Once you have created a Visual Basic application, you may want to distribute it to others. You can freely distribute any application you create with Visual Basic to anyone who uses Microsoft Windows. If you are going to distribute your application, you will need to write or use a setup program that installs your application onto a user's machine. This chapter shows you how to create a professional setup program for your application.

Note If you are using the Control Creation Edition of Visual Basic, some of the material covered in this document may not be applicable. With the full editions of Visual Basic you have the ability to create applications, ActiveX documents, and other types of components. Although some of the terminology may relate to application specific objects such as forms, in most cases the underlying concepts also apply to ActiveX control creation.

1

Contents

2

Creating a Setup Program

Once you've created an application that you would like to distribute to others, you need to create a setup program to properly install your application.

Visual Basic's Setup Wizard does most of this work for you. It can determine which files you'll need to distribute, determine where they'll need to be installed on the user's machine, enter them in the system registry, and create a Program Group and Start Menu links in Windows. You need to decide which media you want to use to distribute your application. For example, you can create disks or install your application to a network server or on the Internet.

In most cases, the Setup Wizard can handle everything you'll need for your setup program. If, however, you want to customize your setup program to meet some special need or functionality, you can use the Setup Toolkit to create a modified or enhanced setup program.

The Setup Wizard vs. The Setup Toolkit

The Setup Wizard is, essentially, a helper program that walks you through the steps necessary to create a professional setup program for your Visual Basic application.

The setup program that the wizard creates for you is compiled from the Visual Basic Setup Toolkit project, which resides in the \Setupkit\Setup1 directory of your Visual Basic installation. Like any other Visual Basic project, the forms, code, and functionality of the setup project can be modified or enhanced.

In most cases, the Setup Wizard is the best way to create a setup for your application. If, however, you want to write a setup program that has features and functionality not provided by the Setup Wizard, you can do so by modifying the Setup Toolkit project.

For More Information See "Using the Setup Toolkit" later in this chapter.

3

Caution If you determine that you need to modify the Setup Toolkit project to add some functionality not already provided, make sure that you copy the Setup Toolkit project into a new directory to serve as a backup before changing the source code in the Setup1 directory. Any modifications you make to the Setup Toolkit project will affect any subsequent setup programs created with the Setup Wizard.

Note The Setup Wizard and the Setup Toolkit create setup programs and distribution media only for Visual Basic applications. To create setup programs for other Windows–based applications, use the Setup Toolkit provided with that development product or in the Microsoft Windows SDK.

4

Setup Wizard Features

Using the Setup Wizard, you can easily create a professional setup program for your application. The Setup Wizard performs these steps, with your input:

- Creates a main Setup program (Setup1.exe). The Setup Wizard compiles the Setup Toolkit project into an executable file called Setup1.exe. You can change the name if you choose.
- Builds your application's executable file. You can optionally chose to have the Setup Wizard rebuild your project as part of the Setup Wizard process.
- Optionally creates a dependency file for a Visual Basic application or component. You can choose to create a dependency (.dep) file for your application or any component built in Visual Basic. Dependency files identify which run-time files need to be included with your application or component when they are distributed.
- Compresses files and assigns them to a disk layout. The Setup Wizard compresses all appropriate files and determines where to place them on the distribution disks, if applicable.
- Notifies you of the number of blank formatted disks needed, if applicable.
- Copies distribution files to blank formatted disks, if applicable, and prompts you to insert additional disks as needed.

- Notifies you when the master distribution media are completed.
- Optionally copies distribution files to your hard disk, which may then be copied directly to a server share for network distribution or be transferred to a compact-disc generation system.

1

The Setup Wizard also provides the following advanced features:

- Supports distribution of your application across the Internet using automatic code download from Microsoft Internet Explorer, version 3.0.
- Supports the installation of remote components using Distributed COM (sometimes called DCOM) and Remote Automation.
- Allows you to copy files to a directory structure that resembles distribution on disks: Disk1, Disk2, Disk3, etc., for distribution on a compact disc or a network.

2

For More Information See “Using the Visual Basic Setup Wizard" later in this chapter.

Creating a Setup Program: Step by Step

Whether you chose to create your setup with the Setup Wizard or by using the Setup Toolkit directly, there are certain steps that must be taken.

1. **Determine the files you need to distribute.** The Setup Wizard automatically determines this information and then generates the Setup.lst file. In addition, for each ActiveX component or control that you create in Visual Basic for use in other projects, the Setup Wizard can create a dependency (.dep) file. All run-time files that are required by any ActiveX controls you used in your application, for example, are listed in this file. The .dep file will then be used by the Setup Wizard if your current project uses the ActiveX component or control created by that project. See "Dependency Files Explained" later in this chapter for more information.
2. **Create or write your setup program (Setup1.exe).** The Setup Wizard creates the setup executable from the Setup Toolkit project. If you want to modify the default setup program, back up the Setup Toolkit project files and make changes as desired. You can rename this project as you see fit, as long as you change the Setup.lst to reflect the new name. See "Writing Your Setup Program" later in this chapter for information on customizing the Setup1.vbp project.
3. **Compress the appropriate files.** The Setup Wizard compresses and copies the appropriate files to the distribution media. Some files (Setup.exe and Setup.lst) are copied to the distribution media uncompressed. If you use the Setup Toolkit, you’ll have to do the compression manually using Compress.exe. See "Compressing the Setup Files" later in this chapter for more information.

4. **If using disks, determine the disk layout.** The Setup Wizard takes care of this automatically, determining the layout and notifying you of the number of disks needed. If you are using the Setup Toolkit, see "Determining the Layout of the Distribution Disks" later in this chapter for more information.
5. **Determine where to install the files on the user's machine.** Different types of files need to be installed to different directories on the user's machine. For example, in Windows 95, setup and program files are usually installed to \Program Files\YourAppName, and system and dependency files are copied into the Windows\System directory. You can specify other paths in the Setup Wizard if you need to override the default settings. When using the Setup Toolkit directly, you will need to manually add this information to the Setup.lst file. See "Creating the Setup.lst File" later in this chapter for more information.
6. **Modify Setup.lst to include all the files on your list.** The Setup Wizard automatically creates the Setup.lst for you, deriving this list from your application's dependencies, program files, etc. When using the Setup Toolkit directly, you need to create this list manually. See "Creating a Setup.lst File" later in this chapter for more information.
7. **Create the distribution media.** The Setup Wizard will prompt you for the type of media you want to use. You may select disks, a network server, or distribution via automatic code download from the Internet. You can also create distribution media manually. See "Creating Distribution Disks" and "Distribution Options" later in this chapter for more information.
8. **Test the setup program.** Whether using the Setup Wizard or the Setup Toolkit, you should test your application's setup prior to distribution. See "Testing Your Application's Setup" later in this chapter for more information.

Advanced Features

Additionally, you may want to add advanced features to your setup program. Many advanced features are integrated in the Setup Wizard; however, you may need to modify some aspect of these features manually. The following list identifies these features and points you to the section within this chapter that discusses them in greater detail:

- **Create Internet component download packages.** Use the Setup Wizard to create .cab files for the ActiveX components that you use in your Internet applications. See "Internet Component Download" later in this chapter for more information.
- **Install ActiveX components.** Use the Setup Wizard to create setup programs for both the client and server in a Remote Automation or Distributed COM (sometimes called DCOM) environment. See "Installing Remote Automation and Distributed COM Components" later in this chapter for more information.

- **Install Data Access Components.** If you used Data Access Objects (DAO) in your application, see “Installing Data Access Components” later in this chapter for more information on how to use the Setup Wizard to install them.
- **Distribute Your Application on a Network.** The Setup Wizard gives you two options for installing your applications on a network. See “Distribution Options” later in this chapter for more information.

4

Files You Are Allowed to Distribute

You can freely distribute any application or component that you create with Visual Basic. In addition to an executable (.exe) file, your application might require other files, such as DLLs, ActiveX controls (.ocx), or bitmaps (.bmp).

You can legally distribute sample application files (found in the \Samples subdirectory) and any files that were originally copied to the \Icons subdirectory of the main Visual Basic directory when you first installed Visual Basic version 5.0 on your system.

Microsoft makes no warranty, express or implied, regarding the merchantability or fitness of these applications, nor does Microsoft assume any obligation or liability for their use.

Professional and Enterprise Editions

If you have purchased the Professional or Enterprise Edition of Visual Basic, you can also distribute any files originally copied to the \Graphics and \ODBC subdirectories of the main Visual Basic directory when you first installed Visual Basic version 5.0 on your system.

Note You may also be able to distribute other ActiveX controls, .exe files, and .dlls that you have purchased. Consult the manufacturer's license agreement for each of the files you plan to distribute to determine whether you have the right to distribute the file with your application.

5

Using the Visual Basic Setup Wizard

The Visual Basic Setup Wizard makes it easy for you to create a setup program for your application. Like other wizards, the Setup Wizard prompts you for information so that it can create what you want for you.

In most cases, the Setup Wizard will be all you need to create a setup program.

To start the Setup Wizard

1 If the project you want to create a setup program for is open, save it and close Visual Basic.

2 Select the Setup Wizard icon from the Visual Basic 5.0 Start Menu in Windows 95 or Windows NT 4.0, or from the Visual Basic 5.0 Program Group in the Windows NT 3.51 Program Manager.

5

Each screen of the Setup Wizard will prompt you for information about your project and will let you choose which options you would like to build into your setup, how it is to be distributed, if you want the Setup Wizard to compile your application, etc.

Each of the screens also explains how they are to be used, when certain information is optional, and what information must be entered before the process can continue (before you can continue on to the next screen). As you are progressing through each screen, if you find that you need more information, press F1 or click the Help button.

Note If your application will be run on a BiDi operating system, you will need to manually include the Vba332me.dll in the Setup.lst created by the Setup Wizard. You can do this by adding the file at the File Summary step when running the Setup Wizard, by editing the Setup.lst directly, or by adding an entry for Vba332me.dll to the Vb5dep.ini file so that it will be automatically added to the Setup.lst whenever you run the Setup Wizard.

6

For More Information See "Setup Wizard Features" earlier in this chapter.

Dependency Files Explained

A dependency (.dep) file contains information about the run-time requirements of an application or component: which files are needed, how they are to be registered, and where on the user's machine they should be installed. You can create .dep files for standard Visual Basic projects or — in the Professional and Enterprise editions of Visual Basic — for ActiveX controls, ActiveX documents, and other Active X components.

The Anatomy of Dependency

Visual Basic contains three sources of dependency information:

- Component .dep files
- Project .dep files
- The Vb5dep.ini file

6

Component Dependency Files

Soon every Visual Basic component will be accompanied by its own .dep file. This applies to controls and other components you purchase, and those you create yourself in Visual Basic. For example, all of the ActiveX controls shipped with Visual Basic have a companion .dep file. These .dep files list all of the dependent files used by the control, plus version and registry information. The following example is a portion of a typical control .dep file:

```
[RichTx32.ocx]
```

```
Dest=$(WinSysPath)
Register=$(DLLSelfRegister)
Version=a.b.c.d
Uses1=RichEd32.dll
Uses2=
CABFileName=RichTx32.cab
CABINFFile=RichTx32.inf
```


These .dep files are used to provide dependency information about the control or other component. When a component is used in another project, this information is incorporated into the project's dependency information. In other words, dependency information in the smaller units provides dependency information for the larger units.

It is strongly recommended that you generate a .dep file for any component that you create in Visual Basic. Creating a .dep file for ActiveX controls, ActiveX document .DLLs, and other Active X components is especially useful when your object is being used as a dependency in another project.

You can generate a .dep file for your component by selecting the Generate Dependency File Only option when you run the Setup Wizard.

Project Dependency Files

When you create a setup program for your application using the Setup Wizard, the .dep files for any components used in your project are incorporated into the project Setup.lst file. The Setup Wizard also derives dependency information about your application from references in your project and from the Vb5dep.ini file. The project Setup.lst file contains dependency information about every element in your project.

If need be, you can also create a dependency file for your application. The .dep files for any components used in your project are incorporated into the project .dep file.

You can generate a .dep file for your application by selecting the Create a Setup Program and Generate Dependency File options when you run the Setup Wizard.

The Vb5dep.ini File

The Vb5dep.ini file is a replacement for the Swdepend.ini file used in previous versions of Visual Basic, and its essential purpose is the same: to provide the Setup Wizard with an all-purpose list of dependencies and references used by Visual Basic. This list is created when you install Visual Basic and resides in the \Windows directory.

The following example is a portion of a typical vb5dep.ini:

```
[DAO350.dll]
Dest=$(MSDAOPath)
Uses1=MSJtEr35.dll
Uses2=MSJInt35.dll
Register=$(DLLSelfRegister)
CABFileName=MSDAO350.cab
CABINFFile=MSDAO350.inf
```

8

As you can see, the vb5dep.ini is very similar to the control .dep file example shown above. When you run the Setup Wizard, it will first check the Vb5dep.ini file to locate dependency information. If it is not available, and if a particular component does not have its own .dep file, the Setup Wizard will notify you of the missing dependency and allow you to either ignore this omission or correct the problem.

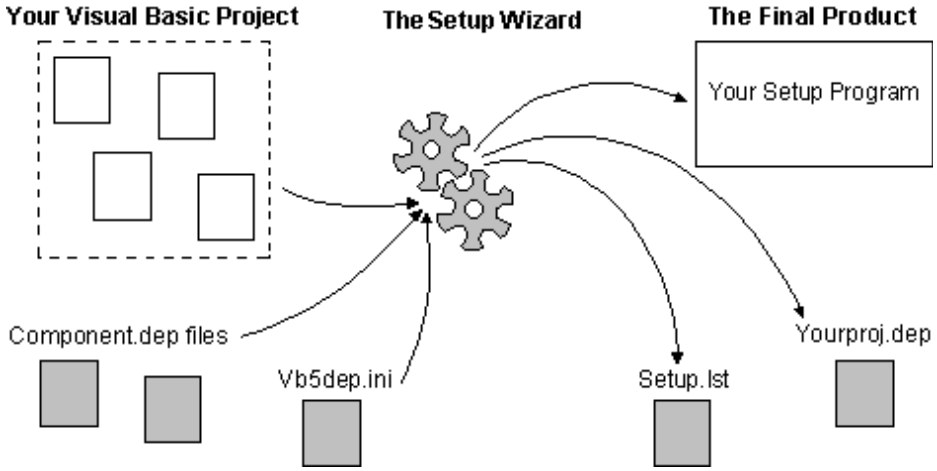
If you ignore the omission, your program may not function properly once installed on the user's machine. If, however, you're aware that a certain dependency is already loaded on the user's machine, you may safely proceed.

You may also add dependency information for a component by editing the vb5dep.ini file (manually adding an entry for a particular component), by creating a .dep file for the component with the Setup Wizard, or by contacting the component vendor and requesting a .dep file.

It may be helpful to think of .dep files as providing a list of dependencies for individual components or applications, and the Vb5dep.ini file as providing a list of dependencies for your entire Visual Basic development environment.

The following illustration shows how the Setup Wizard combines component .dep files with the Visual Basic project and the Vb5dep.ini file to create a setup program, a Setup.lst, and a project .dep file.

Figure 17.1 Creating a setup program, Setup.lst, and project dependency file 7



9

Generating Dependency Files with the Setup Wizard

You can use the Setup Wizard to create a .dep file when you're creating the setup program for your application or use the wizard to simply create dependency information for a component or for the project (if you need the Setup.lst for your

custom setup program). The dependency file is named after your application or component and is located in the same directory.

Note Even if you chose to create a custom setup program, you should use the Setup Wizard to generate a .dep file. This can then be used to help you develop your Setup.lst and distribute your application.

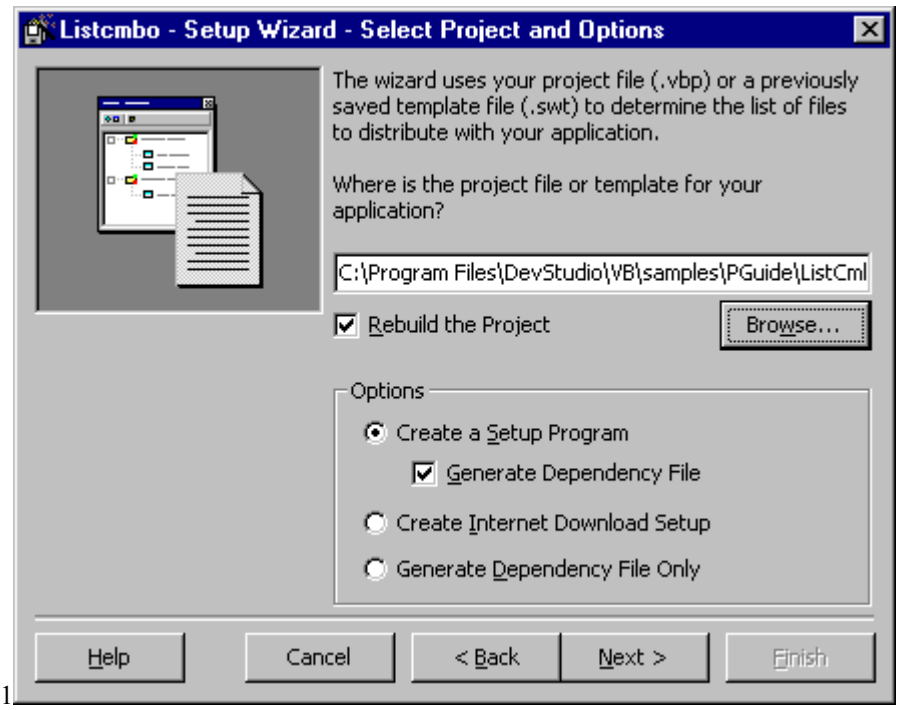
10

To create a dependency file

3 Start the Setup Wizard.

4 Click the **Next** button to select the **Select Project and Options** screen.

1Figure 17.2 The Select Project and Options screen



1

8

5 Choose your Visual Basic project by typing the path and file name into the **Project File** input box or by selecting the **Browse** button.

6 Select either the **Generate Dependency File Only** option to create a .dep file only or the **Create a Setup Program/Generate Dependency File** to create a Setup program and a project .dep file. Click the **Next** button.

1If you want the Setup Wizard to rebuild your project, check the **Rebuild the Project** option. If not, uncheck the option before proceeding.

7 If you are missing dependency information for any component in your project, you will be notified with the **Missing Dependencies** screen.

—9

2If you know that certain components are already installed on the user's machine, or that a listed file does not have dependency files, you may click that file to proceed.

3You may also proceed without confirming the missing dependencies, although your project may not function correctly.

1**Note** If you are missing dependency information for a component that you created in Visual Basic (or the Setup Wizard informs you that this information is out of date), you should first use the Setup Wizard to create a .dep file for that component (using these steps but selecting the .vbp file for the component project), and then restart this process for the current project.

9

4When you've finished resolving the missing dependencies, click **Next** to proceed.

8 The Setup Wizard will then prompt you with the standard file confirmation screens. Step through each screen until you arrive at the **Finished!** screen. Click the **Finish** button.

5A dependency (.dep) file is created with the same name as your project, in the project's source directory.

10

Distribution Options

You can distribute your application applications on floppy disks, compact discs, or on a network. If you're going to be using floppy disks, you can choose either Setup Wizard's Floppy Disk or Single Directory method. If you plan to distribute on a compact disc or on a network, you can choose either the Single Directory or the Disk Directories method.

If you have the Professional or Enterprise Edition, you can distribute your components over the Internet. For more information, see "Internet Component Download" later in this chapter. "Installing Remote Automation and Distributed COM Components" provides specifics about working with Remote Automation and DCOM, which are available in the Enterprise Edition.

11

Figure 17.3 Choosing distribution options with the Setup Wizard

11

If you plan to distribute your application on floppy disks, you can use either the Floppy Disk option to make floppy disks on your computer, or you can create floppy disk images that can be used by a floppy disk duplication service using the Disk Directories option.

If you plan to distribute your application on a network or a compact disc, you also have two options. You can use the Single Directory option or the Disk Directories option. Both methods copy the setup files to a temporary directory on your computer or a network server. You can then either locate the setup files on the appropriate network server or transfer them to a compact disc.

Floppy Disks

To create floppy disks for your application's setup when you don't need disk images, select the Floppy Disk option from the Distribution Method screen of the Setup Wizard. This screen is available when you run the Setup Wizard and select the Create a Setup Program option.

The Setup Wizard will prompt you for the type of floppy disk (1.44, 1.2, 720, or 360) and the appropriate floppy drive on your computer. As with most other Setup Wizard functions, you will then be asked to confirm the project's files and dependencies. Once confirmed, the Setup Wizard will determine the layout of the disks, compress

the appropriate files, and then copy them to the disk drive, prompting you to insert and remove floppy disks as necessary.

You can also use the Disk Directories option to create disk images of your setup program. The Setup Wizard performs essentially the same actions as for the Floppy disk option, except that the images are copied to separate directories in a temporary directory on your computer or a network server rather than to the floppy disk drive. You can then manually copy the files to disks, put the files on a network server and allow users to copy them to disks (providing you're working in a networked environment), or provide them to a floppy-disk duplication service.

Single Directory

You can create a single directory installation by choosing the Single Directory option. You can use this option or the Disk Directories option when you want to distribute your application on a network or a compact disc. This technique simply copies all of the installation files into a single directory.

When you choose this option, the Setup Wizard will ask you to confirm the project's files and dependencies, and then will compress and copy the appropriate files to a temporary directory on your computer or a network server. By default, the files are copied to C:\Windows\Temp\SwSetup.

You can then either put the files on an appropriate network server or transfer them to a compact disc.

Disk Directories

The Disk Directories option copies the files to a directory structure that resembles distribution on disks: Disk1, Disk2, Disk3, etc.

As with the Single Directory option, when you choose this option, the Setup Wizard will ask you to confirm the project's files and dependencies, and then compress and copy the appropriate files to a temporary directory on your computer or a network server. By default, the files are copied to C:\Windows\Temp\SwSetup.

You can put these directories (Disk1, Disk2, etc.) on a network server or transfer them to a compact disc for distribution. The user can install the application by simply opening the Disk1 folder and double-clicking the setup program.

Working with Setup Wizard Templates

If you have an application that you need to create a Setup program for more than once, you can choose to save the steps you took through the Setup Wizard in a template (.swt) file. (In previous versions of Visual Basic, the Setup Wizard template file extension was .vbz.)

To create a template for your Setup program, work through the steps of the Setup Wizard to create your setup program, and then select the Save Template option on the Finished! screen before selecting the Finish button.

The next time you need to create a setup for your program, you can load the template (.swt) file instead of the project .vbp file into the Select Project and Options screen of the Setup Wizard and then immediately click the Finish button instead of stepping through each of the Setup Wizard screens.

Note Because the Setup Wizard in this version of Visual Basic has been greatly enhanced, the Setup Wizard templates (.vbz files) from previous versions of Visual Basic are not compatible.

12

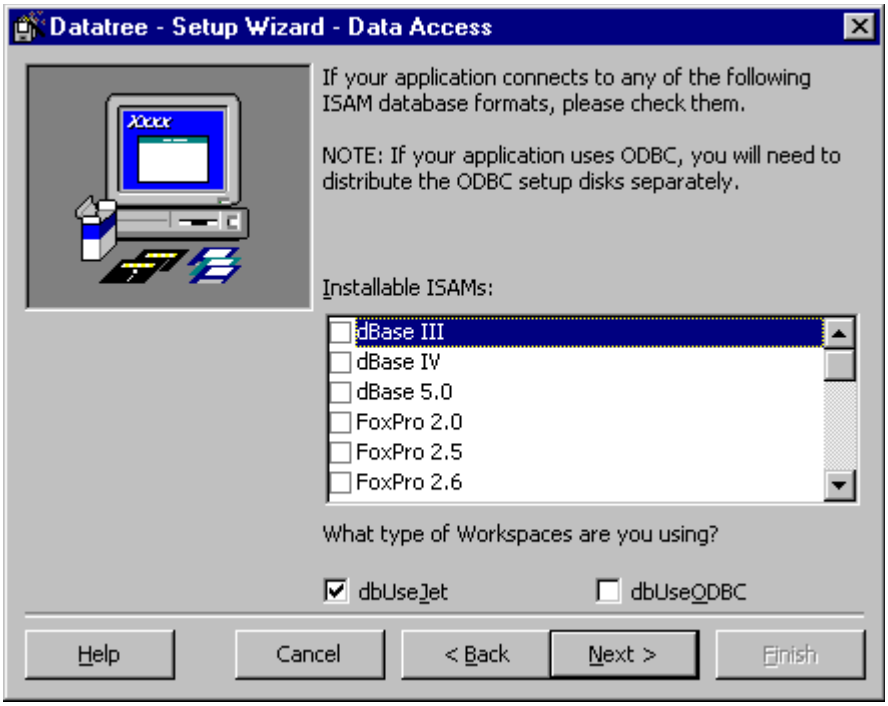
Installing Data Access Components

If your application uses Data Access Objects (DAO), the Setup Wizard will prompt you to choose the appropriate ISAM and workspace components.

You may choose one or more ISAM database formats. You must select at least one workspace; the Setup Wizard will not allow you to deselect both workspaces options — you must select one or both.

If your application requires ODBC drivers, you must install them separately. See below for more information.

Figure 17.4 Data access options



12

—13

Creating a Setup Disk for ODBC Applications

For the Professional and Enterprise editions, if you create an application that uses ODBC and you want to distribute that application, you must create an ODBC Setup Disk. Before you install your Visual Basic application on the user's machine (using a setup program you have written), you must install ODBC on the user's machine.

Note If your application uses Remote Data Objects (RDO), you should instruct the user to install the ODBC drivers before installing your application. If the ODBC drivers are not installed first, the RDO components will not register.

13

To create an ODBC setup disk

9 Copy all the files in \Odbc subdirectory of the main Visual Basic directory to a disk or network directory.

10 Before installing your application on the user's machine, run Setup.exe on the ODBC setup disk or network directory. Either use the Shell function to run ODBC Setup.exe from your setup program, or label the ODBC setup disk with a title such as "ODBC Setup Program: Run Setup.exe before installing the application."

13

Once you have placed the ODBC files on a disk, test the ODBC setup disk on a machine that does not have ODBC files.

Note To install ODBC on a user's machine, you must use the setup program provided in the \Odbc subdirectory of the main Visual Basic directory. Some of the ODBC files are compressed, and the setup program must decompress them to install them correctly.

14

Installing Remote Automation and Distributed COM Components

If your application utilizes ActiveX code components (formerly called OLE servers), the Setup Wizard provides you with the means to create setup programs for both the client and server in a Remote Automation environment or Distributed COM (sometimes called DCOM) environment, with the Enterprise Edition.

Remote Automation and Distributed COM are discussed in detail in *Building Client/Server Applications with Visual Basic,* available in the Enterprise Edition. This example illustrates how to create setup programs for the server and the client application that uses it, and presents the options for both Remote Automation and DCOM.

To create a setup program for the server

11 Start the Setup Wizard.

12 Click the **Next** button to select the **Select Project and Options** screen.

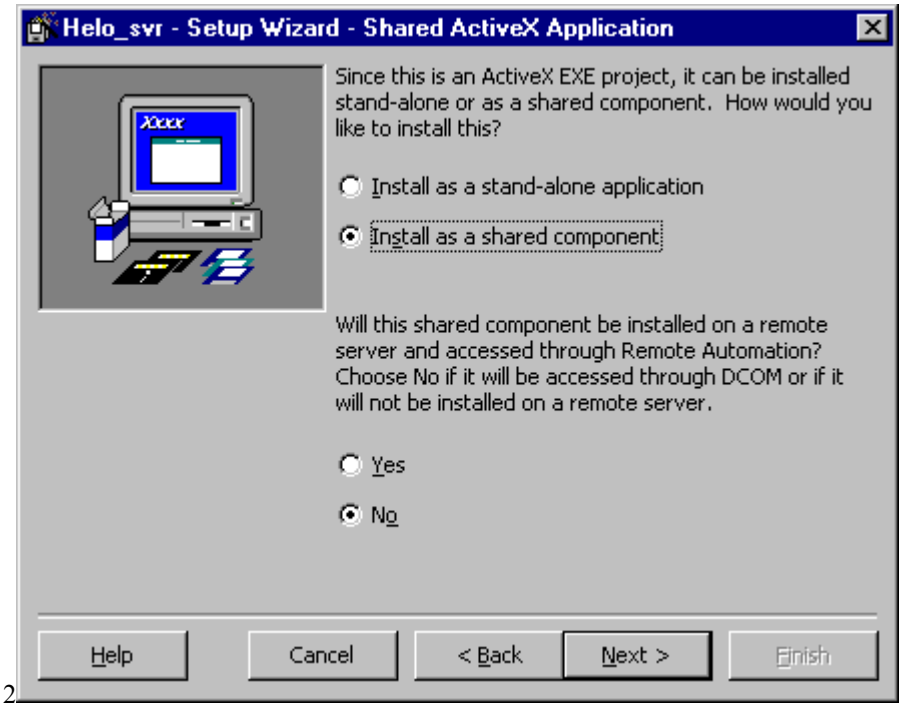
13 Choose the server project by typing the path and file name into the **Project File** input box or by selecting the **Browse** button. In this example, the project is named Helo_svr.vbp.

14 Select the **Create a Setup Program** option and then click the **Next** button to proceed.

6If you want the Setup Wizard to rebuild your project, check the **Rebuild the Project** option. If not, uncheck the option before proceeding.

15 Proceed through the Setup Wizard to the **Shared ActiveX Applications** screen.

2Figure 17.5 Selecting server application options



2

14

7Select **Install as a shared component** and then **Yes** if you are using Remote Automation or **No** if you are using Distributed COM.

8If you select the Remote Automation option, the remote automation files Racmgr32.exe, Racreg32.dll, Autmgr32.exe, and Autprx32.dll will be added to the Setup.lst and installed with the server application into the \Windows\System directory of the remote server.

16 Proceed through the Setup Wizard to create the setup program for the server.

15

The server application is installed on the remote server computer. The setup program for the server application will install the application and the Remote Automation files. *See Building Client/Server Applications with Visual Basic* for more information about configuring the Remote Automation server.

To create a setup program for the client

17 Before proceeding, make sure that you have created a .vbr file for every remote server you reference in your application. Create a .vbr file by selecting the **Remote Server** option in the **Component** tab of the **Project Properties** dialog box in Visual Basic, and then compiling the project.

18 Start the Setup Wizard.

19 Click the **Next** button to select the **Select Project and Options** screen.

20 Choose the client project by typing the path and file name into the **Project File** input box or by selecting the **Browse** button. In this example, the project is named Helo_cli.vbp.

21 Select the **Create a Setup Program** option and then click the Next button to proceed.

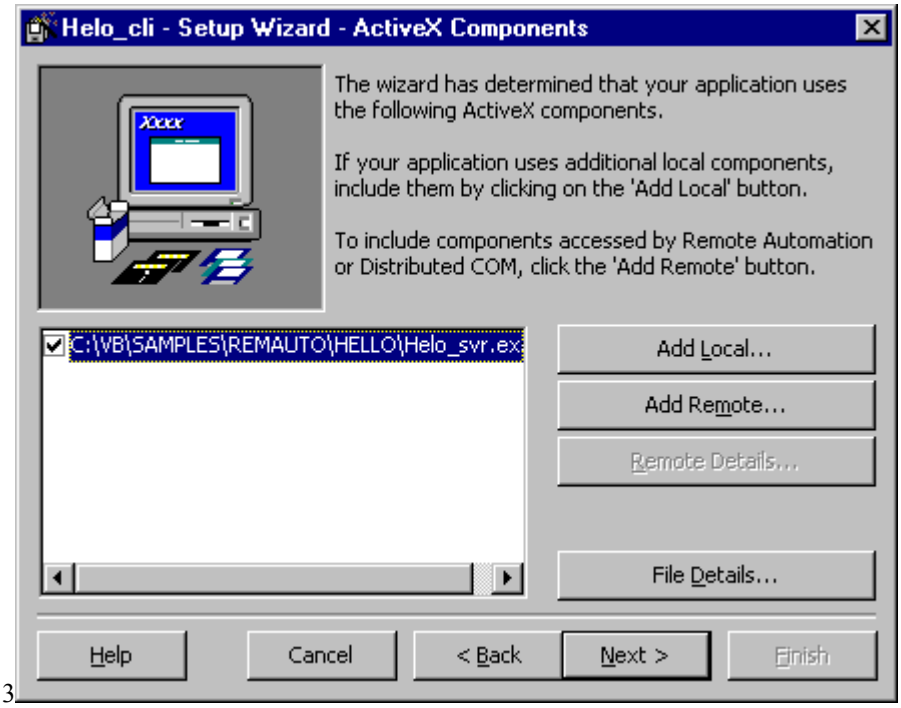
9If you want the Setup Wizard to rebuild your project, check the **Rebuild the Project** option. If not, uncheck the option before proceeding.

2**Note** Before you create a setup program for the client, you must have compiled the application at least once in Visual Basic.

22 Select the distribution media options and location and then proceed to the **ActiveX Components** screen.

10If the client project contains a reference to a local server, the server application will be listed in the file box.

3Figure 17.6 The ActiveX Components screen



3

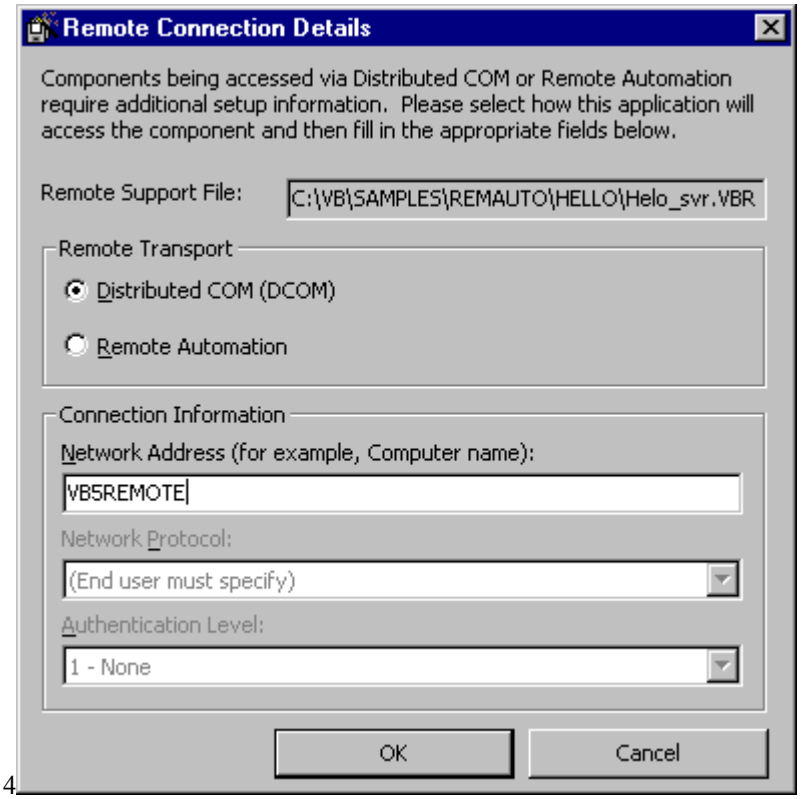
16

11If you want to add a remote server, click the **Add Remote** button.

23 You will be prompted for the .vbr file of the server application. Locate the file in your server project's source directory and choose **Open**. The .vbr file will be added to file list, and the **Remote Connection Details** dialog box will prompt you for the connection information.

—17

4Figure 17.7 Specifying remote transport and connection information



4 17

12If you are using Distributed COM, chose it from the **Remote Transport** options and then type the computer name of the remote server into the **Network Address** input box.

13If you are using Remote Automation, select it and then either specify the network address, network protocol, and authentication level or leave them blank to prompt the client user for this information at run time.

14Select the appropriate options and then click the **OK** button to return to the **ActiveX Components** screen. If both the local (.exe) and remote (.vbr) version of the remote server component appear in the components list, deselect the local (.exe) version so that only the remote server is used by your application when installed.

24 Proceed through the Setup Wizard to create the client application setup.

18

Install the client application on the client user's machine and follow the Remote Automation or Distributed COM procedures for using the client and server applications in unison.

Internet Component Download

With the Professional and Enterprise versions of Visual Basic, you can create ActiveX controls, ActiveX DLLs, ActiveX EXEs, and ActiveX documents for use on Internet Web pages. For example, you might create an ActiveX control that is used in a Web page hosted in Internet Explorer version 3.0. The Setup Wizard provides you with a means to package these components for distribution over the Internet.

How Internet Component Download Works

Without going into great detail, it may be helpful to outline here how ActiveX components are used on the Internet. A detailed explanation of the steps involved in creating Internet applications is available in *Building Internet Applications.*

Visual Basic allows you to create ActiveX controls, ActiveX DLLs, ActiveX EXEs, and ActiveX documents that can be used within an Internet browser, such as Internet Explorer 3.0. For example, you might create a control that prompts a user for a certain type of information and then processes that information in some manner. To use this control in Internet Explorer, you refer to it in the underlying HTML code, perhaps using VBScript. In other words, your control is hosted on a Web page.

The Web page, along with your component and any other dependent files, resides at a specific location on the World Wide Web. When the user accesses this Web page, your control is activated. That is, it is downloaded (in the form of a compressed .cab file) along with the Web page to the user's computer. The control is then verified for safety, decompressed, registered in the Windows registry, installed, and *then* activated.

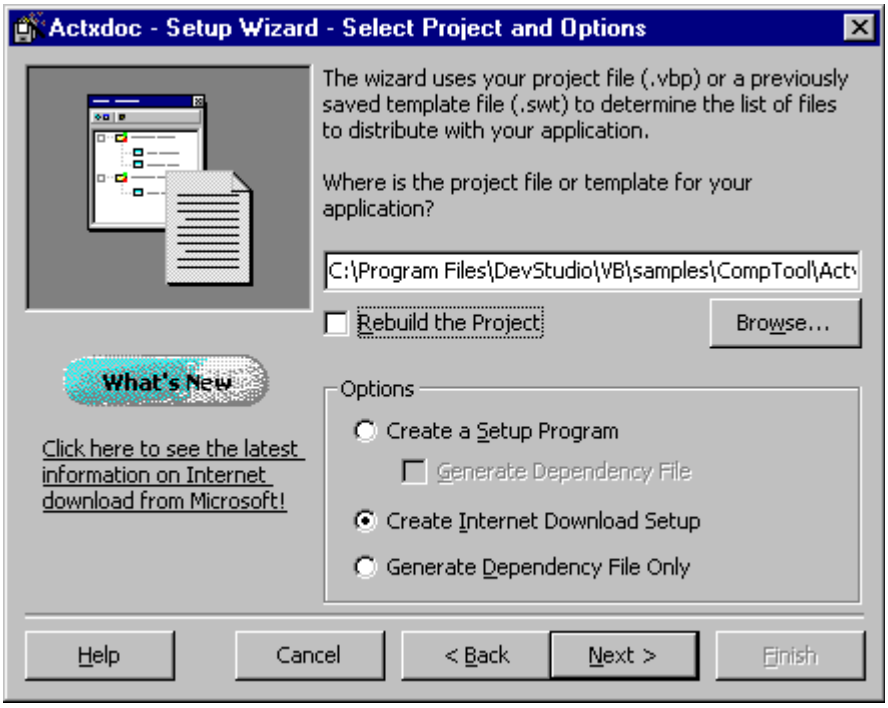
All of this occurs in the background and is controlled by the Internet browser application. Since this form of distribution differs from the more common method of application distribution via disks or compact discs, there are new issues to be considered. Most important of these is safety, both for the user and for your source code. These issues are discussed in *Building Internet Applications*.

The purpose of the Setup Wizard in all this is to package your ActiveX component. Setup Wizard determines which files your project needs at run time, gathers those files, compresses them into a .cab file, and generates the sample HTML code that points to your control.

Packaging ActiveX Components

You create a Internet package by choosing Create Internet Download Setup from the Select Project and Options screen of the Setup Wizard.

Figure 17.8 Selecting the Create Internet Download option



19

The Setup Wizard will create a .cab file, known as the primary .cab file, which contains the following:

- The project components, such as the ActiveX control, ActiveX DLL, or ActiveX EXE.
- The .inf file, which contains links to other .cab files that contain Visual Basic support files and controls, as well as other information, such as whether the control is safe for scripting and safe for initialization and registry information as defined by the user. This file replaces the Setup.lst file that the Setup Wizard creates in the standard setup.
- Reserved space for digital signatures.
- All files that are not in other (secondary) .cab files.

20

Secondary .Cab Files

For ActiveX control projects, ActiveX EXEs, and ActiveX DLLs, all run-time components — such as Msvbvm50.dll, individual controls, Data Access Objects (DAO), and Remote Data Objects (RDO) — are packaged into separate .cab files, digitally signed by Microsoft, and placed on the Microsoft Web site. You can choose to link your files to the .cab files on the Microsoft Web site, or you can download local copies of them.

—20

The benefit of using secondary .cab files from an Web site are:

- You do not need to distribute all of the .cab files required by your application. The only file you need to distribute is the primary .cab file.

  15The .inf file within the primary .cab file points to the Microsoft Web site and downloads the necessary .cab files based on the needs of the end user.

- They provide an efficient means of delivering updates to your product.

21

Note If you cannot or do not want your application setup to require a connection to the Internet, you may place the secondary .cab files on a server within your intranet. An intranet allows for faster downloading while allowing users to remain on a secured network.

15

Safety Issues

If your components are safe for scripting and safe for initialization on a Web page, you can have the Setup Wizard mark them as such. It also reserves space within the .cab file for digitally signing your component.

For More Information See "Packaging ActiveX Component for the Web," "Digital Signing For Internet Distribution," "Component Safety for Internet Distribution," and "Run-Time Licensing for Internet Distribution" in Building Internet Applications for detailed explanations of these issues.

16

Creating an Internet Download Package

Use the following procedure to create an Internet download package for your ActiveX control, ActiveX DLL, ActiveX EXE, and ActiveX document projects.

To create an Internet download package

25 Start the Setup Wizard.

26 Click the **Next** button to select the **Select Project and Options** screen.

27 Choose your Visual Basic ActiveX control, ActiveX DLL, ActiveX EXE, or ActiveX document project by typing the path and file name into the **Project File** input box or by selecting the **Browse** button.

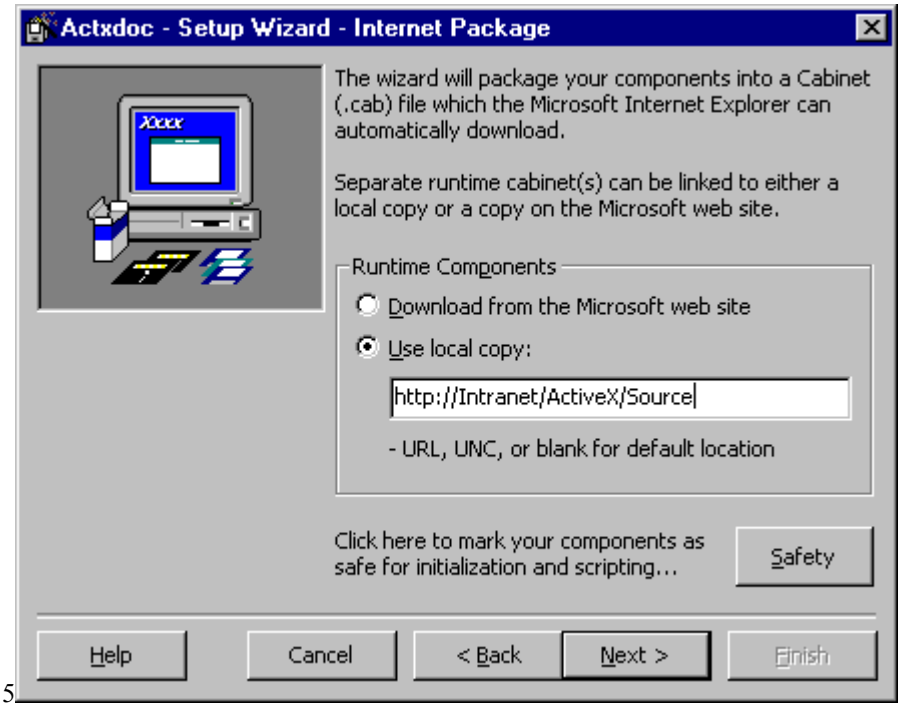
28 Select the **Create Internet Download Setup** and then click the **Next** button to proceed to the **Internet Distribution Location** screen.

16If you want the Setup Wizard to rebuild your project, check the **Rebuild the Project** option. If not, uncheck the option before proceeding.

29 You will be prompted for an Internet distribution location. This should be a temporary location on your machine — later you can upload the relevant files to your Web server. The default is C:\Windows\Temp\SwSetup. Select the location, then click **Next** to proceed.

30 You will then be prompted for download information for the components in your project.

5Figure 17.9 Specifying download information for secondary .cab files



5

22

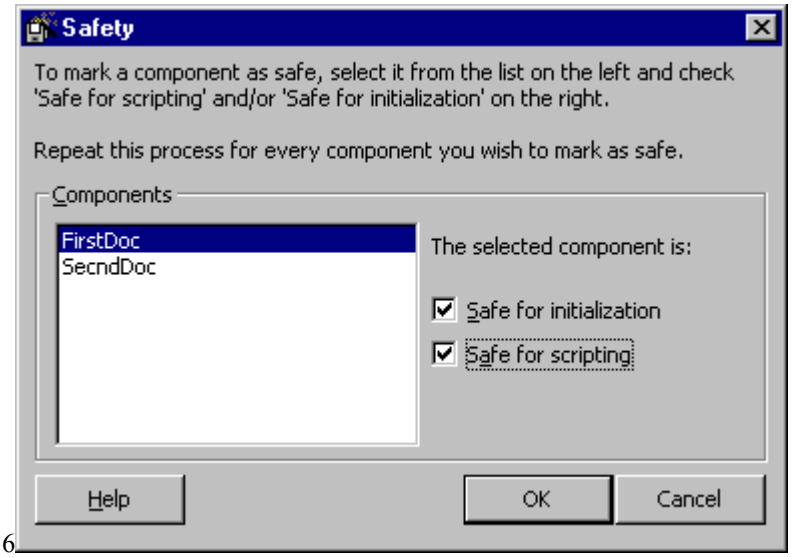
17If you want to link to secondary .cab files for the Microsoft controls or components used in your project, select the **Download from the Microsoft web site** option. All of the ActiveX controls that ship with Visual Basic contain Internet information in their companion dependency (.dep) files. The Setup Wizard writes the appropriate information into the .inf file in the .cab file it creates.

18If you want to use local copies of secondary .cab files, select the **Use Local Copy** option. You then enter a URL or UNC path to where the Internet setup will be able to find these .cab files. Or, you can leave the path blank if the .cab files will be in the same directory as the primary .cab file that the Setup Wizard creates, when it is uploaded to the Web server. Blank is the Default.

19Select the appropriate option.

31 You can now mark your components as safe for initialization or scripting.

6Figure 17.10 Marking components for safety



6

23

20When you mark your components as safe for initialization and scripting, you are guaranteeing that your component can *never* cause undesirable behavior on an end user's machine, even if used in a Web page that you yourself did not author. See "Component Safety for Internet Distribution" in *Building Internet Applications* for more information.

32 You will continue to step through the Setup Wizard to confirm files and dependencies. When you reach the **Finished!** screen, click **Finish** to create the Internet download package.

24

Distributing the Internet Setup Package

The Setup Wizard creates a .cab file that contains your component, an .inf file, and space for digital signing. It also creates a sample .htm file which you can use to refer to your component from the Web page.

In the above example, we created an Internet download package for an ActiveX document. The following files were created:

File	Description
Actxdoc.cab	The.cab file which contains: the ActiveX document (.dll) and the .inf file.
Actxdoc.htm	An HTML file containing sample HTML code.
FirstDoc.vbd	The state file for the FirstDoc object in the ActiveX document.
SecndDoc.vbd	The state file for the SecndDoc object in the ActiveX document.

17

You distribute the .cab and .vbd files along with the Web page on the Web site. The .htm file contains sample HTML code which points the browser to the ActiveX document objects. You can cut and paste sections of the sample code into the underlying HTML code of your Web page to instantiate either of the ActiveX document components.

For More Information See "Progressive Download for Internet Setup" in *Building Internet Applications* for more information on referring to ActiveX components in HTML code.

18

Rebuilding the .CAB File

The Setup Wizard also creates a subdirectory called Support which contains all of the files that were compressed into the .cab file. Again drawing from our previous example, the following files would be found in the Support directory:

File	Description
Actxdoc.dll	The compiled ActiveX document.
Actxdoc.ddf	The .ddf file is used by the compression program.
Actxdoc.inf	The .inf file which contains all dependency, version, setup, and safety information for the component.

19

If need be, you can use these files to rebuild the .cab file (for instance, if you need for any reason to make customized changes to the .inf file). The .cab files are compressed using a utility program called Diamond.exe, which is located in the \SetupKit\Kitfil32 subdirectory of the main Visual Basic directory.

Assuming that the location of the Diamond.exe is in your system's path, you can rebuild the .cab file at the command prompt using this syntax:

Diamond.exe /f <*filename*>.ddf

20

The .ddf file contains compression information used by the Diamond.exe program.

Using the Setup Toolkit

If you determine that you need to create a custom setup to modify or enhance the functionality of the Setup Wizard, you use the Setup Toolkit. The Setup Toolkit is a Visual Basic project which resides in \Setupkit\Setup1 subdirectory of the main Visual Basic directory and a collection of utilities and .dlls located in \Setupkit\Kitfil32.

Caution These are the same files used by the Setup Wizard. Therefore, if you modify a file in the Setup Toolkit (Setup1.exe, for instance), the modified version will be used in subsequent Setup programs created by the Setup Wizard. Before you modify this project, save a copy to another directory.

You use the Setup Toolkit by first loading the Setup1.vbp file into Visual Basic and then making modifications to the look or functionality of the project. In doing so, you may need to manually process the steps that the Setup Wizard would otherwise do for you.

The following sections describe steps in the process and explain how to determine which files you need to include in your setup, how to create a Setup.lst, how to create distribution disks, and how to test your setup.

For More Information See "Creating a Setup Program," "When to Use the Setup Wizard," or "Creating a Setup Program: Step by Step" earlier in this chapter.

21

Modifying the Setup Project

To add other user prompts and other events to the setup sequence, you can modify Setup1.vbp. You can write code in the Setup program just as you would in any other Visual Basic program. Further, a number of function calls are available that are especially useful in setup routines.

Some examples of when you might use the Setup Toolkit are:

- To add special user prompts during installation.
- To create a customized look and feel for your setup program.
- To display billboards during installation. Billboards display information about your product: features, service and support, etc.
- To use your own compression utility to copy your application's files to the distribution media.

25

Important Before you make changes to Setup1.vbp, you should make a backup copy of the Setup1 directory, as all changes made to the project files will be saved to the Setup1.vbp project file.

22

Determining the Files You Need to Distribute

Before creating a custom setup program, you need to determine which files will be included in the release. All Visual Basic applications need a minimum set of files, referred to as *bootstrap* files, along with all of the files specific to your application. In addition to an executable file (.exe), your application may require data files, ActiveX controls, or.dll files.

The files you use to create, run, and distribute your application can be categorized as follows:

Run-time files — The compiled executable (.exe) file and any ActiveX controls (.ocx) or .dlls required by your application. This also includes any other files required by your application, such as a data or initialization file.

Setup files — All of the files required to set up your application on the user's machine. These include the setup executable (Setup.exe), the setup file list (Setup.lst), and the uninstall program (St5unst.exe).

Run-Time Files

The following table lists the typical run-time files that need to be distributed with your application. Depending upon its scope and functionality, your application may require additional files not listed here.

File name	Description
appname.exe	Your application's .exe file.
appname.dat	Any data files you created for your application.
appname.txt	Any text files you created for your application.
<*control*>.ocx	Any ActiveX controls used in your application.
Msvbvm50.dll	The Visual Basic run time.
<*dependency*>.dll and <*dependency*>.exe	Any dependency files that your application requires.

23

Dependencies

Many of the files that need to be distributed with your application will be obvious to you: the executable file, any data files, and any ActiveX controls that you used. The less obvious files are your project's *dependency* files. These include files used by Visual Basic that provide much of the "back-end" to your application and which are usually found in the Windows\System directory. For example, if your application uses data controls, you may need to also distribute the Msjter32.dll or Odbcjt32.dll files.

Determining all the dependency files that your application requires can be made easier by using the Setup Wizard to create a dependency (.dep) file for your project.

For More Information See "Dependency Files Explained" earlier in this chapter for information on using the Setup Wizard to create a list of dependency files for your application.

24

Setup Files

Whether you chose to use the Setup Wizard or create a custom setup program using the Setup Toolkit, your application will use the same setup files for distribution.

File name	Description
Setup.exe	Program that the user runs to install your application.
Setup1.exe	Visual Basic application that you can customize. This is the executable file that is generated by Setup1.vbp. You can rename this file, as long as the new name is reflected in Setup.lst.
Setup.lst	Text file that lists all the files to be installed on the user's machine.
Vb5stkit.dll	Library containing various functions used in Setup1.bas.
St5unst.exe	Application removal utility for use with the Visual Basic Setup Toolkit.

25

In the Enterprise edition, the following files are added to the Setup.lst and are installed when your project contains Remote Automation or DCOM server components:

Clireg32.exe	Remote Automation client registration utility. This is added when your project is a Remote Automation or Distributed COM client, or if the server also acts as a client.
Racmgr32.exe	Remote Automation Client Manager. This is added if your project is a Remote Automation server.
Racreg32.dll	Remote Automation Registry. This is added if your project is a Remote Automation server.
Autmgr32.exe	Automation Manager. This is added if your project is a Remote Automation server.
Autprx32.dll	Automation Proxy. This is added if your project is a Remote Automation server.

26

Determining Where to Install Files on the User's Machine

Before writing your Setup program, you must determine where to install each of these files on the user's machine. The files required by your application can be divided into several categories.

- Program files
- Shared application resources
- Remote Automation server components

26

Suggested locations for each of these classes are described in the following sections.

Program Files

These files are essential for your application to run and are useful only in the context of your application. For example, the application's .exe file or any data files that your application may require are considered *program files*. Program files should be installed in the application directory. The user is prompted for this directory during

setup. The code in Setup1.vbp demonstrates how to do this. Setup1 will use the "Program Files" location as the default root location to install applications with Windows 95. For example, Setup1 will suggest that Project 1 be installed in the \Program Files\Project1 directory.

Caution When installing a file on the user's machine, it is imperative that you do not copy an older version of the file over a new version. The CopyFile function in Setup1.bas uses the VerInstallFile API function to copy files to the user's machine. VerInstallFile will *not* overwrite an existing file with an older version.

Shared Application Resources

Application resources may be shared by more than one application. For example, several different vendors may ship applications that use the same ActiveX control. By installing .ocx files in the \Windows\System directory, you can ensure that all applications use the most current .ocx files.

Remote Automation Components

Install Remote Automation server components to the \Windows\System directory. This ensures that your applications use the most current Remote Automation server components.

Creating the Setup.lst File

Setup.lst is the file that lists all the files to be installed on the user's machine. Setup Wizard writes out the disk layout, file sizes, file segmentation, and file attributes to this file. The file contains three sections: [BootStrap], [Files], and [Setup]. Each section of the Setup.lst file has a specific purpose in the setup process.

If you don't use the Setup Wizard to create the Setup.lst, you'll need to create this file manually. These sections are described in detail below.

The [BootStrap] Section

The [BootStrap] section lists all the files that must be loaded on the user's machine before your application and dependency files can be loaded. These pre-install, or *bootstrap*, files include the core files required to run any Visual Basic application, as well as Setup1.exe. These files are installed by Setup.exe. The following example shows entries in a typical [BootStrap] section:

```
[BootStrap]
File1=1,,setup1.ex_,setup1.exe,$(WinPath),,,9/27/1996,160256,5.0.0.3327,"","",""
File2=1,,VB5StKit.dl_,VB5StKit.dll,$(WinSysPath),,$(Shared),9/27/1996,29696,5.0.33.27,"","",""
File3=1,,VBRun500.dl_,Msvbvm50.dll,$(WinSysPath),$(DLLSelfRegister),$
(Shared),9/27/1996,1385744,5.0.33.27,"","",""
```

The [Files] Section

The [Files] section contains all the other files required by your application, such as your .exe file, data, text, and dependencies. The following example shows entries in a typical [Files] section:

```
[Files]
File1=1,,COMDLG32.OC_,COMDLG32.OCX,$(WinSysPath),$(DLLSelfRegister),$
(Shared),9/27/1996,130320,5.0.33.26,"","",""
File2=1,,TABCTL32.OC_,TABCTL32.OCX,$(WinSysPath),$(DLLSelfRegister),$
(Shared),9/27/1996,191248,5.0.33.26,"","",""
File3=1,,RICHTX32.OC_,RICHTX32.OCX,$(WinSysPath),$(DLLSelfRegister),$
(Shared),9/27/1996,193296,5.0.33.26,"","",""
```

File List Format in the [BootStrap] and [Files] Sections

Each file listed in the [BootStrap] and [Files] sections of Setup.lst must be described in the following format:

File*x*=*y*,[**SPLIT**],*file*,*install*,*path*,*register*,*date*,*shared*,*size*[,*version*]

27

File is a keyword that must appear at the beginning of each file description line.

X is a sequence number, starting at 1 in each section. The numbering sequence must be in order, and it cannot skip any values.

Y is the number of the disk on which this file will be distributed. If you are distributing to a network server or compact disc, this number is always 1.

SPLIT is an optional flag, indicating that this file is not the last part of a segmented file. For subsequent file names that are part of this same file, you should not repeat the fields following *install*. The last piece of the same file will not have the *SPLIT* keyword. A segmented file might be represented like this:

```
File10=2,SPLIT,GAZORNEN.DL1,GAZORNEN.DLL,$(WinSysPath),,10/12/96, 1151756
File11=3,SPLIT,GAZORNEN.DL2,GAZORNEN.DLL
File12=4,SPLIT,GAZORNEN.DL3,GAZORNEN.DLL
File13=5,,GAZORNEN.DL4,GAZORNEN.DLL
```

28

File is the name of the file as it is distributed.

Install is the name of the file that is used during installation (i.e., the name of the file as it exists on the distribution media). This can be the same as *file*. In some situations the file name may be different, such as when you create the media using compressed files where the last character of the file-name extension is an underscore, and you want to install the file (after decompression) with the original character replacing the underscore.

Path is either an actual path, or, preferably, a macro indicating a path specified by the user, or a combination of a macro plus additional subdirectory names separated by backslashes. The following tables list the macros that can be used in the installation.

Macro	Description
$(WinSysPath)	Installs a file in the \Windows\System (Windows 95) or \Windows\System32 (Windows NT) subdirectory.

$(WinPath)	Installs a file in the \Windows directory.
$(AppPath)	Installs a file in the application directory specified by the user, or the *DefaultDir* value specified in the [Setup] section.
$(AppPath)\Samples	Installs a file in the \Samples subdirectory below the application directory.
C:*path*	Installs a file in the directory identified by *path* (not recommended).
$(CommonFiles)	The common directory to which shared application files can be installed: C:\Program Files\Common Files\ (Windows 95 and Windows NT 4.0). C:\Windows (Window NT 3.51) This macro is generally combined with a subdirectory, such as $(CommonFiles)\My Company\My Application.
$(CommonFilesSys)	The same as $(CommonFiles)\System under Windows 95 and Windows NT 4.0, or the same as $(WinSysDir) under Windows NT 3.51.
$(ProgramFiles)	The default root directory to which applications are installed: C:\Program Files (Windows 95 and Windows NT 4.0) C:\ (Windows NT 3.51)
$(MSDAOPath)	Location that is stored in the registry for Data Access Objects (DAO) components. You should not use this for your files.

29

Note In the [BootStrap] section, the only valid macros are $(WinPath) and $(WinSysPath).

30

Register is a key that indicates how the file is to be included in the user's system registry. The four possible keys are listed in the following table.

Key	Use
(no key)	File does not contain linked or embedded objects and does not need to be registered on the user's machine.
$(DLLSelfRegister)	Self-registering .dll, .ocx, or any other .dll file with self-registering information (exports DllRegisterServer and DllUnregisterServer functions).
$(EXESelfRegister)	ActiveX .exe component created in Visual Basic, or any other .exe file that supports the /RegServer and /UnRegServer command-line switches.
$(TLBRegister)	Type Library file.
$(Remote)	Remote Support file (.vbr).
appname.reg	Any component you distribute that needs to be registered but does not provide self-registration. This key indicates a .reg file (that must also be installed) that contains information that updates the system registry. The .reg file should be compatible with Windows NT 3.51, or the

setup will fail when run on this platform.

31

Note Using the <appname>.reg key for distributing components is not the recommended method of getting registration information into the registry. Registry entries added in this manner cannot be automatically uninstalled with the Application Removal program. For the registry to be updated correctly, use the RegCreateKey, RegSetNumericValue, RegSetStringValue, RegOpenKey, and RegCloseKey functions in Setup1.vbp and rebuild Setup1.exe. Refer to commented code in Setup1.vbp.

32

$(Shared) is a keyword automatically added to the list entry of any file installed to \Windows, \Windows\System, or $(CommonFiles), the Common Files directory (\Windows or any subdirectory of \Windows with Windows NT 3.51, \Program Files\Common Files with Windows 95 and Windows NT 4.0). This keyword can be either $(Shared), indicating that it is a shared file and is registered in the registry, or blank, indicating that it is a private file.

Date and *size* are the same as the last date modified and file size as it would appear in the File Manager in Windows NT 3.51 or Explorer in Windows 95 or Windows NT 4.0. This information helps you to verify that you have the correct versions of the files on the setup disks. *Size* is used by Setup1.exe to calculate how much disk space your application requires on the user's machine.

Version is an optional internal version number of the file. Note that this is not necessarily the same number as the display version number you see by checking the file's properties.

Specifying Remote Server Components

When using remote server components in your applications, you must use two entries in the [Files] section: one for the component and one that contains connection information for the component.

Specifying *Remote#* in the file list entry marks the file as a remote server component. You then need to add an entry for the remote connection information immediately following the entry for the component, as in this example:

```
File2=1,,RemSvr1.VB_,RemSvr1.vbr, $(WinSysPath),$(Remote),$(Shared),4/10/1995,1147,1.0.0.0,"","","",""
Remote2="Schweizer","ncacn_ip_tcp",1,RA
File3=1,,RemSvr2.VB_,RemSvr2.vbr, $(WinSysPath),$(Remote),$(Shared),4/10/1995,1147,1.0.0.0,"","","",""
Remote3=,,1,DCOM
```

33

The Remote*x* entry consists of the server address, network protocol, and authentication information, separated by commas. You must also specify whether the component is to be used in a Remote Automation or Distributed COM environment using the *DA* or *DCOM* parameters.

This information is used by the Client Registration utility to register the Remote Automation server. The [Setup] Section

The third section of the Setup.lst file, [Setup], is simply a list of information used by other parts of the Setup program. The following table lists the information contained in the [Setup] section.

Component	Description
Title	The name of the application that will appear in the splash screen during installation. This name will also be used for the Program Manager or Explorer icon group and icon name.
DefProgramGroup	Specifies the default name of the Program Group (in Windows NT 3.51) or folder (in Windows 95 or Windows NT 4.0). Note that by default no folder is created for your application under Windows 95 or Windows NT 4.0. Instead, the program shortcut is created directly in the Programs menu of the Start menu.
DefaultDir	The directory into which the application will be installed, unless the user enters a different destination directory during the Setup program.
Setup	The name of the Setup program file, normally Setup1.exe. If you change the name of that file, be sure to insert the new name here.
ForceUseDefDir	If left blank, the user is prompted for an installation directory. If set to 1, the application will automatically be installed to the directory specified by "DefaultDir" in Setup.lst.
AppPath	Application-specific PATH variable, used by Windows 95 and Windows NT 4.0 to find any additional files needed for application execution.
AppExe	Should be set to the name of your application, such as Myapp.exe.

34

Note You can give your Setup program any name except Setup.exe, which is the name of the pre-installation program. For example, you can name your Setup program Setup1.exe or Mysetup.exe.

35

Determining the Layout of the Distribution Disks

If you are distributing your application on floppy disks, you need to know each file name and the distribution disk on which it resides. It's a good idea to map out the contents of each distribution disk before writing the Setup.lst.

The first distribution disk must always contain the following files:

- Setup.exe
- Setup.lst
- St5unst.exe

27

These files will not completely fill a single disk, so you can include other files. The remaining bootstrap files (Msvbvm50.dll and Stdole2.tlb, for example) must be installed before your application files are installed. If they take up more room than a

single disk, you'll need to determine how to assign the rest of the bootstrap files to the first and subsequent disks.

Following the last bootstrap file, you can start laying out your program files. The program files can be loaded in any order; however, remember that all of the bootstrap files must be installed before any of your program files.

Note Unless you have some special reason to do otherwise, you should use the Setup Wizard to create the layout and distribution disks for your application. See "Distribution Options" earlier in this chapter for more information.

36

Compressing the Setup Files

Once you have determined which files need to be included on the distribution disks, you may discover that all of them do not fit on a single disk. By compressing the files on your distribution disks, you may be able to reduce the number of disks required to distribute your application.

The Setup Toolkit provides a utility called Compress.exe that allows you to compress files. Even if you already have a version of Compress.exe on your system, it is recommended that you use the version of Compress.exe located in the \Setupkit\Kitfil32 subdirectory of the main Visual Basic directory for application setup because the two versions of the compression utility may not be compatible.

To compress a file

- At an MS-DOS command line, type:

  21**compress -r** *filename*

37

  22The **-r** switch automatically replaces the last character of the compressed file with an underscore. The compressed file is encoded so that it is automatically renamed to the original file name when it is decompressed. For example:

  23**compress -r myapp.exe**

28

The result is a compressed file named Myapp.ex_. When decompressed, the file is renamed as specified by the file entry in Setup.lst. By convention, this would usually be Myapp.exe; however, you can rename the expanded file.

You can compress any or all files, except Setup.exe and Setup.lst.

Creating the Distribution Disks

Once you've determined the disk layout for the files in your application and have compressed them using Compress.exe, copy the files to the distribution disks.

It's a good practice to label your distribution disks with the number and name of the disk and instructions for setting up your application. For example:

Disk1: Setup Disk
Insert this disk in drive A
In File Manager in Windows NT 3.51, or Explorer in Windows 95 or Windows NT 4.0, double-click Setup.exe in the directory window for your floppy disk drive.
Follow the instructions that appear on your screen.

38

Repeat this installation message on each of the distribution disks. Now you are ready to copy your files to the distribution disks.

To create distribution disks

9. Copy these files onto the first disk:
   - Setup.exe
   - Setup.lst

29

10. Copy the remaining files as specified by your disk layout, using additional disks as needed.

30

For More Information See "Distribution Options" earlier in this chapter for more information on using the Setup Wizard to create the distribution disks for your application.

39

Testing Your Setup Program

Once you have created the distribution media, you should test your Setup program. Be sure to test your Setup program on a machine that does not have Visual Basic or any of the ActiveX controls required by your application. You should also test your setup on all applicable operating systems.

To test your Setup program using distribution disks

33 Insert the first disk in drive A.

34 In the Windows Program Manager in Windows NT 3.51, choose **Run** from the **File** menu, or in Windows 95 and Windows NT 4.0, from the **Start** menu, choose **Run**, and type:

24 **a:setup**

25– or –

26Double-click Setup.exe from the directory window of the floppy disk or compact disc drive.

31

To test your Setup program using network distribution

35 From another computer on the same network as the distribution server, use the Microsoft Windows File Manager in Windows NT 3.51, or, in Windows 95 and Windows NT 4.0, the Windows Explorer, to connect to the server and directory containing your distribution files.

36 In the distribution directory, double-click the Setup.exe file.

32

As a final test, execute the installed program to be sure it behaves as expected once installed.

Allowing the User to Remove Your Application

When the user installs your application, the application-removal utility (St5unst.exe) will be copied to the \Windows directory. Each time you use the Visual Basic setup program to install an application, an application removal log file (St5unst.log) is generated in the application's installation directory that contains entries indicating:

- Directories that were created during installation.
- Files that were installed and where they were installed. This list contains all of the files in the Setup program, even those that were not installed on the user's machine because a newer version of the same file already existed on the user's machine. The .log file indicates if the file was a shared file and if so, if it replaced the existing shared file.
- Registry entries that were created or modified.
- Program Manager icons and groups that were created with Windows NT 3.51, or links and Start menu entries with Windows 95 and Windows NT 4.0.
- Which .dlls, .exes, or .ocxs were self-registered.

33

In Windows NT 3.51, the application removal utility icon will be added to the Program Manager icon group, with the command line properties set to the appropriate .log file. In Windows 95 and Windows NT 4.0, the application will be added to the list of registered applications displayed by the Add/Remove Programs applet in the Control Panel. Use this applet to remove the application.

In the event of a failed or canceled installation, the application-removal utility will remove all of the directories, files, and registration entries that Setup1 created in the installation attempt.

With Windows 95 and Windows NT operating systems, shared files are reference-counted in the registry; when you remove the installation, the reference count for that item will decrement until the count equals zero, at which time the user will be prompted for final removal of that item.

Application Removal Failure Scenarios

For the application-removal utility to properly remove your application from the user's machine, the log file and registry entries created by your Setup program must

—35

remain unchanged and accurate. The following are a few of the possible scenarios that will cause the Application Removal utility to fail or to work incorrectly:

- The end user copies shared files manually (i.e., reference counting is not updated in the registry).
- The end user deletes installed files or an application directory rather than using the application-removal utility. This creates two problems:
  - Deletes the logfile, so application removal is later impossible.
  - Makes it impossible to remove system registry entries from program files, .dlls, or .ocxs in the application directory, because they must be run for them to remove their system registry entries.
- A setup program for a non-Windows 95–compliant application installs the same shared files that are also installed by your Windows 95–compliant application.
- A shared file is installed into a different directory than the one in which it already exists on the hard drive.
- You create a setup program with the Setup Wizard, with the deployment model set for "install as stand-alone application," and run the setup. Later, another setup program is created for a different Visual Basic application, and the first application is added to this setup program as a component. If it is possible that the end user might install a particular Visual Basic Remote Automation server as a component in the setup program of another application, you must use deployment set to "install as shared component."
- The end user installs the same Visual Basic application in two different directories. Not only will the first installation no longer work, but the Application Removal scenarios will clash. The end user should always remove the first installation before installing an application in a different directory.
- The end user deletes the application setup log (St5unst.log). Without the application setup log file, the application-removal utility will have no installation information, and will fail.

34

Some of these scenarios could degrade the installed file-registry correspondence, cause the Application Removal utility to prematurely reach a zero-reference count for a particular file, and subsequently ask if this file could be deleted. If a file is prematurely deleted, it could cause other applications to cease functioning or function incorrectly due to missing file dependencies, missing components, and so on.

Using the Setup Wizard with the Setup Toolkit

If you want only to customize the installation sequence that the user must follow, you can simplify your work by using the Setup Wizard with the Setup Toolkit project (Setup1.vbp).

Providing Installation Options to the User

You can use the Setup Toolkit and the Setup Wizard together to add dialog boxes to the installation program, prompting the user to specify whether to install optional features in your application. For example, you may have an online Help file that some users would rather not install. You can add as many installation options as you want.

To add an installation option to your Setup program

37 Edit the Setup1.frm code by adding code like the following in the Form_Load event, immediately after the code block calls the ShowBegin Form function.

```
1Dim LoadHelp As Integer
2LoadHelp = MsgBox ("Do you want to install Help? ", _    vbYesNo)
3If LoadHelp = vbYes Then
4    CalcDiskSpace "Help"
5EndIf
6.
7.
8    ' Block of code containing cIcons = _ CountIcons(strINI FILES)
9If LoadHelp = vbYes Then
10   cIcons = CountIcons("Help")
11EndIf
12
13   ' Block of code containing CopySection _ strINI_FILES.
14.
15If LoadHelp = vbYes Then
16   CopySection "Help"
17EndIf
18.
19   ' Block of code containing CreateIcons _
20strINI FILES, strGroupName
```

40~~35~~

38 Close Setup1.frm, save the form and the project, and create an .exe file.

39 Run the Setup Wizard. At the File Summary screen, after you've confirmed all of the file dependencies, add the names of all the files you want to install if the user answers "Yes" to this dialog box.

40 Once you are done with the Setup Wizard, generate the distribution media.

41 Insert Disk1 into a drive and open the Setup.lst File. Cut and paste the optional file(s) from the [Files] section to a new section. This new section will be below the [Files] section and will be given a new section title that corresponds to the string argument (for instance, [Help]) you used in the CopySection statement. Be sure to renumber the copied lines.

```
21[Help]
22File1=5,SPLIT,MyApp.HL1,MyApp.HLP,$(AppPath),,,10/12/96,2946967
23File2=6,SPLIT,MyApp.HL2,MyApp.HLP
```

```
24File3=7,,MyApp.HL3,MyApp.HLP
```


When the user runs the installation program, the Setup program copies all the [BootStrap] files to the user's machine, and then prompts the user to indicate whether to install the Help files. If the user chooses Yes, the CalcDiskSpace statement determines whether there is sufficient disk space on the user's machine for the Help files. The program then installs all of the files listed with the [Files] section in Setup.lst.

Next, the program tests the LoadHelp flag again. If the user chose to install the Help files, Setup1.exe next executes the CopySection statement for the Help files, and installs the files listed in the [Help] section of Setup.lst.