Appendix B
How To: Wise for Windows Installer

Wise for Windows Installer is a powerful tool for creating Windows Installer-based setup packages. This appendix shows you how to use Wise for Windows Installer to create a setup package for a Visual FoxPro application.

What is Wise for Windows Installer?

Wise for Windows Installer, or WfWI for short, is a Windows Installer-based setup tool from Wise Solutions, Inc. Wise also markets other setup tools including Wise Package Studio, Wise for Visual Studio .NET, and Wise Installation System.

As its name implies, Wise for Windows Installer builds setup packages for Windows Installer. Because the fundamental structure of a Windows Installer setup packages is the same no matter what tool you use to create it, the concepts you encounter when working with WfWI are the same as those you encounter when working with other Windows Installer-based setup tools such as the one that ships with Visual FoxPro.

Wise for Windows Installer comes in three editions: Standard, Professional, and Enterprise. We chose to base this appendix on the Standard edition because in our opinion it is the most comparable to the tool that ships with Visual FoxPro. The Professional and Enterprise editions of WfWI are more powerful and offer features not found in the Standard edition. You can find a comparison of features in Wise for Windows Installer Enterprise, Professional, and Standard editions at **http://www.wise.com/wfwi_features_grid.asp**.

The figures and examples in this appendix are based on the Standard edition version 5.21, which is the current version as of this writing. You can download an evaluation copy of Wise for Windows Installer from **http://www.wise.com/downloads.asp**. Evaluation copies of other editions of WfWI and other Wise products are also available at this URL.

Creating a setup package with Wise for Windows Installer

This appendix takes you through the process of creating a setup package for a sample Visual FoxPro application using Wise for Windows Installer. You will learn how to:

- Create a new Wise for Windows Installer project
- Define features and add files to the project
- Add the VFP 8 runtime support files and other dependencies
- Define the type of release you want to build
- Compile, test, and deploy the release

- Create upgrades and patches

The Sample Application

In this appendix we use the DeployFox Demo App to illustrate the construction of a setup package using Wise for Windows Installer. This application, developed in VFP 8, is very simple but representative of the type of application VFP developers often deploy. It consists of a main EXE file, a ReadMe text file, an HTML help file, a data table with an index, and an ActiveX control file, along with the VFP 8 runtime support files and the VFP 8 HTML Help runtime support files.

We use the same sample application, called the DeployFox Demo App, as an example throughout this book in order to provide as much consistency between chapters as possible.

Creating a new Wise for Windows Installer project

When you first start Wise for Windows Installer, it prompts you to create a new project, as illustrated in **Figure 1**.

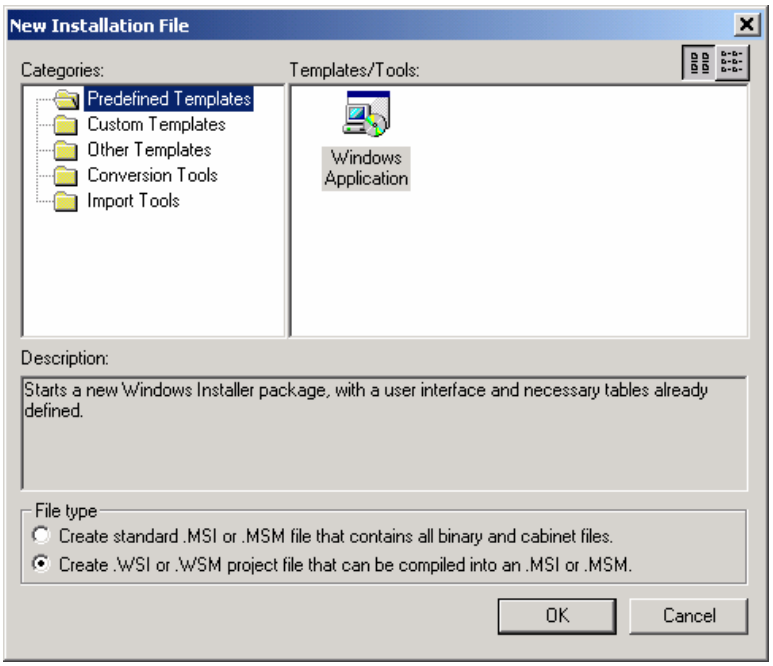

***Figure 1**. Use the predefined Windows Application template when creating a new setup project file for your VFP application.*

Wise for Windows Installer comes with templates to assist you in creating various types of setup projects. For a Visual FoxPro application, choose the Windows Application template from the Predefined Templates category. Near the bottom of the dialog, under File type, choose "Create .WSI or .WSM project file…." Click the OK button to create the new project file.

Using the Installation Expert

The WfWI IDE provides three different views of a project. The one you primarily work with is the Installation Expert view. When you create a new project, the Installation Expert view is the default view. Illustrated in **Figure 2**, you can see the Installation Expert tab selected at the bottom left. In many cases you can build a complete setup package using only the Installation Expert view. The other two views—MSI Script and Setup Editor—provide lower-level access to the setup package and allow for fine tuning if necessary.

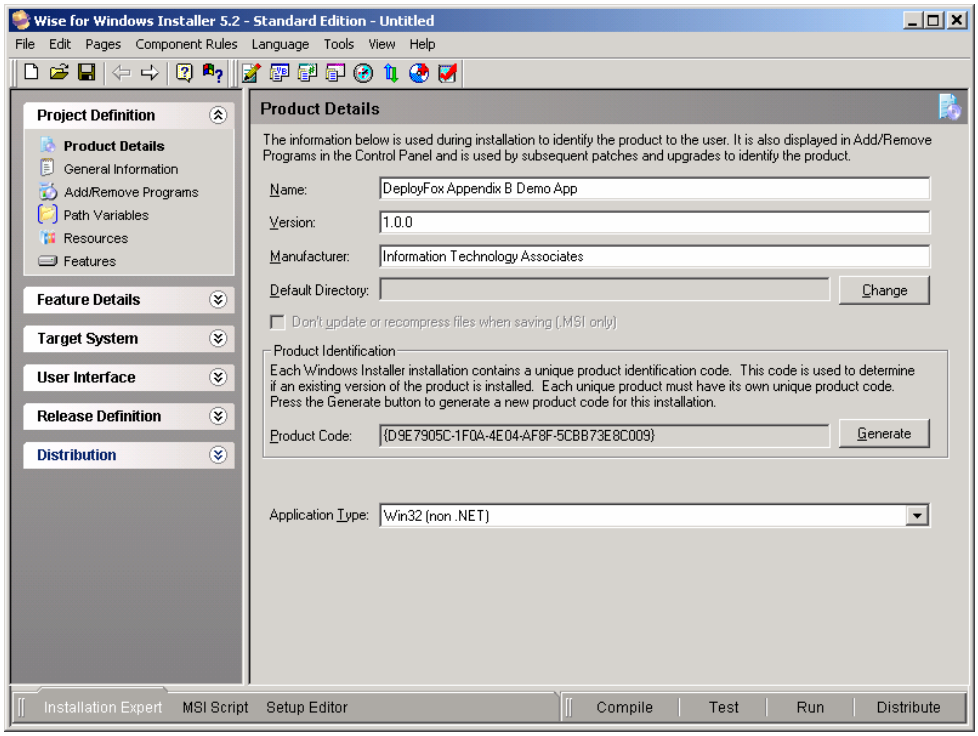

***Figure 2**. The Installation Expert view is your primary interface to the setup project.*

The Installation Expert view consists of a set of page groups on the left side and a page detail area on the right side. Each page group comprises a group of related pages. Clicking the double-headed arrow icon expands or contracts a page group to show or hide the individual page names in that group.

Not all setups require all the pages in each page group. In this appendix, we discuss only the pages you most commonly need to use.

Project Definition group

The first page group in the Installation Expert is the Project Definition group. This is the normal starting place when working with a new project. Figure 2 shows the Project Definition group expanded so you can see the various page names in this group. The Product Details page name appears in bold because it is the currently selected page.

Product Details page

The properties and values for the Product Details page display in the page detail area on the right, as shown in Figure 2. Among other things, this page is where you specify the name of the product, its version number, and its manufacturer or publisher. In Figure 2 this information is already filled in.

The Product Details page provides a space for the Default Directory, which is the default location presented to the user in the Destination Folder dialog during installation. Unfortunately, the value you probably want to set as the Default Directory—typically a sub-folder named for your product under Program Files—does not exist in the drop-down list for this field until you add the necessary folders to the project later on. Therefore, the Default Directory is blank for now.

In a Windows Installer setup each product must have a unique product code. In Wise for Windows Installer the product code GUID can be seen in the Product Identification area of the Product Details page. WfWI generates a product code GUID for you each time you start a new project. Normally you can use the product code generated by WfWI. If you need to change it, click the Generate button for a random new one, or, if you need to use a specific product code GUID rather than a randomly generated one, switch to the Setup Editor view and enter the GUID there. For an explanation of Windows Installer product codes and when to change them, please refer to Chapter 5, “Windows Installer Inside and Out.”

The last piece of information under Product Details is the Application Type. For a Visual FoxPro application use the default setting of Win32 (non .NET).

General Information page

On the General Information page you enter the information you want the user to see when they right-click on the setup program and choose Properties in Windows Explorer. **Figure 3** shows the General Information page for the Demo App.

The Installer Version field on the General Information page specifies the minimum Windows Installer version required on the destination computer. The default is version 2.0, which is the current version of Windows Installer as of this writing. Normally there is no need to change the default setting, but sometimes you might need to. For example, if your product is going to be installed on machines running Windows NT you should be aware that Windows Installer version 2.0 requires Windows NT 4.0 Service Pack 6. If SP6 is not installed on your target machines you need to specify an earlier version of Windows Installer in your setup package. Although you cannot change the minimum Installer Version from the General Information page, you can change it in the Product tab of the Setup Editor view.

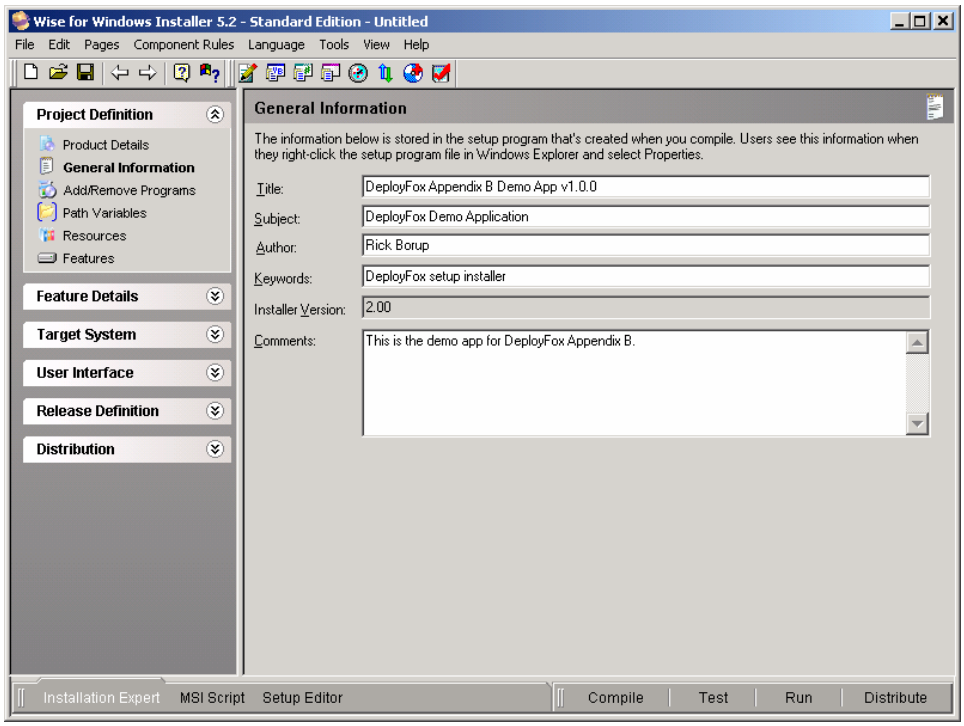

***Figure 3**. Use the General Information page to specify the setup's properties as seen by Windows Explorer.*

Features page

Every Windows Installer-based setup package requires the product to be organized into one or more features. Features, as explained in Chapter 5, are the functional parts of a product as seen from the user's point of view. When performing a custom installation, users see a list of features and can choose which ones to install. One feature is usually set by the developer as required, while the others are required or optional at the developer's discretion.

The DeployFox Demo App is organized into three features. The first feature consists of the main EXE file, the ReadMe file, the VFP 8 runtime support library files, and the ActiveX control file. The second feature includes the application's HTML Help file (CHM) and the VFP 8 HTML Help runtime support files. The third feature comprises the application's data table and associated index file, along with an ODBC data source name (DSN) and the Visual FoxPro ODBC driver.

In Wise for Windows Installer, you use the Features page to specify the features in your product. In a new WfWI project there is one predefined feature named "Complete," as illustrated in **Figure 4**.

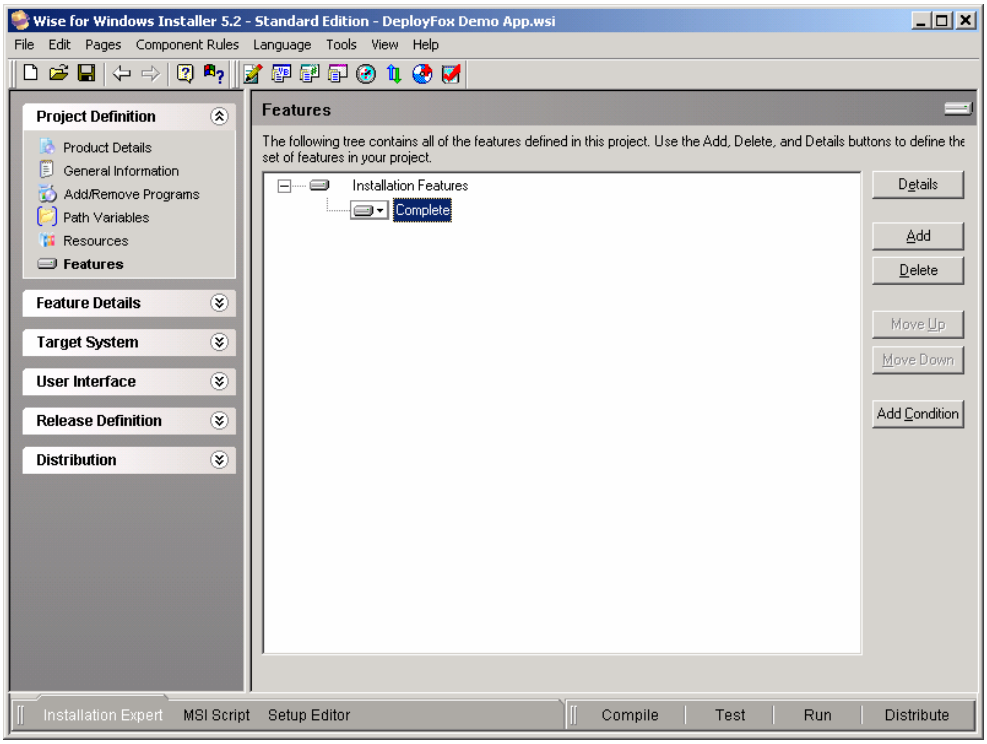

***Figure 4**. Use the Features page to define the features in your product. A new project file has one predefined feature named Complete.*

To change the name of a feature and to specify other details about it, open the Feature Details dialog by selecting the feature and clicking the Details button on the right. In the demo app, we want to name the first feature—the one that contains the EXE file—"Main Program." We also want this feature to be required. **Figure 5** shows the Feature Details dialog reflecting these settings.

In the Feature Details dialog, the Name field contains the name of feature as Windows Installer references it internally. If you enter a Name with spaces, WfWI replaces each space with an underscore character. The Title field is the title of the feature as the user sees it during installation. The default value of the Title field is the value of the Name field with spaces instead of underscores. The entry in the Description field is what the user sees in the description area of the Select Features dialog during a custom installation.

The Main Program feature is at the top level of the feature tree, so its Parent is <None>. Because we want the Main Program feature to be required, we select the Required Feature check box at the lower left of the dialog. Clicking OK saves the settings and returns to the Features page.

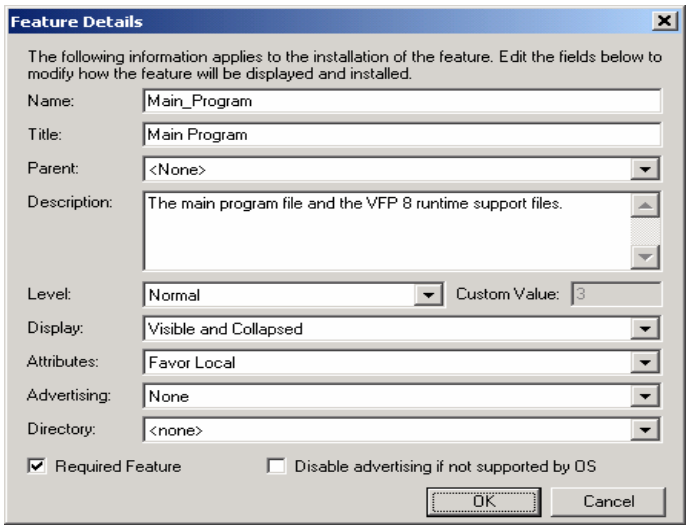

***Figure 5**. Use the Feature Details dialog to specify the name and other information about each feature.*

To add additional features for the product, click the Add button. This opens a Feature Details page for the new feature. Fill in the necessary information for each new feature and click OK to add the feature to the feature tree.

By default, the value of the Parent field of a new feature is the name of the feature selected when you click the Add button. To add new features at the top level of the feature hierarchy, select "Installation Features" at the top of the feature tree before clicking the Add button. If necessary, however, you can change the Parent of a feature by selecting a new Parent from the drop-down list in the Feature Details dialog.

The completed feature tree for the demo app is shown in **Figure 6**. You can see all three features are at the same level of the feature hierarchy.

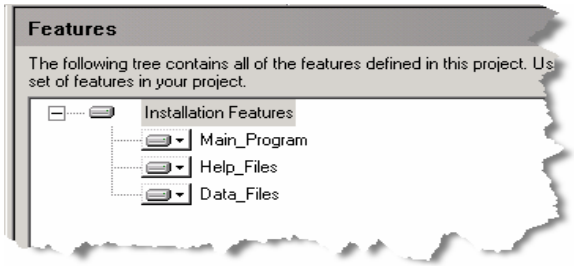

***Figure 6**. The completed feature tree shows all three features for the demo app. The appearance of this tree in WfWI is very similar to what the user sees during a custom installation.*

At this point we are finished with the Project Definition page group. If you are working with a new project file and have not yet saved it to disk, this is a good time to do so. We named the project file for the demo app DEPLOYFOX DEMO APP.WSI.

The DEPLOYFOX DEMO APP.WSI project file is included in the chapter downloads. You need a copy of Wise for Windows Installer to review and edit the contents of this project.

Feature Details group

The next group in sequence is the Feature Details group. This group includes the pages you use to add files to each feature of the product, create registry entries and shortcuts, define ODBC resources, and specify the necessary merge modules such as those for the VFP runtime files.

Merge Modules page

Visual FoxPro programs require the VFP runtime support library files to be installed on the user's computer. A Windows Installer setup does not add these runtime support files to the setup project individually. Instead, they come pre-packaged in the form of merge modules. The Merge Modules page is where you tell Wise for Windows Installer which merge modules to include in the setup package.

When you install Visual FoxPro 7.0 or 8.0 on your computer, the corresponding runtime merge modules are also installed. WfWI does not automatically know where to find all the merge modules on your computer, however, so you need to tell it where they are. The VFP 8 merge modules are typically installed in Program Files\Common Files\Merge Modules. To tell WfWI this folder contains merge modules, go to Edit | Preferences on the WfWI main menu and select the Merge Modules tab in the Preferences dialog as illustrated in **Figure 7**. Click the Add button and navigate to the folder containing the VFP 8 merge modules. Click OK to add this folder to the list of those whose contents WfWI includes when listing merge modules for you to add to your project.

You can add as many additional merge module folders as necessary. If you're not sure where the merge modules are located on your computer, search for all files with an MSM file name extension and note their locations. Figure 7 illustrates you can also change the default merge module directory, which may be sufficient if all the merge modules you need are in a common location.

Once Wise for Windows Installer knows where the VFP 8 merge modules are located, you're ready to add the required merge modules to the setup project. Similar to files and folders, merge modules are added to a project on a feature-by-feature basis. To add a merge module to a project, first select the feature the merge module belongs to from the drop-down list at the top of the Merge Modules page. Click the Add button on the right to bring up the Add Merge Modules Wizard, where you can select the desired merge module(s) from the list.

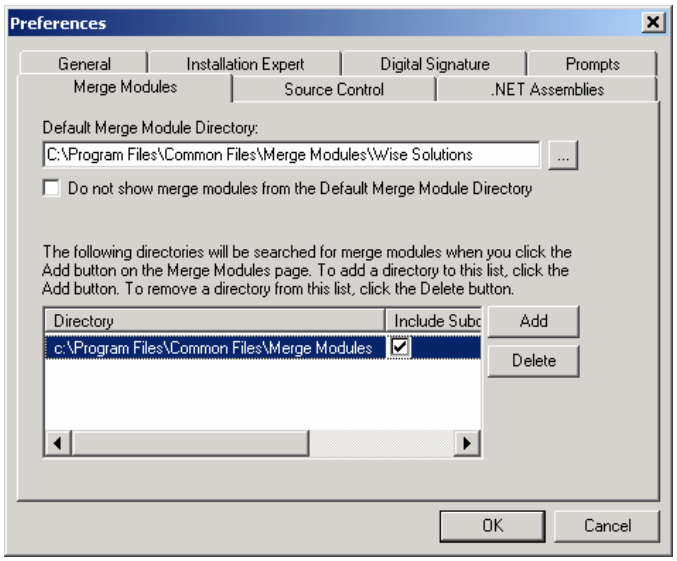

***Figure 7**. Use the Merge Modules tab of the Preferences dialog to tell WfWI about other directories containing merge modules, such as those for VFP 8.*

The demo app requires the VFP 8 runtime libraries merge module, associated with the Main Program feature, and the VFP 8 HTML Help runtime files merge module, associated with the Help Files features. It also requires the MSCOMCTL merge module, which contains an ActiveX control used by the app.

Figure 8 shows the first step of the Add Merge Modules Wizard with the VFP8Runtime merge module selected in the list. Note each merge module has both a name and a description. To help you locate the desired merge modules, click the appropriate column header to sort the list either by name or by description.

Notice the lower section of this dialog, where you can see additional information about the selected merge module, including its actual path and file name. Many merge modules have similar names and even similar descriptions, which sometimes makes it difficult to know which one to select. Verifying the actual path and file name of a merge module is a good way to confirm you selected the one you want.

After selecting the merge modules you want to add to your project, proceed through the next two steps of the Add Merge Module Wizard to finish the process. Step 2 enables you to specify a destination directory for the files installed by a merge module. Most merge modules come with pre-defined destination directories so you can usually skip over this step. Step 3 lets you associate a merge module with other features besides the currently selected feature, if necessary. Repeat the merge module selection process for each feature that installs a merge module.

Figure 9 shows the Merge Modules page for the demo app's Main Program feature after the required merge modules are added. We manually added the VFP 8 runtime files merge module to the project by selecting it from the list shown in Figure 8. The VFP8 runtime files require the Visual C++ 7.0 runtime file and the GDI Plus DLL. Selecting the VFP8 runtime files merge module automatically adds the merge modules for these other files to the project.

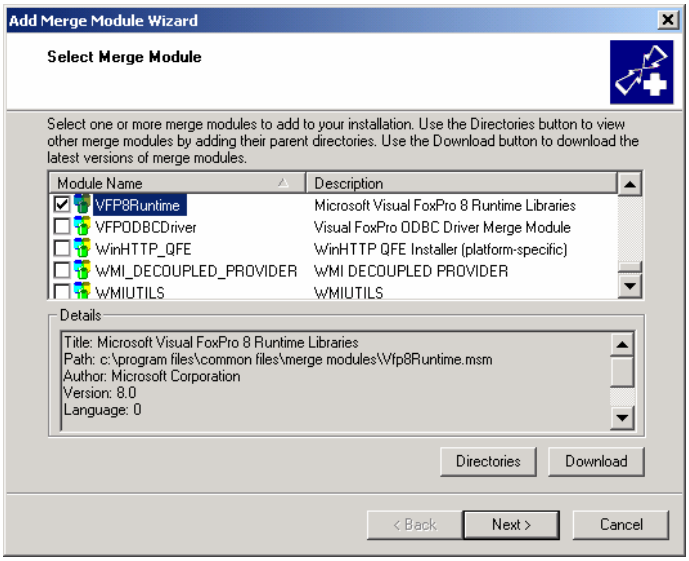

***Figure 8**. Add a merge module to the project by selecting its check box in the list. Additional information about the selected merge module is presented in the lower portion of this dialog.*

The demo app uses the slider control from the MSCOMCTL.OCX ActiveX control, so we want to install that control as part of the Main Program feature. A merge module named MSCOMCTL.MSM contains this control, which we also added to the project. The MSCOMCTL.MSM merge module, in turn, requires the COMCAT and OLEAUT32 merge modules. In total, six merge modules install with the Main Program feature, as shown in Figure 9.

Unlike the VFP 8 runtime merge module, the VFP 7 runtime merge module does not automatically add the other merge modules on which it depends. When deploying a VFP 7 app you need to select these other merge modules manually. For a list of required merge modules, please refer to Chapter 3, "Packaging the Installation."

In a similar manner, add the VFP8 HTML Help support library merge module to the Help Files feature, and the Visual FoxPro ODBC Driver Merge Module to the Data Files feature. The choice of which merge modules belong with which features is, to some extent, up to you as the developer. For example, we could have chosen to install all the merge modules as part of the Main Program feature. This would work for any custom installation regardless of the features the user selects because the Main Program feature is required and therefore always installs. Associating the merge modules with the appropriate feature streamlines the installation, because the merge modules only install if the feature requiring them is installed.

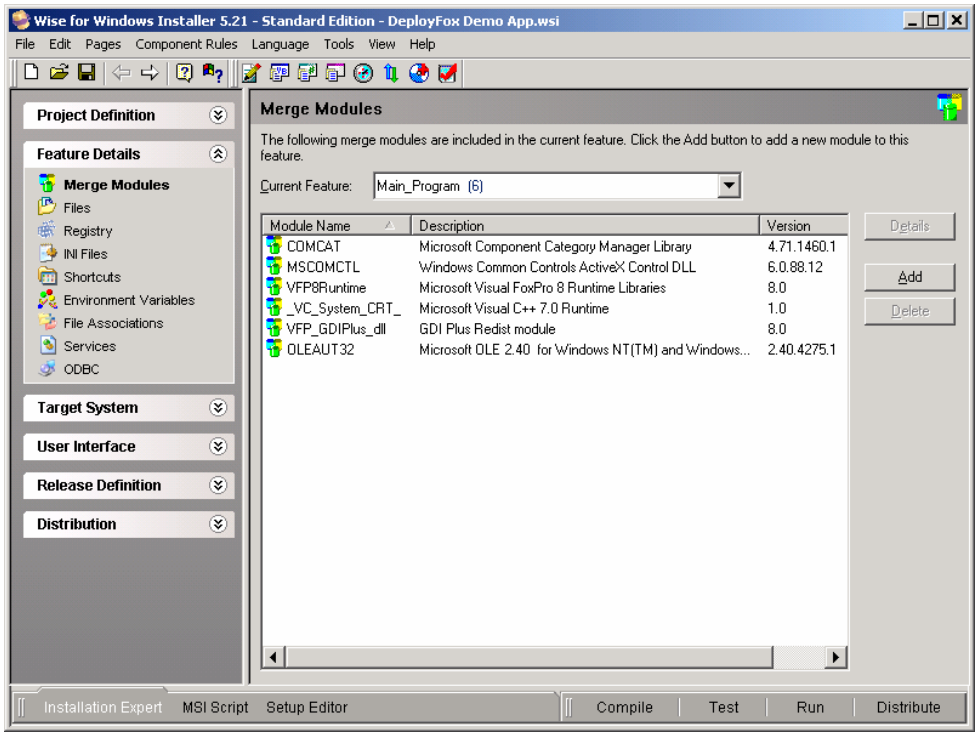

***Figure 9**. The Main Program feature installs the Visual FoxPro 8 Runtime Libraries merge module and its dependencies. It also installs the MSCOMCTL merge module, which in turn requires the COMCAT and OLEAUT32 merge modules.*

Files page

The Files page is where you add the product's files to the setup project. Done on a feature-by-feature basis, the first step is to select a feature from the drop-down list at the top of the page detail area. As you can see in **Figure 10**, the list box has an entry for each of the three features we defined on the Features Page. There is also a fourth choice named All Features (Modify /Delete only). This choice is useful when you want to see all the files for all the features at the same time, but you cannot add new files with this choice selected. In Figure 10 the Main Program feature is selected in preparation for adding directories and files for that feature.

The Files page is organized into four panes. The two upper panes provide access to the folders and files on your computer (the source). The two lower panes represent the folders and files on the user's computer (the destination or target). You can drag the center vertical divider left or right to resize the panes if desired.

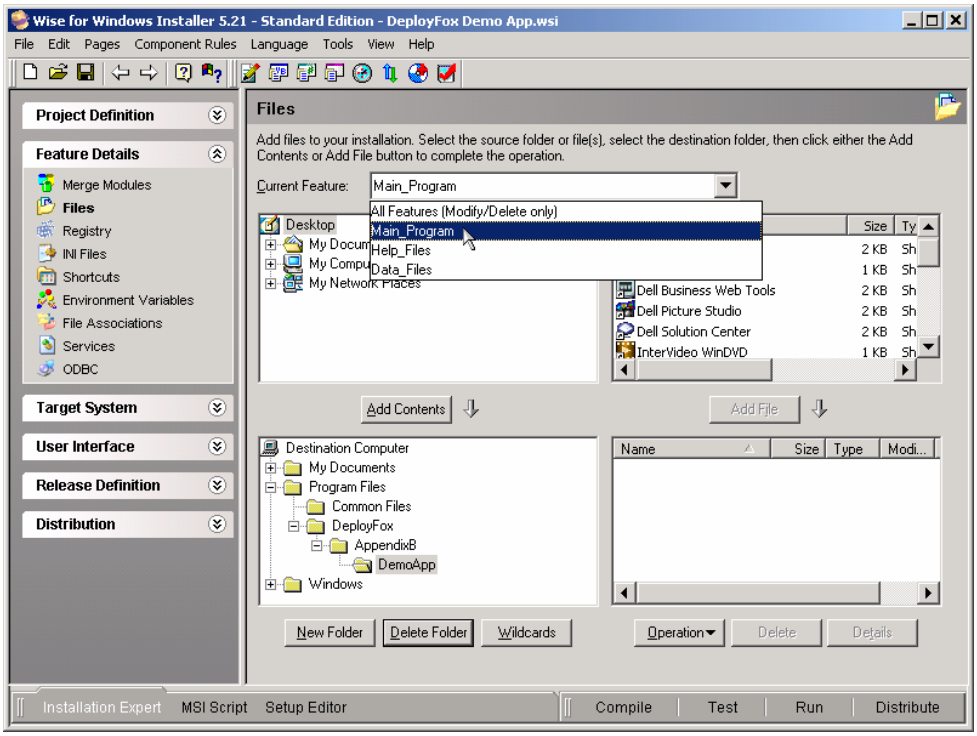

***Figure 10**. On the Files page, first select the desired feature from the drop-down list at the top. Next, add directories and files for that feature as required.*

We want the demo app's EXE file and other related files to install in a folder named DEPLOYFOX\APPENDIXB\DEMOAPP under the user's Program Files folder. Before we can add files to this folder in the setup project, we must first tell Wise for Windows Installer to create this folder structure on the user's machine. Specify this in the lower left pane of the Files page. To add a new folder simply right-click the parent folder in the lower left pane, select New Folder from the shortcut menu, and type in the name of the new folder. In Figure 10 you can see the desired folder structure for the demo app has been created.

Directories, like files, are added to a project on a feature-by-feature basis. Therefore, directories created for one feature may not be visible when another feature is selected. In order to see the directories for all features regardless of the feature currently selected, go to the WfWI Edit | Preferences menu, choose the Installation Expert tab, and select the "View directories for all features on Files page" check box. Also, select the "View registry keys for all features on Registry page" check box to set the corresponding option for registry entries.

Now, with the desired feature selected at the top and the desired folder selected in the lower left pane, you can add files to that folder. To do so, select the source file in the upper right pane and click the Add File button, or drag the source file from the upper right pane to the lower right pane.

Repeat this procedure for each feature, adding the required directories and files for each one. In the demo app, the Main Program feature installs the EXE file and the ReadMe file. The Help Files feature installs the HTML Help (CHM) file. The DBF and CDX files install with the Data Files feature. The demo app requires its data files be located in a sub-folder named Data under DemoApp, so to complete the Data Files feature you must first create the Data sub-folder, and then add the two data files to it.

At this point the list of files and folders for installation by the setup is complete. You can now return to the Product Details page of the Product Definition group (refer to Figure 2) and set the default installation directory for the product. The desired default installation directory—Program Files\DeployFox\AppendixB\DemoApp—now appears in the drop-down list when the default directory Change button is clicked, as illustrated in **Figure 11**.

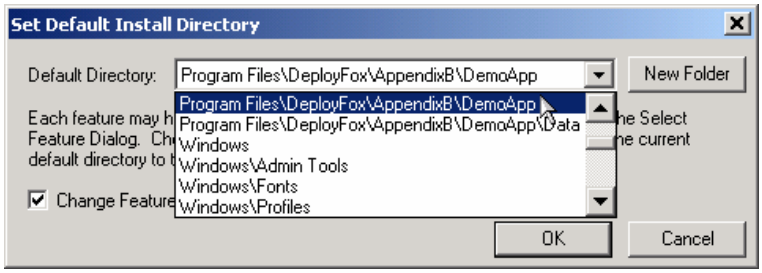

Figure 11 *The folders defined on the Files page appear in the drop-down list when you click the Change button to set the default directory on the Product Details page.*

A note about data files

Data files need special consideration to avoid potential problems when installing future product updates or when the product is uninstalled, which can occur silently during certain types of updates. When installing a product update, the Windows Installer file versioning rules are generally sufficient to avoid overwriting live data files updated by the user since the original installation. However, you can provide an extra measure of protection for data by setting special Windows Installer attributes for these files. In Wise for Windows Installer, this is done from the Components tab of the Setup Editor view. Using this view, select the component(s) containing the data files. Right-click the component name, choose Details from the shortcut menu, and select the check boxes for "Never overwrite if key path exists" and "Leave installed on uninstall," as illustrated in **Figure 12**.

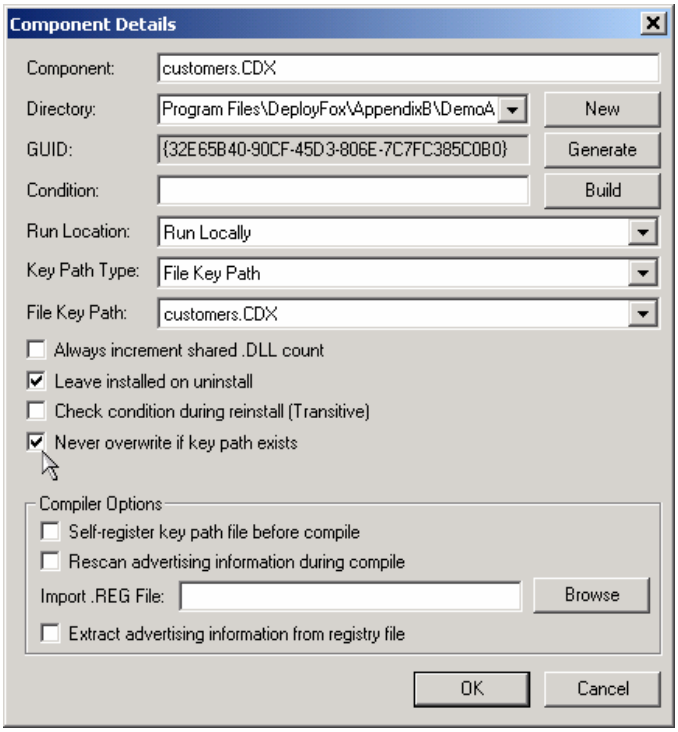

***Figure 12**. To protect data files from being overwritten or uninstalled, mark the component containing them as "Never overwrite if key path exists" and "Leave installed on uninstall."*

In a Windows Installer setup package, components are the building blocks of features. Wise for Windows Installer automatically creates a component structure for you from the information you supply for each feature. Because it's created automatically you do not ordinarily need to be concerned with the component-level structure of a setup package, but WfWI gives you the ability to work at this level if necessary. For more information about features and components please refer to Chapter 5, "Windows Installer Inside and Out."

Registry page

The Registry page makes it easy to add registry entries to the target machine at installation time. As illustrated in **Figure 13**, the Registry page is arranged into four panes much like the Files page. The upper half of the Registry page represents your machine (the source) and the lower half represents the user's machine (the destination).

Remember that registry keys, like files, are always associated with a feature. Be sure to select the desired feature from the drop-down list at the top of the Registry page before adding registry keys and values to your setup project.

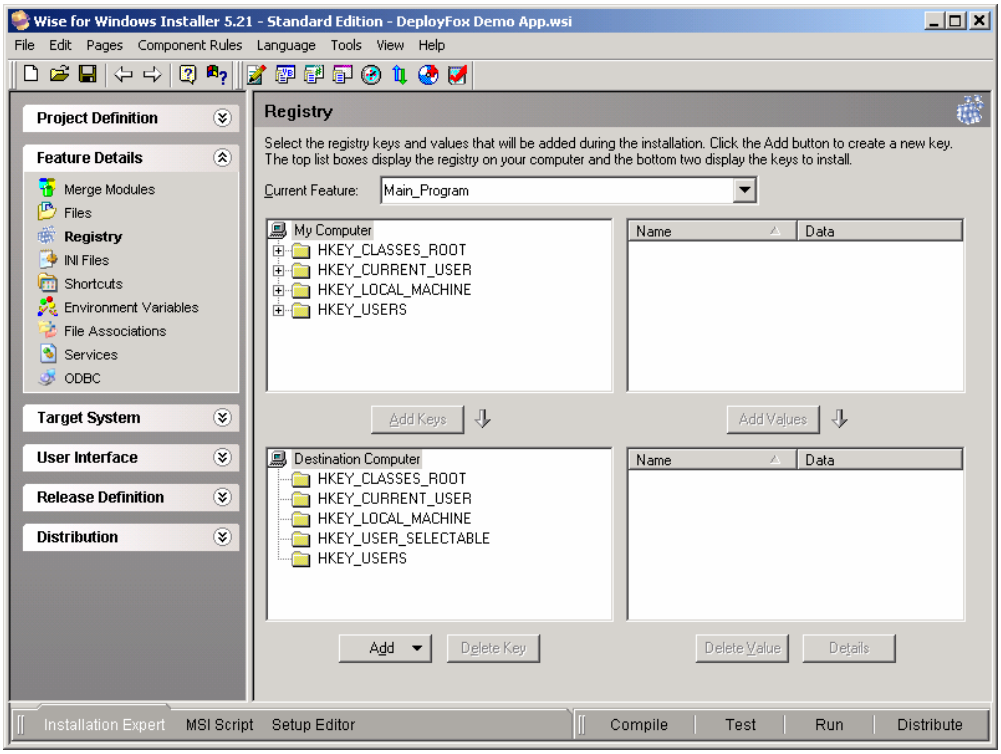

***Figure 13**. Use the Registry page to define registry keys and values to create on the destination computer during installation of your product.*

Registry entries can be added to the setup project in several ways. If the keys and values you want to add already exist in the registry on your own computer, you can select them in the source computer tree (upper left pane of the Registry page) and click the Add Keys button to add them to the corresponding registry location in the destination computer tree (lower left pane). You can accomplish the same thing by dragging a key from the source computer pane to the appropriate location in the destination computer pane.

If the keys and values you want to add do not already exist in the registry on your own computer, you can add them to your setup project by importing a REG file or by entering them manually. To add registry keys and values manually, start by selecting the desired root key in the lower left pane of the Registry page. Right-click on this key and choose Add Key from the shortcut menu. This opens the Registry Details dialog illustrated in **Figure 14**. Use this dialog to enter the information necessary to define the key you want to add.

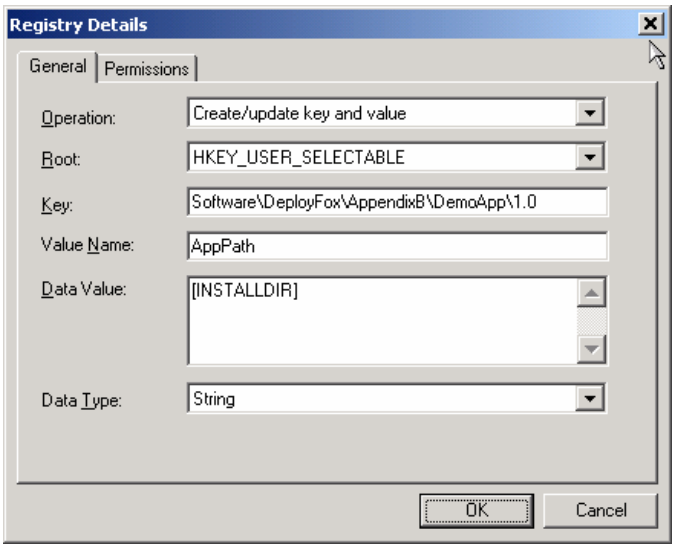

***Figure 14**. Add registry keys manually with the Registry Details dialog.*

Figure 14 shows how to add a key to capture the value of the folder where the user installs the product. The root key HKEY_USER_SELECTABLE shown in Figure 14 is a pseudo-key that resolves to HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE at installation time, depending on whether the users chooses a per-user or a per-machine installation. The data value [INSTALLDIR] is a Windows Installer property whose value is the destination folder selected by the user at installation time.

After filling in the necessary information in the Registry Details dialog, click OK to add the new registry key and value(s) to the setup project. When finished, the new registry entries show up in the destination computer portion (lower half) of the Registry page, as shown in **Figure 15**.

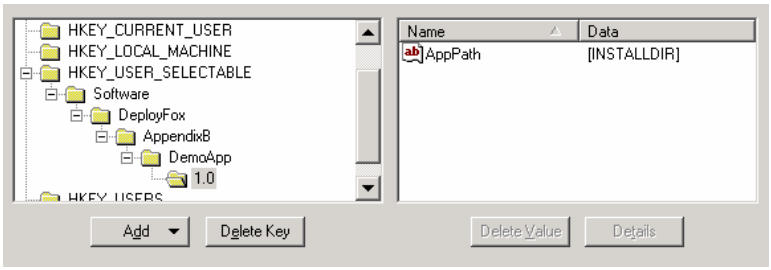

***Figure 15**. Registry keys to add at installation time appear in the destination computer portion of the Registry page.*

Shortcuts page

Use the Shortcuts page to define new shortcuts to be created on the user's machine. For the demo app, we want to create a shortcut to the main executable program, DEMOAPP.EXE, and place it on the user's desktop.

Like files and registry entries, shortcuts need to be associated with a feature in the setup package. To begin the process of creating a shortcut, first select the desired feature from the drop-down list at the top of the Shortcuts page. Click the Add button on the right. This launches the shortcut wizard, which, for a shortcut that points to a file, has three steps:

- Shortcut Type—the target of the shortcut can be a file in the installation or a command line entry
- Shortcut File—if the target is a file in the installation, select the file from the list of files associated with the selected feature (this step is omitted if the target of the shortcut is a command line entry)
- Shortcut Destination—choose the destination from the tree representing folders on the destination computer.

To create the desktop shortcut to DEMOAPP.EXE, first select the Main_Program feature from the drop-down list. Click the Add button and select *File in this installation* as the shortcut type, as illustrated in **Figure 16**. To create a conventional shortcut, clear the *Advertised* check box.

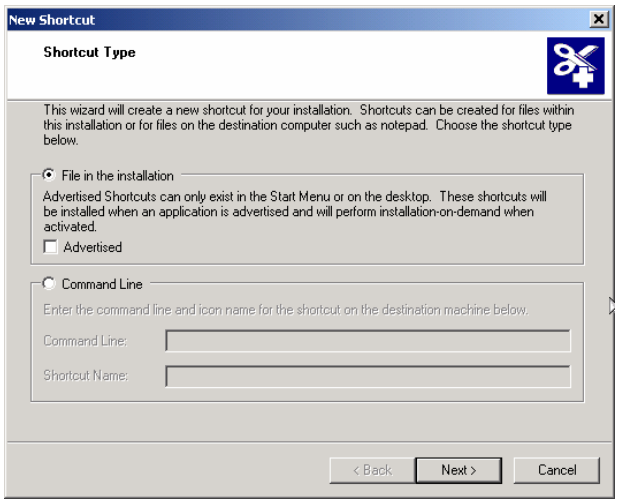

***Figure 16**: To create a shortcut to a file, choose "File in this installation" in the first step of the shortcut wizard.*

In step two of the shortcut wizard, select the shortcut's target file. Because the target is a file in the installation and because the Main_Program feature is selected, the file tree in step

two displays the files associated with the Main_Program feature, as shown in **Figure 17**. Select the DEMOAPP.EXE file as the target of this shortcut.

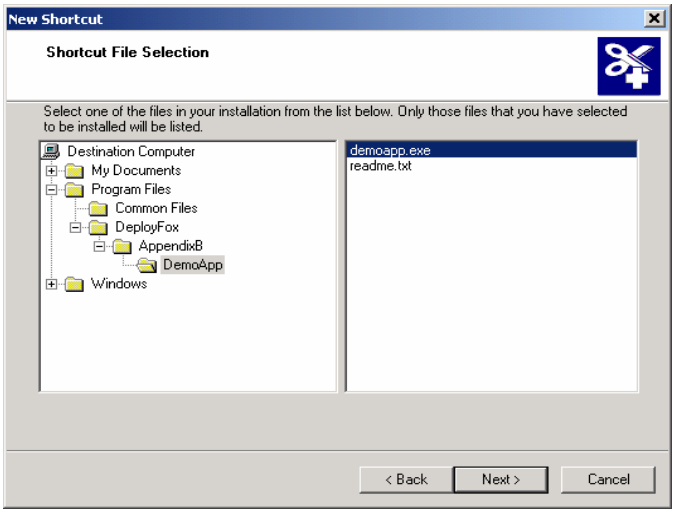

***Figure 17**: Select the shortcut's target file from the list of files associated with the selected feature.*

Step three of the shortcut wizard is where you select the destination folder for the shortcut. Typical choices are the Start menu in the Program folder or the Desktop folder, although other choices are available as illustrated in **Figure 18**.

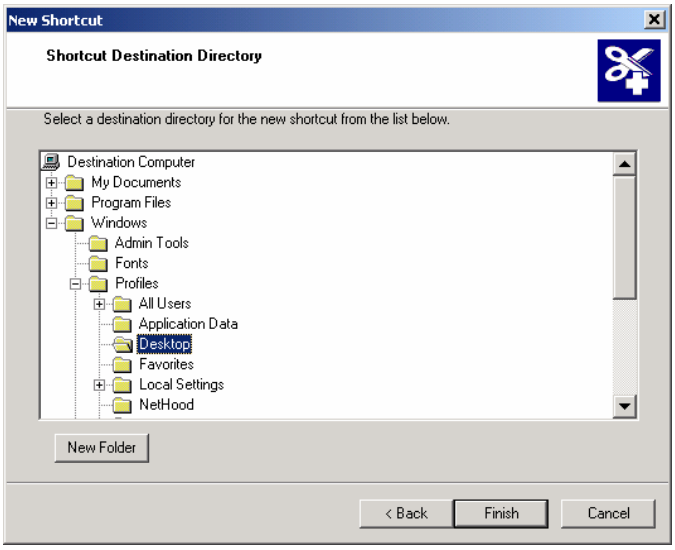

***Figure 18**. To place a shortcut on the user's desktop, choose Desktop under Windows\Profiles in step three of the shortcut wizard.*

When you click the Finish button after step three of the shortcut wizard, Wise for Windows Installer automatically opens the Shortcut Details dialog for the new shortcut. This dialog, shown in **Figure 19**, is the same dialog you get by selecting an existing shortcut and clicking the Details button.

The Shortcut Details page allows you to modify the settings for the shortcut and provides access to a few properties not found in the shortcut wizard. Among the additional properties of interest here are the Description and the Working Directory. The Description is optional and can be anything you want it to be. The Working Directory is more important, because the demo app expects to find its data (the DBF and CDX files) in a sub-directory called Data underneath the directory where the product is installed. In order for the program to work properly when launched from the shortcut, we need to make the Working Directory the directory where the product is installed.

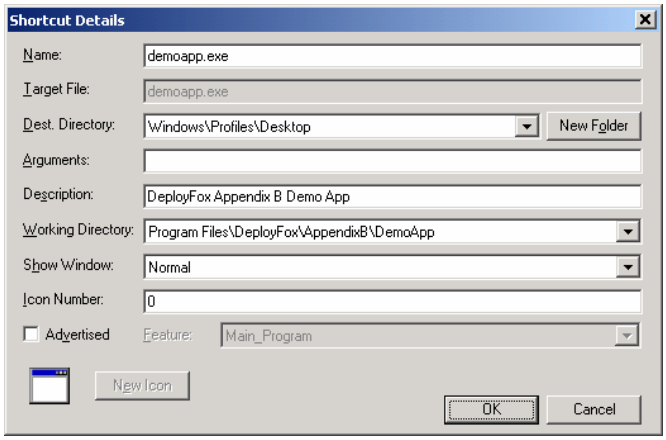

***Figure 19**. The Shortcut Details page provides access to the shortcut's properties.*

Clicking OK on the Shortcut Details dialog closes the dialog and adds the new shortcut to the list on the Shortcut page. Using the above sequence of steps you can add as many shortcuts as you like to the project. Don't forget to select the appropriate feature before adding each new shortcut.

ODBC page

If you want your setup to enable ODBC access to your application's VFP data, you probably want to install a DSN on the user's machine. An easy way to do this is to import a DSN you already created on your own machine. For the demo app, we created a system DSN on our development machine named DeployFoxSample. This DSN points to the demo app's CUSTOMERS.DBF table. In most real apps you would probably create a DSN for a database container (DBC) instead of for an individual DBF file, but the demo app does not use a DBC.

To add the DeployFoxSample DSN to the setup project, first select the ODBC page, and then select the feature to install the DSN with from the drop-down list. Because the DSN is relevant only if the data table is installed, we choose to install the DSN as part of the Data Files feature. Select the Data Files feature, and then click the Add button and choose *Data Source* from the drop-down menu. This opens the ODBC Data Source Details dialog.

In the ODBC Data Source Detail dialog you can create a new DSN or import an existing one. To import an existing DSN, click the Import button. This brings up the Select Data Source dialog that displays a list of DSNs registered on your computer, shown in **Figure 20**.

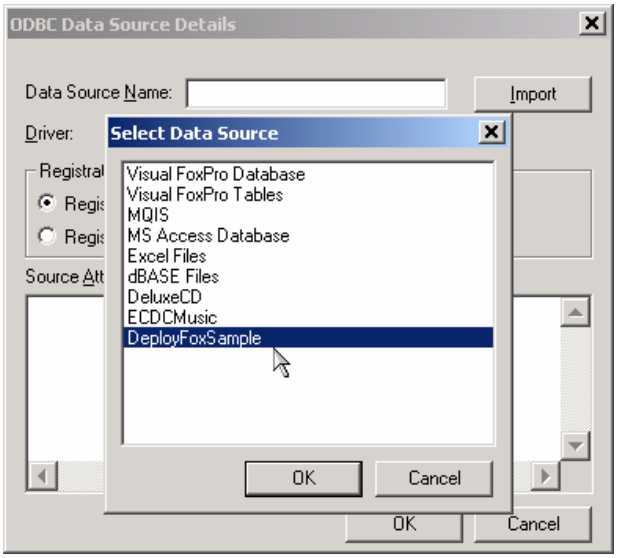

***Figure 20**. You can easily import a DSN by selecting it from the list of data sources registered on your machine.*

For the demo app select the DeployFoxSample DSN from the list. Closing the Select Data Source dialog populates the fields in the ODBC Data Source Detail dialog with the information from the DSN you're importing. The only thing to do manually is select per-machine or a per-user registration for the DSN. **Figure 21** illustrates the completed DSN. Clicking OK at this point adds the DSN to the project. Repeat this procedure for each DSN you wish to add.

To edit an existing data source entry, select it from the list of those shown on the ODBC page and click the Details button on the right. Wise for Windows Installer opens the ODBC Data Source Details dialog shown in **Figure 22**. The Source Attributes area in the lower portion of this dialog is an edit box where you can change the attributes associated with this data source.

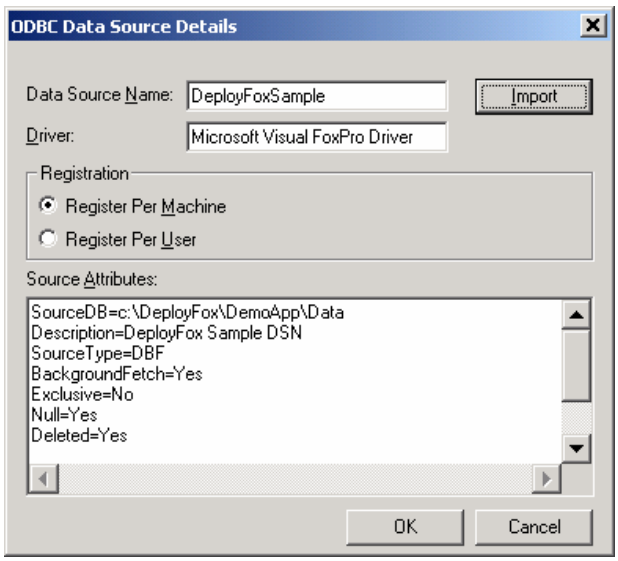

***Figure 21**. The ODBC Data Source Details dialog shows the information from the DeployFoxSample DSN we imported.*

To make your data source portable, change the SourceDB attribute from a literal value (like C:\DeployFox\DemoApp\Data, as shown in Figure 21) to a variable whose value resolves to the location of the data on the user's machine at installation time. In the DeployFox demo app, the data is installed in a sub-folder named Data under the folder where the application itself is installed. The Windows Installer property INSTALLDIR resolves to the folder where the application is installed, so changing the value of the SourceDB attribute to [INSTALLDIR]Data makes the DSN point to the correct sub-folder no matter where the user installs the app.

Visual FoxPro ODBC driver

If you install a DSN that uses the Visual FoxPro ODBC driver, you should also install the VFP ODBC driver itself. There is a merge module called VFPODBC.MSM for this purpose, but as of VFP 8, it is no longer installed with VFP. If you do not have this merge module on your computer from an earlier version of VFP, you can download it from the Microsoft MSDN Web site **at http://msdn.microsoft.com/vfoxpro/downloads/updates/default.aspx**. Look for the Visual FoxPro ODBC Driver download in the Visual FoxPro 6.0 section.

This concludes the discussion of the Feature Details group. At this point most of the work necessary to complete the setup project is done. There are a couple of other pages in the Feature Details group we did not talk about, and there are two other groups—the Target System group and User Interface group—we do not go into here because for many setup

projects you won't need them. Knowing what you now know about Wise for Windows Installer, you should have no difficulty exploring these other two groups on your own.

We should point out, however, it's necessary to include the User Information Dialog if you want to give your users the choice of installing for all users or only for the current user. This dialog is selected by default, and can be found on the Dialogs page of the User Interface group. On that same page is an option for a License Dialog, which by default is not selected.

Release Definition group

Once you define the features of your application and specify the files, merge modules, registry entries, shortcuts, and other resources your setup should install, you are ready to define and build the release package. The pages in the Release Definition group enable you to define the type of release you want to build, which features it should include, what media it's intended for, whether or not to create a SETUP.EXE, and other related choices.

Releases page

Wise for Windows Installer allows you to define more than one release in the same project. For example, you might want to build one release for network distribution and another for CD-ROM or Web distribution. Perhaps you created both a standard and a professional version of your product, with the professional version installing more features than the standard version. Each of these is a potentially different release.

The demo app, however, needs only one type of release. It is designed for distribution on a CD-ROM or for download over the Web, so we want to build it as a single EXE file. To define this release, first select the Releases page in the Release Definition group. In a new project the Releases page contains one pre-defined release named Default, as illustrated in **Figure 22**.

Although you can keep the name Default, we recommend changing it to something more meaningful. On the rest of the pages in the Release Definition Group you refer to releases by name. When there are several releases, meaningful names help you keep them straight.

The name and other attributes of a release can be changed in the Release Details dialog, shown in **Figure 23**. To bring up the Release Details dialog, select the release from the list and click the Details button on the right. In the Release Details dialog, enter the Release Name, MSI File Name, and Description for the release. In most cases you can accept the defaults for Installation Theme, Compression Type, and Release Type. Be sure the "Build this release during compile" is selected, and click OK to save these values.

Release Settings page

One of the main purposes of the Release Settings page is to enable you to select the features that install with each release. The Release Settings page also provides access to certain Windows Installer properties and summary information, which you do not ordinarily need to change. You can also edit the properties and summary information on the Summary and Properties pages under the Product tab of the Setup Editor view.

The Release Settings page of the demo app project is shown in **Figure 24**. Note the drop-down list at the top, with "Single_Image" selected as the current release. As you can see, all three features install with this release.

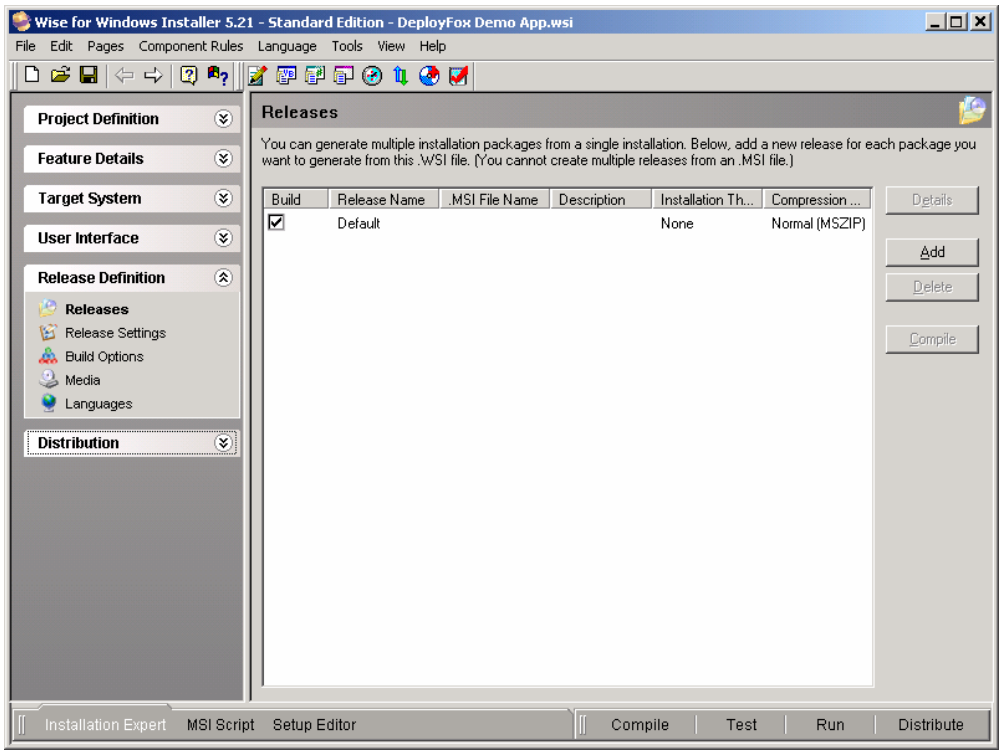

***Figure 22**. Each project must define at least one release. A new project automatically contains a release named Default, which you can modify to suit your needs.*

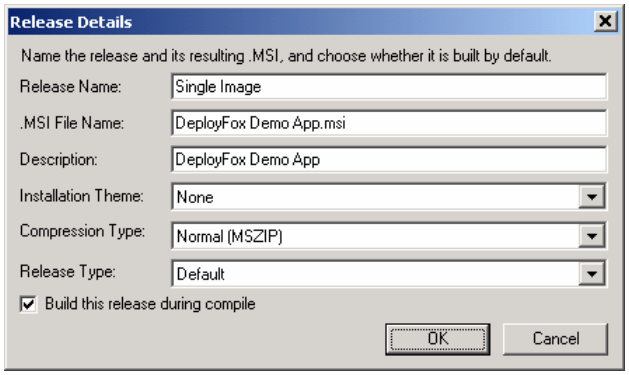

***Figure 23**. Use the Release Details dialog to specify the release name, MSI file name, and description for each release.*

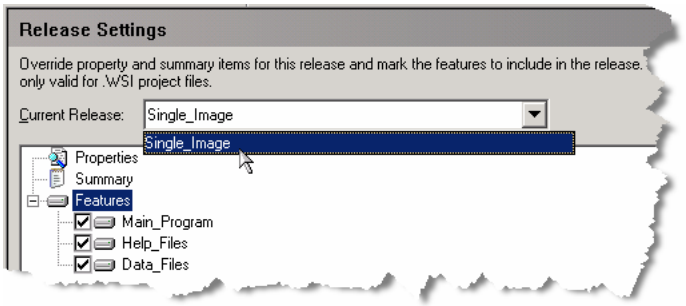

***Figure 24**. The Release Setting page lets you specify which features to install with each release.*

Build Options page

You make two important decisions on the Build Options page. The first is whether to build the release as a single self-contained SETUP.EXE, a SETUP.EXE with a separate MSI file, or just an MSI file. The advantage of creating a SETUP.EXE is the end user can simply run it to install the product. The additional advantage of a single self-contained SETUP.EXE file is you only need to distribute one file to your users. Because we want to be able to deploy the demo app over the Web as well as on CD-ROM, we chose the single EXE build type as shown in **Figure 25**.

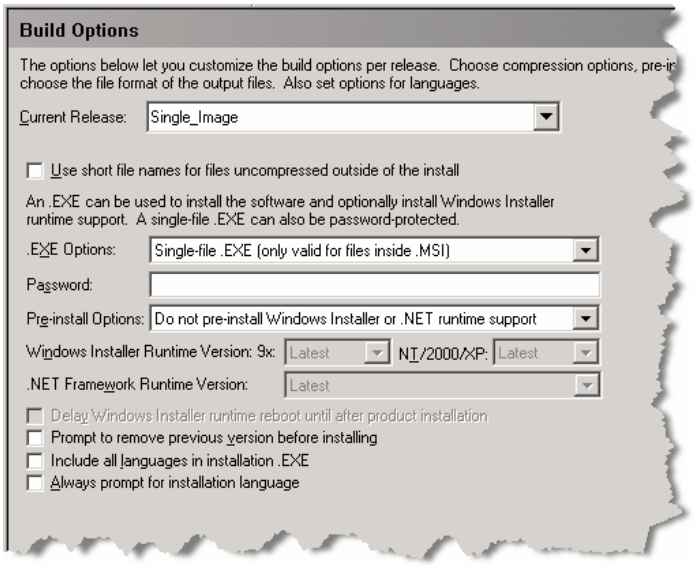

***Figure 25**. Use the Build Options page to determine the type of build (.EXE Options) and whether to install Windows Installer on Windows 9x/NT machines (Pre-install Options).*

The second decision you make on the Build Options page is whether to install Windows Installer on Windows 9x/NT machines. Windows Installer is an integral part of Windows 2000

and Windows XP. Windows Installer is also available as an add-on for older versions of Windows, but you can't be sure it's installed on all computers running those older operating systems. If your product will be installed on computers running Windows 95/98 or Windows NT, you may want to make your setup package install Windows Installer before installing your product. This is easy to do, and the only downside is your setup package is somewhat larger than it would be otherwise.

It's important to note the choice to pre-install Windows Installer as part of your setup is only available if you choose to create a SETUP.EXE. The option to pre-install Windows Installer is not available for the MSI-only build type. Figure 25 shows the various choices and options available on the Build Options page.

Media page

The Media page gives you control over how the product is packaged for the distribution media. Each project requires at least one media definition. In a new project, there is a pre-defined media entry named Basic. For small deployments the default values in the Basic media entry are probably sufficient and you won't need to make any changes. For larger products, such as those requiring more than one CD, you might want to exercise some control over the output media, such as setting different file compression options and controlling which features are written to which CD.

The Media page also enables you to specify the folder(s) on your machine where the WfWI compiler output is written prior to being copied to the distribution media. If you do not specify a destination folder for the compiler, the output files—the MSI file, the EXE file, and any external files being distributed—are written to the same folder as the project file. In our experience it is better to direct the compiler output to a folder of its own.

To demonstrate the use of the Media page, we changed the pre-defined name Basic to Single Image and specified the output files be written to a sub-folder called Release Files under the project folder. Make these changes in the Media Details dialog shown in **Figure 26**. Open the Media Details dialog by selecting a media row and clicking the Details button.

In the Media Details dialog you can edit the Media Name field directly. The Compression Option and Custom Media Settings areas of this dialog are disabled if you chose to create a single-file .EXE under Build Options. You can add a new destination directory by clicking the Add button to the right of the Media Destinations portion of the dialog. In Figure 26, you can see we added the directory C:\DEPLOYFOX\WISE SETUPS\RELEASE FILES. The Features/Components section of this dialog shows that all features of the product are included in this media definition.

Languages page

On the Languages page you can change the language used in the dialogs presented during installation. Choices included with the Standard edition include French, German, Italian, Portuguese, and Spanish. You can even create a multiple-language release. If you are building an English-only release, however, you do not need to change anything on the Languages page.

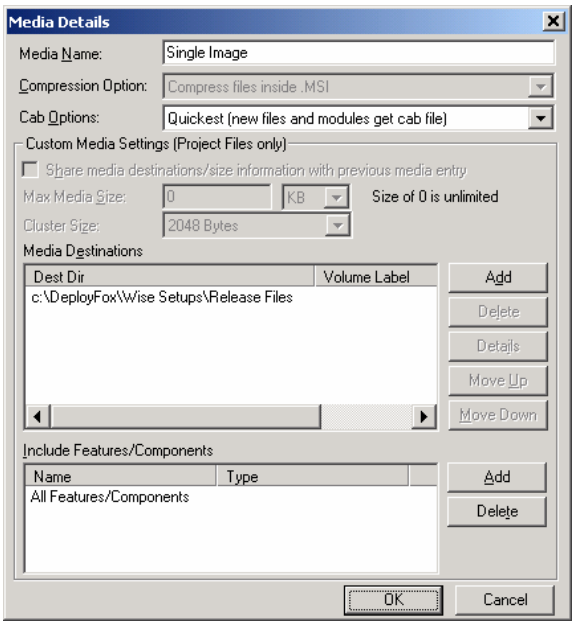

***Figure 26**. Use the Media Details dialog to provide the Media Name and to specify the destination directory for the media output file(s).*

Building the deployment package

Once the release is defined and the project is complete, you need to compile it into the setup package you deploy to your users. You probably also want to test the setup on your development machine and perhaps even install it on one or more test machines before deploying it to your customers.

Compiling the project

Compiling a project is as simple as clicking the Compile button located in the button group along the lower right of the WfWI window, shown in **Figure 27**.

***Figure 27**. Click the Compile button to build the release package.*

Compiling the project creates the output file(s) specified by the release(s) you defined in the project. In the case of the demo app, the output from the compiler is a single-EXE setup file named DEPLOYFOX DEMO APP.EXE, which, on our development machines, is written to C:\DEPLOYFOX\WISE SETUPS\RELEASE FILES per the media destination directory we established in Figure 26. To facilitate the creation of future upgrades or patches, WfWI also creates an MSI file even if you're building a single-EXE setup file. The MSI file is written to

the same folder as the EXE file. The MSI file for the demo app is named DEPLOYFOX DEMO APP.MSI.

If you build a single-EXE setup you do not need to distribute the MSI file; you only need to distribute the EXE file. If you build a release other than a single-EXE setup file, you need to distribute the MSI file and any other files created in the compiler output directory.

Testing the setup

Wise for Windows Installer provides a mechanism for you to test your newly compiled setup package without actually installing the product on your machine. To test your setup, simply click the Test button shown in Figure 27. Testing a setup runs Windows Installer and steps through the entire setup process as the end user sees it, but because no files or other resources are actually installed, your development machine is unaffected. This is especially useful in cases where running the actual setup on your development machine could have undesirable consequences, such as overwriting files, creating registry entries, changing DSN properties, and so on. Using the Test button allows you to verify the setup works the way you want it to without disturbing anything your own computer. It also gives you a chance to review the installation dialogs the way the user sees them, to quickly review any changes you may have made, and to document the install process for the user.

Running the setup

Of course, you can run the actual installation of the product on your development machine if you want to. To do this from within Wise for Windows Installer, simply click the Run button shown in Figure 27. Unlike the Test button, clicking the Run button actually installs the application on your machine: it creates folders, writes files, adds registry entries, creates shortcuts and ODBC resources, etc. Of course, you can also run the installation on your machine by running the setup EXE file from Explorer or the Run command window.

Distributing the setup

Once the release of the product is successfully compiled and tested, deploying the product is simply a matter of distributing the appropriate setup package. Common methods of distribution include Web-based download, distribution on a CD-ROM or DVD, and network deployment. As you worked your way through the Release Definition group, you already had to give some thought to how you intended to distribute your product, so this shouldn't be a new decision by the time you reach this step.

Wise for Windows Installer provides a Package Distribution tool to assist you in deploying your setup package. Launch the Package Distribution tool by clicking the Distribute button shown in Figure 27. The first step of the Package Distribution tool asks you to choose the type of distribution you want to perform. The various choices are shown in **Figure 28**.

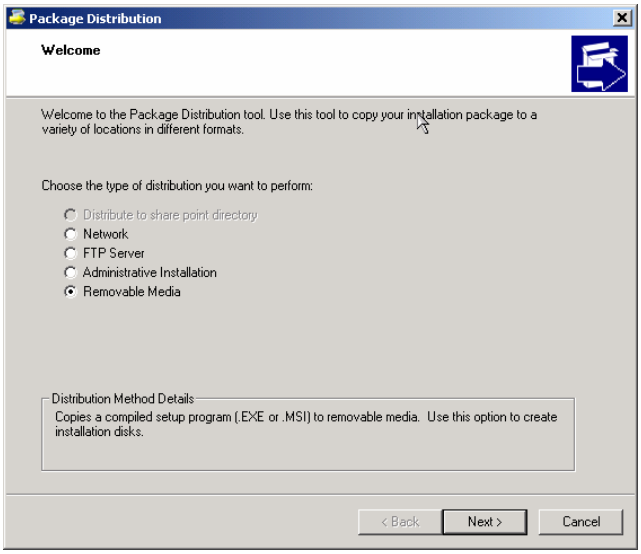

***Figure 28**.The Package Distribution tool helps you deploy your setup package.*

In the case of the demo app, we built a single-EXE setup package. This type of setup is suitable for distribution over the Web or on a CD-ROM. The Package Distribution tool provides FTP-based distribution assistance if you want to copy your setup package to a Web server. It also provides assistance for copying the setup package to a CD-ROM, DVD, Zip disk, or other removable media. This is what we want to do for the demo app, so the Removable Media choice is selected in Figure 28.

The next step of the Package Distribution tool depends on the type of distribution you chose in the first step. If FTP distribution to a server was chosen, the next step prompts you for the server URL, login name, password, and directory on the server. If Removable Media was chosen, the next step prompts you for a drive letter, disk label, and setup file name. Regardless of the type of distribution, WfWI automatically compiles the project after the first step if necessary.

The second step of the Package Distribution tool for a Removable Media type of distribution is shown in **Figure 29**. If you are copying your setup package to a CD-ROM, choose the drive letter of your CD-ROM drive as the Destination Disk.

Optionally, you can provide a Disk Label and a Setup File Name. The Setup Filename is the final name of your setup file on the distribution media. You do not need to enter a file name extension. If you leave the Setup Filename field blank, WfWI uses the name of the currently open project file. Note the Erase Disk Before File Copy check box is selected by default.

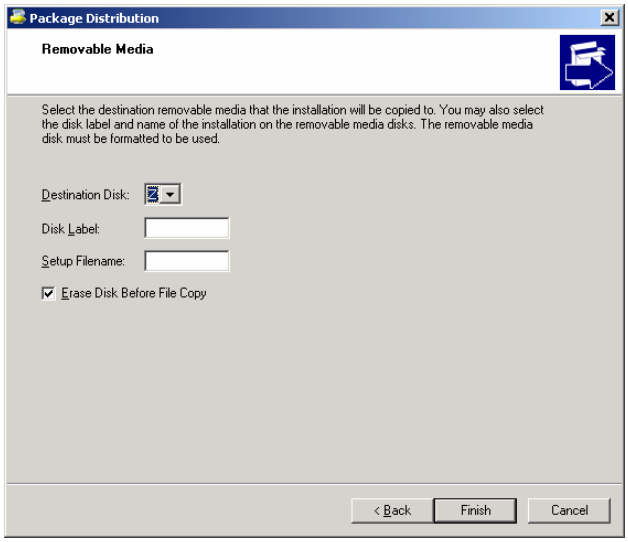

***Figure 29**. The final step is to specify the destination disk where your setup package is to be copied.*

Upgrades and patches

In virtually all real world situations, at some point you need to deploy an update to a product you already deployed previously. In the world of Windows Installer, there are two ways to distribute an updated version of an existing product. One way is to create a new setup package in the form of an upgrade. The other way is to create a patch.

As explained in Chapter 5, "Windows Installer Inside and Out," there are three types of upgrades: small updates, minor upgrades, and major upgrades. A small update and a minor upgrade replace files from the earlier version with updated files from the newer version. A major upgrade uninstalls the previous version of the product and installs the updated version. Wise for Windows Installer enables you to create any of these three types of update packages.

Also as explained in Chapter 5, a patch is not a different kind of update, but rather a different kind of update package. The main advantage of a patch is that it is usually much smaller than the corresponding full upgrade package would be. WfWI also enables you to create a patch package.

Windows Installer has rules governing the changes required to an MSI file in order to configure it as a small update, a minor upgrade, or a major upgrade. WfWI comes with a tool called UpgradeSync to make these changes for you. Use of the UpgradeSync tool is optional, but if you don't use it, you're on your own to make the required changes. Refer to "Using UpgradeSync" in the Wise for Windows Installer Help file for more information.

Creating an upgrade

Upgrades can be created only for products where the earlier version was also installed using Windows Installer. Creating an upgrade is much easier if you have access to the MSI file from the earlier version of the product. If you do not have access to the MSI file from the earlier

version of the product, you at least need to know some of the information it contains, such as the Upgrade Code and the Version number.

To create an upgrade in Wise for Windows Installer, first create a setup project for the new version of the product just as if you were going to run it as a fresh install. An easy way to do this is to open the WfWI project file for the earlier version and save it with a new name. For example, to create a project file for version 2.0.0 of the DeployFox Demo App, we can open the project file for the original version (1.0.0) of the product, which is named DEPLOYFOX DEMO APP.WSI, and save it as DEPLOYFOX DEMO APP 200.WSI.

Naturally, there are changes to make in the new project file before compiling a release. Among the most important changes for a major upgrade is to update the Version number and to generate a new Product Code GUID. You can use the UpgradeSync tool to make these changes for you, or you can make these changes manually on the Product Details page by typing in a new Version number and clicking the Generate button to insert a new Product Code field. **Figure 30** shows the updated Product Details page for version 2.0.0 of the demo app. If WfWI asks you whether to also generate a new Upgrade Code GUID, reply NO. A common Upgrade Code is what enables Windows Installer to identify an existing version of the same product on the user's computer at installation time.

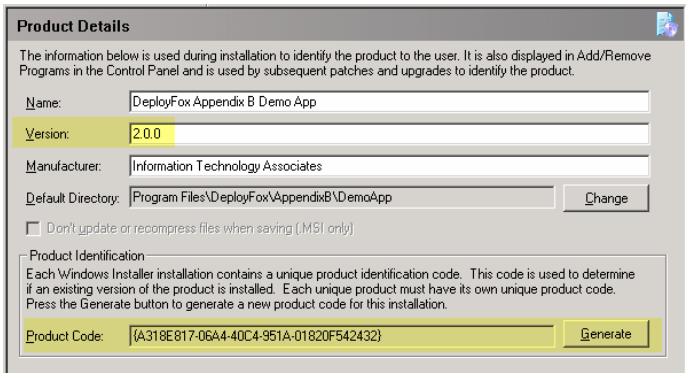

***Figure 30**. To create an upgrade, update the Version number and generate a new Product Code GUID.*

After changing the Version number and Product Code on the Product Details page, step though each of the remaining pages in the Installation Expert and make whatever other changes are necessary for the updated version. At a minimum, the updated app typically includes a new version of the EXE file and maybe an updated ReadMe or Help file. Some updates of course may involve much more extensive changes.

When preparing an upgrade you should also go to the Release Details page and specify a unique MSI file name, such as DEPLOYFOX DEMO APP 200.MSI, and a unique description for the new version. Making the MSI file name unique is particularly important if the compiler output for the update is going to be written to the same location as the compiler output from the original release. Unless you use a unique name the original files will be overwritten the first time you compile the new version. Even if you don't want the original files for deployment any more, you're still going to want them as the basis for future upgrades or patches, so it's a good idea to keep them intact. See Chapter 9, "Support and Ch-Ch-Changes," for more information on this topic.

The key to making the new setup package function as an upgrade is to add one or more entries to the Upgrades page specifying the earlier version(s) of the product the new version is designed to replace. The Upgrades page is found in the Installation Expert's Distribution group. This page is initially empty. To add a new entry, click the Add button to the right of the page detail area.

The first step in creating an upgrade entry is to select the MSI file of the earlier release of the product. WfWI prompts you to do this as soon as you click the Add button on the Upgrades page. In the case of the demo app, the MSI file for the earlier version of the product is the file DEPLOYFOX DEMO APP.MSI created when we compiled the original version of the product. On our development machine, this file is in the C:\DEPLOYFOX\WISE SETUPS\RELEASE FILES folder. As soon as this MSI file is selected, WfWI opens the Upgrade Details dialog and populates it with information from the selected file, as shown in **Figure 31**.

The Upgrade Code and Minimum Version values shown in Figure 31 are from the original version's MSI file. The Maximum Version value of 2.0.0 is the version number of the current project. Note the *Include minimum version in range* check box is selected, while the *Include maximum version in range* is not selected. This tells Windows Installer the current upgrade replaces all earlier versions starting with and including version 1.0.0, up to but not including version 2.0.0. Effectively, this means the current upgrade replaces all 1.x.x versions of the same product.

The Upgrade Action portion of the Upgrade Details dialog allows you to define features contained in the earlier version of the product that this upgrade is removing. By default, a major upgrade removes all features of the earlier version and installs the new ones. If you want to preserve any features from the earlier version, you must limit the removal of features to those specified in the *Features to remove* field. The spelling of feature names in this comma-separated list must exactly match the spelling of the feature names in the original setup. This is another good reason to keep the old installation files.

The Upgrade Action portion of this dialog also defines a Windows Installer property with a value set at installation time to the Product Code of any earlier version of the product found installed on the target computer. Custom actions can then be made conditional on the value of this property. Use of this property is optional, and custom actions are not required.

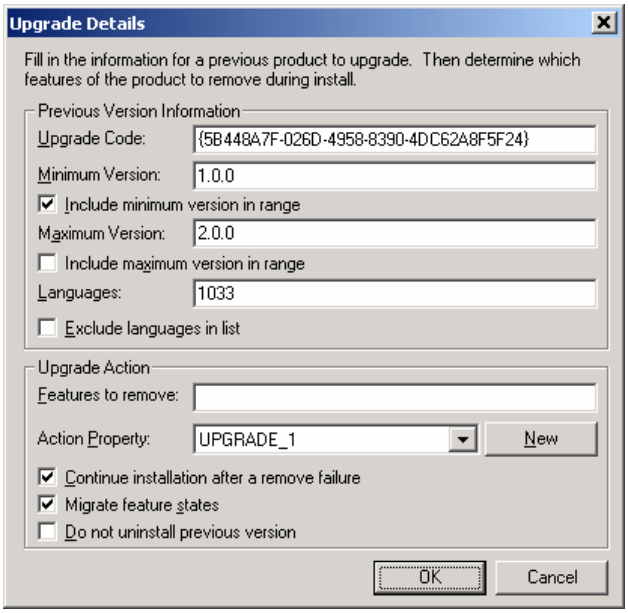

***Figure 31**. Wise for Windows Installer fills in the Upgrade Details page with information it reads from the original MSI file.*

Finally, the Upgrade Actions portion of this dialog provides three check boxes you can use to further control the behavior of the upgrade at installation time. If selected, the *Continue installation after a remove failure* check box tells Windows Installer to go ahead with the installation of the upgrade even if the uninstallation of the earlier version fails in part or in whole. Select the *Migrate feature states* check box if you want the upgrade to install only those features installed with the original version of the product. Select the *Do not uninstall previous version* check box if you do not want the upgrade to uninstall a previous version of the product; this allows two versions of the product to co-exist on the same machine, assuming you used unique folder and file names where necessary to make this possible.

After completing the Upgrade Details page, click OK to add the entry to the Upgrades page. Now compile, test, and distribute the setup package as before.

Creating a patch

A Windows Installer patch file (MSP file) for any given product is essentially the difference between two Windows Installer setup packages (MSI files) for the same product. Like an upgrade, applying a patch to an existing version of a product on the user's machine requires the existing version to have been installed using Windows Installer. Unlike a major upgrade, a patch requires an earlier version of the product to exist on the target machine. A major upgrade functions like a fresh install if no earlier version of the product is installed.

A patch can update one or several earlier versions of a product. Creating a patch requires you have access to the MSI file for each of the earlier versions the patch is designed to update. Depending on how the earlier releases were built and whether there are previous patches, you

may also need access to other files. Refer to "What You Need to Create a Patch" in the Wise for Windows Installer Help file for more information.

In the previous section, we built a major upgrade package to update version 1.0.0 of the demo app to version 2.0.0. In this section we build a patch package to accomplish the same thing. The first step is to create the installation package for version 2.0.0, just as we did in the previous section. If you have not already done so, create the setup project for the updated version, compile it to generate the MSI file, and then close the project.

Now you are ready to begin the actual patch creation process. Select Tools | Patch Creation from the main menu to launch the Patch Creation wizard. In the Patch Creation wizard, choose "Create a new patch file." The next step asks you to identify the MSI file(s) for the previous version(s) of the product. In the case of the demo app, there is only one earlier version. Its MSI file is the original DEPLOYFOX DEMO APP.MSI file, located in C:\DEPLOYFOX\WISE SETUPS\RELEASE FILES on our development machine. **Figure 32** shows the Patch Creation wizard dialog for this step.

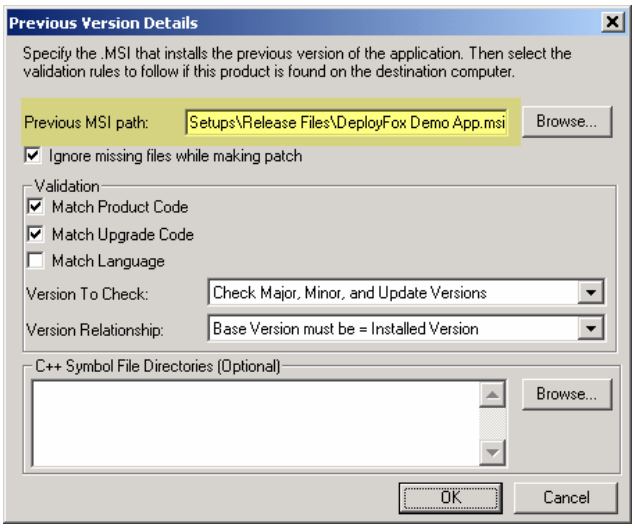

***Figure 32**. Creating a patch requires you to specify the MSI file for the original version of the product.*

The Window Installer patch creation process needs to work with uncompressed base files. If the original setup contains compressed files, which is often the case in order to make the setup package as small as possible, those files need to be uncompressed for use in creating a patch. If the Patch Creation wizard detects the original MSI file contains compressed files, it offers to perform an administrative installation of the product on your machine. An administrative installation creates an uncompressed copy of the files in the setup package without actually installing the product. To create a patch, you must allow the Patch Creation wizard to perform the administrative installation of the product if it prompts you to do so. The uncompressed files are written to a temporary location on your computer.

The next step is to specify the MSI file for the new version of the product. In the case of the demo app, we select the MSI file for version 2.0.0 we built earlier, as shown in **Figure 33**.

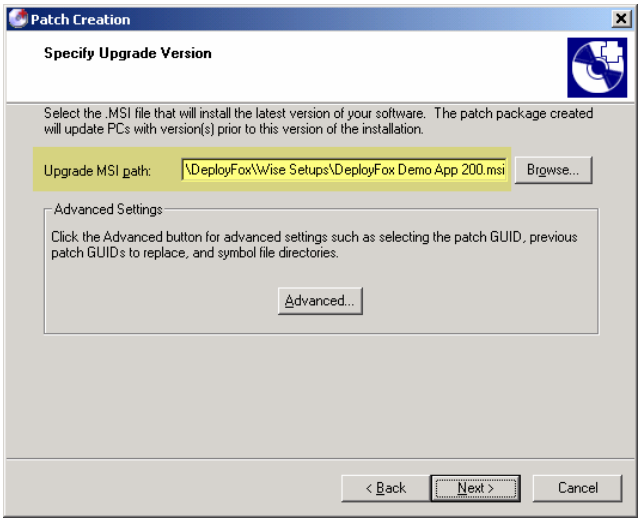

***Figure 33**. Creating a patch also requires you to specify the MSI file for the updated version of the product.*

Because the MSI for version 2.0.0 of the demo app also contains compressed files, the Patch Creation wizard prompts you to perform an administrative installation of this version, too. As before, this is required to complete the patch creation process. The uncompressed files for version 2.0.0 are written to a different temporary location than the uncompressed files for version 1.0.0, resulting in both sets of uncompressed files being present on your machine. This is necessary because the patch creation process needs access to both sets of uncompressed files in order to create the patch file.

*While a patch package is smaller than a full upgrade package, the process of creating a patch package can consume significant amounts of disk space on the developer's machine. The administrative installation creates uncompressed copies of **all** the files in the setup package. This includes not only your application's own files but also the VFP 8 runtime files, ActiveX files, and any other files installed by your product. Even in the case of a simple product like the demo app, this amounts to over 15MB. Because there are two versions involved in this particular patch creation process, more than 30MB of disk space is required to support creation of this patch file. If the patch updates more than one earlier version of the product, uncompressed images of each of those other versions would also be required, consuming even more disk space.*

The last step before compiling the patch is to specify a name and location for the patch file and to specify values for other settings that determine how the patch file is created. This is done in the Compile Patch dialog, illustrated in **Figure 34**.

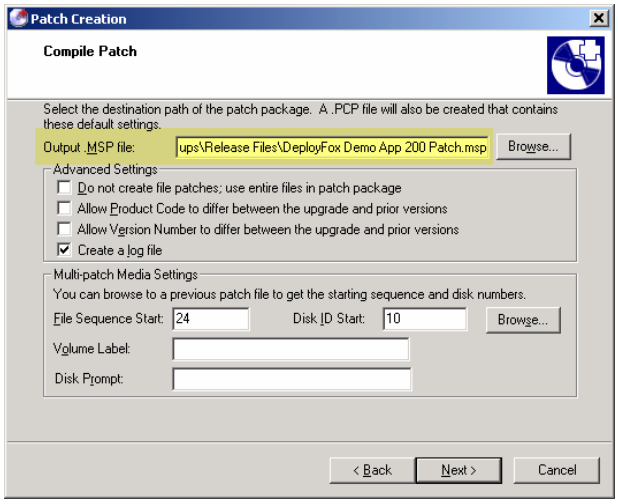

***Figure 34**. You can specify the name for the patch (MSP) file as well as choose other settings to determine how the patch file is compiled.*

In Figure 34 we specified DEPLOYFOX DEMO APP 200 PATCH.MSP as the output patch file name, and indicated it should be written to the same Release Files folder as the other release files built for this app. The other settings all have their default values, which are appropriate for this and most other patches.

Clicking the Next button at this point starts the patch compilation process. This process can be somewhat time-consuming. Upon completion, and assuming no errors, the MSP file is written to the location you specified (see Figure 34). To deploy the patch, distribute the MSP file to your users. To install a patch from an MSP file, right click on the file in Explorer and choose Apply from the shortcut menu.

In addition to the MSP file, a patch creation project (PCP) file is also created. This file is a necessary intermediate file used in the patch creation process. You do not need to do anything with this file.

If the *Create a log file* check box was selected in the Compile Patch dialog, a log file is created with the same file name as the MSP file, but with a LOG file name extension. You can open the log file in any text editor and read the summary of the patch creation process.

In the demo app, the only difference between version 1.0.0 and version 2.0.0 app is the version number changed in the EXE file. The EXE file by itself is 603KB. The full upgrade package for version 2.0.0 of the demo app is 7.77MB, but the patch file for the same upgrade is about 98% smaller at 119KB. In a real application it's likely there would be more of a difference between the two versions, therefore resulting in a large patch file, but this example should give you some idea of the size advantage of a patch as compared to a full upgrade package.

Conclusion

Wise for Windows Installer is a powerful tool for building Windows Installer setup packages. As you have seen in this appendix, it is capable not only of building setup packages for the

initial deployment of a product, but also of building upgrades and patches, which are an important part of a product's life cycle. The Installation Expert hides some of the complexity of Windows Installer and makes it easy to create setup packages. On the other hand, Wise for Windows Installer also lets you dig into the details of a Windows Installer setup package when necessary using the Setup Editor and the MSI Script view, which we did not go into here. WfWI can also open Windows Installer MSI files directly, allowing you to explore these files in a manner similar to what you can do with Orca as described in Chapter 5, "Windows Installer Inside and Out." In summary, Wise for Windows Installer is a full-featured product for creating and working with Windows Installer setups.

Updates and corrections for this appendix can be found on Hentzenwerke Publishing's Web site, **www.hentzenwerke.com**. Click 'Catalog' and navigate to the page for this book.