This article was originally published in the October, 2004 issue of FoxTalk 2.0
www.foxtalknewsletter.com

Inno Setup, Part Two

Rick Borup

In part one of this series (see the July 2004 issue), Rick Borup introduced you to Inno Setup and explained the basics of using it to deploy Visual FoxPro applications. In part two, Rick builds on the example from part one and shows you several things you can do to enhance your Inno Setup scripts, including how to install database files to a location independent of the app itself.

In part one of this series, I showed how to construct a very basic Inno Setup script to install a sample Visual FoxPro application called myVFPApp. That script looked like this:

```
[Setup]
AppName=MyVFPApp
AppVerName=MyVFPApp version 1.0.0
DefaultDirName={pf}\MyVFPApp
DefaultGroupName=MyVFPApp
[Files]
Source: MyVFPApp.exe; DestDir: {app}
Source: Readme.txt; DestDir: {app}; Flags: isreadme
#include "VFP8Runtimes.txt"
[Icons]
Name: {group}\MyVFPApp; Filename: {app}\MyVFPApp.exe
```

This script, although simplistic, is complete and functional. Of course, in the real world you need to handle much more complex installations. Some typical requirements include:

- requiring the user to accept to a license agreement before installation;
- specifying a minimum operating system version on the target machine;
- creating registry entries;
- displaying custom graphics in the setup wizard;
- organizing the installation into components; and
- installing database files to a different location than the application itself.

In addition, there are several things you can do to make your setup scripts easier to work with.

Setup section enhancements

By default, Inno Setup expects to find the files referenced in the script in the same folder as the script itself. If the files are in a different folder you can use a fully qualified drive and path for each file, but this makes the script entries much longer and more cumbersome. The alternative is to add a SourceDir directive to the Setup section, which makes all file references relative to the specified drive and folder, like this:

```
SourceDir=C:\FoxTalk\MyVFPApp
```

Also by default, the output file created by the Inno Setup compiler is named setup.exe and is written to a sub-directory named Output. You can use the OutputBaseFilename and OutputDir directives to specify a different output file name and location. The following example tells Inno Setup to name the compiler output “myVFPApp 1.0.0. Setup.exe” and write it to C:\FoxTalk\MyVFPApp\Distrib.

```
OutputBaseFilename=myVFPApp 1.0.0 Setup
OutputDir=C:\FoxTalk\MyVFPApp\Distrib
```

If your app requires a minimum version of Windows or Windows NT on the target machine, you can specify this with the MinVersion directive. The example below shows how to require Windows 98 Second Edition or Windows 2000. The version numbers are documented in the Inno Setup help file.

```
MinVersion=4.1.2222,5.0.2195
```

To display a license agreement and require the user to accept it before installing your application, simply add a LicenseFile directive to the Setup section of the script. The license file can be either a plain text file (.txt) or a rich text file (.rtf).

```
LicenseFile=license.rtf
```

Update considerations

When you install an update, Inno Setup looks to see if a previous version is already installed. A previous version of the same application is identified by its AppID, which is an internal value you can set with the AppID directive. If the AppID directive is omitted, Inno Setup uses the AppName, but unlike the AppName the AppID is never displayed during setup and can therefore be any value you want it to be.

Whether used implicitly or explicitly, a common AppID value is what ties all versions of a given application together. One important reason to use a common AppID across all versions of the same application is so the uninstall information for an update is appended to the uninstall information from the previous installation. Without a common AppID each version would have its own entry in the Windows Add/Remove Programs list. In this example, the AppID is the same as the AppName.

```
AppID=MyVFPApp
```

When you install an update, Inno Setup defaults to the directory where the previous version is installed even if a different directory is specified in DefaultDirName. This behavior is controlled by the UsePreviousAppDir directive, whose default value is yes. Along the same lines, when you install an update it can be assumed the destination directory already exists and no warning to the user is necessary. When you install a new application, however, Inno Setup warns if the destination directory already exists. This behavior is controlled by the DirExistsWarning directive, whose default value is auto.

If you want to use the default values for these two directives you don't need to include them in your script. However, I like to include them anyway just as a reminder that they affect the setup.

```
UsePreviousAppDir=yes
DirExistsWarning=auto
```

Customizing appearance

You can customize the appearance of the setup wizard dialogs during installation by using custom bitmap files. There are a number of different Inno Setup bitmap files available for download, or you can create your own. Both a large and a small image are required; the maximum sizes are 164x314 for the larger one and 55x55 for the smaller one. Add custom wizard image files to your script like this:

```
WizardImageFile=compiler:images\WizModernImage13.bmp
WizardSmallImageFile=compiler:images\WizModernSmallImage13.bmp
```

The optional prefix 'compiler:' tells Inno Setup the location of the image files is relative to the location of the compiler itself. The advantage of doing this is it makes these two directives independent of the location of the script file itself or the location of files specified in the SourceDir directive, thus making these entries usable verbatim in any setup script you create.

Creating registry entries

Registry entries can be created on the target machine by including a Registry section in the setup script. A typical use for registry entries might be to store the location where the application is installed, as illustrated in the following script.

```
[Registry]
Root: HKCU; Subkey: Software\ITA; Components: workstation; Flags: uninsdeletekeyifempty
Root: HKCU; Subkey: Software\ITA\myVFPApp; Components: workstation; Flags: uninsdeletekey
Root: HKCU; Subkey: Software\ITA\myVFPApp\Settings; ValueType: string; ValueName: AppDir;
ValueData: {code:GetAppDir}; Components: workstation
Root: HKCU; Subkey: Software\ITA\myVFPApp\Settings; ValueType: string; ValueName: DataDir;
ValueData: {code:GetDataDir}; Components: database
```

The references to 'code:' and 'Components:' are explained later in this article.

Types and components

Inno Setup enables you to organize your installation into components, each of which can be associated with one or more different setup types. The default setup types are full, compact, and custom, but you can create your own. Files and other resources such as registry keys can be marked as belonging to one or more components. When the user selects a setup type at installation time, Inno Setup installs the files and other resources belonging to the components associated with that setup type. Files not marked as belonging to any component are always installed unless other conditions take precedence.

Most Visual FoxPro applications are data-centric, so one natural way to organize the setup is to define a workstation component for the files that make up the application itself and a separate database component for the database files. The Types and Components sections to accomplish this are as follows:

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

[Components]
Name: workstation; Description: Workstation files; Types: full workstation
Name: database; Description: database files; Types: full database
```

Looking at the Components section entries, you can see the workstation component is associated with the full and workstation setup types, while the database component is associated with the full and database setup types.

At setup time, the user can choose any of the four setup types. The setup wizard dialog displays a list of the components associated with the selected setup type. The list includes a check box next to each component so the user can see which components are installed with that type. If the user makes any changes to the selected components, the custom setup type is invoked by default.

Individual files are associated with components by adding a Components parameter to the Files section entries, followed by the name of one or more components, as illustrated below.

```
Source: MyVFPApp.exe; DestDir: {app}; Components: workstation
Source: Readme.txt; DestDir: {app}; Components: workstation; Flags: isreadme
```

Making data independent of the app

In a multi-user runtime environment, a common deployment scenario is to install the application itself to one location, such as the Program Files folder on the local machine, and to install the data files to a different location, such as a file server, where they can be shared by all users. The first step is to split the application into a workstation component and a database component, as explained above. However, you still need a way to make the destination directory for the database component completely independent of the destination directory for the application itself.

In the Files section, the value of the each file's DestDir parameter determines where that file is installed. The value of the DestDir parameter is typically relative to the {app} constant, which contains the runtime value of the installation directory chosen by the user, or to another Inno Setup constant such as {pf} or {cf} for the user's Program Files or Common Files directory. If the database component is to be installed to a location that is unrelated to the location of the application itself, you cannot make the value of the database files' DestDir parameters relative to {app} or any other constant. You could use a literal value such as F:\Data\myVFPApp, but that's not a good idea because it makes the setup totally inflexible.

To make the location of the data completely independent of the location of the application without specifying a fixed drive and path, the setup wizard should prompt the user for the database destination directory at runtime. To accomplish this you need to insert a custom page in the setup sequence. Inno Setup provides a way for you to write code in your setup script for this type of customization.

Pascal scripting

Custom code in Inno Setup scripts is written in Pascal scripting, which is placed in the Code section of the script file. Pascal scripting may be unfamiliar to most of us as VFP developers, but don't let that deter you. After all, code is code, right?

The code needed in this example should check whether the database component is selected for installation, and if so should insert a custom wizard page into the setup sequence that prompts the user to specify the destination directory for the database files. The value specified by the user is stored in a string variable called DataDir. A function called GetDataDir returns the value of the DataDir variable.

The code I created for this purpose was adapted from one of the examples that come with Inno Setup. Because of space limitations the full code is not printed in the body of this article, but a small segment is shown below to give you an idea what Pascal scripting looks like. The full setup script, including the entire Code section, is included in this month's subscriber downloads.

```
[Code]
var
  DataDir: String;
function GetDataDir(S: String): String;
begin
  { Return the selected DataDir }
  Result := DataDir;
end;
```

Once the Code section is in place, you can make calls to the GetDataDir function from other sections of the setup script whenever it is necessary to insert the runtime value of the database directory. Among the places where this is necessary is the DestDir parameter for the database files. The File section entries for these files are as follows:

```
Source: Data\customers.DBF; DestDir: {code:GetDataDir}; Components: database
Source: Data\customers.CDX; DestDir: {code:GetDataDir}; Components: database
```

These two files are also marked as belonging to the database component of the setup, so they're installed only if the database component is selected.

The ability to make the location of the data independent of the location of the application is very powerful because it enables you to create flexible setups that satisfy a wide variety of deployment scenarios. This of course is only one example of using Pascal scripting with Inno Setup. There is a great deal more customization you can accomplish with the Code section; see the Inno Setup Help file for details and ideas.

Conclusion

In this article, I showed you several things you can do with Inno Setup to turn the basic example from part one of this series into a more real-life setup script. This has still really only scratched the surface, though. There is a wide variety of information available to help you learn more. I'd recommend you start with the Inno Setup Help file and the examples that accompany the product, then go to the Inno Setup Web site (http://www.jrsoftware.org/isinfo.php) and start looking around. There are also a number of active newsgroups for Inno Setup, links to which can be found on the Web page.

Download the code from www.ita-software.com/papers/FT410Borup.zip.

Rick Borup is an independent developer who has been working with FoxPro and Visual FoxPro for over ten years. He is owner and president of Information Technology Associates, a software design, development, and consulting firm he founded in 1993. Rick has spoken about VFP and related topics to several user group meetings and was a speaker at the Great Lakes Great Database Workshop in Milwaukee. He is an MCSD and MCP in Visual FoxPro, and is co-author of the book Deploying Visual FoxPro Solutions from Hentzenwerke Publishing. rborup@ita-software.com.