

Getting Started with an Install Program

by Leslie Easter
Lesliee@OrangeBrain.com

© 2000 InstallShield Corporation and The Orange Brain Company, LLC. All Rights Reserved

INSTALLSHIELD MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, IN THIS DOCUMENT.

InstallShield is a registered trademark InstallShield Corporation in the United States and/or other countries.
Other product and company names mentioned herein may be the trademarks of their respective owners.

Contents

Introduction _____	1
Questions, Questions _____	1
Ghosts—Not just for Halloween _____	4
The First Install is not Pretty _____	5
Getting a Profile _____	6
Rules for Core Component Installation _____	9
Separating Fact from Fiction _____	10
Wrapping up _____	12
About the Author _____	13

Introduction

It all starts innocently enough. You're assigned the task of writing the install program for the latest version of the company's flagship product, FooSoft. It should only take a couple of days—a week tops. That's what you were told two weeks ago. Now you're behind schedule and the testing team can't seem to find an end to the install bug queue. The install breaks on Windows 98, the application doesn't launch correctly on Windows NT, and you haven't even started testing on Windows 2000. Not only that, but the application team is updating some SQL scripts that they want included—and they need to be run silently. The documentation team needs final screen shots ASAP. It's eight o'clock on Thursday evening and the VP of Development wants to see the installer first thing in the morning. "Where did it all go wrong?" you ask yourself.

If you're like most this not only sends shivers down your spine but it also brings back horrible memories. At times, getting an install perfected seems like black magic. It's a mixture of understanding the application, knowing a bit about the operating system, negotiating your way into a reasonable schedule, and learning a lot about your install development environment. My goal in this article is to get you started. Getting out of the starting block is a big factor in a successful installer. It's also a time-proven technique that will work no matter what kind of application you're shipping. Let's get started.

I've long lost count of the number of install programs I've written, designed or consulted on. The one thing I do know is that each time I've learned something new and the process gets a little bit better. I'd like to share with you some of the things I've learned that has made my installs better. The secret is a successful start. And that start begins with the project team.

Questions, Questions

Believe it or not, every member of the team initially has a different expectation about the install program. They also have varying views on how difficult the install will be. Some picture the install doing lots of neat stuff. Others may see it as a necessary evil. From marketing to fellow developers, your first task is to capture each team member's expectation and generate a common view. Then you need to share that view with the team in a manner that balances schedule with deliverables, functionality with realism. This process is the team interview.

The team interview can be done best starting with an install questionnaire. I've had one that I've updated over the years and it serves its purpose very well. The questionnaire isn't meant to strictly get information; it also alerts the remainder of the team to important install issues early in the development cycle. Too often the install program isn't given the consideration that it should have early enough. The questionnaire gets preliminary information from the project team and, by virtue of the questions raised, informs the project team.

Common questions contained within the install questionnaire range from the simple (Company Name and Application Name) to the complex (NT Service creation parameters, and ODBC DSN values). In some cases, the questions are there to provide a consistent corporate identity. In others, the questions are relevant to how the application works. Table 1 describes a standard list of topics that you may want to consider using:

Table 1. Install Questionnaire Basics

Key developer contact	This is the one person that is accountable to the install process—a representative from the application development team. This person should be able to either answer or track down an answer for any questions that you may have. You should have a reasonably good rapport with this individual as you'll be spending a significant amount of time together. This person should have an open door policy for your questions. You may want to ask for a home number.
Standard string references	It may seem odd, but the team needs to agree on such simple things such as the company name (does it include the 'Inc' or not?), application name (has Marketing finalized on the name?), and application version (is it 1.0 or 1.0.0.0?). These string table references are used to create the default application path and application-specific registry entry.
Minimum system requirements	Hmmm. Sounds easy, but you'd be surprised how many project managers don't have a concrete list of system requirements. You need to make sure such a list is created and followed. For test purposes, you'll need to have access to a system that matches these requirements. Reasonable questions are: which operating systems will be supported; the minimum service pack that should be installed; the hardware constraints such as disk space, video, and processor type; if administrator (machine and/or domain) rights are required; and so on.
Known application dependencies	While most developers aren't going to know the complete software dependencies off the top of their heads, you'll need to plant the thought in their heads. It'll get them thinking about compiling a list. We'll talk about profiling the application later. At that time, you'll generate your own list of application files. An informal list of application dependencies should include system core components (MFC, VB, ADO, OLE DB, etc.), third party redistributables (Crystal Reports, ActiveX controls, etc.), and NT

Table 1. Install Questionnaire Basics

Service parameters.	ODBC drivers and DSN information Installing ODBC has gotten much easier over the years, but there is still information that you must gather and provide to the install program. The biggest of these are the ODBC version to provide and which drivers (if any) that you'll need to ship. You'll also need to gather the complete list of DSN (data source name) values.
Application components/features	Often the seemingly easiest part of this process can consume the most time. I remember spending over four hours in a meeting talking about this one issue: what are the user selectable pieces of the application? Be prepared for this one to change once or twice. The downside is that the bulk of the install architecture hinges off of the answer to this question. Substantial changes could adversely affect the install program schedule.
Setup types	We all know the basics: Typical, Compact, and Custom. But the project team may have others that are more useful: Administrator Install, Server Install, Client Install, and so on. Most install development tools have flexibility along these lines. Pick the ones that will make the most sense for the end-user.
Distribution media	A few years ago, most software was distributed via CD-ROM. Today things are much different. The project team will need to decide how the physical image will be distributed. If you're going to Web-based distribution there are many trade-offs to consider. The team will need to weigh those carefully before deciding.
Shell folders and icons	Not only do you need to know which shortcuts to create, but when to create them. After all it doesn't make sense to create a shortcut for a file that wasn't installed. Maintain Windows User Interface compliance (e.g., don't create a shortcut for the uninstall).
Graphic support	These are things like splash screens and billboarding bitmaps. It's easy enough to integrate these files into the install project, but it's also critical that you get them in time. Bringing up the subject early in the cycle goes a long way to ensuring that they'll be ready on time. Review the color depth and resolution constraints with the team. The team should be reasonably sure that the user base supports the final graphic parameters.

Known file edits	Will the install have to make file changes to new or existing files? If so, you should get an idea of the types of changes you'll have to make. Are they INI-type edits or low-level binary edits? Even simple string edits can be tricky if they involve grep'ing a file contents line-by-line. Get to know what you're in for.
User interface (dialogs)	Every install development tool provides a base dialog set. You should be familiar with the common dialogs and the functionality of the not so common. It may come down to creating a custom dialog. If that's the case, you'll need to know the function of the dialog and have some strategy for getting data on and off of the dialog. Keep in mind that you'll need to support Back and Next functionality, which requires saving the dialog state.
Registry edits	Just like the file edits, you may be required to modify registry keys in the process of configuring the application. You'll need to know the root key, subkey, value name, value data, and data type. You may be asked to query for a specific key or value and use the result to configure a different key.

Once you've got the results of the install questionnaire, it's time to compile the first pass of the install specification. You'll be updating this document through the course of the install development cycle, so it pays to be as thorough and descriptive as possible. For the first release, you should turn the results of the questionnaire into prose. It's as easy as that. Simply write down the view of the install as described by the team members. Be sure to resolve any outstanding conflicts or at least make a note within the specification that points out these contradictions. You may even want to create an "outstanding issues" list to keep track of any issues that need to be resolved.

Your install specification should contain a conversational discussion about the install program. Describe what it does and how it does it. You should also provide a list of files (with versioning information), registry edits, and file changes. After all is done, not only will it be a handy reference for future changes, but also an invaluable for training the folks in phone support.

Ghosts—Not just for Halloween

The next contributor to your success is the clean machine. This is simply a computer at your disposal for testing. But it's a special computer—it's one that you can actually cause damage to with little downside. A key concept in install development is setting a base level for testing. If you can continually refer to a known, clearly defined, base system, you've got instant repeatability. And this is important if you're chasing odd behaviors and transient problems. If your install works on the base level, you've got most of the bases covered.

So what is this base level? It's the oldest version of the supported operating system. It should also contain the minimum requirements for your application. If you're targeting Windows NT 4.0 client, your test machine should be set up with it. There are a number of 'ghosting' tools around and one of the most common is Symantec's Ghost. This software allows you to take a snapshot of a system and store it to a single data file. Create a bootable CD-ROM with this data file and you're 45 minutes away from re-setting a computer back to its original binary status. Simply amazing and a must have for install testing!

If you haven't done so already, plan on spending a couple of days gearing up your clean machine. It doesn't need to be particularly fast, but it should represent the minimum hardware and software requirements for your application. Make sure you format the disk and reinstall the entire operating system from scratch. I recommend using a 'typical' installation of the operating system. Spend an extra few minutes to make sure the system is completely functional and representative. You may also want to make sure the system settings are useful—set the Explorer shell to display file details and 'Hide files of known extensions' should be turned off. Don't forget to map any drives you may need access to. Once you're done create an image of the system so that you can get back to it quickly and painlessly.

If your application has software prerequisites make sure that they are also installed and functional. This includes software like Internet Explorer, Service Packs, and so on. You should remember to test the final install for a detection and reasonable exit process if these prerequisites are not met. Any requirement above the base operating system becomes a LaunchCondition or Application Requirements check.

In our office, we have a series of clean machines with ghosted CD-ROMs for each operating system. You can typically re-install an entire operating system in under 45 minutes. Images range in size from under 60 Mbytes (Windows NT Client) to over 250 Mbytes (Windows 98). We opted to print out a complete file listing from the contents of the Windows and Windows System folders. Key data included with the file list is: version number, time/date stamp, and file size. Storing this information in a spreadsheet can be invaluable when you tracking the origination of a specific system file.

The First Install is not Pretty

Once a clean machine has been set-up, the next step is to capture the current machine state. There are a variety of tools that can accomplish this for you. InstallShield Professional—Windows Installer Edition has a built-in utility called Spy, Rational ships the VeriTest-Rational Install Analyzer as part of its TestFoundation product, and there are other publicly available tools. Ask around and you'll get to know the pros and cons of each.

After a good first picture of the system, the next step is to get the application up and running. This can be done in one of several ways: installing the application by hand or, if the application is already bundled into an install program, running the install. The latter is most common for repackaging software. Make sure you follow all of the steps to get the application working. Initially, you may need to get a few application developers involved. Don't proceed to the final snapshot until the application is working as expected. If the application doesn't work yet (don't laugh, it happens), then don't start the snapshotting process. You need a fully working application for this step to be worthwhile. Don't accept hand waving as an excuse. It all must work, because that's what you're going to install. If the application is broken in the snapshot, it will more than likely be installed that way. Expect to spend some time on this step. Be sure to document as much of the process as possible. You may have to do it again.

Once the application is working, the last step is to take the final snapshot. The difference between the first and last snapshot is what your install program must do. The tricky bit is figuring out how to do it. Most operating systems have a lot of stuff going on behind the scenes. Your snapshot will invariably capture some of it. To minimize the noise factor, make sure you quit any unneeded applications, stop any unnecessary services, and kill any unrelated processes. You should have the system as ambient as possible. Granted this will take some knowledge of the application, but that's what your developer contact is for. Also, keep in mind that the more the system was mucked with to get the application working, the more noise you'll have in the snapshot. You'll need to spot the difference.

As a final note, try not to use snapshotting software that automatically creates an install project. As we'll see, sifting through the snapshotting data is an important part of the design process. Including the wrong files or not including the right ones is only part of the concern. Making sure the configuration process is correctly performed is the other part.

Ideally, the snapshotting process should be repeated for each operating system that you're targeting. It may seem like a lot of work, but when there are operating system dependencies you'll save yourself a lot of troubleshooting time.

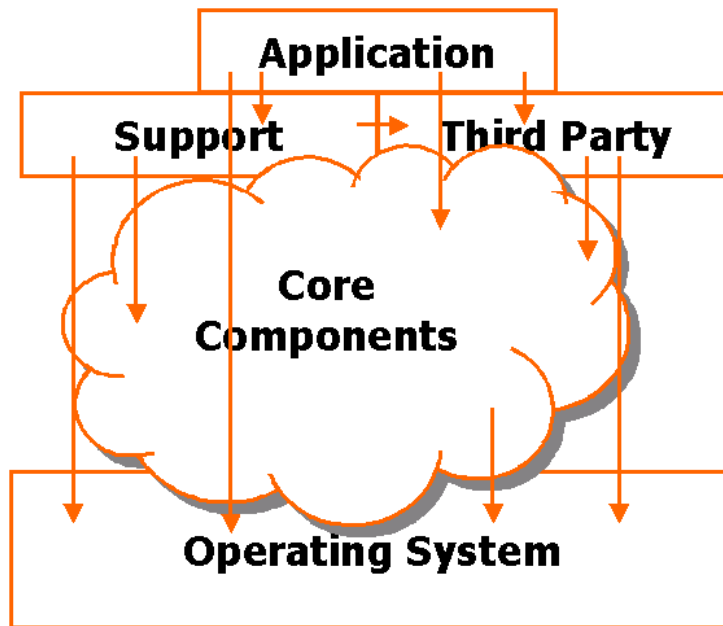
When you're done you should have some kind of text output that lists all the system changes: registry modifications, updated and deleted files, file modifications, and shell objects. Save this file because we'll come back to it in a moment. But first...

Getting a Profile

Did you know that there is an indispensable tool to determine ALL the files that are required by an application? By all the files, I mean statically linked and dynamically called. What are these you may ask? Well, let me take a few minutes to explain.

The Windows operating systems are composed of a collection of dynamically linked library files or DLLs. It's this collection of DLLs that comprise the Windows API set. The complete list of files runs into the hundreds. This list includes files such as KERNEL32.DLL, GUI32.DLL, SYSTEM32.DLL, and so on. These are the very files that make the operating system what it is. Naturally, a great number of applications take advantage of these libraries. Some applications are designed so tightly to these files that they require the files to be located before the application even loads. At other times, the applications are loosely coupled to these same system files. They applications will load, but become unstable when they go to use the library and it's no longer there. The first example is called statically linked. This is accomplished by linking to the library file when the application is compiled. Another way of linking to the file is at runtime. This is often referred to as dynamically linked. The application loads the system file while the application is running and makes a call to an exported routine.

Not only does your application link to files, but also the files they link to may link to other files. This process of a library calling another library calling another library is referred to as chaining. The figure below illustrates just how complicated this process can become. You may be surprised that a relatively small in-house application (6-10 files), may require a support structure of other libraries totaling close to a hundred. Take a look at the following figure for a general idea.



In this figure, you can see via the arrows that all the libraries are related. The application files require support files and perhaps third party files. In turn the support files may require the third party files. At this level, all may require specific core components. Every file sooner or later requires functionality exposed by the operating system libraries. It truly is a messy world.

You may be asking yourself why this matters. Well, it does in an interesting way. As install developers, we'd like to know the complete list of files that an application may need—at any time. While there are a number of tools that can determine what the statically linked files are, it's quite a different matter to determine what files the application loads as it is running. To do this thoroughly, the application must be launched and watched. Once the application is loaded, it must be thoroughly exercised. Every bit of functionality must be used to monitor all of the files that the application requires. This process is called profiling. There aren't as many tools that will do this for you. InstallShield Professional—Windows Installer Edition 2.0 provides built-in functionality for static and dynamic scanning. Another publicly available tool called Dependency Walker is provided from Steve Miller (<http://www.dependencywalker.com>). Both accomplish the same task, but have different objectives. InstallShield's tools are geared towards the Windows Installer project—automatically adding files into the project. Dependency Walker simply provides a list of all linked files.

Keep in mind that the results are only as good as the amount of functionality exposed by running and operating the application. You may find that the software test team is better at getting complete results than you. In fact, since they are continually running and testing the application they'll be able to create and maintain a more complete list than you.

Once you've successfully profiled the application, you'll have a huge list of files. You'll need to sort this list into something meaningful. The files listed can be split into two broad groups: ones you can redistribute and ones you can't. It's your first task to sort out which ones are which.

Redistributable

- In-house files (your company compiles and builds)
- Microsoft core components (ODBC, ADO, RDO, IIS, OLE DB, MFC, VB and so on)
- Third part files (SQL Anywhere, Java Runtime Engines (JRE), JRUN, Crystal Reports, and so on)

Non-redistributable

- System files (part of the operating system)
- Debugging files

There are a number of ways to filter out these files. Let's take a look at a couple of tricks that you can use. The first is to extract the company name from the resource of each file. The easiest way is to right-click on the filename and select Properties from the secondary menu. From the subsequent tabbed dialog, select the Version tab. On this tab, you'll find a number of informational parameters about the file. If it says Microsoft Corporation for the Company Name it's either a part of a Microsoft core component (redistributable), a Microsoft non-redistributable file, or a Microsoft system file. Not much help, but we can delve deeper. If the Company Name refers to anyone else, you'll need to explore the redistribution agreement provided with the software documentation. Ask a developer for this information. For large numbers of files, writing a VBScript is a quick and easy solution.

Another way to filter out these files is to see if they're listed in InstallShield's corecomp.ini file (provided with InstallShield Professional—Standard Edition 6.2) or InstallShield's iswiscan.ini (provided with InstallShield for Windows Installer 2.0). These files list a great number of the Microsoft files. Unfortunately, neither contains a complete list and neither distinguishes between a redistributable core component and a non-redistributable system file.

Let's explore the Microsoft files a bit further. The easiest way to sort out the files is to put Microsoft's Developer's Network (MSDN) to work. For the last couple of years, Microsoft has provided a DLL Help Database that provides versioning information about all the Microsoft files. This reference is located at <http://msdn.microsoft.com>, select the Resources menu, and then select DLL Help Database from the dropdown menu.

As of this writing, the current complete URL is

<http://support.microsoft.com/servicedesks/fileversion/dllinfo.asp?fr=0&sd=msdn>.

Enter the name of the file in question and you'll get a list of all versions of that file. For a particular file, select more information to see which Microsoft product shipped the specified file. Unless you've had some experience working with these files, it's still hard to tell which files are redistributable and which ones are not. The general clue is the 'core component' string. In this context, this core component is not the same as we've been talking about. If you see this string associated with a file several times, you may suspect that it's part of the operating system. The surefire way to know is to search the MSDN Library and Knowledge Base for more information. After poring over a couple developer articles you should get the general idea in a hurry. If in doubt, ask a couple of developers.

Some of the Microsoft redistributables are contained in InstallShield objects (InstallShield Professional—Standard Edition 6.2) and Merge Modules (InstallShield for Windows Installer). If so, you're hunting maybe over. Using the InstallShield Objects and Merge Modules is the simplest way to bundle redistributable files with your install. If you're shipping third party redistributable files, be sure to check out the vendor's website periodically. Many are moving towards Windows Installer and may provide a Merge Module for your Windows Installer project in the near future.

The table below summarizes some of the basic rules of redistributing core components.

Rules for Core Component Installation

1. Install oldest supported versions of core components—don't ship the latest and greatest if your application doesn't require it.
2. Install or self-register only newer core components—force the application team to become compatible with the latest version.
3. Snapshot on the oldest supported minimum requirements system (hardware and software)

4. Keep core components on the user's system—exclude them from the uninstall
5. Install core components only if they are required—not 'just to be safe'
6. Use the specified redistribution mechanism
7. Place third party redistributables in recommended locations (COMMONFILES ^ "Company name")

Separating Fact from Fiction

Okay, now we've taken a snapshot of the system to see how the application is installed. After that, we profiled the running application to generate a complete file list. What do we do with all of this information? Well, we need to integrate it into our install program. The best way to do this is to sort much like we did with the file list we got from profiling.

There is much black magic in this process. Rarely will you get a snapshot that is noise free. Let's spend a few minutes talking about some of the stuff that you'll see and what it means. Or in some cases what it doesn't mean.

First, the golden rule of viewing snapshot results: "Just because it shows up in the snapshot doesn't mean YOU create it." This is closely followed by the silver rule of viewing snapshot results: "Just because it shows up in the snapshot doesn't mean it needs to be created." These will go a long way to helping you sort out the snapshot contents. Let me explain.

The snapshot gives you the end result of changes to the system. It doesn't tell you HOW the changes were made or WHO made them. You'll need to do some research to figure out the differences. Let me give you a simple example—self-registration. If in the snapshot output you see a ton of registry entries that look something like this:

```
[HKEY_CLASSES_ROOT\CLSID\{EAB841A0-9550-11CF-8C16-00805F1408F3}\InprocServer32]
@="C:\\WINNT\\System32\\thumbvw.dll"
"ThreadingModel"="Apartment"
```

Then you're looking at the end results of a DllRegisterServer call from a self-registering file. You do not create these registry entries, you simply need to make sure that the appropriate file is marked as a self-registering file when you link it to your install project. Within InstallShield 6.2, you can do this is done by setting the Self-Registering File Group property. Within InstallShield for Windows Installer, this is done automatically. When you see a bunch of registry keys like the ones above, simply look for the value data associated with the InprocServer32 key. This will give you the name of the self-registering file. After that you can safely ignore any other registry key that uses the same GUID value (the long curly braced hex value).

My point here is if you didn't know better, you'd think that your install program would have to manually create these keys. But now you do know better. Unfortunately there are many more examples of snapshot data that are created as a by-product of some other event. You may have to do some homework to figure out the differences. Let me point out some other examples that are not created manually but via a transient process:

- Type Libraries** Very similar to the CLSID key example, except they will have a TYPELIB string in the registry key. These entries are created by registering a type library file.

- ODBC Drivers and DSNs** These are registry entries that get created via a SQL Windows API call —namely to instantiate the driver if it doesn't exist and to create/modify the data source name. The creation of ODBC drivers and data sources is supported via the InstallShield IDE.

- NT Services** These also show up in the registry, but are created/started via the SCM Windows API calls. The creation of NT Services is supported via the InstallShield IDE.

- Fonts** Fonts are copied to the Fonts folder (a well-known path) and then added to the system via an AddFontResource Windows API call.

There are plenty of others that behave the same way. Carefully sift through the registry snapshot data looking for more. You may need to ask your developer contact for assistance.

Next are file edits. These can also be by-products of other events. This is particularly true if the file resides within the Windows or Windows System folder. There are a number of Control Panel applets that still rely on the settings of system initialization files for their settings. Depending on the snapshotting tool that you're using, you may not capture the file change or what was changed. You may have to run several snapshots to detect content changes within files.

Finally, there are files that added to the system or simply updated. You'll first need to distinguish between files that are created versus files that are installed. Clearly, if it's a binary file the odds are good that the file was installed. Text files and initialization files may be created based on configuration data detected on the computer. While this may be rare, it's worth keeping in mind. Also keep an eye out for the infamous ATL.DLL. There are two versions of this file: an ANSI version for Window9x and an UNICODE version for Windows NT and Windows 2000. You'll need to make sure you get the right one. There may be other files that fall into this category. Thus we have a need for snapshotting on each target operating system.

Updated files are the trickiest. Once you know that a file needs to be installed, you'll need to track down how it gets redistributed. We talked about this during the profiling phase. The next step is to detect which version gets shipped. It's a common misconception that you should always ship the latest version. Resist this temptation and let me explain why.

The goal of an install program is to place the application on the operating system with minimal perturbations. This means we don't want to change anything that already exists if we don't have to. Lots of applications rely on core components. The odds are good that a number of machines that your install will encounter will already have some version of the core component. They just may not have the latest version. But hang on a second. If your application is completely functional with the older version, should you even ship the latest? Especially when you consider that the latest version may break pre-existing applications that are also getting along fine with the older version. The answer is a resounding No! You should ship the oldest version of the core component that your application will work with. If the core component already exists on the target system, you're golden. It's already there so it's not installed. And if it's not installed the odds of breaking anything goes down.

So once you know a core component must be shipped, spend some time tracking down the exact version required. You'll save yourself and the support team some headaches. You'll save your company some money.

My last bit of advice when it comes to files is unorthodox. If you're not sure your application needs a specific file or uses it, don't include it. Make a note of your decision by updating the specification. The reason for this is simple. The install will go through a testing phase. During that testing cycle the install program will hit a large number of system variations. If one of those test scenarios uncover the need for the file then include it. It's easier to detect a missing file than it is to detect an unnecessary file. Sometimes shipping an unnecessary file can cause problems on the end users machine. You'll have enough problems, you don't need to create any more.

Wrapping up

With all of this information now sorted out, we're ready to begin implementing our install program. You'll probably start with the prototype that we created earlier in this cycle. After all, it has the correct dialog sequence and represents a framework project. The first thing to do is create the required Components and File Groups (InstallShield 6.2) or Features and Components (InstallShield for Windows Installer). From there you'll start rolling in the configuration logic from the snapshotting and profiling logic.

Don't forget to keep the install specification up-to-date—especially at this crucial stage. You'll get a number of people asking questions about what the install does or doesn't do. Having a complete and accurate specification at the ready demonstrates your commitment and knowledge of the install process. As an appendix item, you may want to include the entire snapshot and profiling data. When problems arise, you'll find them a handy reference.

The final step is iterative testing. I recommend verifying core logical pieces as they are implemented. Don't wait until you've got a bunch of fixes in the queue. Code-Debug-Test-Code-Debug-Test—it's a good mantra and will save you time. The table below summarizes the steps we've gone through to get to this point. Your mileage may vary, but I believe you'll find it a solid starting point.

Table 3. Process for Creating an Install Program

1. Interview each team member according to the install questionnaire
2. From the interview results, create the first pass of the install specification
3. Create a working prototype
4. Snapshot the application on a clean machine (for each operating system)
5. Profile the application
6. Process the snapshot and profiling results
7. Update the install specification
8. Test and iterate

About the Author

Leslie Easter works as an InstallShield/Windows Installer consultant and trainer for The Orange Brain Company (<http://www.OrangeBrain.com>). When he's not chasing his own install program bugs or fielding student questions, he's working on "Windows Installer Complete" and a second edition of "Bulletproof Installs", both are Prentice-Hall books available early 2001. If you have any questions or comments about this article, Leslie welcomes your comments. He may be contacted at LeslieE@OrangeBrain.com.