

Installing Applications Using Inno Setup

Doug Hennig

Stonefield Software Inc.

Email: dhennig@stonefield.com

Web site: <http://www.stonefield.com>

Web site: <http://www.stonefieldquery.com>

Blog: <http://doughennig.blogspot.com>

Overview

Although a limited edition of InstallShield Express comes with VFP, many developers are unhappy with Windows Installer-based installation tools: the tools are slow and cumbersome, the resulting SETUP.EXE files are huge and slow, and Windows Installer sometimes has a mind of its own. Introducing Inno Setup, a free, fast, easy-to-use installation tool that generates fast, lightweight setup executables. This document shows you how to quickly implement Inno Setup to create great setups for your applications.

You will learn:

- How Inno Setup works
- The basics of creating Inno Setup scripts
- How to obtain and use add-ons that provide extended features
- Advanced Inno Setup scripts

Introduction

Although it's sometimes a last-minute consideration, deploying your application is an extremely important part of the development process. After all, if the user can't install your software, you won't get paid. The installer is often the first interaction the user has with your software; if something goes wrong, it gives them a bad impression about the software and your company in general. Also, problems during installation can be the cause of a lot of technical support calls.

Although a free installer tool, a limited version of InstallShield Express, has shipped with VFP for several versions, it may not be the best choice to use for your software. This document introduces Inno Setup, a free installer tool written by Jordan Russell.

Why I Hate Windows Installer

Perhaps "hate" is too strong a word, but I've had my fill of using Windows Installer-based installation tools (primarily InstallShield Express). Here are the issues I ran into and why I like Inno Setup better:

- The setup executables generated are quite large, especially if you include the Windows Installer itself in the package (perhaps not as necessary as in past years, but you never know when a user will install on an older operating system that doesn't have Windows Installer installed). The executables created by Inno Setup are half the size (your mileage may vary).
- Everything about Windows Installer is slow. It seems to take forever to start InstallShield Express, it takes a long time to generate a setup executable, the setups take quite a while to start (while Windows Installer initializes), and uninstallation is slow. Inno Setup starts up instantly, generates executables somewhat faster, there's no initialization required so the executables start immediately, and uninstallation takes only a few seconds.
- Windows Installer leaves what I call "poops" on your system: Registry entries and files all over the place. I understand why it does that—fixing broken installations is one of the features of Windows Installer—but that doesn't make it any more palatable.
- Related to the previous point: have you ever installed a new application only to find that running an old application now brings up a dialog asking to install it or a completely different application? That happens because when updated ActiveX controls are installed, their InprocServer setting in the Windows Registry, which points to the physical location of the OCX file, is overwritten with a Windows Installer encrypted key. For some reason, when you run an older application that uses those ActiveX controls, Windows Installer thinks there's something wrong with the ActiveX control and so prompts you to reinstall the application. Fixing it is easy if you know which ActiveX controls are involved—hack the Registry and change InprocServer back to the correct value—but figuring out which ones are the culprits may be a challenge.
- Windows Installer uses a database of settings. While that isn't inherently bad, given that we're database people, the database is in a proprietary format, so you have to use third-party tools like InstallShield Express or Orca to maintain the database. The database structure is fairly complex and the tools aren't the easiest things to use.
- The deal-breaker for me: Windows Installer sometimes appears to have a mind of its own. We had a situation where new installs of Stonefield Query for ACT! (one of the products my company makes) worked just fine but upgrades for existing installs didn't; one of our DBFs kept getting corrupted as part of the installation. It was likely some type of versioning issue or something we didn't do quite right, so perhaps it isn't completely fair to blame the installer for this, but I don't want to make it my life's work to figure out every nuance of Windows Installer; I just want to build a simple installation package and move on to more important tasks.

Introducing Inno Setup

Unlike Windows Installer, Inno Setup is a fast, lightweight installation tool. Rather than using a proprietary database for installation settings, it uses script files, which are just text files similar in structure to INI files. Although most people use the graphical interface that comes with Inno Setup, the compiler is actually a command-line application;

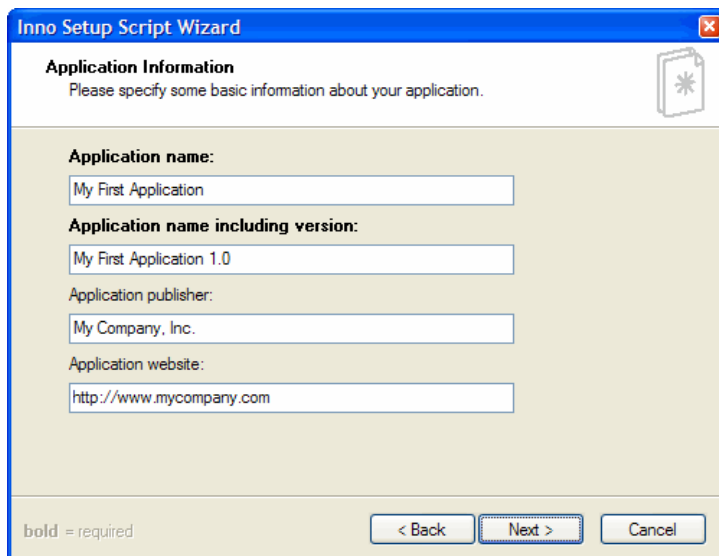
parameters passed to it specify how to build a setup executable. The Inno Setup editor is really just a text editor with a few additional tools, such as access to the Inno Setup compiler. You don't have to use it if you don't want to; you can use any text editor, or even generate Inno Setup scripts programmatically, and run the compiler from a command window.

Installing Inno Setup

The home page for Inno Setup is <http://www.jrsoftware.org/isinfo.php>. From this page, choose the Download Inno Setup link to access the downloads page, and click the link to download the Inno Setup installer. Be sure to also download and install the Inno Setup QuickStart pack, as it contains two invaluable add-ons for Inno Setup, the ISTool script editor and the Inno Setup Preprocessor (ISPP).

Creating Inno Setup Scripts the Easy Way

Let's create our first script, Sample1.ISS. Run Inno Setup and click the New button in the toolbar. This launches the Inno Setup Script Wizard (although that can be turned off in the Tools, Options dialog so New simply creates an empty script). The first step after the welcome screen, shown in **Figure 1**, requests information about the application and your company; required items are shown in bold.



The screenshot shows the 'Inno Setup Script Wizard' window with the 'Application Information' step selected. The window title is 'Inno Setup Script Wizard'. Below the title bar, there is a sub-header 'Application Information' and a prompt: 'Please specify some basic information about your application.' The form contains several input fields: 'Application name:' with the value 'My First Application'; 'Application name including version:' with the value 'My First Application 1.0'; 'Application publisher:' with the value 'My Company, Inc.'; and 'Application website:' with the value 'http://www.mycompany.com'. At the bottom of the window, there is a legend 'bold = required' and three buttons: '< Back', 'Next >', and 'Cancel'.

Figure 1. The Application Information step asks for information about the application and your company.

Specify options for the application directory on the target system in the next step (**Figure 2**). The default base directory is the Program Files directory on the target system, but you can choose Custom from the combo box and enter a different base directory if you wish. The default for the application's directory name is the application name you entered in the previous step, but you can of course change that. You can also specify whether the user is allowed to change the directory or not or whether the application even needs a directory (you might turn the latter option off if, for example, your setup simply installs files in the Windows System directory).

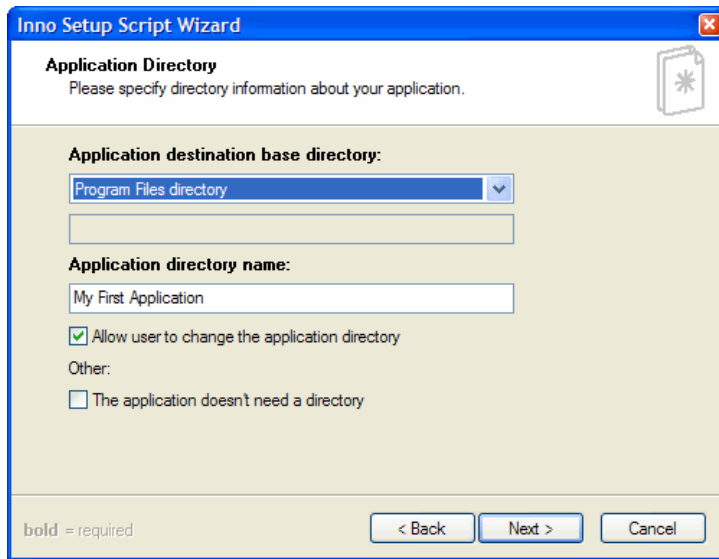


Figure 2. Specify the base and application directories in this step, as well as some directory-related options.

In the next step, shown in **Figure 3**, you specify the files to install: the main executable (if there is one) and other required files. There's a setting that specifies whether a "Run this application" option is available in the setup executable or not.

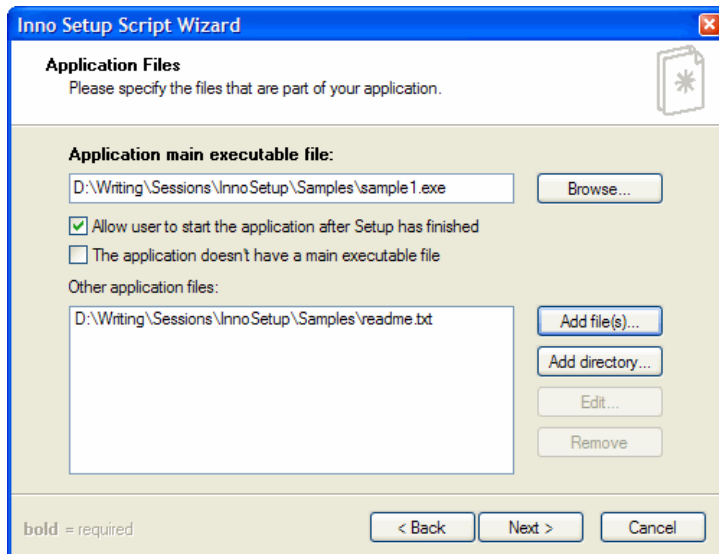


Figure 3. Specify the files the setup should install on the target system.

The Application Icons step (see **Figure 4**) allows you to specify what folder in the Windows Start Menu the installer should create and whether the user can change the name of that folder or disable its creation. You can also indicate that it should create a shortcut to your company's Web site and an uninstall icon in the Start Menu folder. Two other options allow you to specify whether the setup has options allowing the user to create icons for your application's main executable on their desktop or Quick Launch bar.

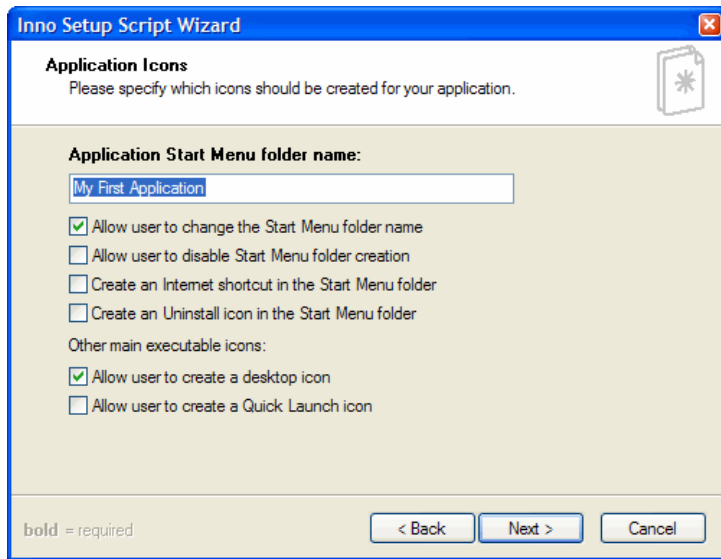


Figure 4. The Application Icons step allows you to specify the name of the Start Menu folder for the application and which icons the installer should create.

You can specify a file that contains license information to display to the user in the next step (**Figure 5**); this file can contain either text or Rich Text Format (RTF), which allows you to display nicely formatted text including colors and bolding. You can also specify files containing pre- and post-installation information; I find the pre-installation information useful for telling the user about the different types of installs they can perform (complete versus workstation-only, for example).

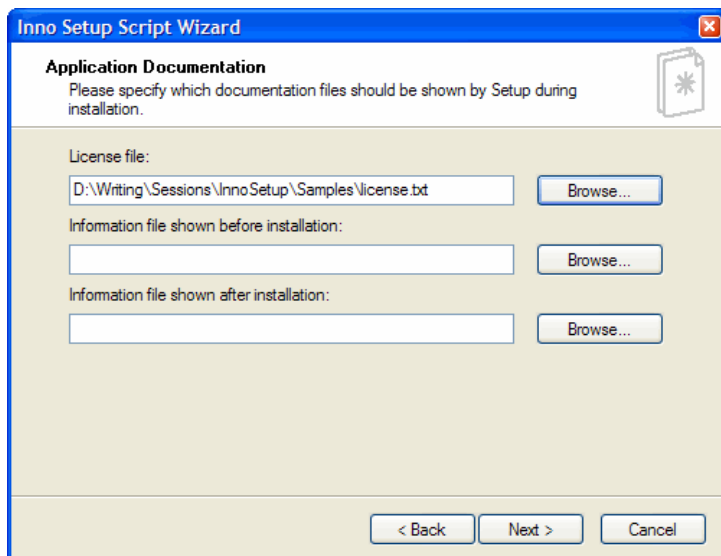


Figure 5. In this step, you can specify license, pre-, and post-installation information to display to the user.

If you have customers who speak languages other than English, you'll like the next step, shown in **Figure 6**: it allows you to create a unilingual or multilingual setup by choosing the appropriate languages to use.

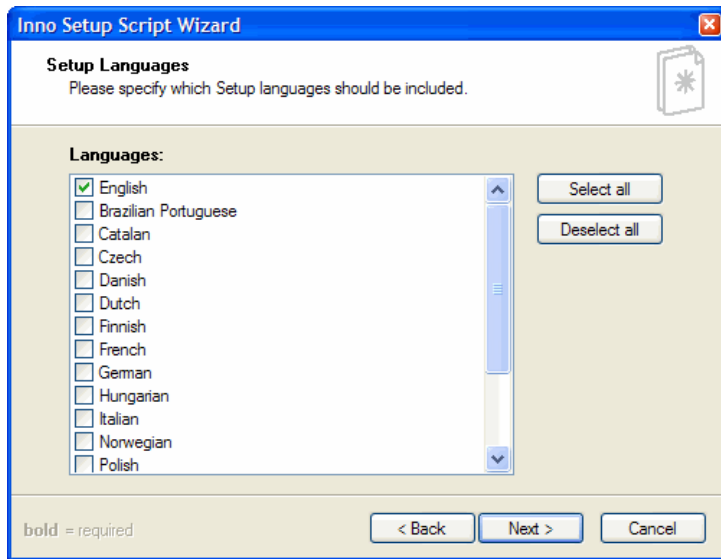


Figure 6. You can create multilingual setups using the Setup Languages step.

The Compiler Settings step (**Figure 7**) allows you to specify the path and file name for the setup executable created when you run the Inno Setup compiler; if you don't specify a path, the compiler places the executable into the Output subdirectory of wherever the script file is located. You can also specify an icon to use for the executable, a password the user must enter to run the installer, and whether that password is used to encrypt the application's files within the setup executable.

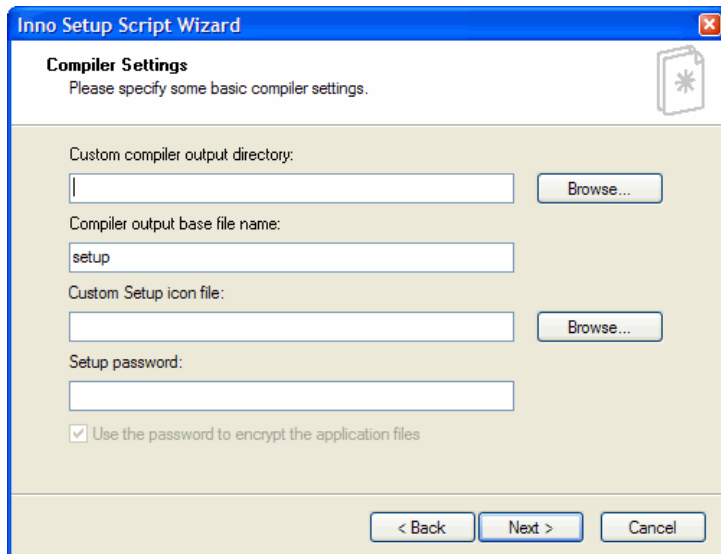


Figure 7. You can specify where the setup executable is created, which icon it uses, and a password for the installation.

The last step, shown in **Figure 8**, only appears if you have the Inno Setup Preprocessor (ISPP) installed. I'll discuss this optional add-on later in this document, but one of the things ISPP does is allow #DEFINE constants to be used in scripts. The benefit of using constants in Inno scripts is similar to that in VFP and other languages: all the constants are defined at the start of the script file, making it easier to edit them later than hunting through the script file looking for the things you want to change.

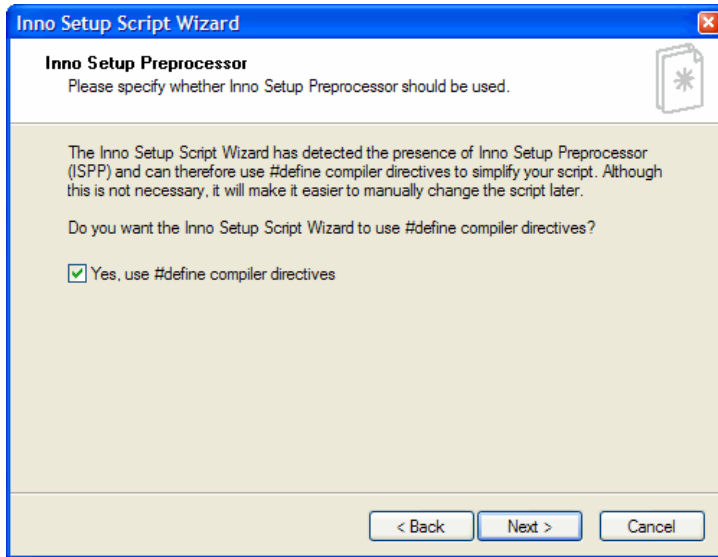


Figure 8. If the wizard detects ISPP on your system, it'll allow you to create #DEFINE constants, which make it easier to edit the script later.

That was a lot of steps, but for the most part you can accept the default settings in each one. Click the Finish button to generate the Inno Setup script for your settings. **Figure 9** shows the results.

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "My First Application"
#define MyAppVerName "My First Application 1.0"
#define MyAppPublisher "My Company, Inc."
#define MyAppURL "http://www.mycompany.com"
#define MyAppExeName "sample1.exe"

[Setup]
AppName={#MyAppName}
AppVerName={#MyAppVerName}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
DefaultGroupName={#MyAppName}
LicenseFile=D:\Writing\Sessions\InnoSetup\Samples\license.txt
OutputBaseFilename=setup
Compression=lzma
SolidCompression=yes

[Languages]
Name: english; MessagesFile: compiler:Default.isl

[Tasks]
Name: desktopicon; Description: {cm:CreateDesktopIcon}; GroupDescription: {cm

[Files]
Source: D:\Writing\Sessions\InnoSetup\Samples\sample1.exe; DestDir: {app}; Fl
Source: D:\Writing\Sessions\InnoSetup\Samples\readme.txt; DestDir: {app}; Fla
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
```

Figure 9. The result of the Inno Setup Script Wizard is an Inno Setup script you can compile to create a setup executable.

Save the script and click the Compile Setup button in the toolbar. This will compile the script, generating a setup executable using the path and name you specified in the Compiler Settings step, and can optionally run the setup so you can test it. **Figures 10 through 16** shows the resulting setup executable being run.

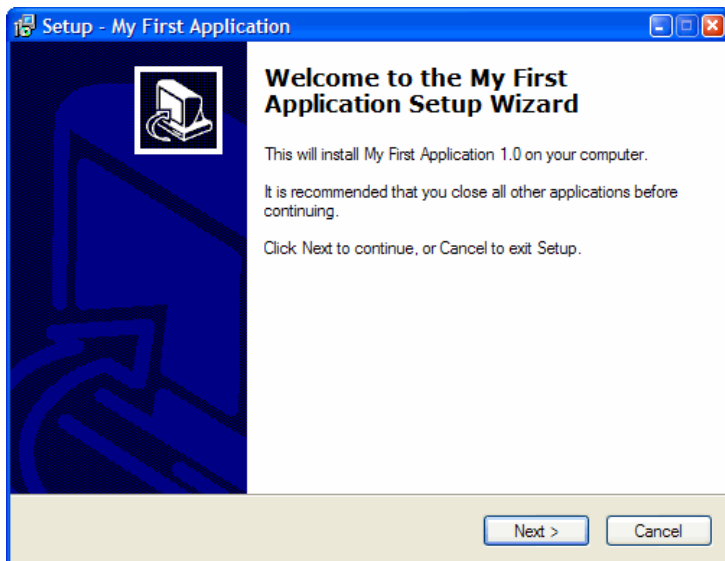


Figure 10. Congratulations: you've created your first Inno Setup executable.

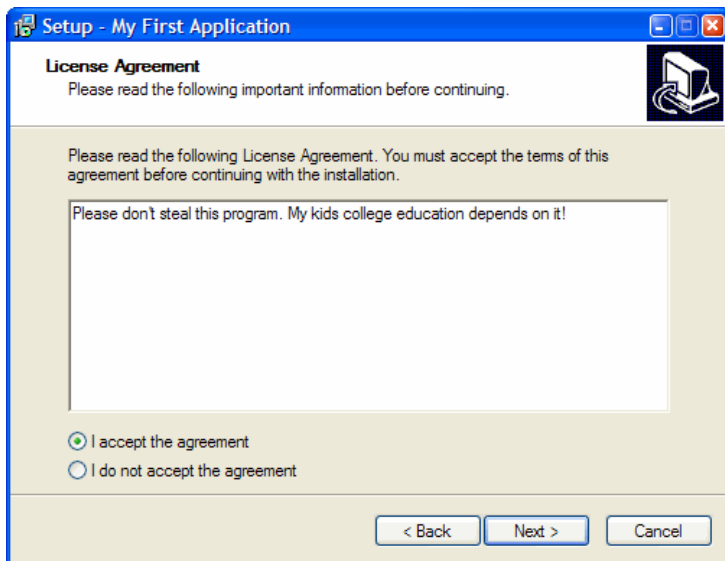


Figure 11. The user must accept the license agreement before going on.

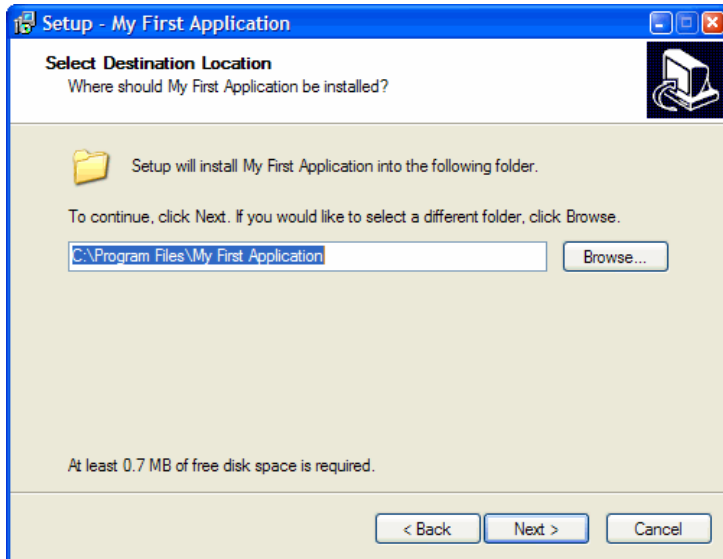


Figure 12. The settings you specified in the Application Directory step of the Wizard define how the Select Destination Location step works.

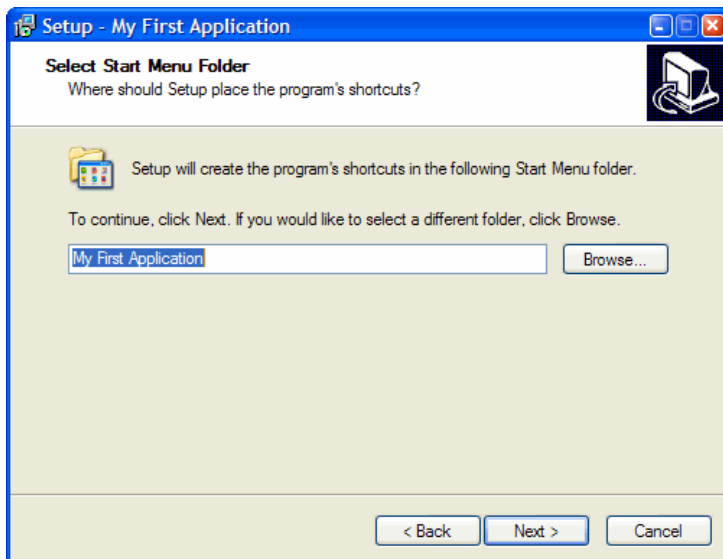


Figure 13. The settings in this setup step depend on the options you selected in the Application Icons step of the Wizard.

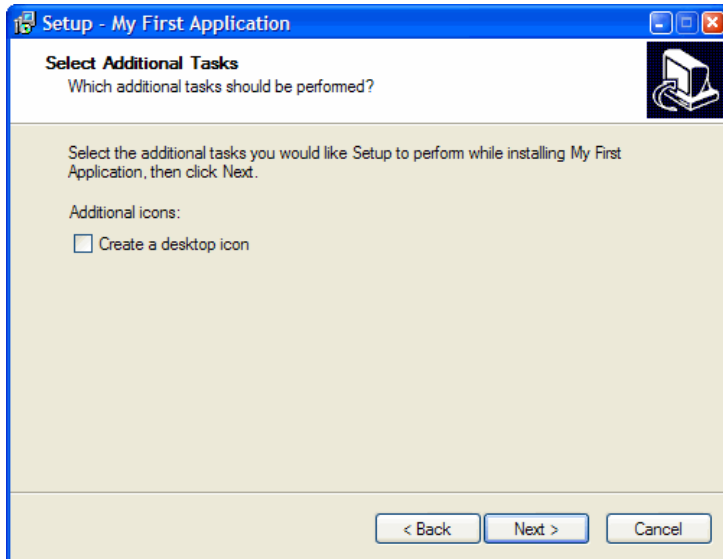


Figure 14. If you turned on the appropriate settings in the Application Icons step of the Wizard, the user can have the installer create desktop and other icons.

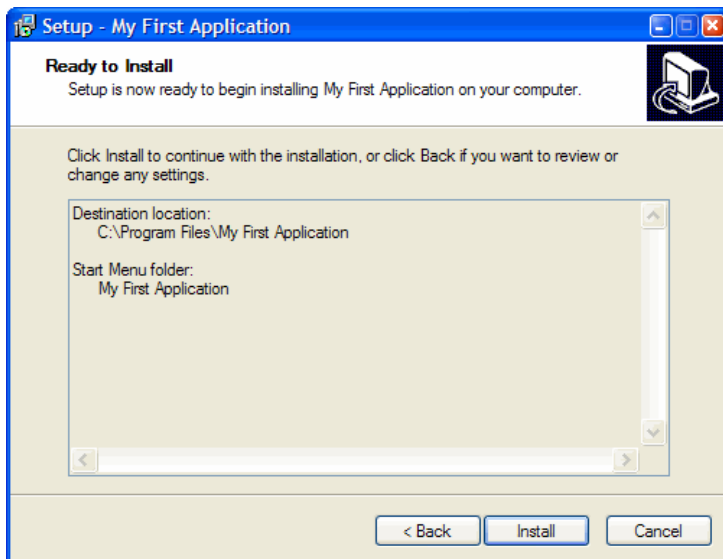


Figure 15. The last step of the installer summarizes the user's settings. Clicking Install starts the installation.

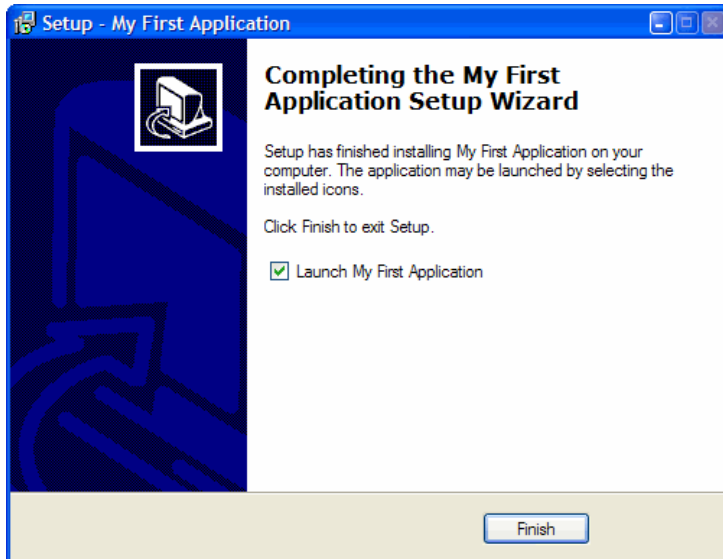


Figure 16. After the installation is complete, the user can launch the application if you turned the appropriate option on in the Wizard.

Go ahead and run the installer to install the application. Once it's finished, you should find the following on your system:

- A shortcut for the main executable on the desktop if you turned that option on.
- A new folder for the application in the Windows Start menu.
- An entry for the application in the Add/Remove Programs Control Panel applet.
- A folder in the location you specified in the Destination Location step in the installer (see **Figure 17**) with the files you indicated to install in the Wizard, along with two other files: UnIns000.EXE and UnIns000.DAT. These two files are the application uninstaller; running the EXE is way faster than opening Add/Remove Programs and removing the application from there.

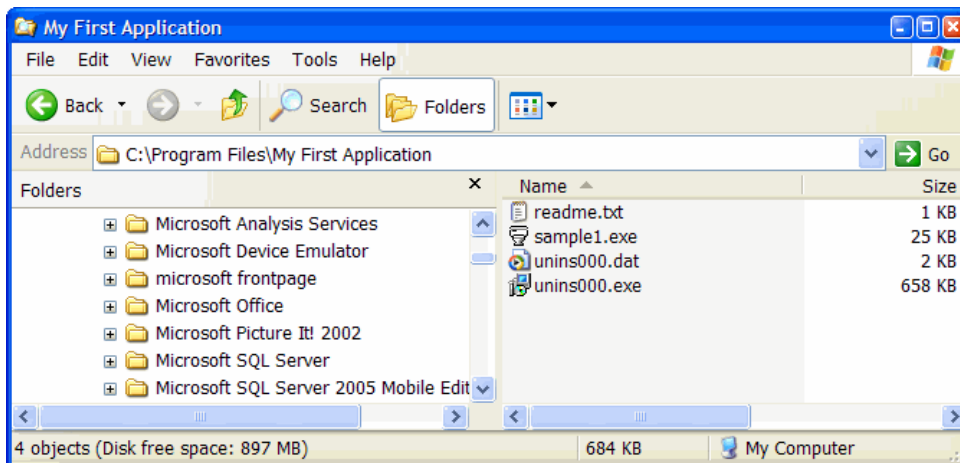


Figure 17. Running the setup executable installs the application.

Here are a few things you may have noticed about this process:

- Starting Inno Setup is significantly faster than starting InstallShield Express.
- The compilation is significantly faster.

- The setup executable is significantly (about 10 MB) smaller.
- The installation process is significantly faster.
- Uninstalling the application is significantly faster.

Overall, it's a better experience both for you and for the user. Now you see why I haven't looked back since I started using Inno Setup.

One thing to be aware of when running the Inno Setup compiler: the files specified in the script must not be open in any other application or you'll get an error message and the compilation will terminate.

The Inno Setup UI

The Inno Setup user interface is fairly sparse, but that's actually one of the things I like about it. As you can see in **Figure 18**, it's not much more than a text editor. However, it does provide functions to compile the current script and run the resulting setup executable with a debugger attached so you can see what happens at every step of the installation. It also uses syntax coloring, making it easier to identify certain elements in a script.

```

sample1.iss - Inno Setup Compiler 5.1.6 (ISPP 5.1.2.0)
File Edit View Build Run Tools Help
|; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "My First Application"
#define MyAppVerName "My First Application 1.0"
#define MyAppPublisher "My Company, Inc."
#define MyAppURL "http://www.mycompany.com"
#define MyAppExeName "sample1.exe"

[Setup]
AppName={#MyAppName}
AppVerName={#MyAppVerName}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
DefaultGroupName={#MyAppName}
LicenseFile=D:\Writing\Sessions\InnoSetup\Samples\license.txt
OutputBaseFilename=setup
Compression=lzma
SolidCompression=yes

[Languages]
Name: english; MessagesFile: compiler:Default.isl

[Tasks]

```

Figure 18. The Inno Setup user interface isn't much more than a text editor.

Some people prefer to use ISTool, one of the add-ons installed with the Inno Setup QuickStart Pack. As you can see in **Figure 19**, it provides a richer, easier-to-user interface to scripts. While you can see the script in its entirety by selecting the Script node in the TreeView, you can also select one of the specific sections and see the settings in a section-specific panel. Figure 19 shows the dialog that appears when you right-click and choose New Item when the Icons section is selected. Rather than remembering somewhat obscure directives for flags and other settings, you can select the settings you want using controls with descriptive captions.

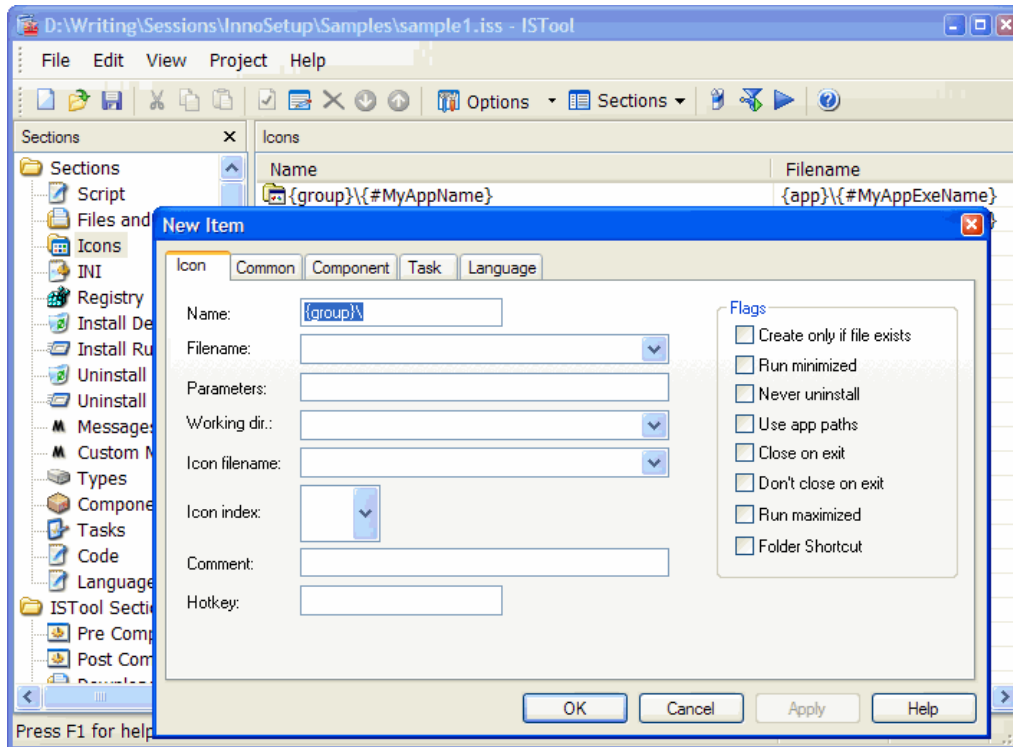


Figure 19. ISTool provides a richer user interface for creating Inno Setup scripts.

Inno Setup Scripts

Now that we've seen how easy it is to create a setup executable, let's dig into the script file to see what they look like.

Inno Setup scripts are just text files with a layout very similar to an INI file: they have named sections (such as "[Setup]") and settings within each section. Here's the script file generated by the Inno Setup Script Wizard in the previous section of this document (although some of the script entries span more than one line due to formatting for this document, each should be a single line of text):

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "My First Application"
#define MyAppVerName "My First Application 1.0"
#define MyAppPublisher "My Company, Inc."
#define MyAppURL "http://www.mycompany.com"
#define MyAppExeName "sample1.exe"

[Setup]
AppName={#MyAppName}
AppVerName={#MyAppVerName}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
DefaultGroupName={#MyAppName}
LicenseFile=D:\Writing\Sessions\InnoSetup\Samples\license.txt
OutputBaseFilename=setup
Compression=lzma
SolidCompression=yes

[Languages]
Name: english; MessagesFile: compiler:Default.isl
```

```

[Tasks]
Name: desktopicon; Description: {cm:CreateDesktopIcon}; GroupDescription: {cm:AdditionalIcons};
Flags: unchecked

[Files]
Source: D:\Writing\Sessions\InnoSetup\Samples\sample1.exe; DestDir: {app}; Flags: ignoreversion
Source: D:\Writing\Sessions\InnoSetup\Samples\readme.txt; DestDir: {app}; Flags: ignoreversion
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: {group}\{#MyAppName}; Filename: {app}\{#MyAppExeName}
Name: {userdesktop}\{#MyAppName}; Filename: {app}\{#MyAppExeName}; Tasks: desktopicon

[Run]
Filename: {app}\{#MyAppExeName}; Description: {cm:LaunchProgram,{#MyAppName}}; Flags: nowait
postinstall skipifsilent

```

Like INI files, lines starting with “;” are comments; you can use this for real comments, like the first couple of lines in the file, or to disable settings you want to keep in the script file.

The #DEFINE statements near the beginning of the script define some constants used later in the script. As you can see in the Setup section, constants are referenced as “{#ConstantName}.”

Inno Setup includes a number of pre-defined constants, such as “pf” for the user’s Program Files directory, “app” for the directory selected by the user to install the application into, “userdesktop” for the user’s Desktop directory, and so forth; see the Inno Setup documentation for a complete list. Similar to #DEFINE constants, built-in constants are referenced using “{ConstantName}.”

Note that hard-coded paths are used in several places in this script; the wizard inserted those. To make the script more portable, you can specify relative paths; since these files are in the same folder as the script file is, you can remove the paths altogether.

The Setup section specifies settings using name-value pairs similar to settings in an INI file. Other sections use parameter lines. The values of values or parameters may be delimited with quotes, although that isn’t required. Parameters are separated with semi-colons.

Setup

The Setup section specifies global settings used by the installer and uninstaller. There are a lot of settings you can use; **Table 1** lists some of the more commonly used ones. See the Inno Setup documentation for a complete list.

Table 1. Commonly-used settings for the Setup section.

Setting	Description
AppName	The application name.
AppVerName	The application name including version number.
AppVersion	The application version number.
AppPublisher	The company publishing the application.
AppPublisherURL	The URL for the publisher.
AppSupportURL	The URL for technical support; this is shown in the Add/Remove Programs Control Panel applet.
AppUpdatesURL	The URL for updates; this is shown in the Add/Remove Programs Control Panel applet.
AppCopyright	The copyright for the application.
DefaultDirName	The default directory to install into.
AppendDefaultDirName	If “yes,” appends the last portion of DefaultDirName onto the path chosen by the user (default is “yes”).
DefaultGroupName	The default folder in the Start menu.
LicenseFile	The name of a file containing license text.
OutputBaseFileName	The file name (without path or extension) for the setup executable.
Compression	The type of compression to use, such as “lzma” or “zip.”
SolidCompression	If “yes,” all files are compressed at once rather than individually (default is “no”).
Password	The password to use.
PrivilegesRequired	The privileges the user requires to run the installer, such as “admin” for administrative rights.
VersionInfoDescription	The description for the installer displayed in Windows Explorer when the user selects the setup executable.
VersionInfoVersion	The version number for the installer.
VersionInfoCompany	The company publishing the installer.
VersionInfoCopyright	The copyright for the installer.

InfoBeforeFile	The name of a file containing pre-installation text to display to the user.
----------------	---

Languages

The Languages section defines the languages used by the setup executable and the names of the message files containing localized strings used by the installer. Although this section is inserted by the Inno Setup Script Wizard, it isn't required if you want an English-only setup.

Tasks

The Tasks section specifies optional choices the user can make on the Select Additional Tasks step of the installer. A task doesn't do anything by itself, but is used by items in other sections to indicate whether something should be installed or not; we'll see an example of this in the Icons section.

The Name parameter specifies the task name, Description is the prompt to display to the user, and GroupDescription is a description to group tasks under ("Additional icons" in Figure 14). Notice in the sample script, messages are used for both Description and GroupDescription rather than hard-coded prompts; this makes it much easier to create a multilingual or localized installer. Messages are referenced as "{cm:MessageName}." Flags indicates how to handle the task; in this example, "unchecked" indicates the task is turned off by default.

Files

The Files section lists the files to install and where to install them. Unlike InstallShield Express and other Windows Installer tools, Inno Setup doesn't natively support merge modules, so every file you want to install must be listed in this section, including runtime files, ActiveX controls, and system DLLs.

The source file is specified in the Source parameter and the destination in the DestDir parameter. The Flags parameter tells the installer how to deal with the file; "ignoreversion" is commonly used with application files because we want them always overwritten if they exist. We'll see different flags in other examples later. Other parameters can also be used; see the Inno Setup documentation for a complete list.

Icons

The Icons section lists shortcuts to create. Name is the path and name of the shortcut file and Filename is the path and name of the target for the shortcut. If the optional Tasks is specified (as it is in the case of the desktop shortcut), it specifies the name of an entry in the Tasks section. The shortcut is only created if that task is turned on by the user.

Run

You can specify applications to execute after the installation is complete in the Run section. Applications are executed in the order they appear in the list. The installer will wait for one application to finish executing before running the next one unless "nowait" is specified in the Flags parameter.

Filename is the path and name of the application to execute. Description is the prompt to display to the user if the "postinstall" flag is specified (discussed later in this paragraph); in this example, a message is used to avoid hard-coding the prompt. Parameters specifies parameters to pass to the application. Flags specifies how to handle the application to execute. "nowait" means the installer won't wait for the application to terminate before carrying on. "postinstall" tells Inno Setup to display the prompt specified in the Description parameter and only execute the application if the user turns the option on. "skipifsilent" means this application isn't executed if the user performs a "silent" install (discussed later in this document). See the Inno Setup documentation for other flags and parameters.

Other Sections

There are several other sections you can include:

- Types: allows you to define different types of setups, such as complete or workstation only.
- Components: this section is related to Types.
- Dirs: specifies additional directories to create.
- Registry: specifies entries to create in the Windows Registry.
- INI: specifies settings to add to INI files.

- InstallDelete: specifies files and folders to delete as the first step of installation.
- UninstallDelete: specified additional files and folders to delete as the last step of uninstallation besides those installed by the setup executable.
- UninstallRun: similar to Run, but specifies applications to run as the first step of uninstallation.
- Messages: overrides the strings displayed for built-in messages.
- CustomMessages: allows you to define custom messages.
- LangOptions: specifies language-specific settings, such as fonts.

We'll see examples of some of these sections later.

Uninstalling

As I mentioned earlier, Inno Setup automatically creates UnIns000.EXE and UnIns000.DAT in the install directory. These two files are the application uninstaller; running the EXE is way faster than opening Add/Remove Programs and removing the application from there.

When the user uninstalls an application, Inno Setup only deletes those files and Registry entries it installed. If the user ran the application and the application created additional files or Registry entries, they aren't deleted. Inno Setup will delete the install directory if it created the directory and there are no other files in that directory; otherwise, the user will have to delete the additional files and the directory themselves.

If the install folder already contains UnIns000.EXE, it names the uninstaller files UnIns001.EXE and UnIns001.DAT, or whatever the next available sequential numbers available are.

Command Line Parameters

Inno Setup supports a number of command line parameters which allow the user to customize the installation behavior. **Table 2** lists some of them; see the Inno Setup documentation for a complete list.

Table 2. Commonly-used command line parameters.

Parameter	Description
/SILENT	Installs the application without displaying the installer; only a progress dialog appears.
/VERYSILENT	Installs the application without displaying the installer or any progress dialog.
/LOG	Creates a log file of setup tasks, useful for debugging if something goes wrong.
/LANG	Specifies the language file to use.

The compiler also accepts command line parameters, allowing you to automate setup creation if desired. The name of the compiler is Compil32.EXE in the Inno Setup program directory; you can find it programmatically from the HKEY_CLASS_ROOT\InnoSetupScriptFile\Shell\Compile\Command key in the Windows Registry. Pass /CC and the name of the script file to compile; for example:

```
compil32 /cc c:\MyApp\Setup\MyScript.iss
```

You can also use the console-mode compiler, ISCC.EXE:

```
iscc c:\MyApp\Setup\MyScript.iss
```

Installing VFP Applications

One of the things you may have noticed about the first example script is that it just installed the EXE for an application. VFP applications have other requirements, including some system DLLs and, of course, the VFP runtime files.

The following script, taken from VFP9Runtimes.ISS (included in the source files accompanying this document), does the necessary tasks to install all runtime files required by VFP applications.

[Files]


```

; VC++ files
Source: "D:\Installer\asycfilt.dll"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist
Source: "D:\Installer\comcat.dll"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist regserver
Source: "D:\Installer\oleaut32.dll"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist regserver
Source: "D:\Installer\olepro32.dll"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist regserver
Source: "D:\Installer\stdole2.tlb"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist regtypelib

; Help system files
Source: "D:\Installer\HH.EXE"; DestDir: "{win}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist
Source: "D:\Installer\HHCtrl.ocx"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist
Source: "D:\Installer\ITIRCL.dll"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist
Source: "D:\Installer\ITSS.dll"; DestDir: "{sys}"; Flags: uninsneveruninstall sharedfile
onlyifdoesntexist

; VFP 9 runtime files. Note MSVCR71.DLL is temporarily installed in the same directory;
; without it, they won't register properly. See http://fox.wikis.com/wc.dll?Wiki~InnoScripts
Source: "D:\Installer\Inno Setup\msvcr71.dll"; DestDir:
"{cf}\Microsoft Shared\VFP"; Flags: onlyifdoesntexist deleteafterinstall
Source: "C:\Program Files\Common Files\Microsoft Shared\VFP\gdiplus.dll"; DestDir:
"{cf}\Microsoft Shared\VFP"; Flags: sharedfile
Source: "C:\Program Files\Common Files\Microsoft Shared\VFP\vfp9r.dll"; DestDir:
"{cf}\Microsoft Shared\VFP"; Flags: sharedfile regserver
Source: "C:\Program Files\Common Files\Microsoft Shared\VFP\vfp9renu.dll"; DestDir:
"{cf}\Microsoft Shared\VFP"; Flags: sharedfile
Source: "C:\Program Files\Common Files\Microsoft Shared\VFP\foxhhelp9.exe"; DestDir:
"{cf}\Microsoft Shared\VFP"; Flags: sharedfile
Source: "C:\Program Files\Common Files\Microsoft Shared\VFP\foxhhelp9.dll"; DestDir:
"{cf}\Microsoft Shared\VFP"; Flags: sharedfile regserver

; Install MSVCR71.DLL in the application directory as per
; http://fox.wikis.com/wc.dll?Wiki~InnoScripts
Source: "D:\Installer\msvcr71.dll"; DestDir: "{app}"; Flags: ignoreversion

[Run]
; Need to run FOXHHELP9.EXE to register it
Filename: {cf}\Microsoft Shared\VFP\foxhhelp9.exe; Parameters: /regserver

```

There are a few interesting things to note about this script:

- Files installed in the user's Windows or System directories (specified as "{win}" and "{sys}", respectively) cannot use your own Windows or System directory for the source location. The reason is discussed in the "Unsafe Files" topic in the Inno Setup help file, so I won't go into details here. So, I've created an Installer folder on the D: drive of my system and copied the files I need to install on the user's system into that folder. Feel free to modify the script and specify a different source location if you wish.
- Those files are also flagged as "uninsneveruninstall" so they're never uninstalled, "sharedfile" since these files are most likely shared by other applications, and "onlyifdoesntexist" so existing files aren't overwritten.
- The "regserver" and "regtypelib" flags on some files tells the installer to register COM components and type libraries in the Windows Registry.
- VFP runtime files are installed in the Microsoft Shared\VFP folder of the user's Common Files folder (specified as "{cf}", usually C:\Program Files\Common Files) rather than the System directory, since that's been the correct place to install them since VFP 8. Included with the VFP runtime files are the files necessary for the help system to work within VFP applications.
- According to the VFP help file, MSVCR71.DLL should be installed in your application directory rather than a system directory, so the final file listed in the Files section does that. However, during installation,

the installer needs to register some of the VFP COM components, and that fails if MSVCR71.DLL can't be found. So, earlier in the script, MSVCR71.DLL is temporarily installed in the same location as the VFP runtime files; the "deleteafterinstall" flag tells the installer to delete the file after installation is complete. Thanks to Rick Borup for discovering this quirk and posting it at <http://fox.wikis.com/wc.dll?Wiki~InnoScripts>.

- Although specifying a flag of "regserver" registers a COM component, that doesn't work for EXE COM components. Instead, you have to run the EXE with "/regserver" as a parameter to register it. That's why the Run section runs FOXHHELP9.EXE; without this, help won't work in VFP applications.

There's no need to copy and paste the lines in VFP9Runtimes.ISS into each script file you create. If you install the Inno Setup Preprocessor (ISPP) optional add-on, which I highly recommend, you can #INCLUDE this file in your scripts much like you can #INCLUDE files in VFP code.

Setup Types

A common scenario is to install an application on a server and only install the minimum files necessary on a workstation along with a shortcut that points to the application's EXE on the server. Like InstallShield Express, Inno Setup allows you to create a setup executable with different setup types so the user can choose whether to perform a complete install (all files) or a workstation install (the runtime files and shortcut). To do that, add Types and Components sections to the script, and then a Components parameter on the entries in the Files section. Here's an example, taken from Sample2.ISS included with this document:

```
[Types]
Name: "complete"; Description: "Full installation"
Name: "workstation"; Description: "Workstation only"

[Components]
Name: "complete"; Description: "Program files"; Types: complete
Name: "required"; Description: "Required files"; Types: complete workstation; Flags: fixed

[Files]
Source: sample1.exe; DestDir: {app}; Components: complete; Flags: ignoreversion
Source: readme.txt; DestDir: {app}; Components: complete; Flags: ignoreversion
```

The Types section defines the type of installations available on the Select Components page of the setup executable. Name is the name of the type and Description is the prompt displayed to the user. If you want the user to be able to select or deselect specific components with a type, add "Flags: iscustom" to one of the types, the one you want to be considered a "custom" installation.

The Components section specifies the components the user can install. Each component is associated with one or more types using the Types parameter; in the example above, the "required" component is associated with both the "complete" and "workstation" installs, while the "complete" component is only associated with the "complete" install. Name is the name of the component and Description is its prompt. A number of different flags are available; in the example, "fixed" means the user cannot select or deselect the component, so it's always installed. If "fixed" isn't specified for at least one component, any types defined as "iscustom" are not displayed. Components can have child components (for example, under a "Data Files", you might have "Production" and "Test Data" subcomponents). These are specified as "*ParentComponent\ChildComponent*".

A component isn't a true component; you associate a file with a component by adding a Components parameter to the file's entry in the Files section. If a file doesn't have such a parameter, it's always installed (that's the case for the VFP 9 runtime files discussed in the previous section). In the sample script, both files listed in the Files section are only installed if the user chooses the "complete" component.

Figure 20 shows the Select Components page of the setup executable created by compiling Sample2.ISS. "Full installation" will install all files while "Workstation only" will install only VFP runtime files and optionally a shortcut to the application's executable on the server (the user has to select the correct server directory in the Select Destination Location step).

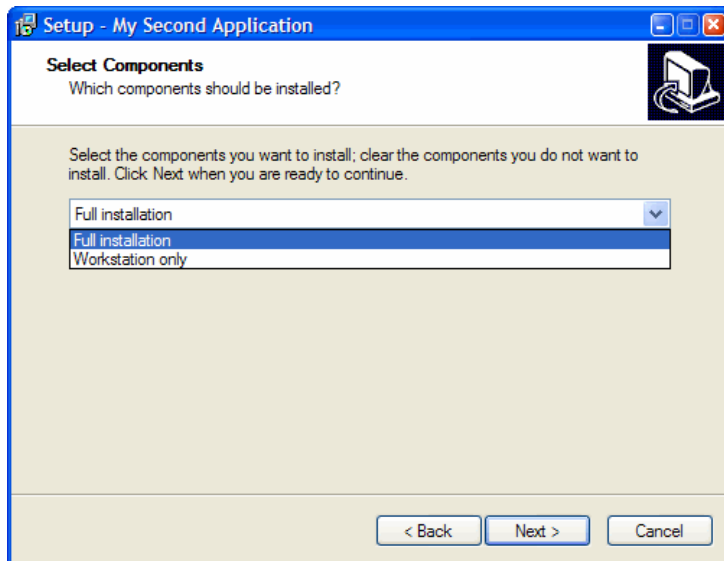


Figure 20. Installation types allow the user to select which components they want to install.

Note that since the “iscustom” flag wasn’t specified for either of the types, the user cannot select individual components. One of the sample files that comes with Inno Setup, Components.ISS, demonstrates a more complete example:

```
[Types]
Name: "full"; Description: "Full installation"
Name: "compact"; Description: "Compact installation"
Name: "custom"; Description: "Custom installation"; Flags: iscustom

[Components]
Name: "program"; Description: "Program Files"; Types: full compact custom; Flags: fixed
Name: "help"; Description: "Help File"; Types: full
Name: "readme"; Description: "Readme File"; Types: full
Name: "readme\en"; Description: "English"; Flags: exclusive
Name: "readme\de"; Description: "German"; Flags: exclusive

[Files]
Source: "MyProg.exe"; DestDir: "{app}"; Components: program
Source: "MyProg.hlp"; DestDir: "{app}"; Components: help
Source: "Readme.txt"; DestDir: "{app}"; Components: readme\en; Flags: isreadme
Source: "Readme-German.txt"; DestName: "Liesmich.txt"; DestDir: "{app}"; Components: readme\de;
Flags: isreadme
```

Figure 21 shows this installer. Note the “Program Files” component is disabled since it’s flagged as “fixed”. Also notice that child components under “Readme File” appear as radio buttons rather than check boxes, since they’re flagged as mutually exclusive. As soon as the user changes one of the options, the installation type automatically changes to the one specified as “iscustom” (“Custom installation” in this case).

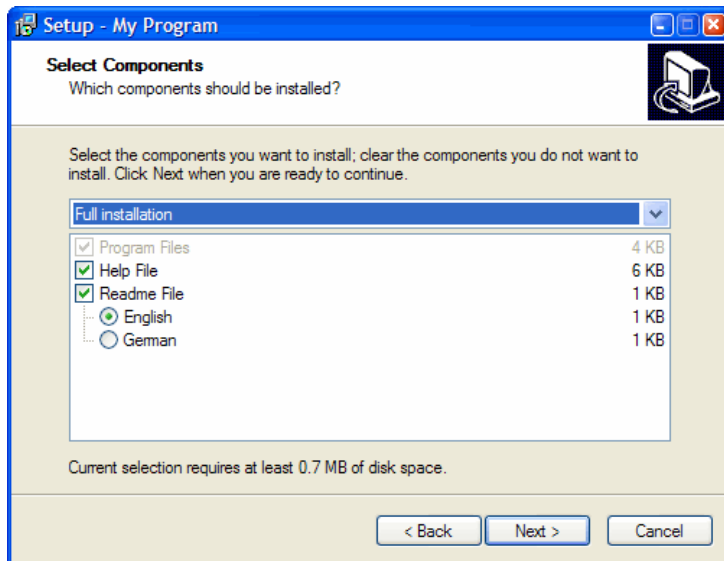


Figure 21. With the “iscustom” flag on a type, the user can select which components to install.

One other thing to note about this script: the “isreadme” flag on the Readme.txt and Readme-German.txt files tells the installer to place a checkbox on the last page, allowing the user the option of reading whichever of these files the user chose to install using the application associated with the file type (for example, Notepad for TXT files).

A More Complete Example

The following script is taken from a real-world application; it’s the installation script for Stonefield Query for ACT! 2005-2006, a commercial product my company sells.

```
#define MyApplication "sfquery.exe"
#define MyPassword "bambi#thumper"
#define MyBeforeFile "actinnosetupbeforefile.rtf"
#define MyAppName "Stonefield Query for ACT! 2005-2006"
#define MyCopyright "Copyright © 1996-2006 Stonefield Software Inc. All rights reserved."
#define MyVersion GetFileVersion(MyApplication)
#define MyURL "http://www.stonefieldquery.com"
#define MyCompany "Stonefield Software Inc."
```

```
[Setup]
Password={#MyPassword}
AppName={#MyAppName}
AppVerName={#MyAppName}
AppVersion={#MyVersion}
AppPublisher={#MyCompany}
AppPublisherURL={#MyURL}
AppSupportURL={#MyURL}
AppUpdatesURL={#MyURL}
DefaultDirName={pf}\{#MyAppName}
DefaultGroupName={#MyAppName}
AppendDefaultDirName=no
LicenseFile=D:\SFQuery\SFQueryDDEditor9\eula.rtf
Compression=lzma
SolidCompression=true
PrivilegesRequired=admin
UninstallDisplayIcon=D:\SFQuery\SFQueryDDEditor9\SFQuery.ico
AppCopyright={#MyCopyright}
VersionInfoDescription={#MyAppName} Setup
VersionInfoVersion={#MyVersion}
VersionInfoTextVersion={#MyVersion}
VersionInfoCompany={#MyCompany}
VersionInfoCopyright={#MyCopyright}
WizardImageFile=compiler:WizModernImage-IS.bmp
WizardSmallImageFile=compiler:WizModernSmallImage-IS.bmp
```

```

InfoBeforeFile={#MyBeforeFile}

#include "D:\SFQuery\Inno Setup\SQ9Runtimes.iss"

[InstallDelete]
Type: files; Name: "{app}\sfscript.*"
Type: files; Name: "{app}\sfconfig.*"

[Files]
; Stonefield Query
Source: "{#MyApplication}"; Components: complete; DestDir: {app}; Flags: ignoreversion

; Project files
Source: "repmeta.*"; DestDir: "{app}"; Components: complete; Flags: ignoreversion
Source: "sfquery.ini"; DestDir: "{app}"; Components: complete; Flags: ignoreversion
Source: "sfquery.chm"; DestDir: "{app}"; Components: complete; Flags: ignoreversion
Source: "sfquery.pdf"; DestDir: "{app}"; Components: complete; Flags: ignoreversion
Source: "Reports\*"; DestDir: "{app}\Reports"; Components: complete; Flags: ignoreversion

; ACT 2005-2006 specific files
Source: "fieldutil.dbf"; DestDir: "{app}"; Components: complete; Flags: ignoreversion
Source: "D:\SFQuery\Inno Setup\RichTx32.ocx"; DestDir: "{sys}"; Flags: onlyifdoesntexist
sharedfile regserver
Source: "SFQueryAct2005COM.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "SFQUtil.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "RegAsm.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "qwmeta.*"; DestDir: "{app}"; Components: complete; Flags: ignoreversion

[Run]
Filename: "{app}\sfquery.exe"; Description: "{cm:LaunchProgram,Stonefield Query}"; Flags: nowait
postinstall skipifsilent
; ACT 2005-2006 specific execution
Filename: "{app}\RegAsm.exe"; Parameters: "SFQUtil.dll /codebase"; WorkingDir: "{app}";
Flags: runhidden
Filename: "{app}\RegAsm.exe"; Parameters: "SFQueryAct2005COM.dll /codebase"; WorkingDir: "{app}";
Flags: runhidden

[Icons]
Name: {group}\{#MyAppName}; Filename: {app}\sfquery.exe; WorkingDir: {app}
Name: {group}\{#MyAppName} Help; Filename: {app}\sfquery.chm
Name: {commondesktop}\{#MyAppName}; Filename: {app}\sfquery.exe; WorkingDir: {app}; Tasks:
desktopicon

[Registry]
Root: HKCU; Subkey: Software\Stonefield Software Inc.\{#MyAppName}; Flags: uninsdeletekey noerror
; ACT 2005-2006 specific key: Rich Text Control license
Root: HKCR; Subkey: "Licenses\DC4D7920-6AC8-11cf-8ADB-00AA00C00905"; ValueType: string;
ValueName: ""; ValueData: "iokouhloohrojhhhtnooiokomiwmmohosmsl"

```

Here are the interesting things about this example:

- The MyVersion constant is assigned the return value of the GetFileVersion function with the name of the main EXE (which comes from another constant). GetFileVersion is a function included in ISPP that does what its name says: it gets the version for the specified file. Assigning the version number this way means you don't have to assign the version number in two places—the project manager, which then assigns that number to the EXE, and the installation script.
- UninstallDisplayIcon specifies the icon used for UnIns000.EXE.
- The VersionInfo* settings apply to the setup executable itself. That way, when the user selects SETUP.EXE in Windows Explorer, they'll see information about the installer, including a description, company name, and version number.
- WizardImageFile and WizardSmallImageFile allow you to specify different images on the left side and upper right corner of the installer dialogs, respectively. I downloaded these images from a link on the Inno Setup Third-Party Files page of the Inno Setup Web site. Rather than hard-coding the path, you can use the

built-in “compiler:” prefix, which specifies the location of the Inno Setup compiler (if that’s where you placed the files).

- InfoBeforeFile specifies some help information I want displayed before the user is asked where to install the application. This tells them to choose a server location even if they do a workstation install so the installer creates a shortcut with the proper path to the application’s EXE.
- This script #INCLUDEs another script, SQ9Runtimes.ISS, which, in addition to specifying the VFP 9 runtime files, has Types and Components sections defining complete and workstation only installations, a Tasks section for the desktop icon as I discussed earlier, and a Run section to run FOXHHELP.EXE. It also includes some custom code I’ll discuss later.
- An older version of our application included a couple of files we no longer need, so the InstallDelete section deletes these files if the user installs the new version in the same location as the older one (which is normally the case).
- All the files in the Reports subdirectory are installed because of the wildcard in the Source="Reports*" parameter.
- The Run section lists three applications to execute (actually, four, since SQ9Runtimes.ISS runs FOXHHELP.EXE): the application if the user chooses to do so and RegAsm.EXE, which register a couple of .NET components as COM objects so our VFP application can use them. Notice the WorkingDir parameter, which specifies “{app}” so we don’t have to pass the path to the DLLs on the command line, and the “runhidden” flag, which prevents a DOS window from appearing while RegAsm.EXE does its thing.
- The installer creates two entries in the Windows Registry. One is the parent for all Registry entries used by Stonefield Query, installed in HKEY_CURRENT_USER (“HKCU” in Inno Setup); it’s there so the entire set of entries is deleted, thanks to the “uninsdeletekey” flag, if the user uninstalls the application. The second installs a license for the Microsoft Rich Text ActiveX control in HKEY_CLASSES_ROOT (“HKCR”); because we instantiate that control programmatically rather than adding it to a form or class, the license isn’t included in our EXE, so we need to ensure it’s installed in the Registry or the user will get a “license not found” error when they run the application.

Scripting

What makes Inno Setup really powerful is the ability to script various stages of the installation process to customize the appearance and behavior of the installer. Since Inno Setup was actually written in Delphi, it uses Pascal as its scripting language. Although you may not be familiar with Pascal, it doesn’t take long to learn and you likely don’t have to learn it in a great amount of detail to accomplish the tasks you need in Inno Setup. Script code goes in the Code section of a script file.

Example 1: Custom Type Descriptions

Let’s look at an example. Notice the descriptions for the two install types in Sample2.ISS is fairly sparse; they don’t indicate to the user how to decide which type to use. It would be nice if we could provide a more detailed explanation of what each type is for.

Sample3.ISS is based on Sample2.ISS with a few changes: the Type section was changed and two new sections were added. Here’s the changed script text:

```
[Types]
Name: "complete"; Description: {cm:T_Complete}
Name: "workstation"; Description: {cm:T_Workstation}

[CustomMessages]
T_Complete=Full installation
TD_Complete=Choose this option to install My Second Application on a network server or an
individual computer that will not run the application from the server.
T_Workstation=Workstation only
TD_Workstation=Choose this option if My Second Application is already installed on a network
server and you want to run it from this computer.

[Code]
```

```
#include "DescriptiveTypes.isi"
procedure InitializeWizard();
begin
    InitializeDescriptiveTypes();
end;
```

The CustomMessages section defines four custom messages used by the setup. Two of these messages are used as the descriptions for the types in the Types section; note the {cm:MessageName} syntax to specify a custom message.

The code section #INCLUDEs another script file, DescriptiveTypes.ISI, and defines an InitializeWizard function. This function is automatically called from Inno Setup so it is normally used for initialization tasks. In this case, it calls a function named InitializeDescriptiveTypes, which is defined in DescriptiveTypes.ISI.

I won't discuss the code in DescriptiveTypes.ISI; it's well-commented if you want to examine it yourself. It was written by Gavin Lambert, a Inno Setup guru who generously posted in on his Web site (<http://www.mirality.co.nz/inno.php>). It creates a panel on the Select Components page displaying the custom message matching the selected install type (the message with the name "TD_" followed by the install type name). **Figure 22** shows what this page looks like.

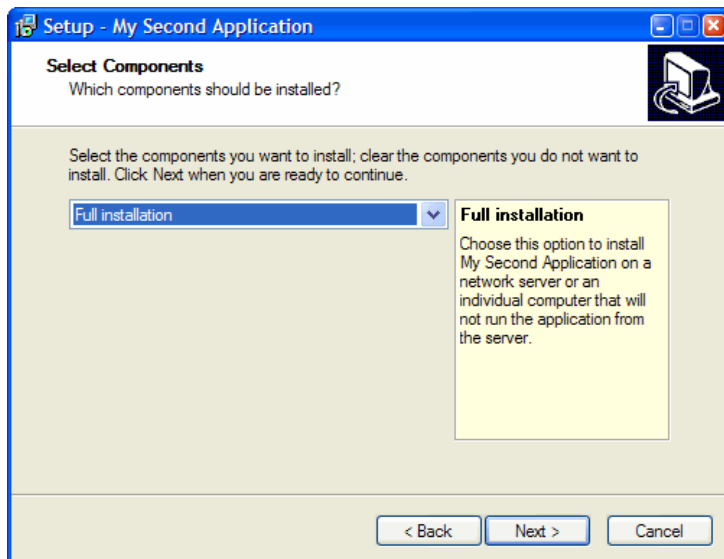


Figure 22. Using scripting, you can change the appearance of setup pages, including adding a panel of descriptive text.

Example 2: Prompting for the Data Location

My second example comes courtesy of Rick Borup. Rather than assuming the data files for an application are installed in a particular folder, you could ask the user where they should go. This allows someone to install the application on their local machine but have the data stored on a server.

Sample4.ISS is based on Sample2.ISS but defines some additional types and components and has script code that adds a new page to the setup executable. It also stores the locations the user selected for the application and the data files in the Registry so the application can determine where its data is located. **Figure 23** shows the new page.

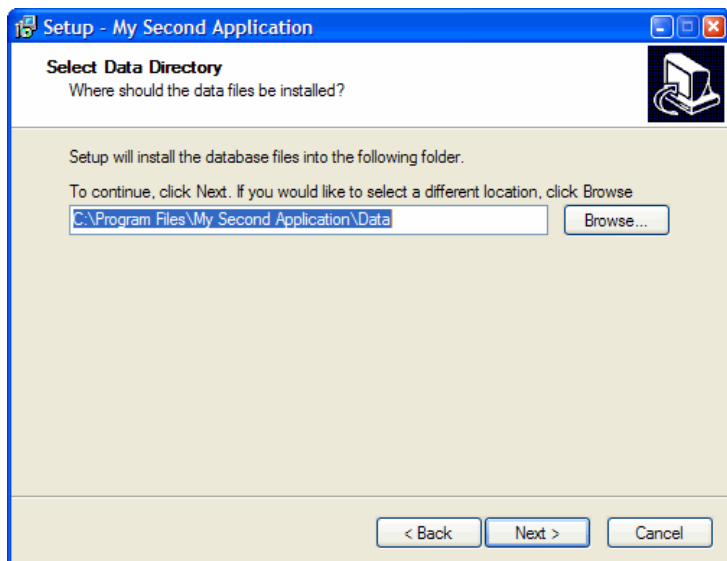


Figure 23. You can add new pages to a setup executable, such as asking the user for the location of the data.

Here are the pertinent sections of the script (the Code section is shown separately later in this document):

```
[Types]
Name: "complete"; Description: "Full installation"
Name: "workstation"; Description: "Workstation only"
Name: "database"; Description: "Database installation"
Name: "custom"; Description: "Custom installation"; Flags: iscustom

[Components]
Name: "complete"; Description: "Program files"; Types: complete
Name: "required"; Description: "Required files"; Types: complete workstation custom; Flags: fixed
Name: "database"; Description: "Database files"; Types: complete database

[Files]
Source: sample1.exe; DestDir: {app}; Components: complete; Flags:
ignoreversion
Source: readme.txt; DestDir: {app}; Components: complete; Flags:
ignoreversion
Source: data\customers.dbf; DestDir: {code:GetDataDir}; Components: database; Flags:
onlyifdoesntexist
Source: data\customers.cdx; DestDir: {code:GetDataDir}; Components: database; Flags:
onlyifdoesntexist

[Registry]
Root: HKCU; Subkey: Software\{#MyAppPublisher}\{#MyAppName}\Settings; ValueType: string;
ValueName: AppDir; ValueData: {code:GetAppDir}; Components: required
Root: HKCU; Subkey: Software\{#MyAppPublisher}\{#MyAppName}\Settings; ValueType: string;
ValueName: DataDir; ValueData: {code:GetDataDir}; Components: database
```

Sample4.ISS adds two new types: database, which allows the user to install just the database files, and custom, which allows the user to choose any combination of the components to install. There's also a new component, database, which installs the data files.

The Files section includes the customer table and its index as part of the database component, but they're also flagged as "onlyifdoesntexist" so existing files aren't overwritten with empty copies when the user installs an update. Note the destination is specified as "{code:GetDataDir}"; this tells Inno Setup to call the GetDataDir function in the Code section and use the return value as the destination.

The Registry section adds new keys to the Registry, storing the location of the application and data files. Again, these get their values from script functions using "{code}".

Let's look at the Code section. The first statement defines a variable named DataDirPage of type TInputDirWizardPage, which is a wizard page class containing edit boxes and Browse buttons for selecting directories.

```
[Code]
var
  DataDirPage: TInputDirWizardPage;
```

The InitializeWizard function creates a new page into that variable using the built-in CreateInputDirPage function. The first parameter is the ID of the page after which the new page will appear (wpSelectComponents is a built-in constant specifying the Select Components page). The next three parameters are the caption, description, and subcaption for the page. The False parameter indicates that the last parameter should not be appended to any directory name the user selects. The following line of code adds a directory selection box to the page; the parameter specifies the caption.

```
procedure InitializeWizard;
begin
  { Create the data directory page }
  DataDirPage := CreateInputDirPage( wpSelectComponents,
    'Select Data Directory', 'Where should the data files be installed?',
    'Setup will install the database files into the following folder.',
    False, 'New Folder');
  DataDirPage.Add( 'To continue, click Next. If you would like to select a different location,
click Browse');
end;
```

The RegisterPreviousData function, automatically called by Inno Setup, calls SetPreviousData to save the value entered by the user in a DataDir key; this allows it to be restored as a default value the next time the user runs the installer.

```
procedure RegisterPreviousData( PreviousDataKey: Integer);
begin
  { Save the DataDir setting so we can restore it next time }
  SetPreviousData( PreviousDataKey, 'DataDir', DataDirPage.Values[0]);
end;
```

ShouldSkipPage is called from Inno Setup to determine if the specified page should be skipped or not. We want to skip the new page if the user didn't select the database component, so the WizardSelectedComponents function is called to return a list of selected components and the Pos function, like the VFP AT() function, returns the location of one string within another. The Result variable, which contains the value of the return code of the function, is set to True if Pos contains 0 (that is, the database component was not selected), so the page is skipped.

```
function ShouldSkipPage( PageID: Integer): Boolean;
begin
  { Skip the DataDir page if the database component is not selected }
  if ( PageID = DataDirPage.ID) then
    Result := pos( 'database', WizardSelectedComponents( false)) = 0
end;
```

Inno Setup calls the NextButtonClick function when the user clicks the Next button, passing it the ID of the current page. For the Select Directory page (wpSelectDir), this code retrieves the data directory they chose the last time they ran the installer. If there is one, that value is placed into the directory control on the data directory page. If not, the code calls the ExpandConstant function to expand the {app} constant, which returns the directory selected by the user, appends "\\Data" to it, and stores that in the control.

```
function NextButtonClick( CurPageID: Integer): Boolean;
begin
  { If we're leaving the wpSelectDir page we now have the value for the 'app' variable,
  which we can use to create the default value for the DataDir variable if there is no
  value already available from a previous install. }
  if ( CurPageID = wpSelectDir) then
```

```

begin
  { Set the value of the data directory to the value of DataDir from the previous
  install, if any. }
  DataDirPage.Values[0] := GetPreviousData( 'DataDir', '' );
  { If there was no previous value then supply a default value. }
  if DataDirPage.Values[0] = '' then
    DataDirPage.Values[0] := ExpandConstant( '{app}' ) + '\Data';
  end;
  Result := True;
end;

```

If it exists, Inno Setup calls the UpdateReadyMemo function to update the display on the Ready to Install page. This code adds a couple of blank lines to the end of the current display (MemoComponentsInfo and NewLine are both passed as parameters). If the user chose the workstation component, the code adds the install directory, passed in the MemoDirInfo parameter, to the text. If the user chose the database components, the code adds the data directory to the text (Space is a string of spaces passed as a parameter). Finally, it returns the string to Inno Setup by setting Result.

```

function UpdateReadyMemo( Space, NewLine, MemoUserInfoInfo, MemoDirInfo, MemoTypeInfo,
  MemoComponentsInfo, MemoGroupInfo, MemoTasksInfo: String): String;
var
  S: String;
begin
  { Fill the 'Ready Memo' with the normal settings and the custom settings }
  S := MemoComponentsInfo + NewLine + NewLine;

  { Workstation component is selected }
  if pos( 'workstation', WizardSelectedComponents( false)) > 0 then
    S := S + MemoDirInfo + ' (workstation)' + NewLine;

  { Database component is selected }
  if pos( 'database', WizardSelectedComponents( false)) > 0 then
    if pos( 'workstation', WizardSelectedComponents( false)) > 0 then
      { Workstation component is also selected }
      S := S + Space + DataDirPage.Values[0] + ' (database)' + NewLine
    else begin
      { Workstation component is not selected }
      S := S + 'Destination Directory:' + NewLine;
      S := S + Space + DataDirPage.Values[0] + ' (database)' + NewLine;
    end;

  Result := S;
end;

```

GetDataDir, which as we saw earlier is called from a couple of places in the script file, simply returns the selected data directory from the value of the directory control on the new page.

```

function GetDataDir( S: String): String;
begin
  { Return the selected DataDir }
  Result := DataDirPage.Values[0];
end;

```

Similarly, GetAppDir returns the install directory by retrieving the value of the {app} constant.

```

function GetAppDir( Default: String): String;
begin
  { Return the install dir }
  Result := ExpandConstant( '{app}' );
end;

```

Other Examples

Inno Setup comes with several sample scripts in its Example folder. Two of the more interesting ones are CodeAutomation.ISS, which shows how to use COM objects in Inno Setup scripts—the examples it uses are creating

a SQL Server database and an IIS virtual directory—and CodeClasses.ISS, which shows how to use the various types of controls built into Inno Setup on custom pages.

Vista Issues

Although Microsoft has been telling us for years not to run our systems while logged in as administrators, Windows Vista finally enforces that with the new User Access Control (UAC). As a result, many users will run as a standard user, which limits their rights on the machine but also limits the damage malware can do.

Most installations require administrative privileges because they need to install files in certain protected folders, such as Program Files. If a standard user runs a setup program, Vista displays the UAC dialog, asking them to enter the user name and password of an administrative user. It then “elevates,” or runs the installer as that user.

A result of running as an administrative user is that user-specific tasks relate to that user rather than the standard user. For example, suppose the installer stores some settings in the HKEY_CURRENT_USER hive in the Registry. The problem is that it stores those settings for the administrative user, so once the installer is done and the user runs the application (now running it as the standard user), they can't see those settings. The same is true if the user launches the application from the installer using the [Run] section in an Inno script. Since Vista was still elevated when it started the application, the application runs as the administrative user. If the user specifies configuration settings and the application stores those settings in a user-specific location (the HKEY_CURRENT_USER hive in the Registry or a user-specific folder), the next time the user runs the application, they will do so as the standard user and won't see those settings.

The solution is to not allow any user-specific tasks in an Inno script. This means:

- Don't write settings to HKEY_CURRENT_USER in the Registry.
- Don't write to files in user-specific folders.
- Create icons for all users rather than the current user: use {commondesktop} rather than {userdesktop}.
- Don't allow the application to be launched from the installer: add OnlyBelowVersion: 0,6 to the [Run] command so the option is only available for Windows XP or lower.

Resources

Because it's a very popular tool, there are lots of resources available for Inno Setup.

The Inno Setup Web site, <http://www.jrsoftware.org/isinfo.php>, is a great place to start. It has a Frequently Asked Questions page, a Knowledge Base, and links to third-party add-on tools (such as a custom page editor) and information Web sites. There's also a fairly active newsgroup available at news.jrsoftware.org.

Rick Borup has a couple of white papers and a video on using Inno Setup on his Web site, <http://www.ita-software.com/foxpage.aspx>. These are excellent resources for getting started.

In addition to the DescriptiveTypes.ISI file I mentioned earlier, Gavin Lambert's Web site, <http://www.mirality.co.nz/inno.php>, has several white papers on using Inno Setup.

Nicholas Sherlock's Web site, <http://www.sherlocksoftware.org/page.php?id=11>, includes a downloader utility you can use to download files as part of your installation.

For ideas about deployment in general, I highly recommend *Deploying Visual FoxPro Solutions*, by Rick Schummer, Rick Borup, and Jacci Adams, Hentzenwerke Publishing, ISBN 1-930919-32-8, available from www.hentzenwerke.com.

Summary

Inno Setup is a free, fast, easy-to-use installation tool that generates fast, lightweight setup executables. I've been using it exclusively for several years now and can attest to its power and freedom from the problems that drove me away from Windows Installer-based installation tools. I recommend giving it a try; you'll likely find you don't want to go back.

Biography

Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT), the award-winning Stonefield Query, and the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro,

and the My namespace and updated Upsizing Wizard in Sedna. Doug is co-author of the “What’s New in Visual FoxPro” series (the latest being “What’s New in Nine”) and “The Hacker’s Guide to Visual FoxPro 7.0.” He was the technical editor of “The Hacker’s Guide to Visual FoxPro 6.0” and “The Fundamentals.” All of these books are from Hentzenwerke Publishing (<http://www.hentzenwerke.com>). Doug wrote over 100 articles in 10 years for FoxTalk. He has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over the world. He is one of the administrators for the VFPX VFP community extensions Web site (<http://www.codeplex.com/Wiki/View.aspx?ProjectName=VFPX>). He has been a Microsoft Most Valuable Professional (MVP) since 1996. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (<http://fox.wikis.com/wc.dll?Wiki~FoxProCommunityLifetimeAchievementAward~VFP>).



Copyright © 2006-2007 Doug Hennig. All Rights Reserved.