup Types
Upgrade Paths
Specify Application Data
Configure the Target System
Customize the Setup Appearance
Enable Automatic Updates
Define Setup Requirements and Ac
Prepare for Release
realtimestocks
setuprts
stockanalysis
stockcalc
stockcount
stockprice
stockpurchase
Solution Ex...
Toolbox
Properties

Product Properties	
Product Name	Stock Suite
Product Version	2.00.0000
Product code	{ACD59C72-F309-41F2-A9B4-CE85DE3A6F7A}
Upgrade code	{698817CC-C5EF-4F48-AA88-F71C09F47872}
INSTALLDIR	[ProgramFilesFolder]cmReady/Stock Suite
DATABASEDIR	[INSTALLDIR]Database
Default Font	Tahoma:8
Summary Information Stream	
Title	Installation Database
Author	Your Company Name
Authoring Comments	Contact: Your local administrator

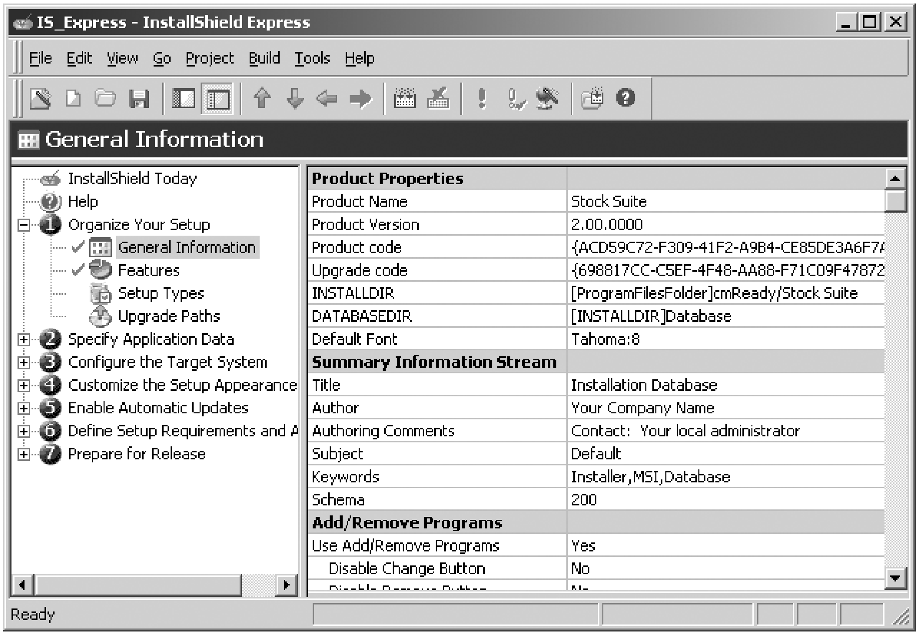
Title Property

Specify the default destination folder where your application's database files will be installed, and press Enter. Instead of hard-coding a path, you can select a Windows Installer folder property (in square brackets) from the drop-down list. Separate further levels of subfolders with a backslash—for example, [ProgramFilesFolder] MyApp\Bin.

Figure 10-19. Viewing a setup project in the InstallShield Express Visual Studio integration

Your last step in configuring the installation is to navigate through all the properties and double-check all of the settings. Although the InstallShield wizard gave you a good start, it didn't require you to fill in several important fields.

The notable changes you make in this step are to update the general information page, set descriptions for the features, remove the "minimal" setup type, and let the license dialog box know where your End User License Agreement (EULA) text file is located on the local hard drive. Feel free to explore and change any desired settings. At this point, you might also want to explore the stand-alone IDE that was installed with the Express package, as displayed in Figure 10-20. Start the IDE by clicking the Start button and navigating to the InstallShield program group.

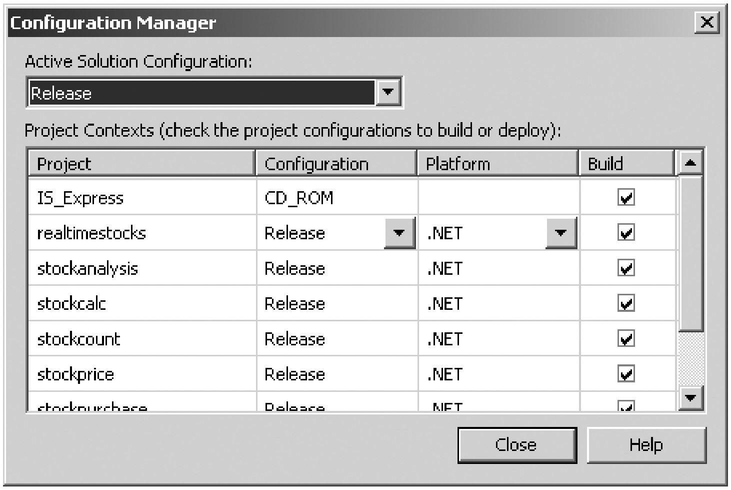


Product Properties	
Product Name	Stock Suite
Product Version	2.00.0000
Product code	{ACD59C72-F309-41F2-A9B4-CE85DE3A6F7A
Upgrade code	{698817CC-C5EF-4F48-AA88-F71C09F47872
INSTALLDIR	[ProgramFilesFolder]cmReady/Stock Suite
DATABASEDIR	[INSTALLDIR]Database
Default Font	Tahoma:8
Summary Information Stream	
Title	Installation Database
Author	Your Company Name
Authoring Comments	Contact: Your local administrator
Subject	Default
Keywords	Installer,MSI,Database
Schema	200
Add/Remove Programs	
Use Add/Remove Programs	Yes
Disable Change Button	No

Figure 10-20. Viewing the same project in the InstallShield Express IDE

Once you've created the installation, it's time to build and test it. First, open the Configuration Manager dialog box and specify via which output media you plan to distribute the application. InstallShield Express supports distribution via several media types including CD-ROM, DVD-ROM, and Web download. If the installation will span disks (such as when distributing on a floppy disk), Install-Shield automatically splits the installation into multiple parts as necessary. For your project, pick the CD-ROM installation. Figure 10-21 displays how the release configuration for your solution might appear.

NOTE *Don't forget to put the installation at the end of the build order or to make it the startup project! For more information about the Configuration Manager and build orders, see Chapter 9, "Builds for Windows .NET."*


Configuration Manager
Active Solution Configuration:
Release
Project Contexts (check the project configurations to build or deploy):
Close
Help

Project	Configuration	Platform	Build
IS_Express	CD_ROM		✓
realtimestocks	Release	.NET	✓
stockanalysis	Release	.NET	✓
stockcalc	Release	.NET	✓
stockcount	Release	.NET	✓
stockprice	Release	.NET	✓
stockpurchase	Release	.NET	✓

Figure 10-21. Changing propeties using the Configuration Manager

Once you've set the active configuration and chosen CD_ROM for your IS_Express project, you can build the product from within Visual Studio using one of three ways:

- The first is to choose the Build ➤ Build Solution menu item. This builds all of the release components in the solution and then packages them into the installation as defined in the active configuration.
- The second way to build is to right-click the installation project itself and choose Build.
- Finally, you can also open the Prepare for Release folder in the installation project and choose the Build Your Release item.

NOTE *If using project outputs, you must build the installation from within Visual Studio. The Express IDE is unable to compile and build .NET components.*

Once built, there are a couple of ways to test the new setup. First, you can choose the Prepare for Release folder under the installation project and choose the Test Your Release item. This allows you to navigate through the user interface without actually installing any components on your hard drive. Alternatively, you can use the Debug ➤ Start menu item to run the application. Keep in mind that, by default, you can only uninstall Express installations by using the Add/Remove Program Files feature in the Control Panel.

Lastly, the following are some tips on InstallShield Express:

Use the Help icon: The Help for Express isn't integrated into the dynamic Visual Studio Help. To access it, make sure to use the Help icon that appears on the InstallShield Express toolbar when the project is active. This means that it's not possible to use F1 context-sensitive help.

You'll have to stick to one language: You can't create multilanguage installations with InstallShield Express. The language you choose in the Setup Wizard is the language with which you're stuck. It's possible to import Express installations into the Developer product that support multilanguage installations. Of course, you could create multilanguage installations by simply copying the existing project and then changing the string tables, but this isn't a good solution according to SCM standards. Frankly, if you need to have a multilanguage installation, Express isn't the tool for you.

Forget source control: Visual Studio's source control via SourceSafe doesn't work with the Express product.

You can't reorder dialog boxes: Although it's possible to select or deselect user interface dialog boxes, there's no way to reorder them using Express. If this is a requirement, you'll need to switch to the default Visual Studio MSI integration or another product. Developer supports both the reorder of dialog boxes and the creation of custom dialog boxes.

You have to use the InstallShield branding: As included with all InstallShield products, the Express EULA requires that you not change or remove the InstallShield branding or copyright information. The term *branding* refers to the InstallShield name that's displayed in several dialog boxes of the installation. The marketing spin from the InstallShield folks is that customers trust the InstallShield label and that the branding makes both the developer and the customer feel more comfortable—though I bet the marketing aspect of the branding play a part in the way it writes the license. Many development houses choose not to allow third-party software to brand their products. In this case, you'll need to switch to the default Visual Studio MSI integration or to one of the Wise products discussed later in this chapter.

.NET is installed automatically: The Microsoft .NET Framework is installed automatically with an Express installation when it isn't found on an end user's machine. This is a handy feature that isn't available through the default Visual Studio MSI integration without putting in some effort. I found that Install-Shield Express tried to reinstall the .NET Framework on the same machine where the components were built—an action that seems rather redundant. Although it's possible that the installation was simply checking versions of the Framework's components without changing the system, it was an extremely time-consuming process that I couldn't cancel or avoid.

Getting More Options—Using InstallShield Developer

At first glance, InstallShield Developer doesn't appear to be all that different from Express. The Setup Wizard is similar to that of Express, and you've still got the choice to engage MSI as your installation engine. And that's where most of the similarities end. The Developer interface offers quite an increase in customization and features. Some of the additional features include the following:

- A visual dialog box editor
- An installation debugger
- A direct MSI editor
- Visual Studio source control integration
- Multilanguage installations available at an additional price

I suggest that the most significant difference between Developer and Express is the ability to use InstallShield's scripting mechanism as the install's engine in addition to MSI.[6] This allows a level of customization not possible with the Windows Installer—including the inclusion of 100-percent custom dialog boxes. By using InstallScript, MSI doesn't fade completely out of the picture…InstallShield

6. InstallShield recommends that the basic MSI setup project be used in most cases and that InstallShield scripting be used only in special cases.

still calls it for much of its basic functionality. Developer supports both scripting and typical MSI installations. You might choose a nonscripting Developer scenario for multilanguage support or if a dialog box reorder is necessary.

NOTE *To be eligible for the "Optimized for Enterprise" logo provided by Microsoft, you must use an MSI-based installation tool. Professional and Developer installations using InstallScript aren't the tools to use to meet this requirement. Eligibility for the basic Windows XP logo is still available to script-based installation programs.*

The scripting language in the latest version of Developer[7] is similar to the InstallScript available in previous incarnations of InstallShield. Unfortunately, folks new to the technology may suffer when choosing to go down this road—even simple tasks are often daunting. For example, adding a default, noncustom dialog box to the installation must be accomplished completely through scripting. Luckily, InstallShield provides a default script and has many examples on its support site and in documentation. A complete InstallScript language reference is also included when Developer is installed.

Listing 10-3 displays a complete Developer script and the steps that can be taken to add a new dialog box, use strings tables for multilanguage support, and put up a customized "ready-to-install" dialog box. Bolded text indicates where I made changes to the default code provided by InstallShield.

NOTE *All custom and some provided dialog boxes require the InstallScript engine.*

Listing 10-3. An Example Developer InstallScript Script Used for the StockSuite Product Installation.

```
////////////////////////////////////////////////////////////////////////////////
//  File Name:        Setup.rul
//  Description:      InstallShield script
//  Comments:         This script was generated based on the selections you made in
//                    the Project Wizard.  Refer to the help topic entitled "Modify
//                    the script that the Project Wizard generates" for information
//                    on possible next steps.
////////////////////////////////////////////////////////////////////////////////
```

7. The example included in this section was created with InstallShield Developer 8.0.

```
////////////////////////////////////////////////////////////////////////////////
// Both the IDS_C_WINDOWTITLE and IDS_C_SDWELCOMETITLE objects
// were added to the English string table as follows:
// IDS_C_WINDOWTITLE = "cmReady Stock Suite 2.0"
// IDS_C_SDWELCOMETITLE = "Welcome to the cmReady Stock ⮠
//     Suite 2.0 setup program.  Click "next" to install and use this product."
////////////////////////////////////////////////////////////////////////////////

// Include header files
#include "ifx.h"

////////////////////// string defines ////////////////////////////

//////////////////// installation declarations ///////////////////
// ----- DLL function prototypes -----
    // your DLL function prototypes

// ---- script function prototypes -----
    // your script function prototypes

    // your global variables

///////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION:   OnFirstUIBefore
//
//  EVENT:      FirstUIBefore event is sent when installation is run for the first
//              time on given machine. In the handler installation usually displays
//              UI allowing end user to specify installation parameters. After this
//              function returns, FeatureTransferData is called to perform file
//              transfer.
//
///////////////////////////////////////////////////////////////////////////////
function OnFirstUIBefore()
    number  nResult,nSetupType;
    string  szTitle, szMsg;
    string  szLicenseFile, szQuestion;
    string  szName, szCompany;
    string  szFile;
    string  szTargetPath;
    string  szDir;
    string  szFeatures, szTargetdir;
    number  nLevel;
    LIST    listStartCopy;
    LIST    list;
    number  nvSize;
    number  nUser;

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    string szSetupType;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

begin
    // TO DO: if you want to enable background, window title, and caption bar title
    // SetTitle( @PRODUCT_NAME, 24, WHITE );
    // SetTitle( @PRODUCT_NAME, 0, BACKGROUNDCAPTION );
    // Enable( FULLWINDOWMODE );
    // Enable( BACKGROUND );
    // SetColor(BACKGROUND,RGB (0, 128, 128));
    SHELL_OBJECT_FOLDER = @PRODUCT_NAME;
    nSetupType = TYPICAL;
    szDir = INSTALLDIR;
    szName    = "";
    szCompany = "";

Dlg_Start:
    // beginning of dialogs label

Dlg_SdWelcome:

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
    szMsg   = @IDS_C_SDWELCOMETITLE;
//NON INSTALLSHIELD DEFAULT CODE STARTS HERE

    nResult = SdWelcome( szTitle, szMsg );
    if (nResult = BACK) goto Dlg_Start;

Dlg_SdLicense:
//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code tells the Installer which license file to use
    //when displaying the license textbox.  The SUPPORTDIR
    //variable indicates that I've dropped the license.txt file
    //into the Support Files window listed under Behavior
    //and Logic.  Respective text files should be dropped into
    //each language supported by your instllation.
    szLicenseFile = SUPPORTDIR ^ "license.txt";
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    szTitle    = "";
    szMsg      = "";
    szQuestion = "";
    nResult    = SdLicense( szTitle, szMsg, szQuestion, szLicenseFile );
    if (nResult = BACK) goto Dlg_SdWelcome;

Dlg_SdShowInfoList:
    //sk
    szFile = SUPPORTDIR ^ "readme.txt";
    list = ListCreate( STRINGLIST );
    ListReadFromFile( list, szFile );

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    szMsg    = "";
    nResult  = SdShowInfoList( szTitle, szMsg, list );
    ListDestroy( list );
    if (nResult = BACK) goto Dlg_SdLicense;

Dlg_SdCustomerInformation:

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    szMsg   = "";
    nResult = SdCustomerInformation( szTitle, szName, szCompany, nUser );
    if (nResult = BACK) goto Dlg_SdShowInfoList;

Dlg_SdAskDestPath:

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
```

```
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    szMsg   = "";
    nResult = SdAskDestPath( szTitle, szMsg, szDir, 0 );
    INSTALLDIR = szDir;
    if (nResult = BACK) goto Dlg_SdCustomerInformation;

Dlg_SetupType:

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    szMsg      = "";
    nResult = SetupType ( szTitle , szMsg , "" , nSetupType , 0 );
    if (nResult = BACK) then
        goto Dlg_SdAskDestPath;
    else
        nSetupType = nResult;
        if (nSetupType != CUSTOM) then
           szTargetPath = INSTALLDIR;
            nvSize = 0;
            FeatureCompareSizeRequired(MEDIA,szTargetPath,nvSize);
            if (nvSize != 0) then
                MessageBox( szSdStr_NotEnoughSpace, WARNING );
                goto Dlg_SetupType;
            endif;
        endif;
    endif;

Dlg_SdFeatureTree:
    if ((nResult = BACK) && (nSetupType != CUSTOM)) goto Dlg_SetupType;

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE
```

```
    szMsg      = "";
    szTargetdir = INSTALLDIR;
    szFeatures = "";
    nLevel = 2;
    if (nSetupType = CUSTOM) then
        nResult = SdFeatureTree(szTitle, szMsg, szTargetdir, szFeatures, nLevel);
        if (nResult = BACK) goto Dlg_SetupType;
    endif;

Dlg_SdStartCopy:

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
    //This code gets strings from the currently loaded language
    //string table and places them in variables.
    szTitle = @IDS_C_WINDOWTITLE;
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    szMsg   = "";
    listStartCopy = ListCreate( STRINGLIST );
    //The following is an example of how to add a string(szName) to a
list(listStartCopy).
    //eg. ListAddString(listStartCopy,szName,AFTER);

//NON INSTALLSHIELD DEFAULT CODE STARTS HERE
           //This code creates a string array which will be loaded into the
           //textbox for the Ready to Copy dialog box.  The customer's
           //name, company and install directory will be displayed.  The
           //customer can cancel or continue at this point depending on
           //what they read on this dialog box.
           ListAddString(listStartCopy, "Stock Suite 2.0 Setup", AFTER);
           ListAddString(listStartCopy, "", AFTER);
           ListAddString(listStartCopy, "CUSTOMER INFORMATION", AFTER);
           ListAddString(listStartCopy, "Customer Name = " + szName, AFTER);
           ListAddString(listStartCopy, "Customer Company = " + szCompany, AFTER);
           ListAddString(listStartCopy, "", AFTER);
           ListAddString(listStartCopy, "INSTALL INFORMATION", AFTER);
           ListAddString(listStartCopy, "Install Directory = " + INSTALLDIR,
AFTER);
           switch (nSetupType)
               case CUSTOM:
                   szSetupType = "Custom";
               case TYPICAL:
                    szSetupType = "Typical";
                case COMPACT:
                    szSetupType = "Compact";
            endswitch;
            ListAddString(listStartCopy, "Install Type = " + szSetupType, AFTER);
//NON INSTALLSHIELD DEFAULT CODE ENDS HERE

    nResult = SdStartCopy( szTitle, szMsg, listStartCopy );
    ListDestroy(listStartCopy);
    if (nResult = BACK) goto Dlg_SdFeatureTree;

    Enable(STATUSEX);

    return 0;
end;

///////////////////////////////////////////////////////////////////////////////
//
//  FUNCTION:   OnMaintUIBefore
//
//  EVENT:      MaintUIBefore event is sent when end user runs installation that
//              has already been installed on the machine. Usually this happens
//              through Add/Remove Programs applet. In the handler, installation
//              usually displays UI allowing end user to modify existing
//              installation or uninstall application. After this function
//              returns, FeatureTransferData is called to perform file transfer.
//
///////////////////////////////////////////////////////////////////////////////
function OnMaintUIBefore()
    NUMBER nResult, nType;
    STRING szTitle, szMsg, svDir, svResult, szCaption;
begin
    // TO DO: if you want to enable background, window title, and caption bar title

    // SetTitle( @PRODUCT_NAME, 24, WHITE );
    // SetTitle( @PRODUCT_NAME, 0, BACKGROUNDCAPTION );
    // SetColor(BACKGROUND,RGB (0, 128, 128));
    // Enable( FULLWINDOWMODE );
    // Enable( BACKGROUND );
Dlg_Start:
    Disable(BACKBUTTON);
    nType = SdWelcomeMaint(szTitle, szMsg, MODIFY);
    Enable(BACKBUTTON);
    if (nType = REMOVEALL) then
        svResult = SdLoadString(IFX_MAINTUI_MSG);
        szCaption = SdLoadString(IFX_ONMAINTUI_CAPTION);
        nResult = SprintfBox(MB_OKCANCEL,szCaption,"%s",svResult);
        if (nResult = IDCANCEL) goto Dlg_Start;
    endif;

    nResult = NEXT;

Dlg_SdFeatureTree:
    if (nType = MODIFY) then
        szTitle = "";
        szMsg = "";
        nResult = SdFeatureTree(szTitle, szMsg, INSTALLDIR, "", 2);
        if (nResult = BACK) goto Dlg_Start;
    endif;

    switch(nType)
        case REMOVEALL: FeatureRemoveAll();
        case REPAIR:    FeatureReinstall();
    endswitch;

    Enable(STATUSEX);
end;
```

The following are some final tips on InstallShield Developer:

Integrates gracefully with Visual Studio: I found that Developer's integration into the Visual Studio environment is much better than that of Express. This especially includes the support of a project's right-click functionality.

Provides mulilanguage support: One language comes for free with Install-Shield Developer. Support for 32 other languages is available for an additional cost from InstallShield. Custom dialog boxes or strings, of course, aren't included in purchased language packages. In those cases, string tables are used for complete translations.[8]

Gives you debugging capabilities: InstallShield Developer comes with a fully featured debugging tool—this is especially helpful when using the scripting engine of the program.

8. From my understanding, there are some branding strings that aren't included in the string table and are only available through language package purchases. These strings will appear in English when installation occurs on different language Windows operating systems.

Brands you with the InstallShield name: Again, as with all InstallShield products, the Express EULA requires that you not change or remove the InstallShield branding or copyright information. The term *branding* refers to the InstallShield name that's displayed in several dialog boxes of the installation. The marketing spin from the InstallShield folks is that customers trust the InstallShield label and that the branding makes both the developer and the customer feel more comfortable—though I bet the marketing aspects of the branding play a part in the way they write the license. Many development houses choose not to allow third-party software to brand their products. In this case, you'll need to switch to the default Visual Studio MSI integration or to one of the Wise products discussed later in this chapter.

Offers redistribution packages: A bevy of system file redistribution packages are available for inclusion in your installation by using a point-and-click interface.

Using Wise

If InstallShield owns the majority of the install market,[9] Wise is generally known as the upstart competitor. Started by a couple of guys in a garage back in 1992, the creators of Wise wanted to provide an easy-to-use alternative to InstallShield, which was then completely script based, considered relatively difficult to use, and held the Windows install monopoly. Wise began as a shareware program but became so popular that the product became an official offering in the late 90s.

At the time of this writing, Wise has several product offerings for installations:

Wise for Visual Studio .NET: Like InstallShield, Wise integrates cleanly into the Visual Studio .NET IDE. According to the marketing spin, Wise is the only third-party installation software that has partnered with Microsoft, thereby making its integration more robust and easier to use than that of the "limited Visual Studio add-in"[10] provided by its competitor.[11]

9. Neither Wise nor InstallShield officially track market share—however, representatives from both companies told me that InstallShield owns a majority of the market. From the information I've gathered both from the reps and other sources, it appears that InstallShield has about 60 percent of the Windows market and Wise about 30 percent. These numbers won't hold up in court—I think pulling out the teeth from the people-in-the-know might have been easier than getting an answer to this question. I even started to wonder if I had unearthed some strange Watergate-style installation coverup.
10. Their words—not mine.
11. InstallShield strongly disputes this claim and states that its integration has been at least as strong as Wise's since April 2002.

Wise for Windows Installer: This is the separate integrated development environment that has the same functionality as Wise for Visual Studio .NET. The only real difference between the two products is that this is a completely stand-alone product and can't take advantage of some Visual Studio features.

NOTE *Unlike the equivalent products of InstallShield, Wise Corporation doesn't provide the separate IDE and Visual Studio integration in one package at a single price point. You must buy each product separately.*

Wise Installation System: This is a continuation of the legacy Wise script-based product line that has been on the market for many years. If you're starting your installation from scratch, it's advisable that you start with either the Visual Studio .NET or Windows Installer versions of Wise.

Wise for Visual Studio .NET[12]

I'll put the cards on the table here—Wise touts that the comparative ease of use of its tools is a large differentiating factor between it and the InstallShield Developer product. This is generally true. There's no need to script—even for customized dialog boxes—and you can do almost everything via a point-and-click interface. This product, however, still adheres to one of the basic rules of software: As the power of a tool increases, so does the difficulty of its use. I personally wouldn't say that Wise for Visual Studio .NET is "easy to use." I was, however, able to put out a fairly customized installation for the Stock Suite in just a few hours.

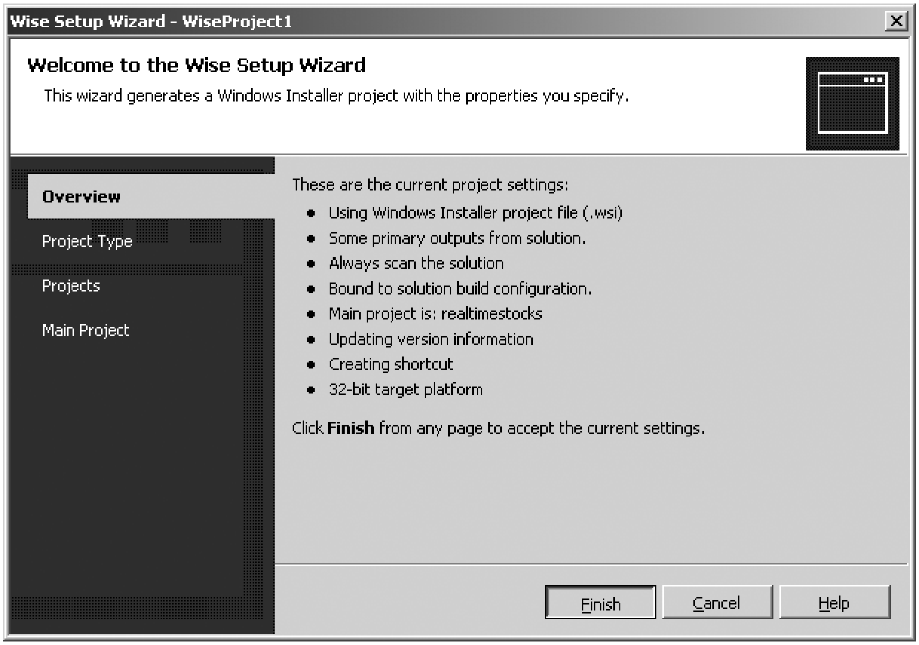
NOTE *Wise products are available for evaluation download via the Wise Web site at* `http://www.wise.com`.

There's a major difference between the two competitors: The MSI-enabled Wise products aren't procedural in the same way as the script-based InstallShield Developer. Although they do have scripting ability, the scripts are object oriented in nature and are used only at particular times—much like the MSI custom actions discussed earlier in the chapter.

12. This evaluation used Wise for Visual Studio .NET 4.2.

Figure 10-22 displays the first and only dialog box you encounter in the Wise Setup Wizard.

NOTE *As usual, the naming of the new project takes place before the wizard starts. Remember to name your project appropriately when choosing which project to create.*


Wise Setup Wizard - WiseProject1

Welcome to the Wise Setup Wizard

This wizard generates a Windows Installer project with the properties you specify.

These are the current project settings:

- Using Windows Installer project file (.wsi)
- Some primary outputs from solution.
- Always scan the solution
- Bound to solution build configuration.
- Main project is: realtimestocks
- Updating version information
- Creating shortcut
- 32-bit target platform

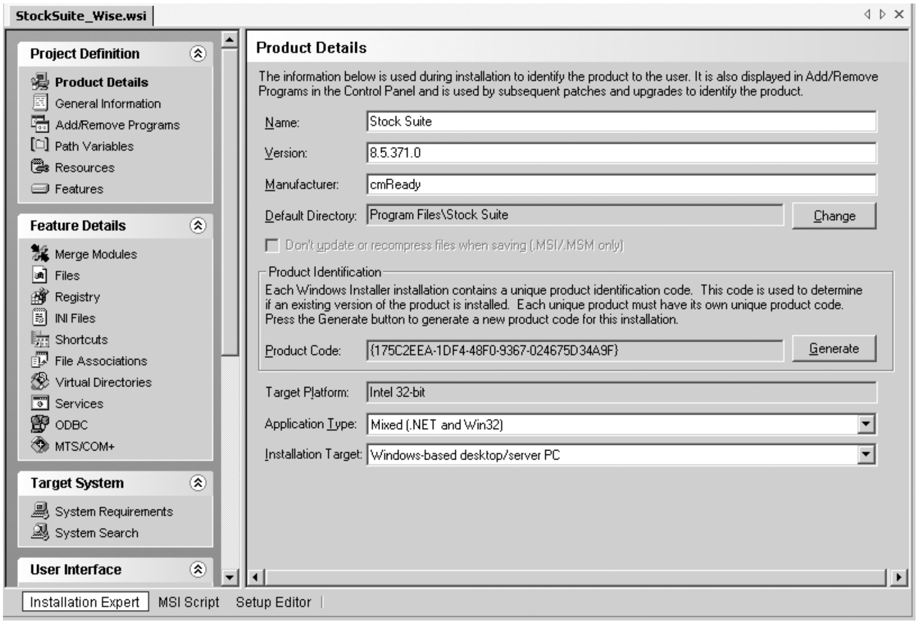
Click **Finish** from any page to accept the current settings.

Figure 10-22. Using the Wise Setup Wizard

Use the four tabs to the left of the dialog box to choose the basic options—such as which projects in the solution should be included in the installation. But that pretty much covers it for the wizard—after clicking Finish, you're thrown directly into the tool itself. This is a shame because much of the tool isn't intuitive. Wise would be wise (okay, bad joke) to include an InstallShield-like wizard that allows you to set most of an installation's basic options.

On the positive side, Wise's integration with Visual Studio is indeed impressive. Although InstallShield has small imperfections with its add-in compatibility, Wise looks like it's always been a part of the tool. From accessing the Help all the way to building setup projects, you can access almost all of Wise's functionality through the usual Visual Studio methodologies. Only its debugger appears to be a separate application.

The integration is split into three different modes. The first is the Installation Expert, as displayed in Figure 10-23. In contrast to its name, it's not the "expert mode" as you might expect. This is the beginner's step-by-step creation mode. Much like you did in InstallShield, navigate through the list of properties and set them appropriately for your installation.



Product Details

The information below is used during installation to identify the product to the user. It is also displayed in Add/Remove Programs in the Control Panel and is used by subsequent patches and upgrades to identify the product.

Name: Stock Suite
Version: 8.5.371.0
Manufacturer: cmReady
Default Directory: Program Files\Stock Suite
☐ Don't update or recompress files when saving (.MSI/.MSM only)

Product Identification
Each Windows Installer installation contains a unique product identification code. This code is used to determine if an existing version of the product is installed. Each unique product must have its own unique product code. Press the Generate button to generate a new product code for this installation.
Product Code: {175C2EEA-1DF4-48F0-9367-024675D34A9F}

Target Platform: Intel 32-bit
Application Type: Mixed (.NET and Win32)
Installation Target: Windows-based desktop/server PC

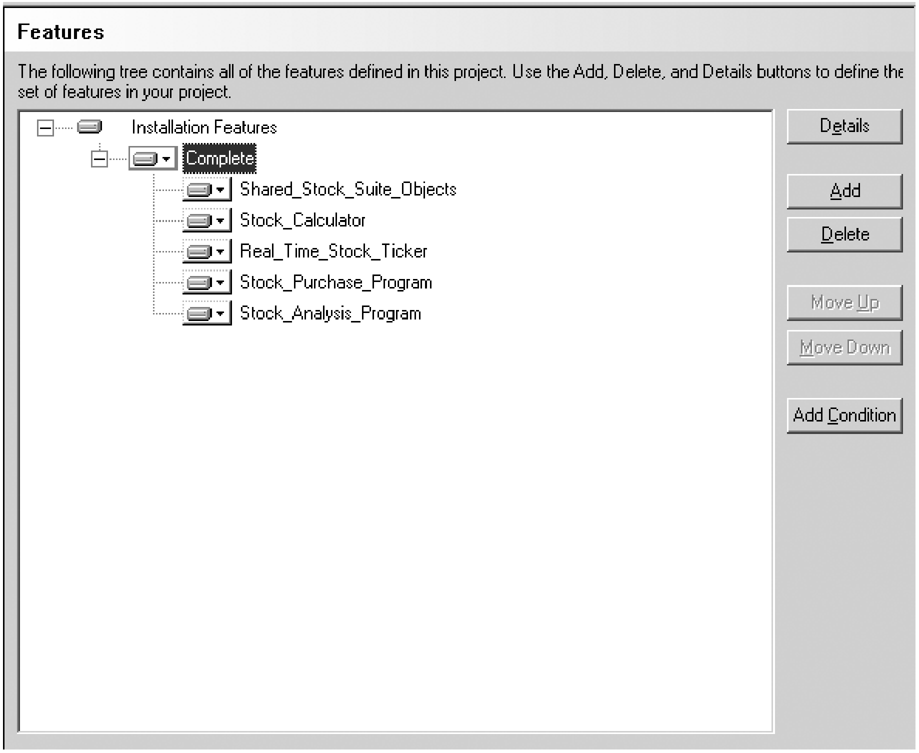
Figure 10-23. Using the Installation Expert mode of Wise for Visual Studio .NET

Change the title, manufacturer, and default directory in the Product Details screen as displayed in Figure 10-23. Note that the version is set automatically—it's tied to the default project in the solution (which in this case is the Real Time Stock Ticker).

NOTE *Setting the default directory on this screen doesn't automatically cascade the change throughout the installation. In fact, in one of my largest criticisms of this product, I was often unable to find a simple way to change properties without making the same change numerous times throughout the interface.*

After navigating to and filling in properties on the screens located in the Product Details and General Information sections of the Installation Expert, it's time to define the installable parts (hereafter known in this section as Wise *features* to be consistent with the product's user interface) of your setup. This isn't as easy as it might seem—by default, Wise automatically gathers all of the outputs created by the solution into a single feature called Complete. You have to undo this default action to split up the application properly.

Navigate to the Features screen of the Project Definition section. Using the Add button, create five separate Wise features—the shared objects, the Real Time Stock Ticker, the Stock Purchaser, the Stock Analyzer, and the Stock Calculator. Add them as children to the currently existing Complete feature—the Complete choice should still install the entire suite. Once completed, the Features screen should look similar to Figure 10-24.


Features

The following tree contains all of the features defined in this project. Use the Add, Delete, and Details buttons to define the set of features in your project.

```
Installation Features
  Complete
    Shared_Stock_Suite_Objects
    Stock_Calculator
    Real_Time_Stock_Ticker
    Stock_Purchase_Program
    Stock_Analysis_Program
```

Figure 10-24. Specifying features using Wise's Installation Expert

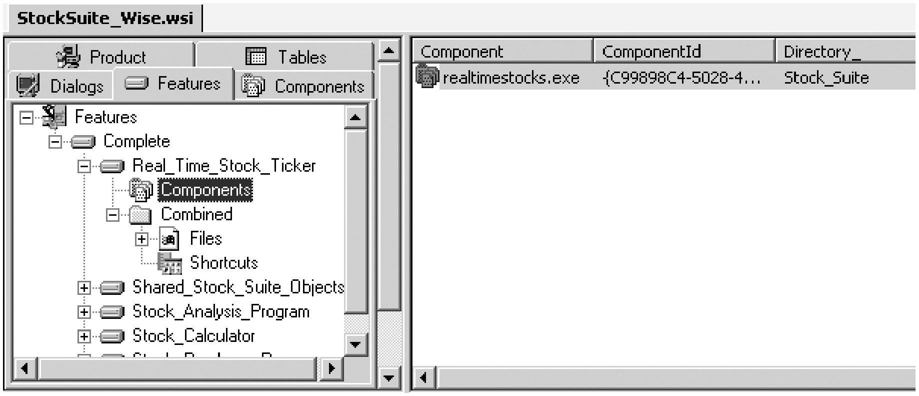
Unfortunately, there's no easy way from within the Installation Expert to set which files are applicable to which Wise features. This change affects how the rest of the install takes place, so you have to change to Setup Editor mode before continuing. The Setup Editor consists of several tabs that allow you to make changes to the installation in a more power-user-friendly manner. These tabs include the ability to turn on and off dialog boxes, create and change your Wise features, and detail how and under what conditions files should be installed.

Let's concentrate on the Features tab for a moment. This tab allows you to update properties that are out of sync because of the changes you made earlier. First, you need to move the project outputs associated with the Complete feature to the respective features you just created. Do this by navigating to the Components file folder listed directly underneath the Complete feature and dragging and dropping the objects as necessary to the new features. After moving the files, choose to view the details on the shared objects feature and click the Required Feature checkbox so that it's always installed.

NOTE *It's my opinion that installations created with Wise show installable parts/features in a more pleasing manner than either InstallShield or the default MSI integration of Visual Studio. This allows you to list the calculator as a removable feature instead of automatically including it with the Stock Analyzer and Stock Purchaser programs.*

Figure 10-25 displays how the Features tab might look after accomplishing this goal. Once your files are properly associated with features, you need to correct another problem that developed because of an earlier change. By default, Wise decided that the Program Files install directory for your product would be the same name as that of your solution: stocks. This isn't a professional title for the product, so change the default directory to *Stock Suite* on the Product Details screen of the Installation Expert's Project Definition section.

Alas! This change doesn't propagate automatically throughout your installation. All of your Wise installation features still point to the original STOCKS directory and must be changed by hand. To do this, right-click the Wise feature and choose Details. A dialog box appears where you can set a descriptive phrase that describes the Wise feature and that also gives you the ability to designate a new installation directory. Change all the directories to match the one you set earlier: Program Files\Stock Suite.



Component	ComponentId	Directory_
realtimestocks.exe	{C99898C4-5028-4...	Stock_Suite

Figure 10-25. Specifying features using Wise's Setup Editor mode

While you're in Setup Editor mode, make a few other quick changes. First, create the shortcuts that should appear for your applications after installation. To do so, right-click any Wise feature and choose New ➤ Shortcut. This creates a shortcut object beneath the feature and pops up a helper wizard. On the first dialog box of the wizard, choose the File in This Installation radio button—this allows you to specify the project outputs associated with this feature rather than a file from the file system. On the second page of the wizard, choose the executable for which you want to create a shortcut. Wise doesn't give you an opportunity to choose a "friendly" name for your shortcut in the wizard. Instead, you make that change afterward by double-clicking the newly created shortcut and changing the Name property to whatever you want.

After creating the shortcuts, it's time to improve the user interface. First, click the Dialogs tab. Much like in the Visual Studio default MSI integration, this displays a tree whose roots contain different kinds of installations—the Install, Maintenance, and Administrative sets of dialog boxes are listed. The Install set refers to the default end user installation. The Administrative set is what administrators see when installing from the network. The Maintenance set holds the user interface that appears if the product is already installed—in this case, an uninstall or repair are offered to the end user.

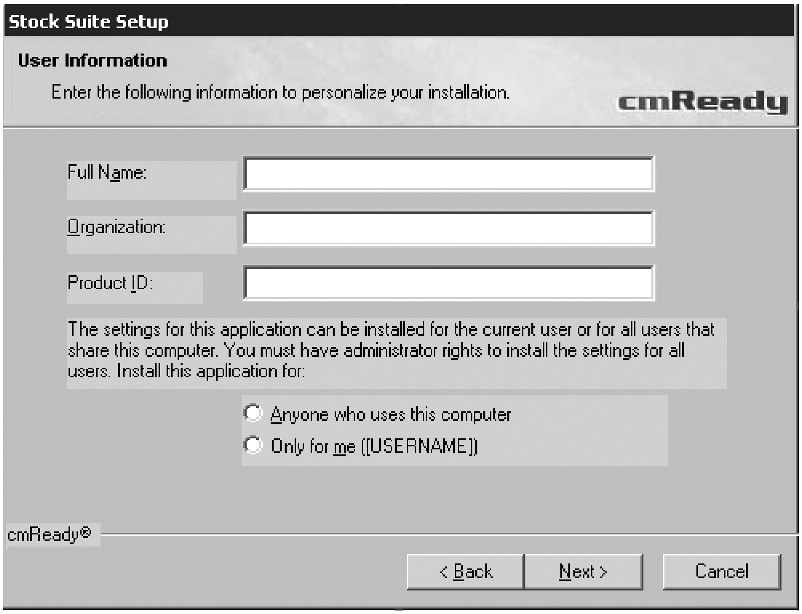
For the purposes of this example, concentrate on the Install set dialog boxes. In real life, of course, you'd make sure the dialog boxes for all three trees work properly.

After clicking Install Dialogs to expand its tree if necessary, add the Select Features dialog box. With this dialog box, you can choose which Wise Features to install. That's the only dialog box you're going to add for the time being, but because you have the user interface editor open, go ahead and personalize some of the interface dialog boxes. Double-click a dialog box—any dialog box—to make it appear in the editor. Go ahead and change the banner for the dialog box; use your company's bitmap by double-clicking it, choosing the Graphic tab, and then clicking the Set button to find the proper bitmap on your hard drive.

TIP *You can also choose to set pictures anchored on the left side of dialog boxes instead of the top if you like that look better.*

Unlike the default MSI integration in Visual Studio, you aren't required to perfectly size the bitmap—the dialog box editor stretches it to fit. You can choose only bitmaps for this task. If you have other types of images, you must convert them using a program such as Adobe Photoshop before adding them to the project.

You can customize all dialog boxes in Wise without the need for script—a significant difference from the way InstallShield Developer handles the same task. Like the example displayed in Figure 10-26, change the branding located near the bottom of the screen from Wise Corporation to your company name. Be sure to extend the line object next to it to align with the newly sized text. This particular change is small—if you make other changes, be careful that you don't accidentally lose functionality by deleting required text or choice objects.


Stock Suite Setup

User Information

Enter the following information to personalize your installation.

Full Name:

Organization:

Product ID:

The settings for this application can be installed for the current user or for all users that share this computer. You must have administrator rights to install the settings for all users. Install this application for:

- Anyone who uses this computer
- Only for me ([USERNAME])

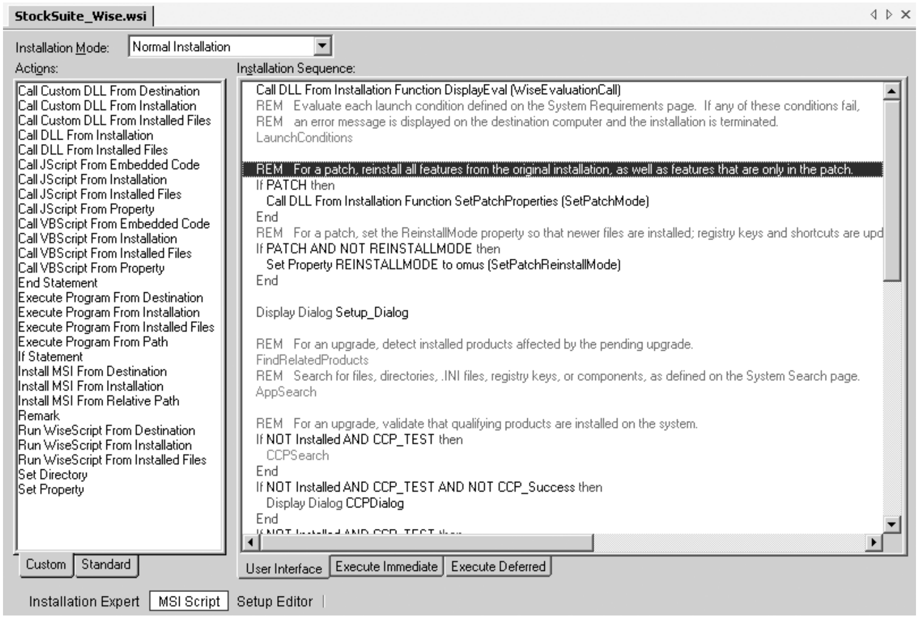
Figure 10-26. Editing the User Information screen of the Stock Suite Setup dialog box

And now for the bad news—you must make that same change on every dialog box you include in the installation. However, it's a small price to avoid another company branding the product.

TIP *Double-click graphical objects in the user interface editor to display the object's x and y coordinates, width and height, respectively. Then you can make the same movement or size change in more than one dialog box by copying and pasting values in other dialog boxes. This assures the consistency of your installation's look as the user navigates through it.*

After you've made any desired aesthetic changes to the dialog boxes, finish the user interface by opening the License Dialog box editor and then double-clicking the text box located in the center. Cut and paste your company's license information into the text box. The box supports rich text—so feel free to format it using your favorite word processor.

You've now completed customizing your installation's user interface. You could have also added custom dialog boxes as necessary to gather information from your users. Had you chosen to do so, you could've used the MSI Script mode, as displayed in Figure 10-27, to instruct Wise how to display and react to it.



```
Call Custom DLL From Destination
Call Custom DLL From Installation
Call Custom DLL From Installed Files
Call DLL From Installation
Call DLL From Installed Files
Call JScript From Embedded Code
Call JScript From Installation
Call JScript From Installed Files
Call JScript From Property
Call VBScript From Embedded Code
Call VBScript From Installation
Call VBScript From Installed Files
Call VBScript From Property
End Statement
Execute Program From Destination
Execute Program From Installation
Execute Program From Installed Files
Execute Program From Path
If Statement
Install MSI From Destination
Install MSI From Installation
Install MSI From Relative Path
Remark
Run WiseScript From Destination
Run WiseScript From Installation
Run WiseScript From Installed Files
Set Directory
Set Property
```

```
Call DLL From Installation Function DisplayEval (WiseEvaluationCall)
REM   Evaluate each launch condition defined on the System Requirements page.  If any of these conditions fail,
REM   an error message is displayed on the destination computer and the installation is terminated.
LaunchConditions

REM   For a patch, reinstall all features from the original installation, as well as features that are only in the patch.
If PATCH then
  Call DLL From Installation Function SetPatchProperties (SetPatchMode)
End
REM   For a patch, set the ReinstallMode property so that newer files are installed; registry keys and shortcuts are upd
If PATCH AND NOT REINSTALLMODE then
  Set Property REINSTALLMODE to omus (SetPatchReinstallMode)
End

Display Dialog Setup_Dialog

REM   For an upgrade, detect installed products affected by the pending upgrade.
FindRelatedProducts
REM   Search for files, directories, .INI files, registry keys, or components, as defined on the System Search page.
AppSearch

REM   For an upgrade, validate that qualifying products are installed on the system.
If NOT Installed AND CCP_TEST then
  CCPSearch
End
If NOT Installed AND CCP_TEST AND NOT CCP_Success then
  Display Dialog CCPDialog
End
```

Figure 10-27. Using MSI Script mode in Wise for Visual Studio .NET

Although still somewhat procedural in nature, MSI script isn't typical script. Instead of typing the code, double-click a command to the left of the editor to display a dialog box with options for the command. After filling and closing it, Wise adds a text statement that reflects the command in MSI script on the right side of the window. From this point forward, you must always double-click that line to bring up the command dialog box for any changes you want to make. You can copy and paste code listed in this screen to other locations within Wise but, unfortunately, not outside of the application. For those used to coding via the keyboard, this can be a bit of a culture shock.

The three tabs near the bottom of the MSI Script mode screen relate to "moments in time" in the setup process. The Execute Immediate script runs as the installation starts. The Execute Deferred script runs just before the installation finishes. The User Interface script runs as the dialog boxes display. Each default script has many conditionals based upon whether the installation is an upgrade, uninstall, or other nonstandard setup—so be careful to place commands in the proper location.

Wise also allows using Windows JScript or VBScript to carry out some custom actions. Earlier in the chapter, you created a console application to support custom actions using default MSI. You can still make calls to outside applications if you'd like to do so, but it's much easier to use an interface such as this.

Once you've completed creating the Wise installation, build and run it using the default Build and Debug commands in Visual Studio. The Wise debugger starts automatically when run in debug mode—this allows you to step through code as necessary to root out errors.

That does it for the Wise tutorial. However, I do have these parting thoughts:

Avoid reordering dialog boxes: It's possible to reorder the dialog boxes as created by the Setup Wizard; however, my technical contact at Wise advises against it—it's easy to accidentally break the default installation's flow.

Provides mulilanguage support: Five (count 'em!) languages are included by default with the Wise for Visual Studio .NET product. This includes all strings present in the default dialog boxes—just remember to translate any strings you may have created yourself. To add your own strings, switch to the Installation Expert mode, navigate to the Languages screen listed under the Release Definition section, and click the Strings button.

Remember to add the .NET Framework: The .NET Framework and MSI tool installations aren't included by default with installations created with Wise. To add them to your installation, use the Build Options screen in the Release Definition section of the Installation Expert mode. It's not possible to bundle the .NET Framework and/or the MSI Installer tool in an MSI file, so if you choose to include them in your installation, you must allow Wise to build a setup executable file. It's possible to specify which version of the Framework to include, but Wise automatically downloads the latest version for you if desired.

Watch for the binaries: The Visual Studio SourceSafe integration works fine. Keep in mind that, by default, Wise includes the project output from your other projects as members. If you choose to add the Wise install to your source, you might accidentally check in large and unnecessary binary object files.

Installations for Linux

Although they share quite a few similarities, the Linux and other Unix operating systems often differ in their methods of installing software.

The lack of a common Registry-type system database and the ability to create powerful command-line scripts mean that installations can take the form of simple text files that simply copy files to their proper location. Of course, there are so many different hardware architectures to support in the Linux/Unix world that binary files are often not included as part of deployments. Instead, "installation" packages, especially open-source products, consist of source and makefiles that the end user prepares. Of course, the lack of an uninstall or any sort of rollback capabilities leaves a lot to be desired.

Introducing RPM

Sun and some Linux vendors have taken steps to make software installation and uninstallation more efficient by introducing databases that keep track of software installed on their operating systems. I'll talk about one tool in particular—the Linux installation tool called RPM.

RPM—short for RPM Package Manager[13]—has become the standard for many Linux-flavored environments. I discuss it in this section because it has been ported to many different operating systems, including Sun Solaris and Microsoft Windows. Although this doesn't necessarily mean that the same installation packages can be used on all the operating systems, it does mean you can present your products to the end user with a common interface. Keep in mind, of course, that the RPM packager software needs to be installed on the end user's machine in order for these packages to be used. Although it's common for the packager to be installed by default on many Linux systems, it's less commonly found on Solaris and almost never on Windows.

13. In the old days, it stood for the Red Hat Package Manager. The name changed, however, as it became more of a standard Linux project.

WORD OF THE DAY

sys'tem, *n. a.*

"There and back again."—Bilbo Baggins

Yes, it's absolutely true. The more things change, the more they change for the worse and then go back to what they were before.

In the beginning, back in the old DOS days, applications and their settings were encapsulated into their own directories. Needed to move a version of Lotus 1-2-3 to a different machine? You simply zipped it up the folder and copied it via a floppy disk.

But then everything changed. Microsoft created Windows and instructed software manufacturers to keep settings in text-based INIT files—which stands for ***initialization files***—that were located in the Windows directory. The complications began. The Windows directory and subdirectories became a free for all.

All was made much worse by software manufacturers who placed (or ***replaced***) whatever files in the SYSTEM directory they fancied to support their own products. Suddenly, after installing seemingly innocuous applications, the end user's machine wouldn't restart—or the printer stopped working for no apparent reason.

The software development companies weren't 100 percent at fault. Microsoft released new versions of their system DLLs one by one with no list of dependencies or instructions. It was impossible for a developer to know whether an installation would render the system inoperable before it was run—and then you could only figure out what happened through trial and error.

"Reinstall Windows," was always the retort from technical support.

This, my friends, was known as "DLL hell." Fear it.

Things began to change once again. When Microsoft released Windows 2000, it came with a welcome new feature: The system wouldn't allow rogue applications to install files willy-nilly. In addition, the Windows installer memorized which files and what changes to the system had been installed and could completely clean applications off a system at the user's bequest.

And now comes the .NET Framework with its encapsulated assemblies. Applications keep their files and settings in a single directory just like in the good old days.

Yes, it's absolutely true. The more things change, the more they change for the worse and then go back to what they were before.

RPM allows your end users to install application files from the command line and the graphical programs provided in some operating systems. This robust tool keeps track of an application's version as well as its files for easy uninstall or upgrade. In addition, you can set up package dependencies in order to inform end users of other packages that might need to be applied before your application is installed. In many cases, installing a RPM[14] package is as easy as typing this:

```
rpm --install [package-name]
```

14. I'm using RPM version 4.1.1 on Red Hat Linux 8.0.

Once installed, an RPM package is uninstalled by typing the following:

```
rpm --erase [package-name]
```

For the full functionality of RPM, refer to its associated MAN pages or type the following at the command line:

```
rpm --help
```

Creating installation packagers with RPM may seem a little overwhelming at first. It's true that it's always a bit more challenging to work in nongraphical environments, but it's not nearly as difficult as it appears. These two simple rules apply to building RPM packages:

- You must have access to your application's code to build either source or binary RPM packages.

- When creating binary packages, you must be able to build the source for all of the hardware architectures and operating systems that you want to support.

Creating an RPM Package

It's time to create a sample RPM installer package. To follow this example, you'll need to be on a Unix-flavored operating system with the RPM package manager and the standard GNU C compilers installed. I created this example on an Intel-based machine with Red Hat 8.0 installed as the operating system.

NOTE *So far I've used uppercase letters to denote directories and filenames. Because this section concerns Linux, which is a file case-sensitive operating system, I'll leave file and directory names lowercase so that they match the examples.*

Creating the Source Tree

When creating an RPM package, the first thing you want to do is gather your product's source into a single directory and create a makefile that builds the source into an application.

NOTE *For more information regarding makefiles and building in the Linux world, see Chapter 8, "Basic Builds."*

First, create a directory called rpm-example and place the source and makefile into it. It doesn't matter where this directory lives on your drive. The project you're going to make installs two small executable files. Listings 10-4, 10-5, and 10-6 display the C code for these executables and their associated makefiles. Specifically, Listing 10-4 displays hello.c, Listing 10-5 displays welcome.c, and Listing 10-6 displays the makefile. Be sure to create them in the rpm-example directory.

Listing 10-4. hello.c

```
#include <stdio.h>

int main()
{
     printf("An RPM example!\n");
}
```

Listing 10-5. welcome.c

```
#include <stdio.h>
int main()
{
     char szName[128];
     printf( "\n\nPlease enter your name: " );
     gets( szName );
     printf("Hello %s!  Welcome to the RPM example!\n\n", szName);
}
```

Listing 10-6. makefile

```
all: build
build:
        cc -c welcome.c
        cc -c hello.c
        cc -o welcome_rpm_example welcome.o
        cc -o hello_rpm_example hello.o
install:
        cp welcome_rpm_example /usr/bin
        cp hello_rpm_example /usr/bin
clean:
        rm *.o
        rm *_example
```

NOTE *The makefile should reflect paths relative to its position to the source. In Listing 10-6, the C files have no parent path attached to them—that's because the makefile and the C files reside in the same directory. This distinction is important to the RPM compiler.*

These programs don't do much—they simply output some text to the console. They will, however, be sufficient to demonstrate how to create an RPM package. Go ahead and ensure they build by changing to the rpm-example directory and compiling the files by typing `make` at the command line.

Tarring the Source

After you've confirmed that the source has been built properly, collect the source files into a TAR file (also known as a *tar ball*). This is a single file that contains many other files—it's analogous to a Windows ZIP file without the compression. Using a TAR file, you can easily transport collections of directories and files.

Before you create the TAR file, be sure to delete the object files you created in your test build by typing `make clean` in the source directory.

NOTE *From this point forward, you may have to be logged in as root to follow these instructions. The root account has administrator privileges on Linux and other Unix-flavored operating systems. If you don't have root access on the machine for which you're building the RPM, try to create a similar directory structure in your home directory and test build the RPM there.*

For this example, you should follow basic RPM standards by placing the TAR file you created in the special directory designed for RPM source tar balls. On my Red Hat Linux 8.0 machine, this directory is called /usr/src/redhat/SOURCES, but this may differ depending on your installed operating system. Be sure to check with your operating system distributor for the proper RPM build directory.

NOTE *If you don't have rights to the SOURCES directory, it's still possible to create an RPM package. Simply emulate the directory structure in another directory. Be aware that you will not be able to test the installation of your package if you don't have the appropriate rights.*

Navigate to the parent directory of rpm-example and create the TAR file by typing the following:

```
tar cvf /usr/src/redhat/SOURCES/rpm-example.tar rpm-example
```

Don't forget the second rpm-example parameter! It's the directive that tells TAR which files to add to the new rpm-example.tar file. If your source tree had child directories, the same command would recursively add the entire tree to the TAR file.

After creating the tar ball, compress it using the GNU ZIP application:

```
gzip /usr/src/redhat/SOURCES/rpm-example.tar
```

Navigate to the /usr/src/redhat/SOURCES directory to confirm the existence of the rpm-example.tar.gz file. You may find other files in that directory—it's the standard place where source tar balls are stored—but just leave them be.

Creating the SPEC File

You've now created a makefile for your source and made sure it builds properly. You've also packaged it and readied it for the RPM build tool. Before you can build the RPM package, however, you must create a text file that tells the packager how to build the application. These files, typically known as *SPEC files*, are placed in the /usr/src/redhat/SPECS directory.

NOTE *Again, the directory on your operating system may differ. Most of these directories (for instance, the SPECS directory mentioned previously) are likely to be in the same relative position to whatever parent directory you may find on your machine. Simply substitute out the /usr/src/redhat directory with the proper directory for your operating system.*

The SPEC file consists of contextually grouped sections. The first section of the file is called the *preamble* and allows you to tell RPM general information about your package. This general information includes the following fields:

- **Summary**: A one-line description of the application.
- **Name**: The name of the package. Try to be unique with this field—otherwise, your application may interfere with others of the same name.
- **Version**: The release version number of the product.
- **Release**: The release number for the package—but not necessarily the product. For instance, if you were to release an incomplete package, you might need a new package that contains the same "release." In that case, increment this field.
- **Copyright**: Any copyright information regarding this product. For instance, GPL or ©2002 by cmReady.
- **Group**: The group that the package will belong to in a group packager. Examples might be Games, Text Editors, or Utilities. See your operating system's distribution notes for more information.

- **Source**: This is the name of the compressed TAR file containing the source for the product. It should exist in the SOURCES directory as directed earlier in this section and shouldn't be pathed.
- **URL**: The Uniform Resource Locator (URL) where the source distribution is available for download by the general public.
- **Distribution**: The operating system distribution for which the application has been created. In many cases, this can be left blank if you don't plan to be distributed by a major product line.
- **Vendor**: The name of the company or person who created the application.
- **Packager**: The name and email of the person/entity who created the package.

Create a file called rpm-example.spec in the /usr/src/redhat/SPECS directory and add the preamble text from Listing 10-7.

Listing 10-7. The Preamble for the RPM-Example SPEC File

```
#
# Example spec file RPM-Example...
#
Summary: A program that does absolutely nothing but demonstrate RPM!
Name: rpm-example
Version: 1.0
Release: 1
Copyright: GPL
Group: Applications/Sound
Source: rpm-example.tar.gz
URL: http://www.cmready.com
Distribution: Example Linux
Vendor: cmReady, Inc.
Packager: Sean Kenefick <skenefick@cmready.com>
```

The second section of the SPEC file contains a paragraph detailing the product description. Try to keep your description to five or so lines long. The text doesn't wrap automatically, so use the Return key periodically to keep lines shorter than 75 characters. From this point on in your SPEC file, you need to place section titles in the SPEC file with a preceding percentage sign (%). Listing 10-8 details the "description" section.

Listing 10-8. The Description Section for the rpm-example SPEC File

```
%description
It really does nothing except show how RPM works.  Really.  That's
all.  I don't suggest running it unless you want to see it work.
```

The prep section contains information and whatever steps might be necessary to prepare for the build of the source. You might use the prep section to clean up any object files that exist before starting a new build. For instance, you're going to build your application in the /usr/src/redhat/BUILD directory—again, the BUILD directory is a standard—so you want to clean up whatever builds you might have previously created before the source is made again.

NOTE *Yeah, yeah, yeah…you've never built your source in the BUILD directory so there are no previous builds to clean up! Absolutely true. But in the future, there could be. If you build the source into a package today but need to update the package tomorrow, today's build would still exist in the BUILD directory and would need to be removed. In a time travel scenario worthy of a* Star Trek: The Next Generation *episode, you're planning for future previous builds that will need to be removed when you're building in the "future present." Just be careful not to cause a tremor in the space-time continuum.*

The prep section allows you to use operating system commands as if you were writing a script file. Use the typical `rm` command to clan the directory. Afterward, use the `zcat` command to uncompress the source tar balls into the BUILD directory so the source is ready to be built. Listing 10-9 displays both of these steps.

Listing 10-9. The Prep Section for the rpm-example SPEC File

```
%prep
rm -rf $RPM_BUILD_DIR/rpm-example
zcat $RPM_SOURCE_DIR/rpm-example.tar.gz | tar -xvf -
```

NOTE *Notice the introduction of the $RPM_BUILD_DIR and $RPM_SOURCE_DIR variables? These are provided to you for free by RPM and don't need to be declared—hence one of the reasons why you want to use the standard RPM directories. If you're using a nonstandard directory path, you'll have to change these variables in your SPEC file. Keep in mind that this can break the RPM when it runs on an end user's machine.*

Use the build section of the SPEC file to specify the commands RPM should use to build the product. Luckily, the build steps for this product are pretty simple. As displayed in Listing 10-10, the specific steps you'll instruct RPM to follow include changing the source directory and then building the executables by calling the MAKE utility. Keep in mind that these steps are specific to this makefile and set of source—different applications may have different steps.

Listing 10-10. The Build Section for the rpm-example SPEC File

```
%build
cd $RPM_BUILD_DIR/rpm-example
make
```

You've now instructed RPM how to build the source. The next step is to instruct the tool on how to install the product on an end user's machine. As displayed in Listing 10-11, use the install section of the SPEC file to fulfill this goal. Again, you'll simply call the makefile, which has detailed install instructions in it.

Listing 10-11. The Install Section for the rpm-example SPEC File

```
%install
cd $RPM_BUILD_DIR/rpm-example
make install
```

Almost done! The last step in creating the SPEC file is to add a files section. This section tells RPM which application files should be installed on the end user's machine. RPM stores this information on the end user's machine, so it can easily uninstall or upgrade the application as necessary. Because the two executables are installed to /usr/bin, indicate such in the files section, as displayed in Listing 10-12.

Listing 10-12. The Files Section for the rpm-example SPEC File

```
%files
/usr/bin/welcome_rpm_example
/usr/bin/hello_rpm_example
```

NOTE *The files section can be a little confusing because you must specify every file by the location in which it* will *be installed and not where it's built. In addition, the installation instructions you specify in the install section must reference the file and the path. In the previous example, the makefile installs a file entitled welcome_rpm_example to the /usr/bin directory. The files section must list it exactly as the install will output it—otherwise, the file will not be copied.*

TIP *Copy all the files being "installed" to a single outputted directory by listing only the directory name. For example, typing* /usr/bin *by itself under %files in Listing 10-12 would achieve the same ends. On the other hand, if the install section of the makefile had 12 files being installed to /usr/bin, you'd want to specify only the files you want copied.*

Listing 10-13 displays the rpm-example.spec file in its entirety.

Listing 10-13. The Complete rpm-example SPEC File

```
#
# Example spec file RPM-Example...
#
Summary: A program that does absolutely nothing but demonstrate RPM!
Name: rpm-example
Version: 1.0
Release: 1
Copyright: GPL
Group: Applications/Sound
Source: rpm-example.tar.gz
URL: http://www.cmready.com
Distribution: Example Linux
Vendor: cmReady, Inc.
Packager: Sean Kenefick <skenefick@cmready.com>

%description
```

```
It really does nothing except show how RPM works.  Really.  That's
all.  I don't suggest running it unless you wish to see it work.

%prep
rm -rf $RPM_BUILD_DIR/rpm-example
zcat $RPM_SOURCE_DIR/rpm-example.tar.gz | tar -xvf -

%build
cd $RPM_BUILD_DIR/rpm-example
make

%install
cd $RPM_BUILD_DIR/rpm-example
make install

%files
/usr/bin/welcome_rpm_example
/usr/bin/hello_rpm_example
```

NOTE *SPEC files have many more properties that you can use. For more information about them and more detailed RPM instructions, refer to the RPM standards organization Web site at* `http://www.rpm.org`.

Building the RPM

Whew! The hard part is over. You've gathered the source together and told RPM how to make, package, and install it to an end user's machine. Your last task is to build the installer package. Change into the /usr/src/redhat/SPECS directory and, as displayed in Listing 10-14, use the RPM package creation utility, RPMBUILD, to build your package. Use the `-ba` parameter to specify that you want to build both binary and source packages. I'll talk about source packages at the end of this section.

Listing 10-14. The RPMBUILD Output from rpm-example.spec

```
$ cd /usr/src/redhat/SPECS
$ rpmbuild -ba rpm-example.spec
```

Listing 10-15 displays the output you might receive from the RPMBUILD command on Red Hat Linux 8.0.

Listing 10-15. The RPMBUILD Output from rpm-example.spec

```
$ rpmbuild -ba rpm-example.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.104
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ rm -rf /usr/src/redhat/BUILD/rpm-example
+ zcat /usr/src/redhat/SOURCES/rpm-example.tar.gz
+ tar -xvf -
rpm-example/
rpm-example/welcome.c
rpm-example/hello.c
rpm-example/makefile
+ exit 0
Executing(%build): /bin/sh -e /var/tmp/rpm-tmp.104
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ cd /usr/src/redhat/BUILD/rpm-example
+ make
cc -c welcome.c
cc -c hello.c
cc -o welcome_rpm_example welcome.o
welcome.o: In function `main':
welcome.o(.text+0x2e): the `gets' function is dangerous and should not be used.
cc -o hello_rpm_example hello.o
+ exit 0
Executing(%install): /bin/sh -e /var/tmp/rpm-tmp.104
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ cd /usr/src/redhat/BUILD/rpm-example
+ make install
cp welcome_rpm_example /usr/bin
cp hello_rpm_example /usr/bin
+ /usr/lib/rpm/redhat/brp-compress
+ /usr/lib/rpm/redhat/brp-strip
+ /usr/lib/rpm/redhat/brp-strip-comment-note
Processing files: rpm-example-1.0-1
Finding  Provides: /usr/lib/rpm/find-provides
Finding  Requires: /usr/lib/rpm/find-requires
PreReq: rpmlib(PayloadFilesHavePrefix) <= 4.0-1 rpmlib(CompressedFileNames) ↲
     <= 3.0.4-1
Requires(rpmlib): rpmlib(PayloadFilesHavePrefix) ↲
    <= 4.0-1 rpmlib(CompressedFileNames) ↲
    <= 3.0.4-1
Requires: libc.so.6 libc.so.6(GLIBC_2.0)
Checking for unpackaged file(s): /usr/lib/rpm/check-files %{buildroot}
Wrote: /usr/src/redhat/SRPMS/rpm-example-1.0-1.src.rpm
Wrote: /usr/src/redhat/RPMS/i386/ rpm-example-1.0-1.i386.rpm
```

You should be aware of the important notes output during the build process. The first is that RPMBUILD detected several dependencies that must exist on your end users' machines before this package can be installed. In this case, these packages are extremely common and are likely included in the original operating system distribution. If they weren't, you might distribute those packages with your own application (when licensed to do so) or note the dependency on the download page of your Web site.

Another interesting fact is that you built two RPM packages as part of this process. RPM has placed the first, called rpm-example-1.0-1.src.rpm, into the /usr/src/redhat/SRPMS directory. The second, called rpm-example-1.0-1.i386.rpm, has been placed into the /usr/src/redhat/RPMS directory. You can distribute either of these packages to customers—the end result is always that the application gets installed on their systems. The important distinction between the two files, however, is that the rpm-example-1.0-1.src.rpm package doesn't contain binary files. It contains the source necessary for RPM to build and install on the end user's machine using the SPEC file instructions. Many Linux/Unix users prefer to receive source RPMs so they can evaluate the source for security flaws and distribute it to different hardware architectures. The second file, rpm-example-1.0-1.i386.rpm, contains only binaries. Keep in mind that the binaries can only be installed on machines with the same operating system and hardware architecture as the one from which you built the package.

CAUTION *If you distribute the rpm-example-1.0-1.src.rpm file to your customers,* you distribute your source in its entirety to anyone who might want to see it. *In other words, don't distribute source RPM packages if you're trying to protect your company's intellectual property through secrecy. You can use the* `-bb` *parameter with RPMBUILD to build only binary packages.*

Once created, distribute your RPM packages in all the usual ways—including posting the package to your Web site or placing it on a distributable CD.

Summary

This chapter discussed how to create installations for both Windows and Unix/Linux-flavored operating systems.

Specifically, I spoke about the following:

- How Microsoft introduced the MSI.
- The default MSI integration in Visual Studio and how to use it with your other projects to create a robust installation.
- Third-party installation tools such as InstallShield and Wise. I also detailed the usage of their most popular products.
- The RPM tool for Linux and Unix machines.
- How to best redistribute the Microsoft .NET Framework to your customers.