SETUP FACTORY 7.0™

User's Guide

NOTE: This is a **low-quality version** of the user's guide with highly compressed screenshots in order to minimize the download size of the evaluation package.

If you would like to download the full print-quality version of this document, simply go to www.indigorose.com/sf and choose "User's Guide" from the Explore menu.

IndigoRose
SOFTWARE DESIGN CORP.
www.indigorose.com

Note: This User's Guide is also available as a professionally printed, perfect-bound manual. To order your copy, please visit www2.ondemandmanuals.com/indigorose.

Welcome!

Introduction

Setup Factory has been raising the bar for over a decade, and we’re proud to present Setup Factory 7.0—the most important release in years. It’s compatible with all versions of Windows from 95 through XP. We’ve improved the development environment to streamline your workflow and introduced unprecedented flexibility with the new customizable Screen Manager, Project Themes, Action Editor and Scripting Engine. Of course, we’re true to our roots; we still make the easiest to use installer builder available anywhere. But now, it’s one of the most powerful solutions as well.

What is Setup Factory?

Setup Factory is a sophisticated tool that gives you complete control over the installation process. Full-featured, fast, and easy to use, Setup Factory's intuitive design maximizes your ability to deploy software products, data files, graphic images, or anything else that you want to distribute.

Creates Compact and Reliable Software Installers

Setup Factory 7.0 creates single-file, compact, and bulletproof professional software installations. Its visual design environment makes it a snap to assemble your files, customize the wizard interface, add runtime support modules, and package your software for deployment via diskette, CD-ROM, LAN, email or Web.

A Proud Heritage

Since its first release, Setup Factory has defined innovation in Windows based setup and installation tools. Over the years, the product line has been the recipient of numerous awards, accolades and glowing reviews.

Trusted by Professionals

Thousands of software developers trust Setup Factory to distribute their software to millions of customers and clients around the world.

While others have tried to imitate it, only Setup Factory 7.0 offers such a perfect combination of ease, flexibility and control.

Key Features in Setup Factory 7.0

Install Any File – Anywhere!

Setup Factory 7.0 features an unbeatable development environment that puts *you* in control of your files. Simply drag and drop your files and folders onto the project window and you're ready to build. Setup Factory is smart enough to maintain your folder structure, automatically query version resource information, create shortcut icons and ensure 100% data integrity with reliable CRC-32 checking. Of course, everything can be customized and overridden if you wish, but for most projects the improved Project Wizard can handle it all with just a few clicks.

Compact Single-File Setups

Smaller and faster means a better experience for your customers and Setup Factory 7.0 delivers. Compare our tiny ~450 KB runtime overhead (including uninstall) to the competition and see for yourself. Additionally, with no "unpacking" step required, Setup Factory 7.0 installers are also much faster to initialize and install than those created by competitive tools. What's more, our Publishing Wizard walks you through the build process with a few easy steps. The single-file setup.exe is ready for distribution by web, email, LAN, CD, DVD and even floppy disk.

Project Quick-Start

Spend five minutes with Setup Factory's easy to use Project Wizard and come away with a complete, ready to build installation project. You'll be walked through each option so you can get your project started as quickly as possible.

Customizable Install Wizard Screens

With a library of more than twenty different screen templates to choose from and a pre-built wizard sequence suitable for the majority of installation tasks, Setup Factory 7.0 is miles ahead of both previous versions *and* the competition. There are pre-built layouts to handle just about any task your installer could want, and it's easy to adjust them to fit your needs exactly. There's everything from check boxes, radio buttons and edit fields to popular screens like license agreements, serial number verification, shortcut folder selection and other advanced options. The Screen Manager allows you to add and remove screens at will and adjust the sequence with a simple drag-and-drop motion. Each screen features a real-time preview so you can see the result of your changes as you work.

Themes and Skins

Choose from dozens of pre-made themes (skins) for your screens or even make your own. It's as easy as viewing a live dialog preview and picking your favorite style. You can configure everything from fonts (face, color, size, style) and banner images to body/background graphics, control colors (buttons, check boxes, radio buttons) and more. Whether your installer needs a corporate feel or a hip attitude, you can do it with Setup Factory 7.0!

Background Window Options

Choose between the traditional setup-wizard style or the modern Windows Installer style interface. You can customize the background window with gradients, images, color washes, headlines and footer text with 3D effects. Now you can also select a custom taskbar icon, force the setup window to remain on top of other windows or even hide the taskbar icon completely for totally silent installs.

Extensive Action Library

Previous versions of Setup Factory have included a handful of system actions for doing routine tasks such as running programs and creating shortcut icons. Setup Factory 7.0 takes it to another level and gives you full control over your installer. You get a built-in library of more than 250 easy to use actions, so you can do whatever you need without having to be an advanced programmer. There are actions to handle everything from text file editing to system registry changes. You can execute programs, call DLL functions, query drive information, manipulate strings, copy files, enumerate processes, start and stop services, interact with web scripts, display dialog boxes and much more.

Easy to Use Action Wizard

You don't have to be a wizard to create powerful installers with Setup Factory 7.0. We've built the wizard into the software! Simply choose the action you want from a categorized list (complete with on-screen interactive help), fill in the requested information fields and the wizard does the rest. You don't have to know anything about scripting or programming – just fill in the blanks and you're done. Making changes is just as easy. Click on the line you want to change and press the "edit" button to go back to the original form. It's really that easy.

Powerful Scripting Engine

We've taken the classic Setup Factory action engine and replaced it with an all-new and incredibly powerful free-form scripting engine. Based on the popular LUA language, this easy to understand scripting language features everything from “for, repeat and while” loops, to “if/else” conditions, functions, variables and tables (associative arrays). Paired with the built-in action library, full mathematical evaluation and Boolean expressions, there’s simply nothing you can’t achieve. Of course, we’ve also built in an “Action Wizard” and “Code Wizard” so even complete novices can create powerful installers that handle even the most demanding installation tasks.

Color Syntax Highlighting Action Editor

If you’ve outgrown the Action Wizard interface or simply want to unleash the power of the fast and efficient scripting engine, we’ve got you covered. The Setup Factory 7.0 freeform action editor features all of the professional features you’d expect. There’s color syntax highlighting, code completion, function highlighting, as-you-type action prototypes, Ctrl+Space function listings and even context-sensitive help. If you’re used to programming in Microsoft® Visual Basic, Microsoft® Visual C++ or any other modern development language, you’ll be right at home.

Expandable with Action Plugins

Setup Factory 7.0 can be easily expanded with Action Plugins. These plugin modules can extend the product in infinitely powerful ways, such as adding support for databases, XML, data encryption and FTP file transfers. Tight integration with the design environment – including IntelliSense style code completion and syntax highlighting – makes them just as easy to use as built-in actions. Plugins are available through Indigo Rose as well as third-party developers thanks to Indigo Rose’s freely available plugin development kit.

Dependency Modules & Runtime Support

Instantly add runtime support for many popular third-party technologies, including Visual Basic 5.0, Visual Basic 6.0, DCOM, JET, MDAC, Visual C++ and more. Additional dependency modules are easy to create and integrate seamlessly into the development environment. Setup Factory 7.0 gives you complete control over the detection and installation of each module, so you can easily adjust the behavior to suite your particular requirements.

Flexible Uninstaller

Setup Factory 7.0 features a new and improved uninstall feature. Simply turn it on and let Setup Factory worry about removing your files. Anything that is installed during the setup will be marked for removal by the uninstaller. However, if you want to go in and tinker with the settings, you’ll find that the uninstaller is now fully customizable.

You can take full control of what files are removed, what shortcut icons are deleted, what registry changes are made and anything else you require. Even the dialog screens and sequence can be adjusted to suit your needs!

System Requirements Checking

You can easily enforce system requirements just by clicking a few boxes. Built-in checks include operating system, memory, screen resolution, color depth and administrator privileges. If the user's system does not meet the minimum requirements you have chosen, the installer can either warn the user or abort the setup entirely.

Serial Numbers, Security and Access Features

Setup Factory includes a variety of features designed to help you manage access to your software, including serial number lists and date-based expiration. With its powerful serial number generator you can quickly create thousands of unique serial numbers, which are stored internally as secure MD5 hashes. Hidden from prying eyes, your installer data is compressed using proprietary algorithms and only accessible to those users who supply a valid serial number. Of course, if you'd rather do it yourself, Setup Factory 7.0 is flexible enough to support your own custom validation and authorization schemes.

Packages, Categories and Install Types

Creating installers for complex and multi-part products is now easier than ever. With dramatically improved support for grouping files into packages, it's a snap to group products and features into a single installer. There's even support for categories so you can group your packages into logical components. It's the ultimate in flexibility for those large projects that need to support different installation types like typical, minimum, complete and custom.

International Language Support

Setup Factory 7.0 offers unsurpassed support for multilingual installations right out of the box. While some products charge you extra for this essential feature (or don't offer it at all), Setup Factory 7.0 gives you everything you need to support your customers and clients around the world. Installers created with Setup Factory can automatically determine the language of the client operating system and adjust the display of screens and messages appropriately. Whether you need to support English, French, German, Spanish, Italian or any other language recognized by Windows, with Setup Factory 7.0 you simply provide the text and your installer takes care of the rest!

Built-in Spelling Checker

Now it’s easier than ever to make sure that typos don’t creep into your projects. Basically anywhere you can type, you can perform a spell check to ensure error-free text. Dictionaries are available for over a dozen languages including English, French, German, Italian, Spanish, Dutch, Swedish, Danish, Croatian, Czech, Polish and Slovenian.

Reports and Logs

Keeping track of the essential details of your installation project is now just a couple of clicks away. With improved HTML-based project reports (featuring CSS formatting) and text-based install-time log files, you’ll have an accurate record of everything you need. New options let you control the level of detail being logged, including options for recording errors and script actions.

Silent Installs

Create silent installations that operate without displaying user interface dialogs, prompts, messages or errors. Easily read command defaults from a response file and control the installation automatically. Silent installs let you maintain control over hundreds or thousands of workstations while enforcing corporate standards. New options let you enable silent installs with a command line switch, or even force the installer to always run in unattended mode. Setup Factory 7.0 installers automatically return command line status codes and can be easily called from batch files and automatic build processes.

Visual Basic Project Scanner

If you’re creating installers for your Visual Basic projects, then look no further! Setup Factory can analyze your VB project and automatically add the necessary runtime files and dependencies for you. There’s even an advanced executable scanner that can assist you in determining required DLL’s and libraries for those difficult to manage multi-team projects.

Unattended Builds

Setup Factory 7.0 fits seamlessly into your daily build process. Creating your product installer every time you build your source code makes it easy to test early and often. Simply include your Setup Factory project in your build process to run automatically and output a finished setup.exe.

Works with Windows 95 and Up

While other installer tools are dropping support for older operating systems such as Windows 95 and 98, we believe that your choice of installation builder shouldn’t limit your potential market. Setups created with Setup Factory 7.0 work just fine on every Windows operating system from Windows 95 to XP and beyond.

About this Guide

This user's guide is intended to teach you the basic concepts you need to know in order to build a working installer. You'll learn the ins and outs of the program interface and how to perform many common tasks.

The guide is organized into 12 chapters:

Chapter 1:	Getting Started
Chapter 2:	Working with Files
Chapter 3:	Creating the User Interface
Chapter 4:	Actions, Scripts and Plugins
Chapter 5:	Session Variables
Chapter 6:	Packages
Chapter 7:	Languages
Chapter 8:	Security and Expiration
Chapter 9:	Dependencies
Chapter 10:	Uninstall
Chapter 11:	Building and Distributing
Chapter 12:	Scripting

Each chapter begins with a brief overview and a list of the things you will learn in that chapter.

Document Conventions

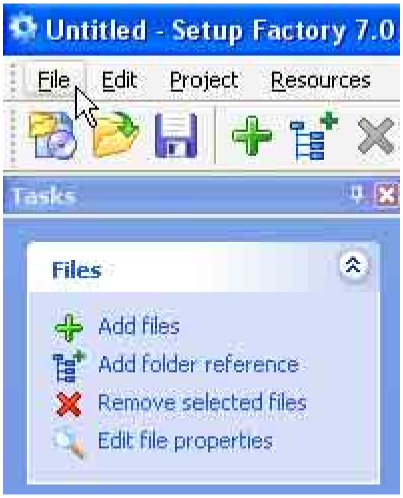
This user's guide follows some simple rules for presenting information such as keyboard shortcuts and menu commands.

Keyboard Shortcuts

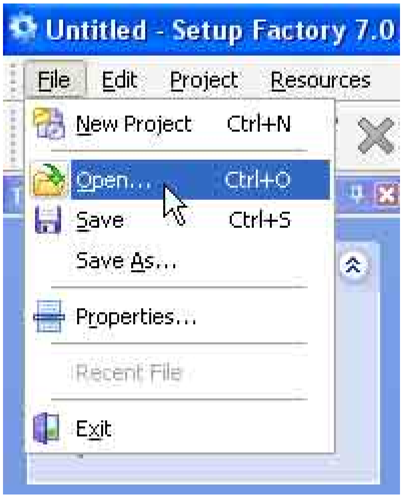
Keyboard shortcuts are described like this: press Ctrl+V. The “+” means to hold the Ctrl key down while you press the V key.

Menu Commands

Menu commands are described like this: choose File > Open. This means to click on the File menu at the top of the Setup Factory program window, and then click on the Open command in the list that appears.



Click on the File menu...



...and click on the Open command

Typed-In Text

When you're meant to type something into a text field, it will be presented in italics, like this: type *"Setup Factory makes me happy"* into the Message setting. This means to type in "Setup Factory makes me happy", including the quotes.

Chapter 1:

Getting Started

Every journey begins with a first step. In this chapter, we’ll walk you through the creation of a new project and introduce you to the Setup Factory program interface.

1

In This Chapter

In this chapter, you'll learn about:

- Starting a new project
- Making sure you have the latest version
- Learning the interface
- Getting help
- Setting preferences
- Planning your installation

Starting a New Project

Everything has to start somewhere. In Setup Factory, the design process starts with the creation of a new project.

A project is simply the collection of files and settings and everything else that goes into building an installer. A typical project will contain all of the files that you want to distribute, some screens that inform or gather information from the user, and maybe a few actions to take care of any "extras" (such as storing the installation path in a registry key for future use by your patching tools).

The project file

Each file that you add to a Setup Factory project has individual settings that control where, when and how the file will be installed at run time. Likewise, each screen that you can display has its own properties that determine everything from the text that appears on the various parts of the screen to the color of the screen itself.

These settings are all stored in a single file called the *project file*. The project file contains all of the properties and settings of a project and the list of source files that need to be gathered up each time the project is built.

The project file is automatically created for you when you start a new project.

When you start a new project, Setup Factory's project wizard walks you through the first few steps of project creation. This helps you get your project started quickly without missing any of the basics.

Let's open the Setup Factory program and start a new project.

1) Open Setup Factory.

Use the Start menu to launch the Setup Factory program.

You'll find Setup Factory under:

Start > Programs > Indigo Rose Corporation > Setup Factory 7.0

2) Read the Tip of the Day, and click the Close button.

When you open Setup Factory for the first time, the Tip of the Day dialog appears to give you one of many helpful tips for working with the program.

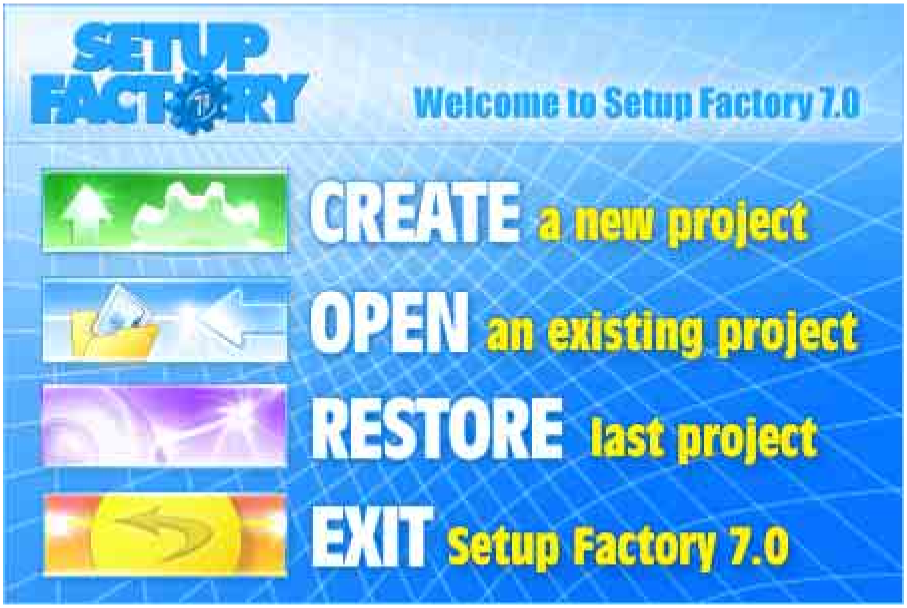
You can browse through the tips right away by clicking the Next button. If you don't want to be shown a new tip each time you start the program, click on the "Show tips at startup" option to deselect it.

Tip: You can also access the tip of the day by choosing Help > Tip of the Day while you're working on a project.

Once you're done reading the tip of the day, click the Close button to move on to the Welcome dialog.

3) When the Welcome dialog appears, click on "Create a new project."

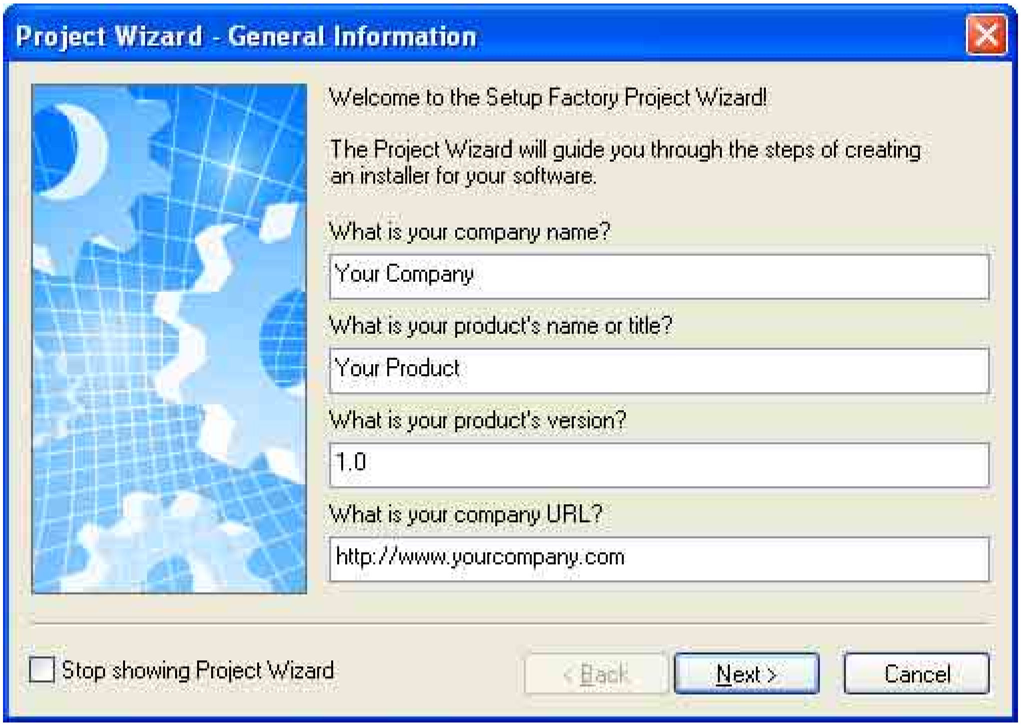
The Welcome dialog appears whenever you run Setup Factory. It not only welcomes you to the program; it also lets you easily create a new project, open an existing one, or restore the last project you worked on. (Restoring the last project automatically opens the project you were working on the last time you ran Setup Factory.)



When you click on "Create a new project," the Welcome dialog closes and the project wizard appears.

4) Enter your information and click Next.

The first step of the project wizard asks you for four pieces of information related to your project. Simply enter your company name, product name, product version, and web address into the appropriate fields.



Project Wizard - General Information

Welcome to the Setup Factory Project Wizard!

The Project Wizard will guide you through the steps of creating an installer for your software.

What is your company name?

Your Company

What is your product's name or title?

Your Product

What is your product's version?

1.0

What is your company URL?

http://www.yourcompany.com

□ Stop showing Project Wizard

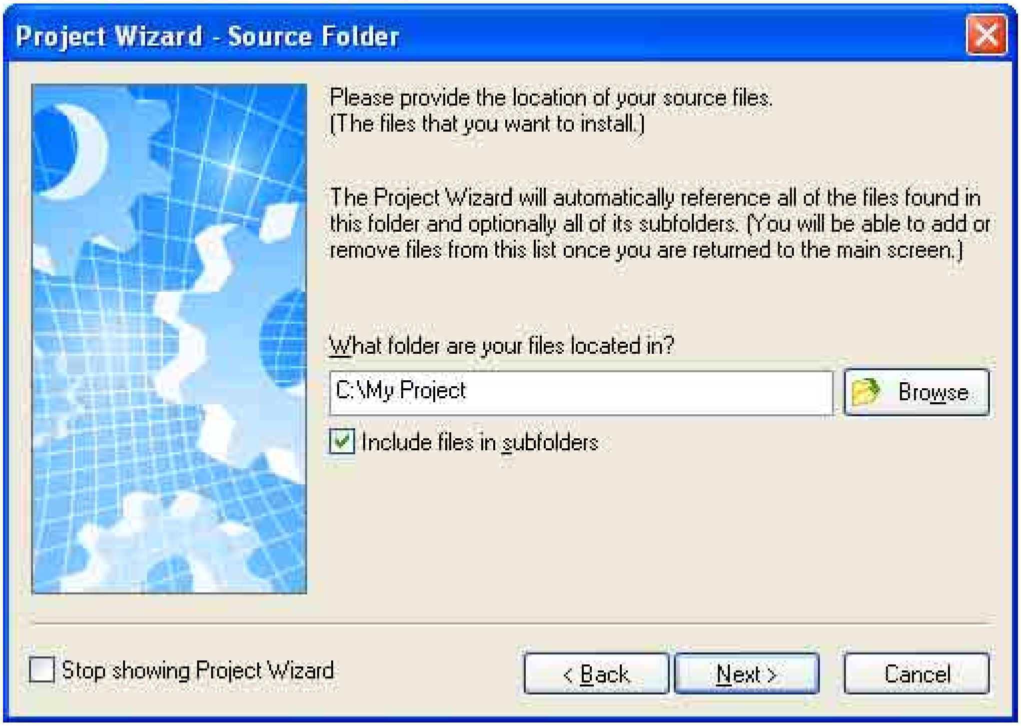
When you've entered all your information, click Next to move to the next step in the project wizard.

Tip: At any step in the project wizard, you can click Cancel to go straight to the program window with no files added and all of the default settings untouched. If you find that you don't use the project wizard, and want something a little more permanent than clicking Cancel, you can turn on the "Stop showing Project Wizard" option. Turning on that option will make Setup Factory open a blank project all the time, without showing you the project wizard. For now though, leave this option turned off so you can see what the project wizard contains.

5) Provide the full path of the folder where your files are located, and click Next.

The next step in the project wizard is to specify the folder where your source files are located.

Note: In this case, “source files” refers to the original files on your local hard drive, i.e. the files that you want to create an installer for, and not files containing source code. Setup Factory doesn’t need access to your source code files at all (unless you intend to distribute them in your installer).



Project Wizard - Source Folder

Please provide the location of your source files.
(The files that you want to install.)

The Project Wizard will automatically reference all of the files found in this folder and optionally all of its subfolders. (You will be able to add or remove files from this list once you are returned to the main screen.)

What folder are your files located in?

C:\My Project

☑ Include files in subfolders

□ Stop showing Project Wizard

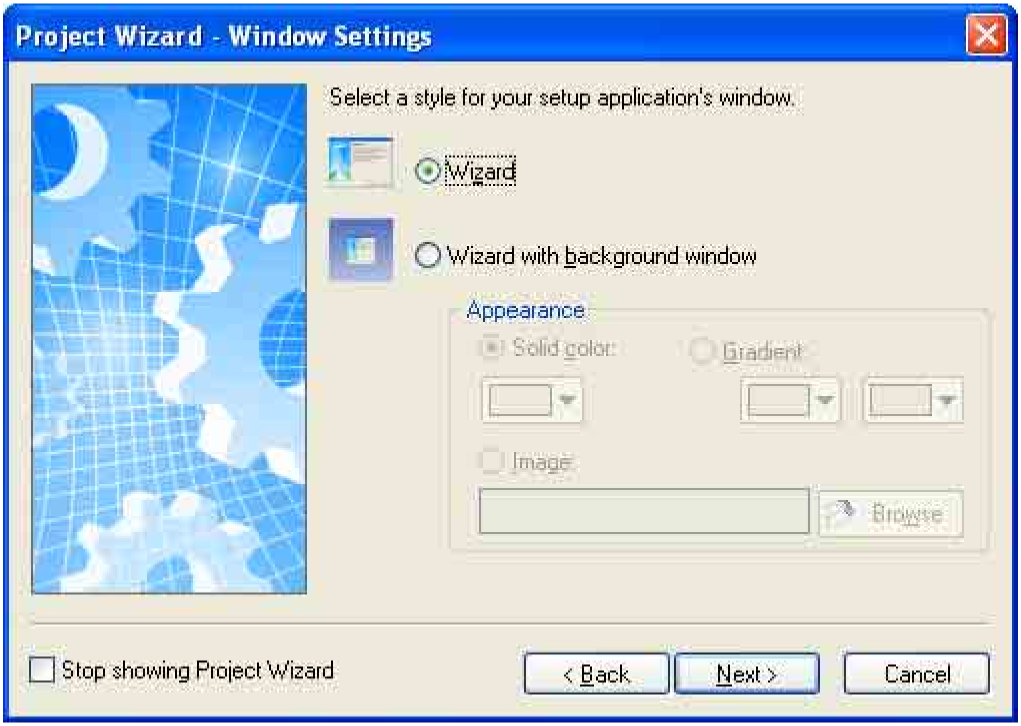
Tip: You can use the Browse button to browse for a folder.

If your product files are spread across several folders, choose the main folder that represents the core of your product. By default, all of the files and folders contained within the folder that you select will be included as well, so you want to choose the folder at the “top” of your product’s folder hierarchy.

Once you’ve selected the folder, click Next to proceed to the next step.

6) Select a style for your setup application window, and click Next.

This step determines whether or not your installer will have a background window behind it covering the user's desktop.



Project Wizard - Window Settings

Select a style for your setup application's window.

● Wizard

○ Wizard with background window

Appearance

● Solid color: ○ Gradient

□ Image

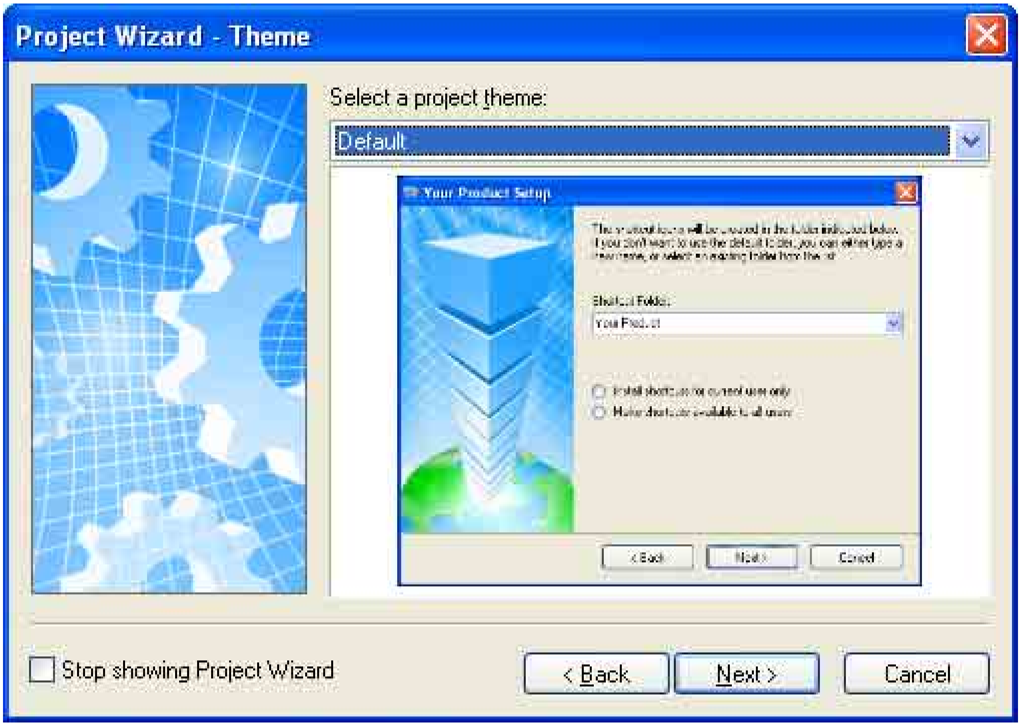
Browse
□ Stop showing Project Wizard
< Back
Next >
Cancel

Note: If you choose to have a background window, you can customize its appearance using the settings in this project wizard step.

Once you've selected your window style, click Next to proceed to the next step.

7) Select a visual theme for your installer's interface, and click Next.

Setup Factory 7.0 has brand new support for *project themes*. Project themes are a set of visual attributes that are applied to all of the screens in your installer to give them a uniform look and feel. These attributes include color, banner styles, images, fonts, and more. You can change the look throughout your entire installer by simply switching to a different theme.



Project Wizard - Theme

Select a project theme: Default

□ Stop showing Project Wizard

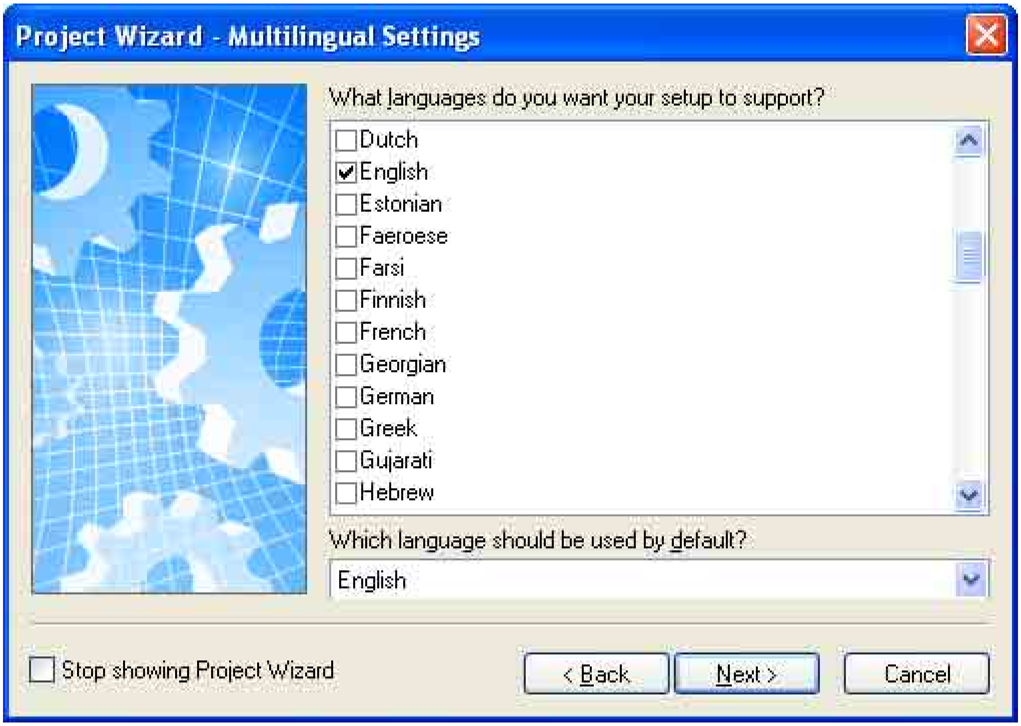
Below the theme selector, a preview image shows what a sample screen would look like with the theme applied.

Note: The selected theme will be applied to all screens in your project. However, you can override the theme settings on a per-screen basis using the “Override project theme” option on each screen’s Style tab.

Feel free to try out some of the different themes. Once you’ve picked one that you like, click Next to proceed to the next step in the project wizard.

8) Select the languages that you plan to support in your installer, and click Next.

The “Multilingual Settings” step of the project wizard asks you to select the languages that you want your installer to support. Every language in the list that has a checkmark next to it will be supported in your setup.



Project Wizard - Multilingual Settings

What languages do you want your setup to support?

- □ Dutch
- ☑ English
- □ Estonian
- □ Faeroese
- □ Farsi
- □ Finnish
- □ French
- □ Georgian
- □ German
- □ Greek
- □ Gujarati
- □ Hebrew

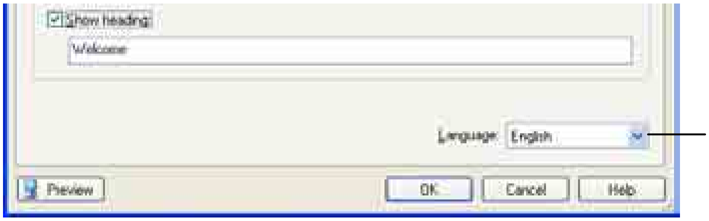
Which language should be used by default?

English

□ Stop showing Project Wizard

Two things happen when a language is supported. First, Setup Factory will use the text from that language's message file (if one exists) for the installer's built-in messages when that language is detected at run time. (If a message file doesn't exist for a particular message, the default language's message file will be used.)

Second (and more important), you will be able to localize the text in your project for each supported language. For example, if you choose English, French and German as your three supported languages, you will be able to enter different English, French and German text on your project's screens, package descriptions, etc. This is done by simply choosing a different language from the *language selector* that appears wherever there is text that you can translate in your project.



☑ Show heading:

Welcome

Language: English

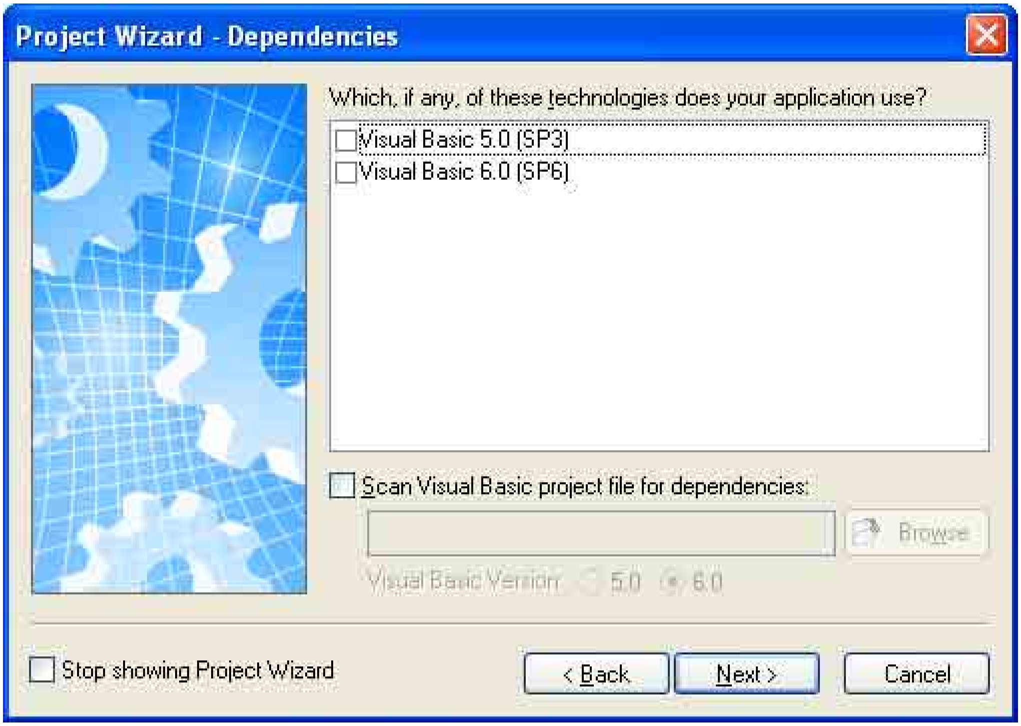
Language selector

Note: Supporting a language adds it to the list of languages available in the language selector.

Once you've selected the languages you want to support, click Next to proceed to the next step.

9) Select any technologies that your product requires, and click Next.

If your product has any requirements, such as the VisualBasic or .NET runtime, select them in the list. Setup Factory will automatically check for the selected technologies at run time and will require the user to install them before performing the installation.



Project Wizard - Dependencies

Which, if any, of these technologies does your application use?

□ Visual Basic 5.0 (SP3)
□ Visual Basic 6.0 (SP6)

□ Scan Visual Basic project file for dependencies:

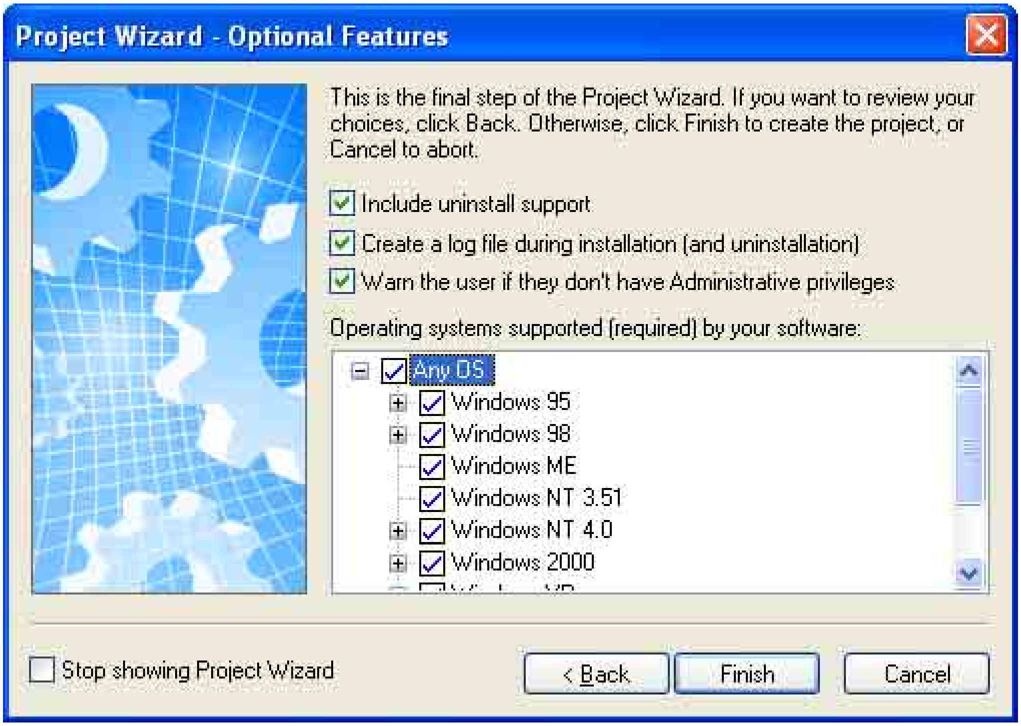
Visual Basic Version ○ 5.0 ● 6.0

□ Stop showing Project Wizard

Select your product's requirements in the list, and click Next to proceed to the final step.

10) Select the optional features you want, and select the operating systems that your product will work on. Once you're ready, click Finish to apply your settings to your new project.

The final step of the project wizard lets you configure a number of optional features, such as whether to automatically provide an uninstaller, and whether to alert the user if they attempt to install without Administrative privileges. (Usually you'll want to leave all of these features turned on.)



Project Wizard - Optional Features

This is the final step of the Project Wizard. If you want to review your choices, click Back. Otherwise, click Finish to create the project, or Cancel to abort.

☑ Include uninstall support

☑ Create a log file during installation (and uninstallation)

☑ Warn the user if they don't have Administrative privileges

Operating systems supported (required) by your software:

- ☑ Any OS
  - ☑ Windows 95
  - ☑ Windows 98
  - ☑ Windows ME
  - ☑ Windows NT 3.51
  - ☑ Windows NT 4.0
  - ☑ Windows 2000

☐ Stop showing Project Wizard

You can also configure the operating system requirements for your project. Setup Factory will automatically limit the installer to only perform the installation on operating systems that are selected (checked) in this list. So, for example, if you do not want anyone to install your software on Windows 95, simply turn off support for that operating system, and the installer will automatically abort (with a short explanation) when it's run on that OS.

When you click Finish, the project wizard closes and the new project is loaded into the design environment with the settings you chose.

Note: The Setup Factory program interface is also known as the *design environment*.

If you selected a folder with files in it in step 5, those files will automatically be added to the file list for you.

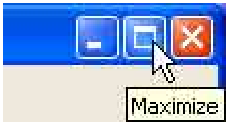
At this point, you could build the project and generate a basic installer for those files. Of course, you'd probably want to customize the screens before distributing it. To learn how to customize the screens in your installer, see Chapter 3.

Tip: Once you're in the design environment, you can start a new project by choosing File > New.

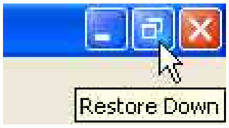
11) Maximize the Setup Factory program window.

The easiest way to work with Setup Factory is with the program window *maximized* so it covers the whole screen. This way, you have the whole desktop area to work with, and you won't have any other programs or windows in the background to distract you.

To maximize the window, click on the little Maximize button, which is the second button from the right on the Setup Factory title bar (right next to the Close button).



If the maximize button looks like this:



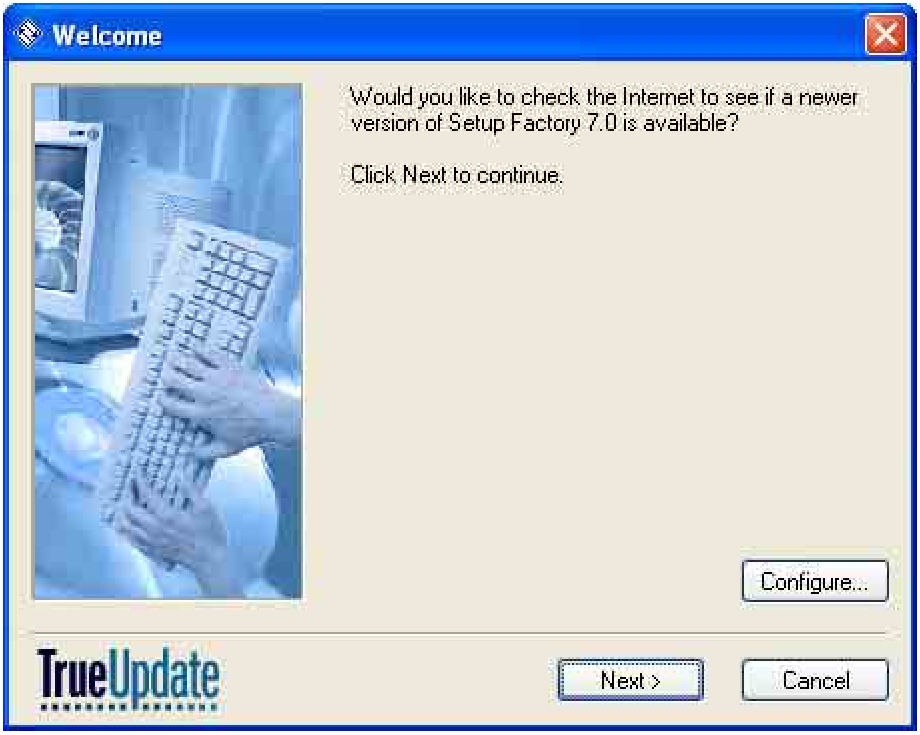
...it means you already have the program window maximized. (That button is actually the Restore button, which takes the place of the Maximize button while the window is maximized. If you click the Restore button, the window will return to the size and position it had before you maximized it.)

Making Sure You Have the Latest Version

Setup Factory has the built-in ability to check the Internet to see if there is an update available. Before we start exploring the program interface, let's use this feature to make sure you have the latest version of the program.

1) Choose Help > Check for Update.

The TrueUpdate wizard will open.



Would you like to check the Internet to see if a newer version of Setup Factory 7.0 is available?

Click Next to continue.

TrueUpdate is an Internet update technology developed by Indigo Rose Software that makes it easy to add automatic updating capabilities to any piece of software.

You can use the Configure button on this dialog to adjust the connection settings if required. If you're accessing the Internet from behind a proxy, you will probably need to make some changes in order for the update to work.

(The default settings work 99% of the time, though, so you probably don't need to worry about it.)

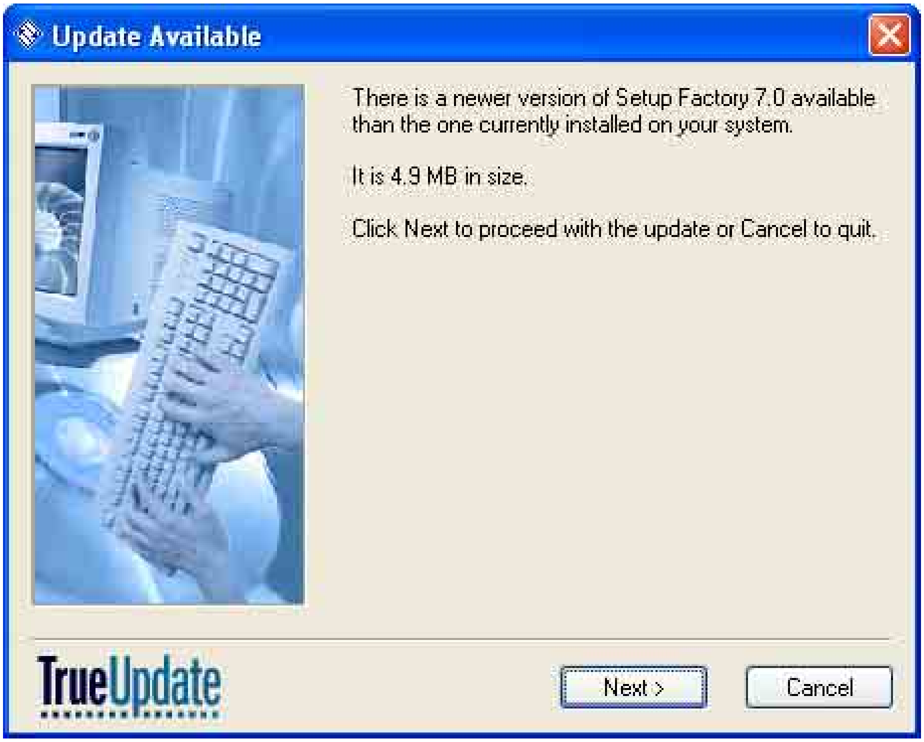
2) Click Next.

When you click Next, the TrueUpdate wizard will connect to the Indigo Rose website and determine whether there is a newer version of Setup Factory available for you to download. If there is, it will give you the option to download and apply a patch that will update your copy of Setup Factory to the latest version.

Note: If you are running any Internet firewall software such as ZoneAlarm, it may ask you whether to permit the TrueUpdate Client to connect. You will need to allow the client to connect in order for the update to work.

3) If an update is available, click Next and follow the instructions to update your software to the latest version. Otherwise, click Close to exit the TrueUpdate wizard.

If an update is available, you will see something like this:



Update Available

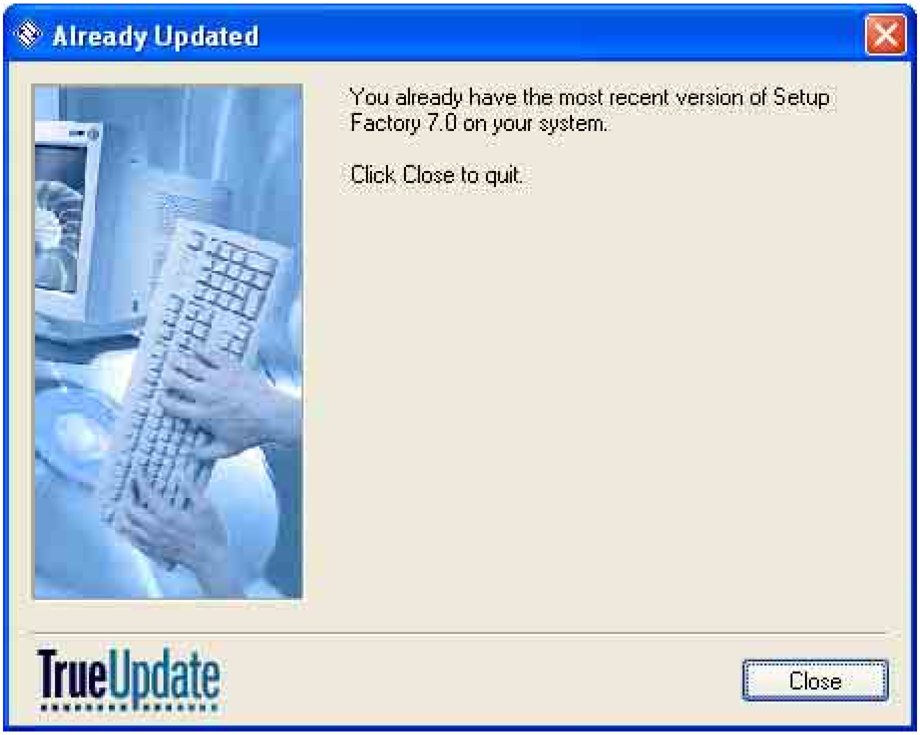
There is a newer version of Setup Factory 7.0 available than the one currently installed on your system.

It is 4.9 MB in size.

Click Next to proceed with the update or Cancel to quit.

Just follow the instructions on the wizard to complete the update.

If an update isn't available, you will see this message instead:



Already Updated

You already have the most recent version of Setup Factory 7.0 on your system.

Click Close to quit.

In that case, you already have the latest version installed, so just click Close to exit the TrueUpdate wizard.

Learning the Interface

Now that you have Setup Factory started and you've made sure that you're using the latest version, it's time to get comfortable with the program interface itself.

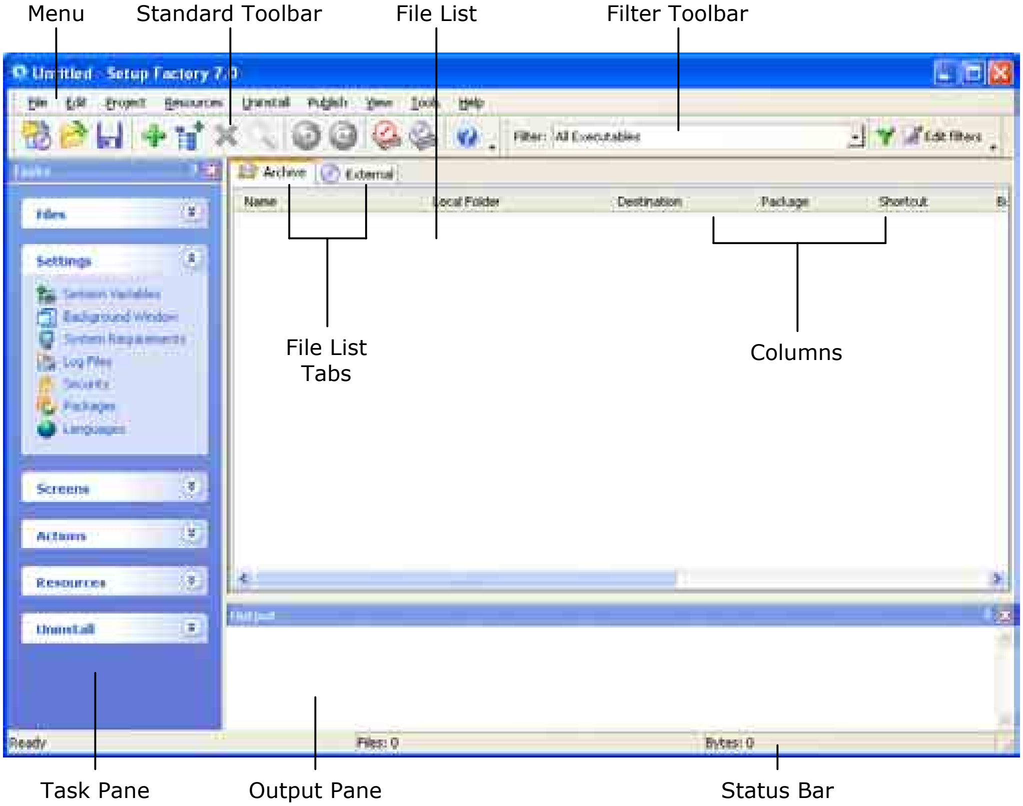
1) Explore the Setup Factory program window.

The Setup Factory program window is divided into a number of different parts.

At the top of the window, just under the title bar, is the program menu. You can click on this program menu to access various commands, settings and tools.

Below the program menu are a number of toolbars. The buttons on these toolbars give you easy access to many of the commands that are available in the program menu.

Tip: You can create your own custom toolbars or edit the existing ones by choosing Tools > Customize.



Most of the program window is taken up by the file list, which is where all the files in your project are listed. There are actually two file lists—one for files that are packed into the setup executable's data archive, and one for external files that are distributed along with your setup executable. You can switch between the Archive file list and the External file list by clicking on their respective tabs.

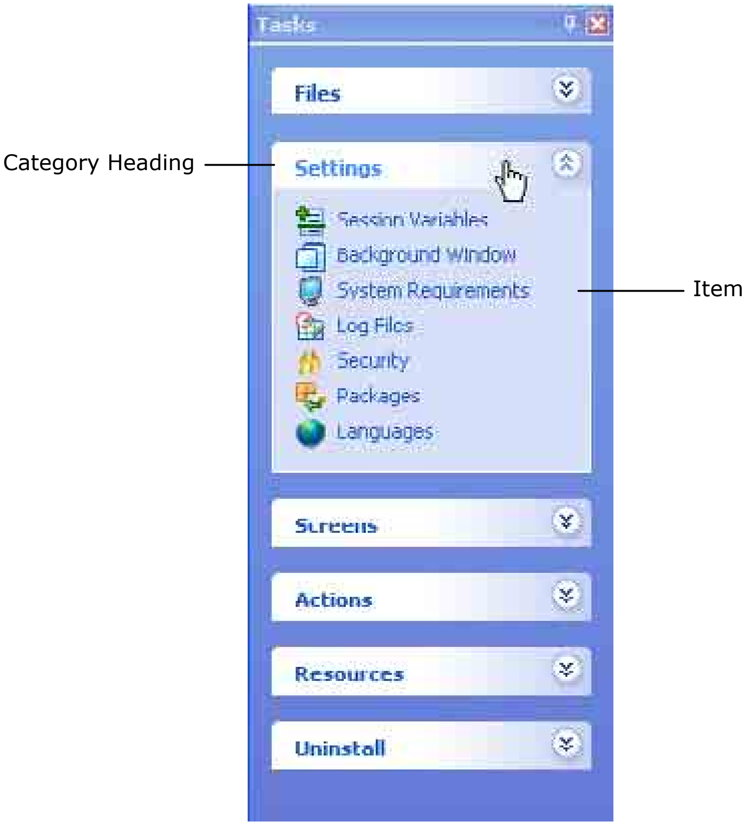
At the very bottom of the window, a status bar reflects your interaction with the program and offers a number of informative readouts.

The rest of the program window is made up of individual sub-windows known as *panes*. Each pane can be docked, tabbed, pinned, resized, dragged, and even made to float on top of the design environment.

The long horizontal pane near the bottom of the program window (just above the status bar) is the output pane. This is where Setup Factory displays the progress of lengthy operations, such as report generation, project conversion, and building the project (unless the publish wizard is being used, since it has its own output window).

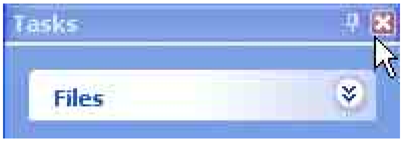
The tall pane on the left is the task pane. The task pane provides easy access to the parts of Setup Factory you will use the most. It provides an alternative to the program menu for accessing the various parts of the Setup Factory design environment.

The items on the task pane are organized into categories. You can expand or collapse these categories by clicking on the category heading.



2) Close the task pane.

You can close a pane by clicking on the little x on its title bar.

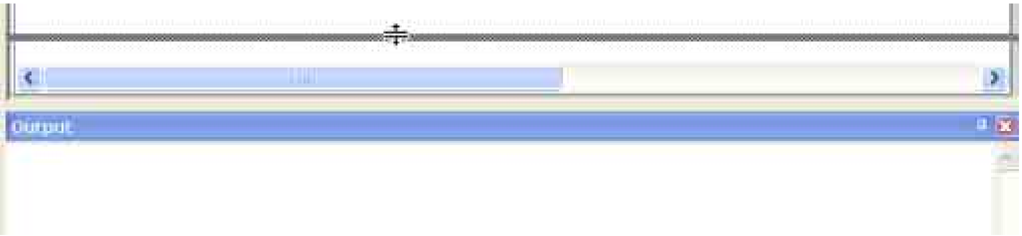


3) Choose View > Tasks to open the task pane again.

All of the panes can be toggled on or off in the View menu. When you choose View > Tasks, the task pane is restored to the same position it occupied before you closed it.

4) Make the output pane taller by dragging its top edge up.

You can resize panes by dragging their edges. In this case, you want to drag the part “between” the output pane and the file list...specifically, the little bit of window surface below the file list and above the output pane’s title bar. As you begin to drag the edge of a pane, a line will appear to show where the edge will move to when you release the mouse button.

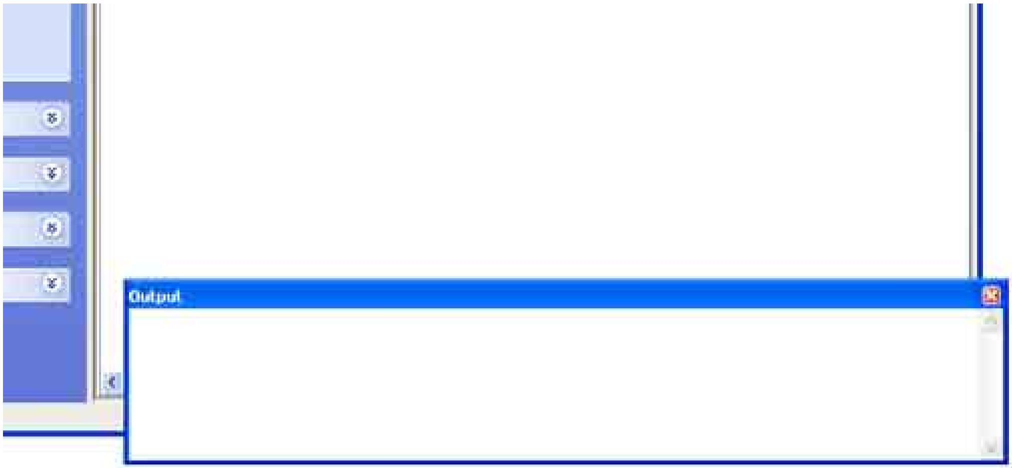


When panes are docked alongside each other, dragging an edge will resize both panes at the same time. As you make the output pane taller, you will be making the file list shorter at the same time.

5) Double-click on the output pane’s title bar to un-dock it.

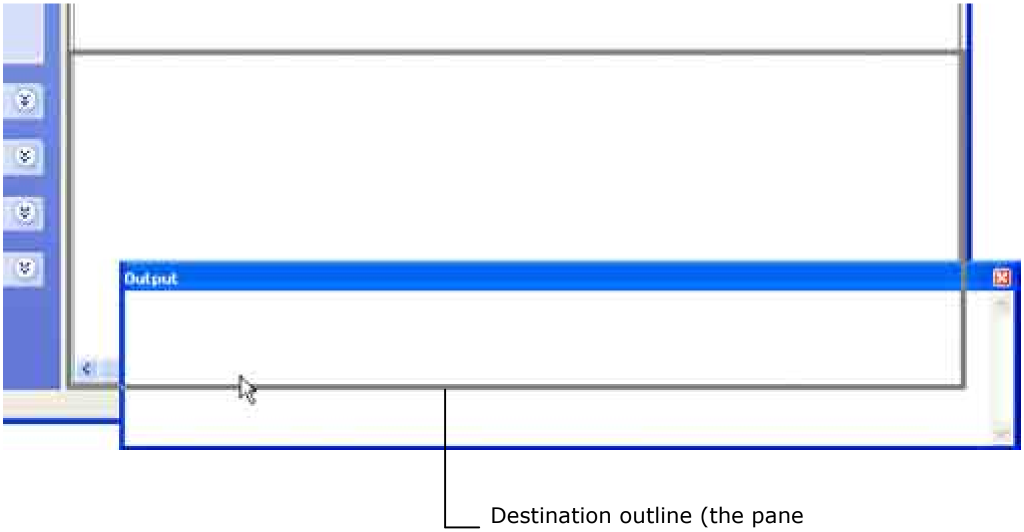
The output pane is *docked* by default. A docked pane is seamlessly integrated into the program window. You can “un-dock” it from the program window by double-clicking on its title bar.

When you un-dock the output pane, it floats above the program window, and the file list expands to fill in the space that the output pane left behind.



6) Drag the output pane to the bottom edge of the program window to dock it again.

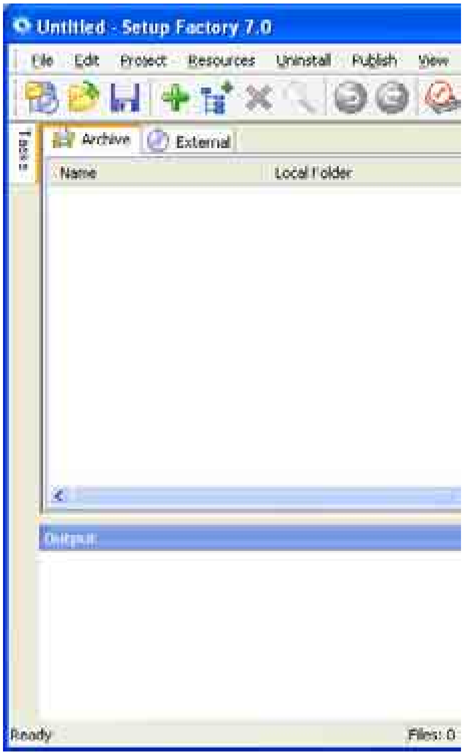
You can move panes around by dragging them by their title bars. As you move a pane, an outline shows you the general area where the panel will end up. If you drag the pane near the edge of the program window, or near another pane, the outline will “snap” to show you how the pane can be docked, tabbed, or otherwise combined with the target area.



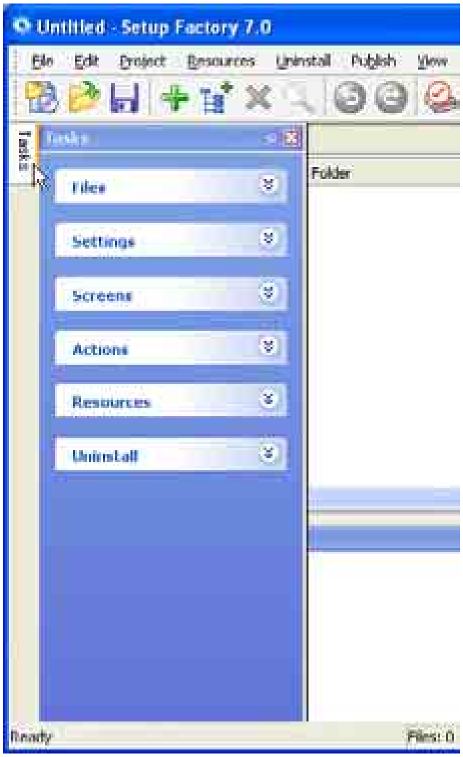
Destination outline (the pane will be docked below the file list)

Tip: When you're dragging panes, it's the position of the mouse cursor that determines where the outline snaps into place. For example, to dock a pane below another one, drag the pane so the cursor is near the bottom edge of that pane. To "tab" one pane with another, drag the pane so the cursor is on top of the other pane's title bar.

Docked panes can also be "pinned" or "unpinned." *Pinned* panes remain open when you're not using them. (All of the panes in the default layout are pinned.) *Unpinned* panes stay out of the way until you click on them or hover the mouse over them. Whenever you need them they "slide" open, on top of everything else, and then slide closed when you're done.

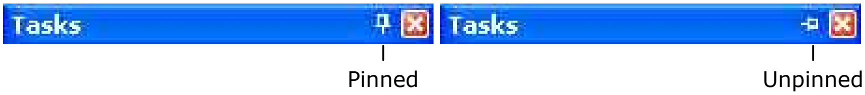


The task pane unpinned ...



...and after "sliding" open

You can pin or unpin a pane by clicking the little pin icon on the pane's title bar.



Panes remember their positions even after you unpin them. If you unpin a pane, and then pin it again, it will return to the position it had before it was unpinned.

Getting Help

If you still have questions after reading the user's guide, there are many self-help resources at your disposal.

Here are some tips on how to quickly access these self-help resources.

1) Press the F1 key.

The online help is only a key press away! Setup Factory comes with an extensive online program reference with information on every object, action, and feature in the program.

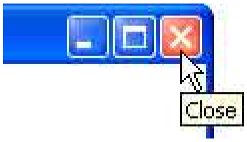
In fact, whenever possible, pressing F1 will actually bring you directly to the appropriate topic in the online help. This context-sensitive help is an excellent way to answer any questions you may have about a specific dialog or object.

Note: You can also access the online help system by choosing Help > Setup Factory Help.

There are three ways to navigate the online help system: you can find the appropriate topic using the table of contents, or with the help of the keyword index, or by searching through the entire help system for a specific word or phrase.

2) Close the online help window and return to the Setup Factory design environment.

To exit from the online help, just click the Close button on the help window's title bar.



3) Choose Help > User Forums.

Setup Factory is used by thousands of people worldwide. Many users enjoy sharing ideas and tips with other users. The online forums can be an excellent resource when you need help with a project or run into a problem that other users may have encountered.

Choosing Help > User Forums opens your default web browser directly to the online user forums at the Indigo Rose website.

4) Close your web browser and return to the Setup Factory design environment.

Exit from your web browser and switch back to the Setup Factory program.

Alternatively, you can press Alt+Tab to switch back to Setup Factory while leaving the web browser open in the background.

5) Choose Help > Technical Support > Support Options.

This takes you to the Setup Factory web site, where a variety of online technical support resources are available to you, including a large knowledge base with answers to some common questions.

This is also where you can find information about ordering one of our premium support packages and submitting a support request.

6) Close your web browser and return to the Setup Factory design environment.

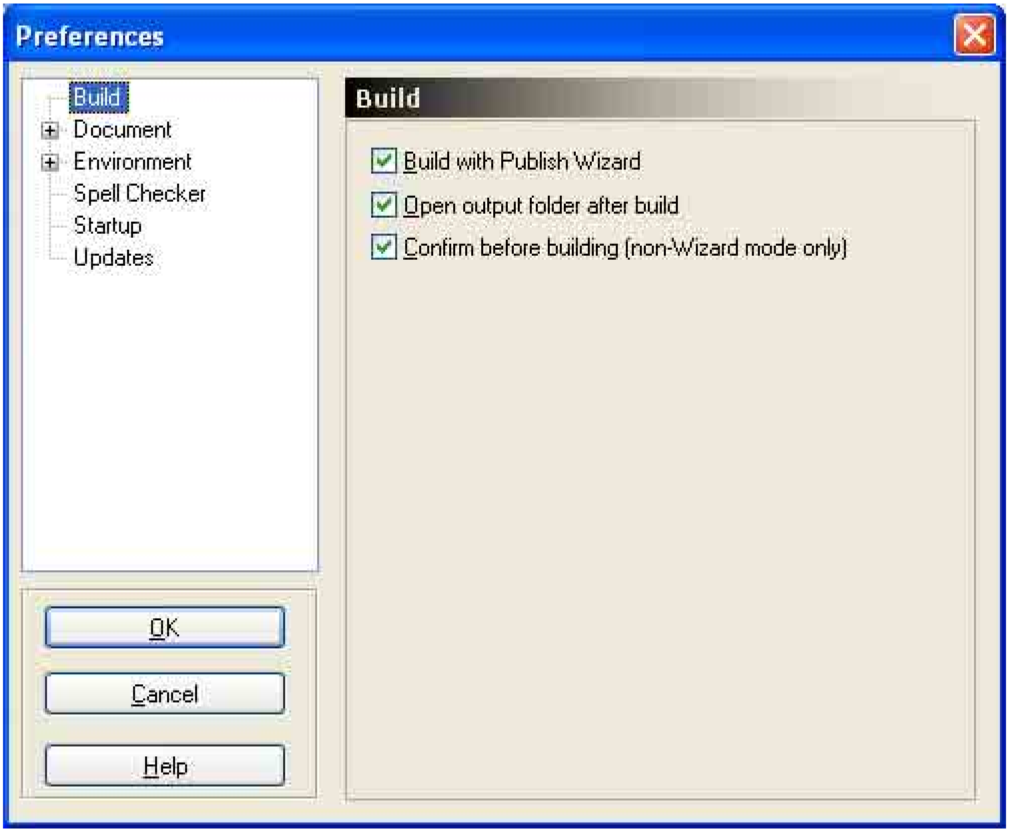
When you’re done browsing the technical support information, return to the Setup Factory design environment to continue with this chapter.

Setting Preferences

There are a number of preferences that you can configure to adjust the Setup Factory design environment to suit your work style. Let's have a look at some of them.

1) Choose Edit > Preferences.

This will open the Preferences dialog, where all of Setup Factory's preferences can be found.



☑ Build with Publish Wizard

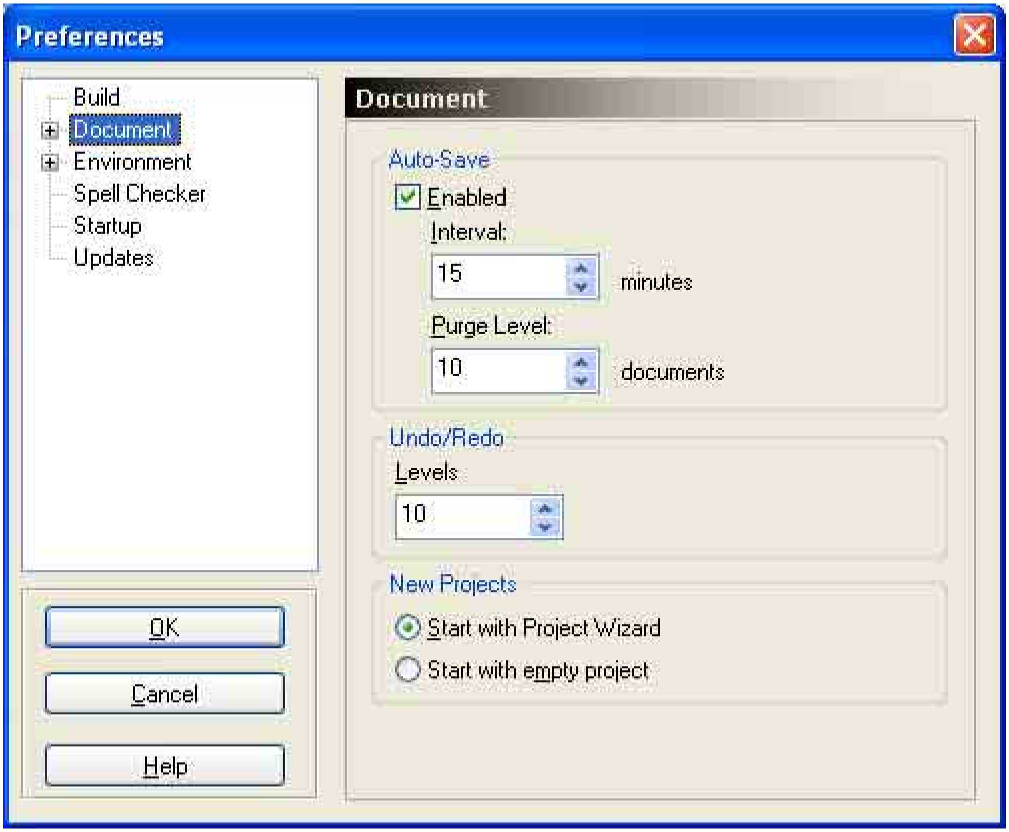
☑ Open output folder after build

☑ Confirm before building (non-Wizard mode only)

The preferences are arranged into categories. The categories are listed on the left side of the dialog. When you click on a category, the corresponding preferences appear on the right side of the dialog.

2) Click on the Document category.

The Document preferences allow you to change settings that affect the project file. For example, you can configure the auto-save feature that can automatically save your project file as you're working on it to avoid any accidental loss of data. You can also configure the number of undo/redo levels, and choose either to use the project wizard to create new projects or to simply start with a new, blank project.



Document

Auto-Save

☑ Enabled

Interval: 15 minutes

Purge Level: 10 documents

Undo/Redo

Levels 10

New Projects

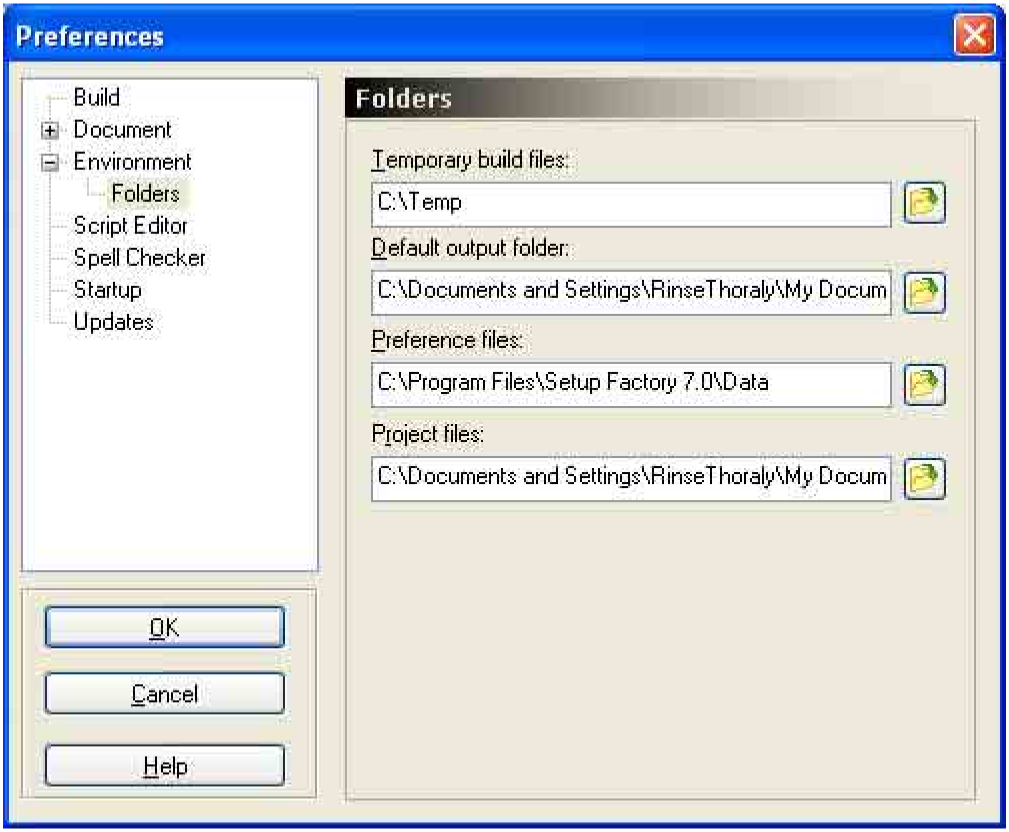
● Start with Project Wizard

○ Start with empty project

Tip: It can be helpful to set the number of undo levels to a larger value, like 25 or 50. That way you can undo even more "steps" back if you change your mind while you're working on a project.

3) Expand the Environment category by double-clicking on it, and then click on the Folders category.

You can also expand the Environment category by clicking on the little plus symbol to the left of it.



Temporary build files: C:\Temp

Default output folder: C:\Documents and Settings\RinseThoraly\My Docum

Preference files: C:\Program Files\Setup Factory 7.0\Data

Project files: C:\Documents and Settings\RinseThoraly\My Docum

The Folders category allows you to specify the locations of various folders that are used by the project. For example, you can specify the location of the project folder (where project files are stored) and the default output folder (where setup executables are built to by default).

4) Feel free to explore some of the other categories. When you're done, click OK to close the Preferences dialog.

There are many other preferences that you can set, such as what to do when the design environment is started (in the Startup category) and what happens when you add files to the project (in the Document > Adding Files category). Take some time to look through the categories and familiarize yourself with the different options that are available.

Remember that you can click Help or press F1 to get more information about any of the settings in a specific category.

Planning Your Installation

The first step in creating a successful software installation is to plan it out.

In fact, planning your installation is one of the most *important* steps in designing a professional installer. Knowing what your installer needs to accomplish in advance will give you a clear goal to aim for and a solid plan to follow.

Tip: As with most things, the more planning you do now, the less repairing you'll have to do later. Investing some time in planning at the start can save you a lot of time in the long run.

There are several things you need to know in order to create an installer:

- What files do you need to distribute?
- Where do your files need to be installed?
- What system changes need to be made?
- What information do you need from the user?

This section will provide you with the information you need to answer these questions as quickly and accurately as possible.

What Files Do You Need to Distribute?

The first step in preparing your installation is to determine exactly what files your software requires for proper operation.

There are four basic types of files that you may need to distribute: program files, configuration files, operating system components, and shared application resources.

Program Files

Program files are the “main files” of your application. These files are essential to your application and are only useful in the context of your application. They can be executables, help files, documents, templates, or any other data files that your application requires. Program files usually make up the bulk of your software.

Configuration Files

Configuration files are used to store startup options, user settings, and other configuration options for your software. Information can be read from and written to these files during the normal operation of your application. These files are often referred to as “config” files, and come in many different formats, from traditional “INI” files (which adopt the same internal structure as Windows .ini files) to modern XML files.

Operating System Components

These files are usually included with the Windows operating system, but you may want to distribute the newest versions of certain files, or you may have developed custom system files of your own. These files are generally DLLs, OCX components, or hardware drivers like SYS files and VxDs.

Shared Application Resources

These are files that may be shared by more than one application, such as ActiveX controls, OCX components, and DLL files.

Some of the files that you need to distribute will be obvious, such as the main executable and help files. Others may be less apparent, such as DLLs and ActiveX controls installed in the Windows directories of your development system.

Note: Many of today’s development tools require that you distribute runtime support files along with your application. Please consult your development tool’s documentation to determine what files you need to distribute with your software.

Often an executable in your application absolutely requires other files in order to work properly. These “absolutely required” files are known as *dependency files*.

Dependency Files

Dependency files are external support files that an executable requires for proper operation. In other words, they are external files that a program file “depends on” in order to function properly. Dependency files may include INI files, DLLs, ActiveX controls, OCX components, or any other support file type. Although it’s generally preferable to install dependency files in the same folder as the program that needs them, they are often installed in other locations, such as the Windows system directories.

Tip: Setup Factory has a built-in dependency scanner that you can use to identify dependency files and add them to your project. You can access this feature by choosing Tools > Scan File Dependencies.

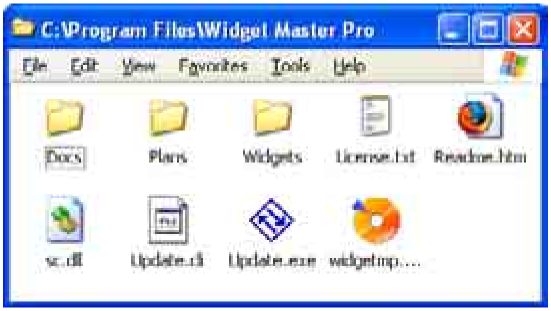
Preparing the Directory Structure

The ultimate goal of a good installer is to get your software onto the user’s system in an easy and accurate manner. Although Setup Factory ensures both you and your users’ ease of use, the accuracy of the installation itself is largely up to you. Whether your files are installed in a structure in which they can function properly depends largely on where you tell Setup Factory to install those files.

The best way to ensure an accurate installation is to prepare the software in its finished state on your development system before you begin creating the installer. This means setting up the entire directory structure and all of the file locations exactly as you want them to appear on the user’s system.

The end result should be that your software is fully installed on your development system exactly as you want it installed on the user’s system.

Not only does this make it easy to test your software in its intended directory structure, but it also makes the process of designing the installer much quicker and easier for you. All you’ll have to do is drag your application folder onto the Setup Factory project window, and the files and sub-folders will be recreated exactly the same on the user’s system.



A well-prepared directory structure

Tip: You should fully test your software in its “final” structure before creating the installer.

Where Do Your Files Need To Be Installed?

Once you have your software organized, you need to determine exactly where each file needs to go on the user’s system. Although Setup Factory does a lot of this work for you by maintaining directory structures when you add files to your project, there are still some files that may need to be directed to different locations on the user’s system.

Here are some guidelines to help you determine where you should install your files on the user’s system. Once again, there are four basic types of files that you may need to distribute: program files, configuration files, operating system components, and shared application resources.

Installing Program Files

Program files should be installed in a folder that the user chooses during the installation process. Throughout this manual and in the Setup Factory program, this folder is referred to as the *application folder*. The path to the application folder is represented by the built-in session variable %AppFolder%.

Tip: For more information on session variables, see Chapter 5.

It’s okay to install program files in sub-folders within the application folder—in fact, organizing your program files into sub-folders is a very good idea. Your files don’t all have to be in the application folder itself; the folder that the user chooses can be used as a common application folder, with sub-folders for all of your program files.

If the users aren’t given an opportunity to choose an installation folder, the path that you provided as the default value for %AppFolder% will be used. You can provide a default path for the application folder on the Session Variables tab of the Project Settings dialog. (To access this tab, choose Project > Session Variables from the menu.)

Installing Configuration Files

In the past, initialization or “INI” files were often installed in the Windows folder (%WindowsFolder%). This is definitely not necessary or even beneficial in all cases. Unless your application shares a configuration file with other programs, it’s best to install the file in the application folder along with your program files.

You should avoid installing any files in the Windows folder, or in any other folders where critical operating system files are stored, unless it is absolutely required by your application.

Installing Operating System Components

Operating system components, such as DLL or OCX files, are traditionally installed in the WINDOWS\SYSTEM folder (%SystemFolder%). If your application doesn't need to share these files with other applications, it's best to install them in your application folder (%AppFolder%) instead.

Installing Shared Application Resources

Shared application resources, such as some ActiveX controls or DLL files, are generally installed in the WINDOWS\SYSTEM folder (%SystemFolder%).

Note: Many DLL and OCX files are COM servers, also known as "OLE components" and "ActiveX controls." As such, they will need to be registered with the operating system before they will be available for use by your application.

Setup Factory will automatically try to detect files that require registration when you add them to your project. You can also manually indicate files that you want Setup Factory to register by using the options on the Advanced tab of the File Properties dialog. Or you can use an action like System.RegisterActiveX to register a file manually at run time.

Keep in mind that some DLL and OCX files have dependency files themselves, and these dependency files need to be distributed and possibly also registered before the DLL or OCX files can be used. You should consult the documentation for your components to determine what dependencies they have.

Note: Many OCX and DLL controls ship with a dependency file. The dependency file typically has the same filename as the control with a .DEP extension, e.g., Control.ocx would have a dependency file named Control.dep. These dependency files can be opened and viewed with a text editor (such as Notepad) and can tell you a lot about what, if any, dependencies the control may have.

Tip: You can also use Setup Factory's built-in dependency scanner to identify any dependency files your components may have. You can access the dependency scanner by choosing Tools > Scan File Dependencies.

What System Changes Need To Be Made?

The next step is to think about what changes, if any, need to be made to the user's system. This can include changes to system files (such as WIN.INI, CONFIG.SYS, SYSTEM.INI and AUTOEXEC.BAT), changes to the Registry, and even such things as installing and registering fonts. All of these changes can be handled easily using Setup Factory actions.

For more information on actions, see Chapter 4.

What Information Do You Need from the User?

The final step in planning your software installation is to consider what information you will need from the user, and what information you will need from the user's system. For example, in order to personalize your software or register it to a specific user, you may want to ask the user to provide their name on a User Information screen. Or, you may want to ask the user for a serial number on a Verify Serial Number screen.

For more information on screens, see Chapter 3.

Tip: You can also use actions to prompt the user for information (e.g. Dialog.Input) and to gather other information directly from the user's system (e.g. System.GetUserInfo). See the help file for a complete list of the actions available in Setup Factory 7.0.

Chapter 2:

Working with Files

Installing the right files to the right places is the key to performing a successful installation. The files that make up your software need to get from your development system to the appropriate locations on the user's computer. Regardless of whether you're distributing files on CD-ROMs, floppies, or in a single, downloadable setup executable, Setup Factory allows you to install all of your files with the utmost precision.

In fact, the installation of files is the main purpose of an installer. As such, you are expected to be completely familiar with files and folder structures in order to use Setup Factory. If you need more information on the basics of files and hierarchical folder structures, please consult the "Windows Basics" section of the Setup Factory help file.

Note: Files *are* your software—and you are responsible for getting those files onto the users' systems without causing any harm. A solid understanding of files and folders is critical to building a professional installer.

2

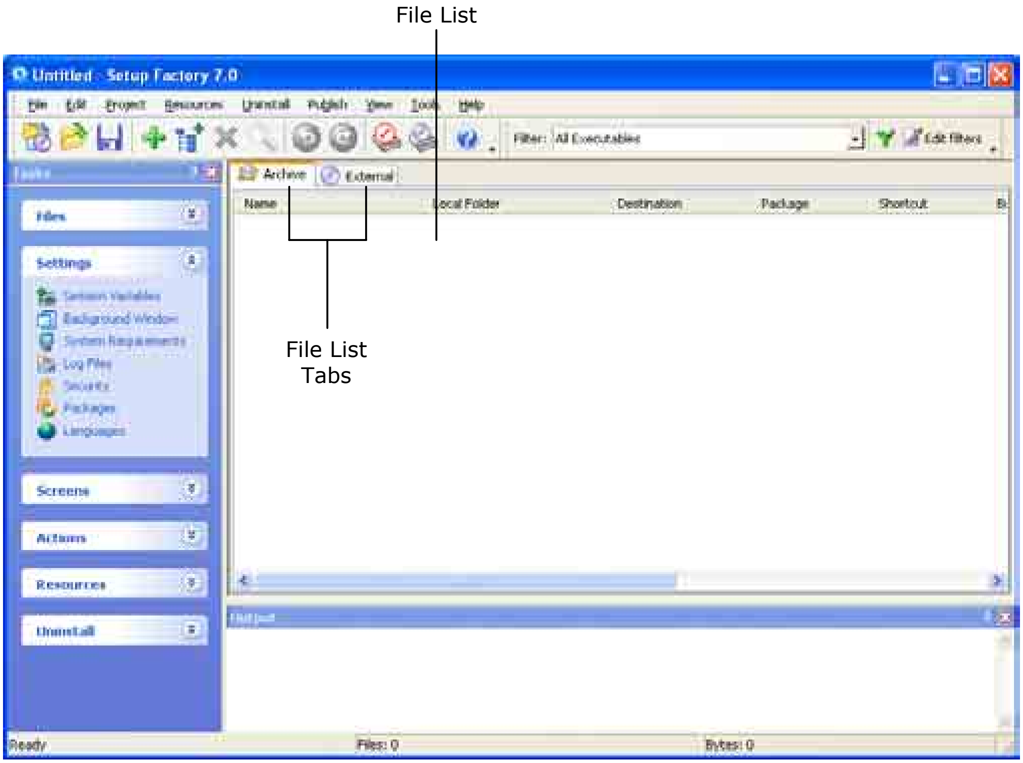
In This Chapter

In this chapter, you'll learn about:

- The project window
- The Archive and External file lists
- Filtering the file list
- Folder references
- Adding files and folder references
- Removing files and folder references
- File and folder reference properties
- What %AppFolder% is
- Working with multiple files
- Missing files
- Primer files

The Project Window

The project window takes up most of the main screen and contains a list of all the files that you have added to your project. From this list you can highlight specific files and view or edit their properties.



The file list tabs on the project window allow you to switch between the two separate file lists in Setup Factory: the Archive file list, and the External file list.

The Archive File List

The Archive file list (or “Archive tab”) is for files that you want compressed and stored in the setup executable. In other words, this list is for files that will be included *in* your installer.

When Setup Factory builds the setup executable, it compresses all the files in the Archive file list and packs them right into the setup executable file. The part of the setup executable where the files are stored is known as the *setup archive*. The Archive file list is where you add the files that you want to store in the setup archive.

The External File List

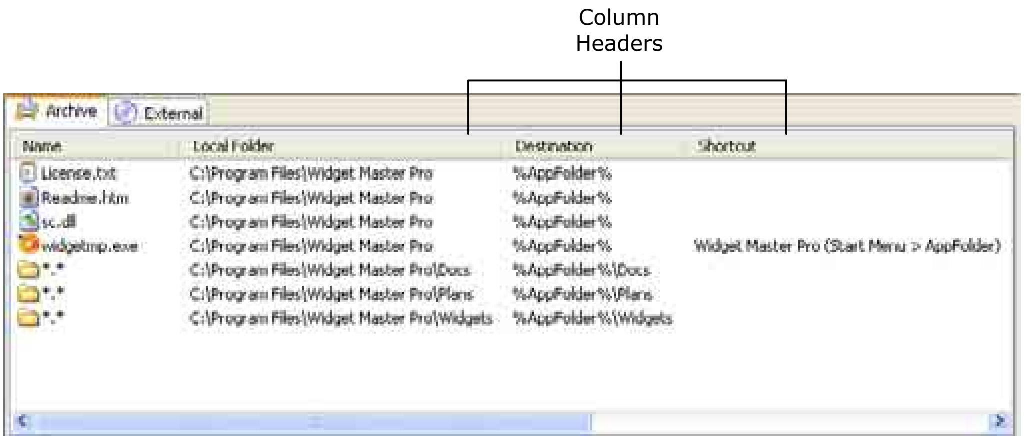
The External file list (or “External tab”) is for files that you don't want compressed and stored in the setup executable. In other words, this list is for files that will be distributed *with* your installer.

When Setup Factory builds the setup executable, it includes all the information it knows about the files on the External tab, such as where the files are expected to be at run time (the run-time source location), and where you want them to be installed to (the run-time destination folder). But it doesn't actually include the files in the setup executable. Instead, you're expected to make sure that those files will be where the installer expects them to be at run time. Usually this is done by distributing the files along with the setup executable on the same CD-ROM.

Note: In previous versions of Setup Factory, the External file list was known as the “CD-ROM tab.”

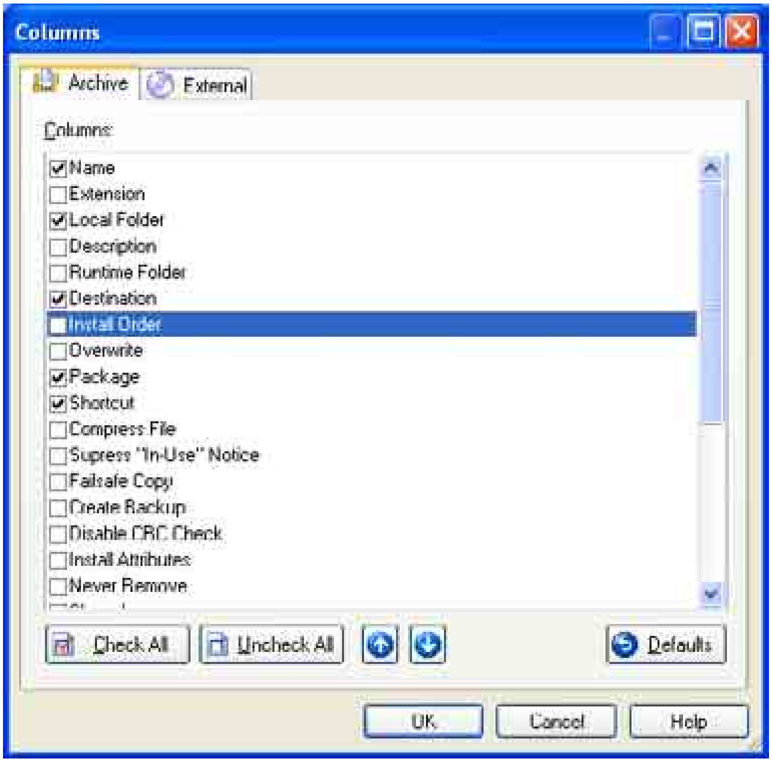
File List Columns

Both file lists have columns that display different information about each file in your project. You can sort the information along any of the columns by clicking on the header for that column. If you click on the same header again, the files will be sorted by that category in reverse order.



Name	Local Folder	Destination	Shortcut
License.txt	C:\Program Files\Widget Master Pro	%AppFolder%	
Readme.htm	C:\Program Files\Widget Master Pro	%AppFolder%	
sc.dll	C:\Program Files\Widget Master Pro	%AppFolder%	
widgetmp.exe	C:\Program Files\Widget Master Pro	%AppFolder%	Widget Master Pro (Start Menu > AppFolder)
.	C:\Program Files\Widget Master Pro\Docs	%AppFolder%\Docs	
.	C:\Program Files\Widget Master Pro\Plans	%AppFolder%\Plans	
.	C:\Program Files\Widget Master Pro\Widgets	%AppFolder%\Widgets	

You can customize the columns by choosing View > Columns. This opens the Columns dialog, where you can choose which columns to display in the list, and change the order (left to right) that the columns are listed in.



Columns:

☑ Name
☐ Extension
☑ Local Folder
☐ Description
☐ Runtime Folder
☑ Destination
☐ Install Order
☐ Overwrite
☑ Package
☑ Shortcut
☐ Compress File
☐ Supress "In-Use" Notice
☐ Failsafe Copy
☐ Create Backup
☐ Disable CRC Check
☐ Install Attributes
☐ Never Remove

Tip: Moving a column up in the columns list moves it to the left in the file list; moving a column down in the columns list moves it to the right.

File List Items

The items that appear in the Setup Factory file lists refer to files on your system. When you add a file to your project, the file is *not* copied into the project. It is only *referenced* by the project.

In other words, only the path to the file (and a few other properties, such as the size of the file) is stored in your project. If the original file is renamed, moved or deleted, Setup Factory won't be able to build it into the setup executable. (In fact, it will show up in the file list as a missing file.)

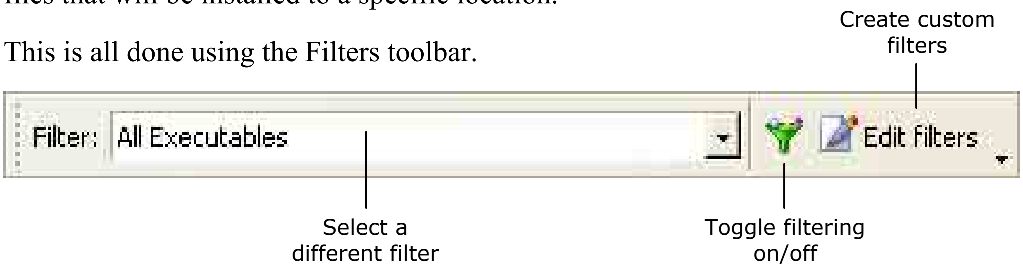
The file itself, however, is not part of the project. If you copy your project from one system to another, you will not be able to build it unless you also copy the files, and place them in the exact same folder structure—in other words, you must place the files where the project expects them to be located.

Note: The files that your project references are commonly referred to as *source files*.

Filtering the File List

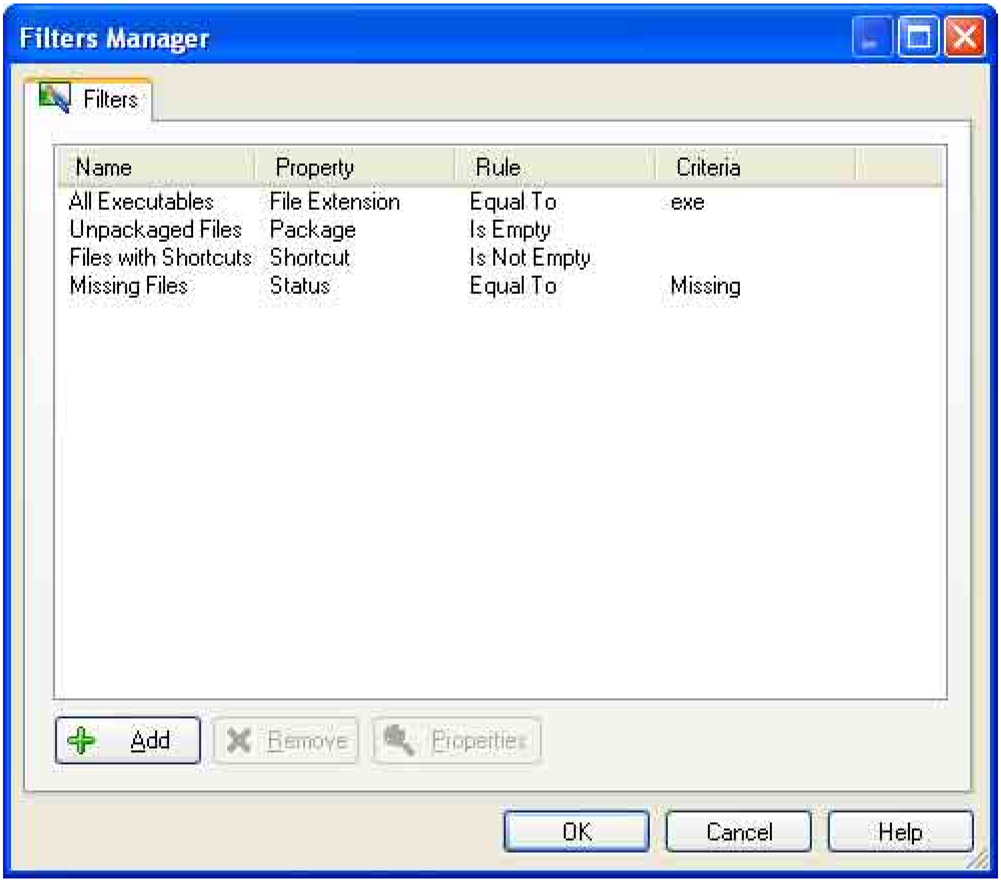
You can apply a filter to the file list so it only shows files that meet specific criteria. For example, you could filter the list to only show files that are larger than 5 MB, or files that will be installed to a specific location.

This is all done using the Filters toolbar.



Filter: All Executables

Clicking the Edit filters button on the toolbar opens the filters manager, where you can configure the list of filters that appear in the drop-down selector on the toolbar.

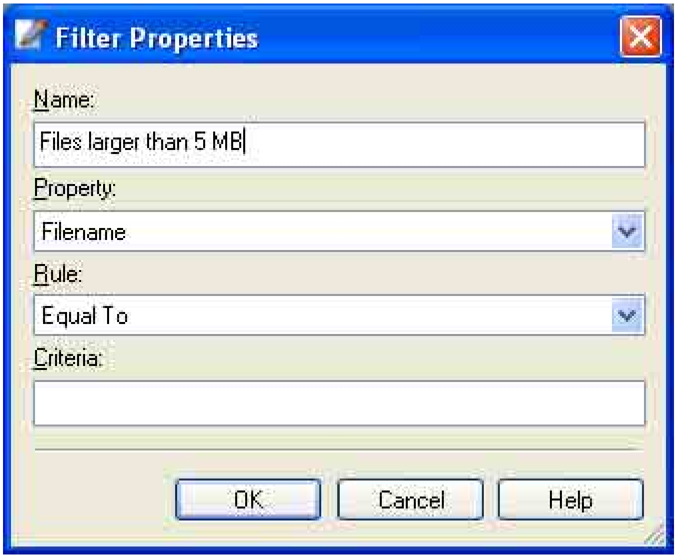


Name	Property	Rule	Criteria
All Executables	File Extension	Equal To	exe
Unpackaged Files	Package	Is Empty	
Files with Shortcuts	Shortcut	Is Not Empty	
Missing Files	Status	Equal To	Missing

The filters manager allows you to define custom filters using a wide range of criteria.

Creating a Filter

Setting up a new filter is incredibly easy. After clicking on Add in the filters manager, you simply give the filter a name:



Filter Properties

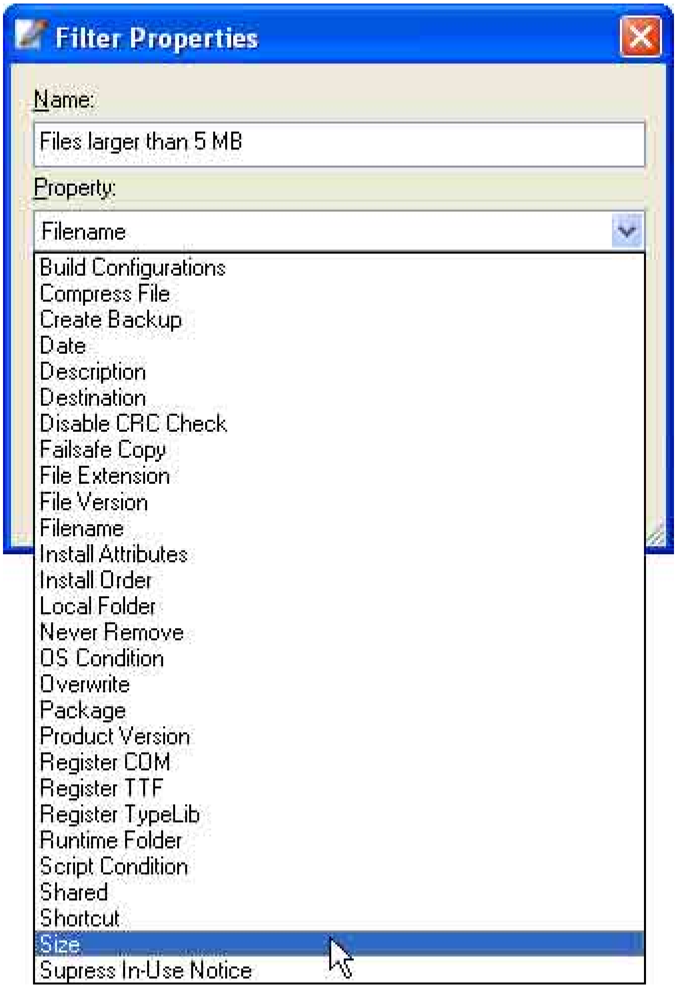
Name: Files larger than 5 MB

Property: Filename

Rule: Equal To

Criteria:

…then select the property you want to filter on:



Filter Properties

Name:

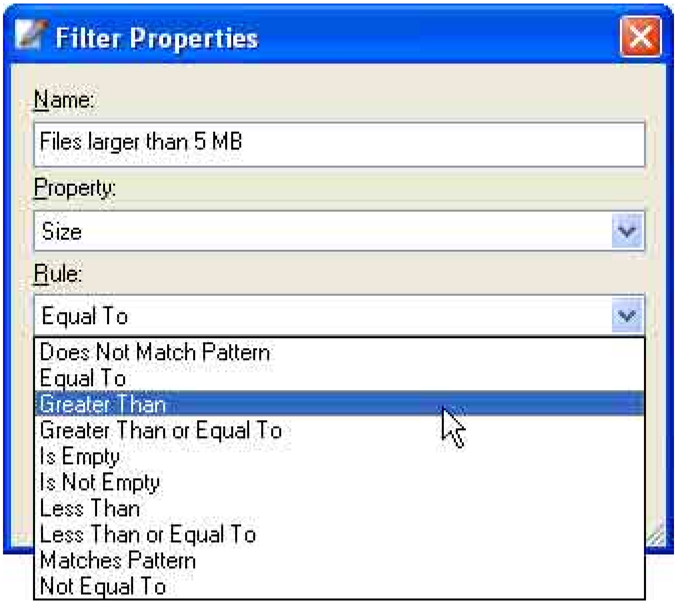
Files larger than 5 MB

Property:

Filename

- Build Configurations
- Compress File
- Create Backup
- Date
- Description
- Destination
- Disable CRC Check
- Failsafe Copy
- File Extension
- File Version
- Filename
- Install Attributes
- Install Order
- Local Folder
- Never Remove
- OS Condition
- Overwrite
- Package
- Product Version
- Register COM
- Register TTF
- Register TypeLib
- Runtime Folder
- Script Condition
- Shared
- Shortcut
- Size
- Supress In-Use Notice

…select the rule that you want to use:



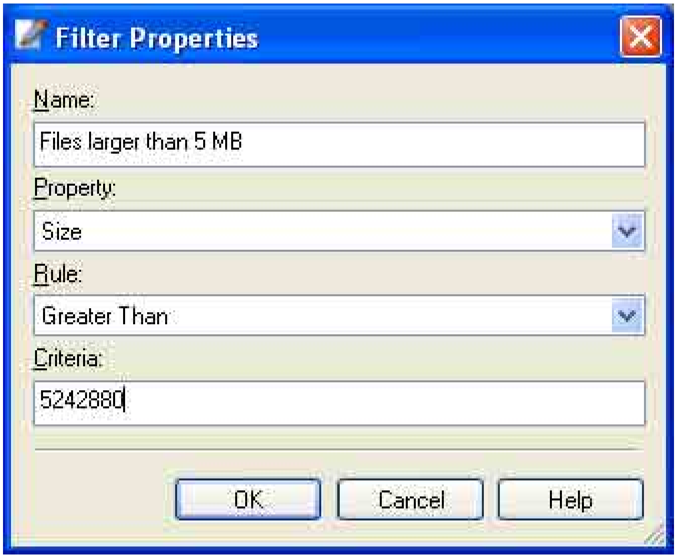
Filter Properties

Name: Files larger than 5 MB

Property: Size

Rule: Equal To

…and finally provide the criteria:



Filter Properties

Name: Files larger than 5 MB

Property: Size

Rule: Greater Than

Criteria: 5242880

Once filters have been defined, you can easily switch between them using the drop-down selector. The filters are saved on a global basis, so they can be used in all of your projects.

Folder References

Folder references are similar to files, but they reference a folder on your system instead of a single file. They're useful for including folders full of files without having to reference each file individually.

For example, if you have a folder full of images that you need to install, you can use a folder reference to add the entire folder to your project. This also lets you configure the settings for all of those files from a single item in the file list.

You can even choose whether or not to reference folders recursively. In other words, you can choose whether or not to include the files in any subfolders (and subfolders of subfolders, and so on) that may be in the selected folder. (This is in fact the default behavior of folder references; however, you can choose to only include the files in the immediate folder if you want by simply turning off the "Recurse subfolders" option.)

Each folder reference also has a File Mask setting that you can use to include only files that match a specific pattern. For instance, if you wanted to use different settings for the .jpg files and the .bmp files in your product's Images folder, you could use two folder references—one for each type. So, for example, you could set up one folder reference to add everything beneath C:\Program Files\GiddyApp\Images that matches "*.jpg" and another one to add everything in that same path that matches "*.bmp." You could then configure their settings differently—for example, you could turn off compression for the .jpg files so Setup Factory won't try to compress them when you build the project, which could shorten the build process if you have a lot of image files.

Note: During the build process Setup Factory performs a test compression on each file, and automatically turns compression off for any file that isn't reduced in size after it's compressed. JPEG files are already compressed, so normally Setup Factory ends up turning compression off for them anyway.

Tip: Folder references are one of the most powerful new features in Setup Factory 7.0! Be sure to use them to your advantage.

Overriding Individual Files

There may be cases where you want to include a folder full of files, but you want to use different settings for some of the files in the folder. Setup Factory makes this incredibly easy to accomplish because you can override a folder reference with individual file items.

The rule is simple: the settings for individual file items always take precedence over folder references. So if you have a folder reference that happens to include files A, B, and C, and you also add file B to your project on its own, the settings for the standalone file item will be used for file B, and not the settings for the folder reference.

Note: It doesn't matter what order the files are in…individual file items always override folder references.

Before Adding Files

Before you start adding files to your project, make sure you have all of your files properly installed on your system. Your local directory structure affects the destination paths that Setup Factory assigns to files as they are added to your project, so it's important to have your application's directory structure laid out on your system the same way you want it to end up on the user's.

Adding Files

There are two ways to add files to your Setup Factory project: you can add files from within Setup Factory, or you can drag and drop files onto the project window.

Adding Files from Within Setup Factory

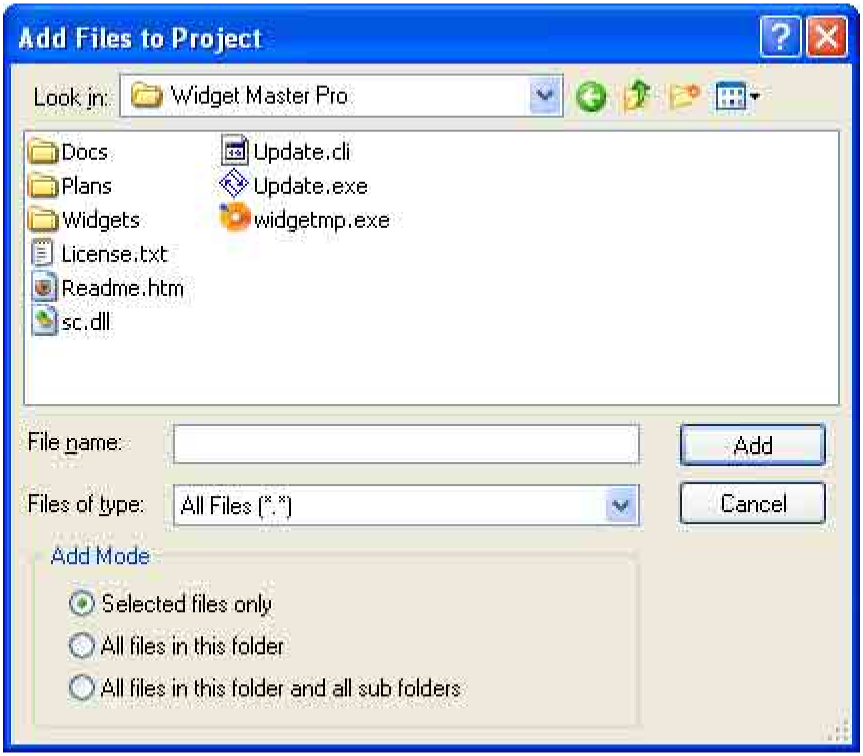
To add files from within Setup Factory:

1. Click on the Archive tab or the External tab.

Use the Archive file list if you want the files to be included in the setup executable, or the External file list if the files will be distributed separately.

2. Choose Project > Add Files from the menu.

This will open the Add Files to Project dialog.



Add Files to Project

Look in: Widget Master Pro

Docs
Plans
Widgets
License.txt
Readme.htm
sc.dll
Update.cli
Update.exe
widgetmp.exe

File name:

Files of type: All Files (*.*)

Add Mode

● Selected files only
○ All files in this folder
○ All files in this folder and all sub folders

Tip: You can also click the Add Files button on the toolbar, press the Insert key, or right-click on the project window and select Add Files from the right-click menu.

3. Select the add mode that you want to use.

The “Selected files only” option will only add the files that you select.

The “All files in this folder” option will add all of the files in the folder that the Add Files to Project dialog is currently displaying.

The “All files in this folder and all sub-folders” option will add all of the files in the current folder, and all of the files in any sub-folders as well.

4. Select the files that you want to add, or navigate to the folder where all the files (and possibly sub-folders) that you want to add are located.

Depending on the add mode you chose, you either need to select one or more individual files in the browse dialog, or navigate into the folder that you want to add files from.

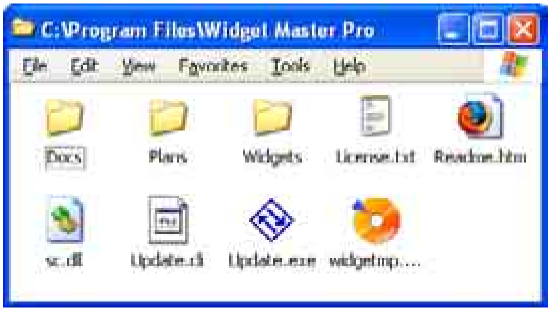
5. Click the Add button to add the files to your project.

Once you click the Add button, the files will appear in the file list.

Dragging Files onto the Project Window

To add files by dragging and dropping them:

1. Open the folder where your files are located.



The "C:\Program Files\Widget Master Pro" folder in Windows

2. Select the files that you want to add to your project.

You can hold the Ctrl key down while clicking on files to select more than one file at a time.

3. With your files still selected, hold the left mouse button down and drag the files over the Setup Factory project window.

This is just like dragging icons around on the desktop.

4. Let go of the mouse button to “drop” the files onto the project window.

Once you drop the files onto the project window, they will appear in the file list.

Adding Folder References

There are two ways to add folder references to your Setup Factory project: you can add them from within Setup Factory, or you can drag and drop folders onto the project window.

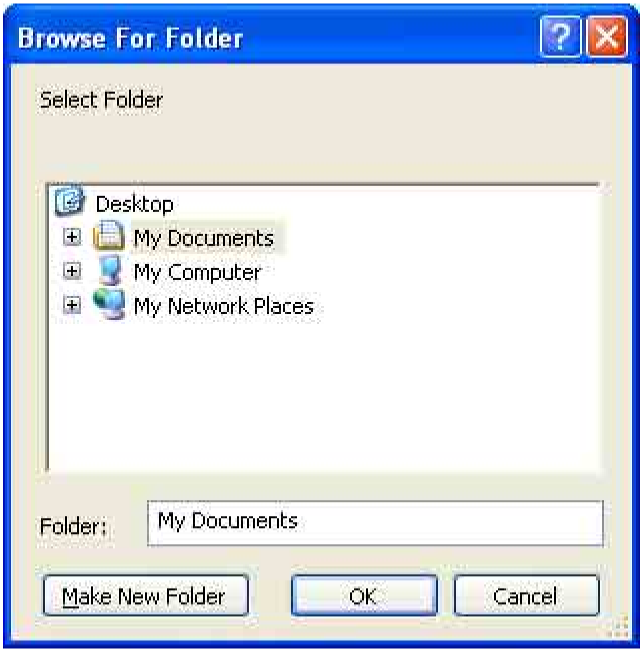
Adding Folder References from Within Setup Factory

1. Click on the Archive tab or the External tab.

Use the Archive file list if you want the folders to be included in the setup executable, or the External file list if the folders will be distributed separately.

2. Choose Project > Add Folder Reference from the menu.

This will open the Browse For Folder dialog.



Folder: My Documents

Tip: You can also click the Add Folder Reference button on the toolbar, press Ctrl+Insert, or right-click on the project window and select Add Folder Reference from the right-click menu.

3. Select the folder that you want to add.

Simply browse for the folder that you want to add. When you select a folder, its name will appear in the Folder field near the bottom of the dialog.

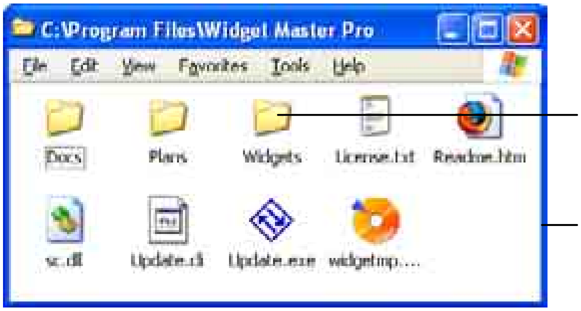
4. Click OK to add the folder to your project.

When you click the OK button, the folder reference will appear in the file list.

Dragging Folders onto the Project Window

To add a folder reference by dragging and dropping a folder:

1. Open the containing folder where your folder is located.



A folder...

...its containing folder (open)

2. Select the folder that you want to add to your project.

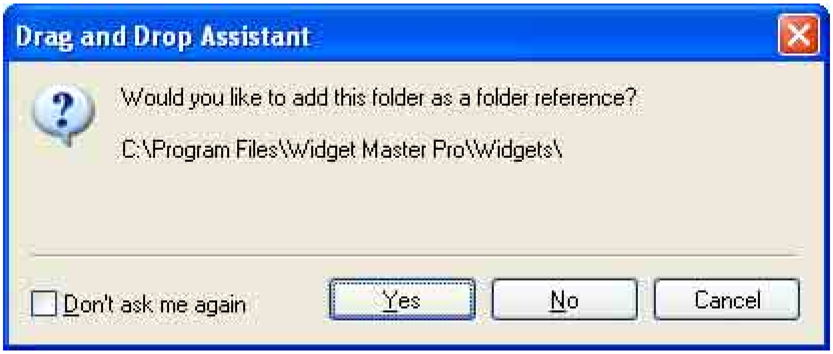
In other words, click once on the folder that you want to add. (But don't navigate into the folder, or you won't be able to drag it.)

3. With your folder still selected, hold the left mouse button down and drag the folder over the Setup Factory project window.

This is just like dragging icons around on the desktop.

4. Let go of the mouse button to "drop" the folder onto the project window.

Once you drop the files onto the project window, the drag and drop assistant will appear, asking if you want to add the folder as a folder reference.



Drag and Drop Assistant

Would you like to add this folder as a folder reference?

C:\Program Files\Widget Master Pro\Widgets\

5. Click Yes to add the folder as a folder reference.

Clicking Yes on the drag and drop assistant will add the folder to your file list as a folder reference.

Note: Clicking No will add all of the files inside the folder as individual file items instead.

Removing Files

To remove files from your Setup Factory project:

1. Select the files that you want to remove on the project window.

Simply click on the file items that you want to remove.

2. Choose Project > Remove Selected.

Choosing Project > Remove Selected from the program menu will remove the selected file items.

Tip: You can also click the Remove Files button, press the Delete key, or right-click on the project window and select Remove from the context menu.

Note: Removing files from your project removes the file items from the file list, but does not delete the original files. The original files remain completely unaffected.

Removing Folder References

Removing folder references is exactly like removing files. Simply select the folder references that you want to remove and choose Project > Remove Selected (or press the Delete key, etc.).

File Properties

You can use the File Properties dialog to view and edit the settings for any file in your project.

To access the File Properties dialog:

1. Select a file on the project window.

Simply select the file item that you want to edit.

2. Choose Project > File Properties.

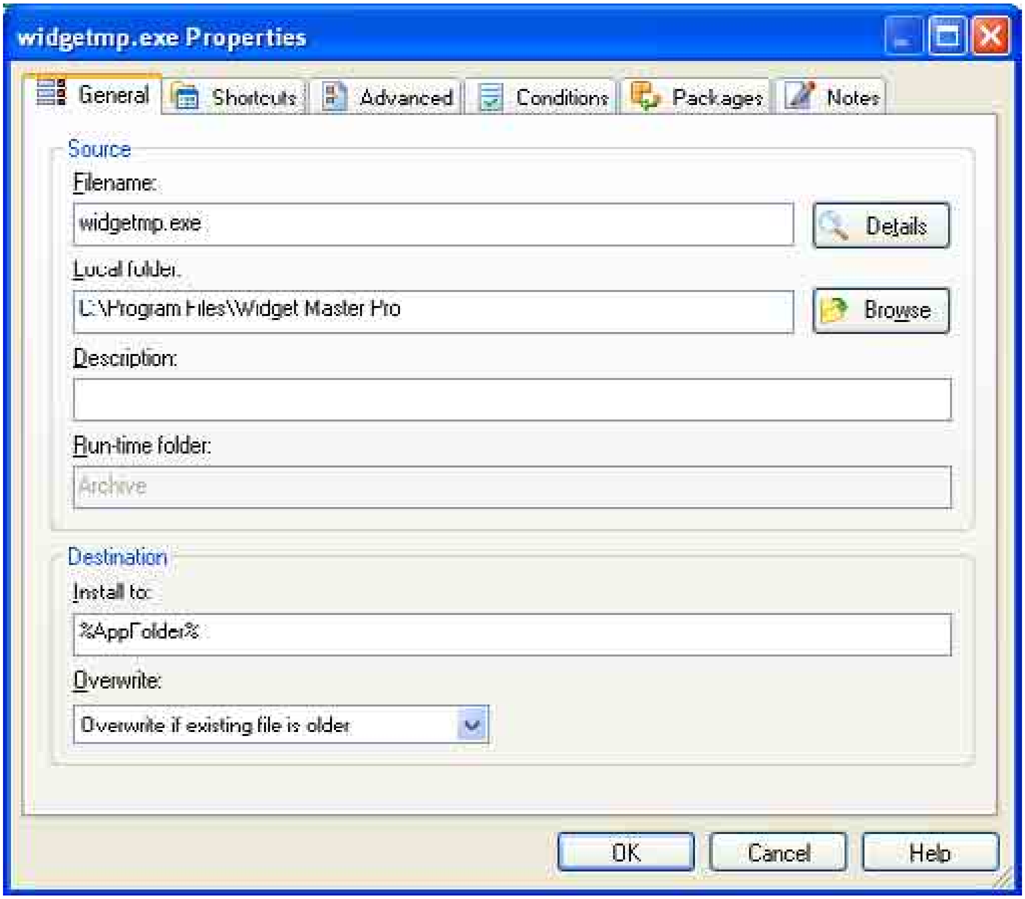
This will open the File Properties dialog, where you can view and edit the properties of the file you selected.

Tip: There are several other ways to access the file properties as well. You can click the File Properties button, press Ctrl+Enter, right-click on the file item and select File Properties from the right-click menu, or simply double-click on a file in the list.

There are six tabs on the File Properties dialog: General, Shortcuts, Advanced, Conditions, Packages, and Notes.

General

The General tab is where you will find information about the file itself, such as its name and location on your system. This is also where you can view or edit the destination path, and the overwrite settings for the file.



Filename: widgetmp.exe

Local folder: C:\Program Files\Widget Master Pro

Description:

Run-time folder: Archive

Install to: %AppFolder%

Overwrite: Overwrite if existing file is older

Source Folders

Since a Setup Factory project only contains references to files, each file has a property that specifies *where* the file is located. This location is known as the source folder.

There are two kinds of source folders: local folders, and run-time folders.

Local Folders

Local folders are folders that are located within local access of your system or LAN. These specify the *current location* of the file on your system.

Setup Factory uses the local folder to retrieve the file's information (e.g. when you open a project) and to locate the file when it builds the setup executable (i.e. when you build the project).

Run-time Folders

Files in the External file list have an additional property known as the "run-time folder." The run-time folder specifies the *run-time location* where the file will be when the user runs your setup file. In other words, it represents the location of an external file at the time of the installation.

Note: The run-time source path only applies to files on the External tab. The files on the Archive tab are always located in the setup executable archive at run time.

Destination Paths

Destination paths specify where the files are intended to end up on the user's system. Though this seems simple enough, it is complicated by the fact that you can't predict the layout of the folder structure on the user's system. For instance, if you want to install a file in the user's "My Documents" folder, how do you know what the full path to that folder is? It could be anything from "C:\Documents and Settings\JoeUser\My Documents" to "F:\Joe's Docs."

Setup Factory gets around this uncertainty by providing you with a number of built-in session variables for common system folders. These are essentially placeholders (like %MyDocumentsFolder%) that you can use in your file paths, and that will be replaced by the corresponding path on the user's system at run time. You simply use the appropriate placeholder, and Setup Factory will take care of investigating what the actual path is when your user runs the setup file.

For more information on session variables, see Chapter 5.

%AppFolder%

Most installations involve asking the user to confirm where they would like the software installed. In order to accommodate the user's choice in this matter, Setup Factory uses a session variable to represent the main folder that your software will be installed in. The name of this session variable is %AppFolder%, which is short for *Application Folder*, i.e. the folder where your software application is to be installed.

Tip: You can set a default path for %AppFolder% by choosing Project > Session Variables, and you can allow the user to change it (i.e. select a different path) by including a Select Install Folder screen in your project.

Automatic Destination Folder Prediction

When files are added to a Setup Factory project, Setup Factory tries to set a "best guess" value for the destination folder of each file. It does this in two ways.

The first time you add any files to the Archive file list, Setup Factory assumes that you are adding files from the “main” folder where your application is installed on your system. It sets these files to be installed in %AppFolder%, or in sub-folders of %AppFolder% if the files were in subfolders of the main folder—in other words, it preserves the relative folder structure of the files that you add.

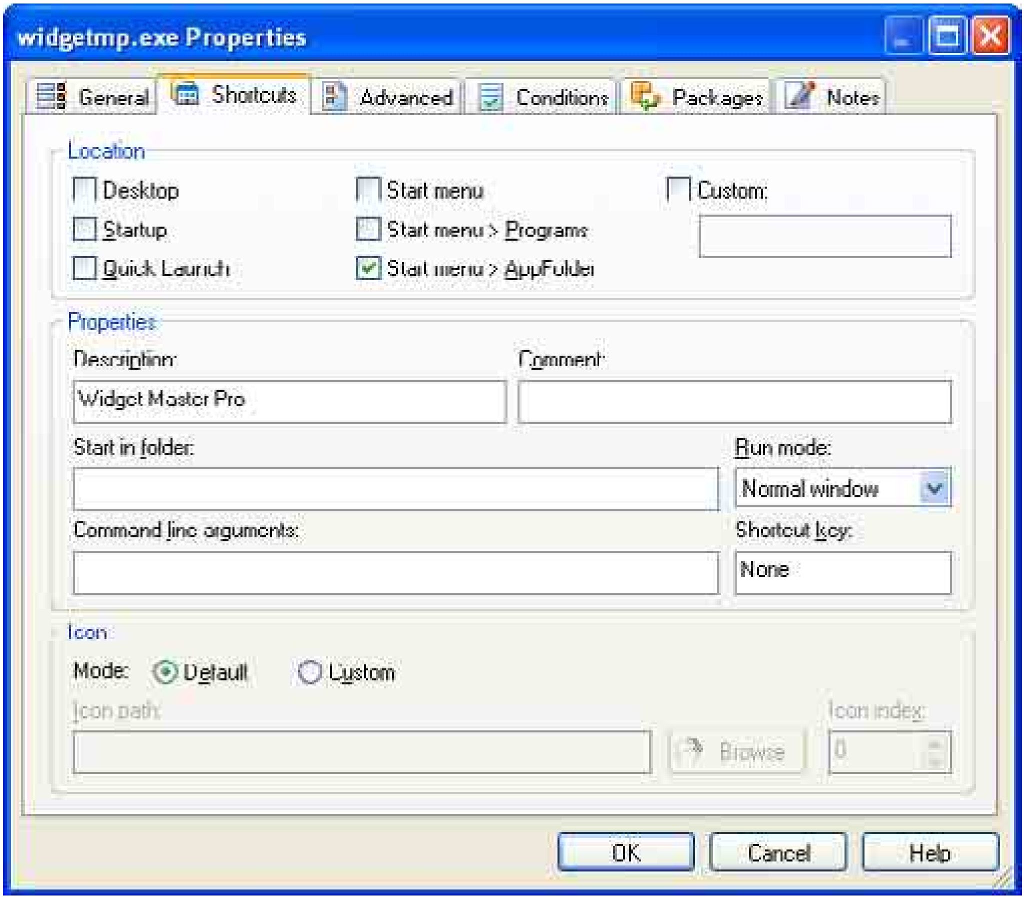
Once you already have files in the file list, Setup Factory goes even further when you add new files to the project. First, it compares the path of each file that you’re adding to the local source path of each file in your project. If it finds another file in your project that was from the same source folder on your system, it uses that file’s destination path to guess what the new file’s destination path should be. (Normally, files that are located in the same place on your system will be installed to the same place on the user’s system.)

In other words, Setup Factory tries to base each new file’s destination path on the destination path of a file in your project that is in the same location on your system.

If there are no files in your system from the same source location, the new file is configured to simply use %AppFolder% as the destination folder path.

Shortcuts

The Shortcuts tab is where you can configure the shortcut options for the selected file. You can use this tab to have Setup Factory automatically create a shortcut icon for the file in the user's Start menu or on the user's desktop at run time.



widgetmp.exe Properties

Location

□ Desktop
□ Startup
□ Quick Launch
□ Start menu
□ Start menu > Programs
☑ Start menu > AppFolder
□ Custom:

Properties

Description: Widget Master Pro

Comment:

Start in folder:

Run mode: Normal window

Command line arguments:

Shortcut key: None

Icon

Mode: ● Default ○ Custom

Icon path:

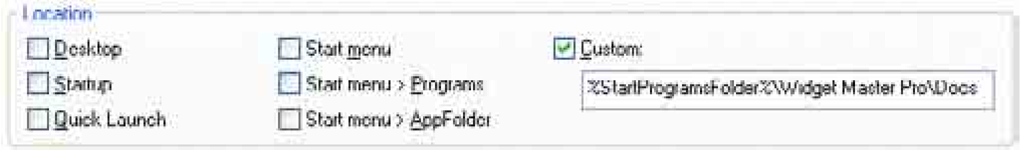
Icon index: 0

Creating Nested Shortcuts

A nested shortcut is a shortcut file that is located in a sub-folder of a shortcut folder. In other words, it’s a shortcut in a folder that is located or “nested” in another folder.

You can easily create nested shortcuts with Setup Factory 7.0 by specifying a custom shortcut path. Simply include the nested folders in the path that you provide in the Custom field on the Shortcuts tab.

In other words, just enter the full path to where you want the shortcut file to end up, and if any of the folders in the path don't exist, Setup Factory will automatically create them for you.



Location

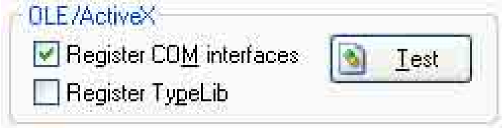
□ Desktop
□ Startup
□ Quick Launch
□ Start menu
□ Start menu > Programs
□ Start menu > AppFolder
☑ Custom:
%StartProgramsFolder%\Widget Master Pro\Docs

Advanced

The Advanced tab is where you can configure advanced options for the selected file. This is where you can override certain default installation settings, such as whether the file will be compressed before storing it in the setup executable, whether it will be uninstalled automatically by your installer's uninstall routine, and what attributes will be assigned to the file after it's installed. You can also specify whether the file should be registered as a TrueType font, or even whether it should be registered as a shared OLE or ActiveX component using DLLRegisterServer.

Registering Files

There are two ways that you can register files (such as ActiveX controls) in Setup Factory. The easiest way is to simply select the “Register COM interfaces” check box on the Advanced tab of the File Properties dialog for the file you want to register.



OLE/ActiveX

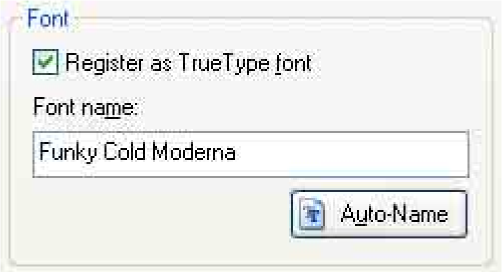
☑ Register COM interfaces
□ Register TypeLib

The alternative method is to use a System.RegisterActiveX action to register a file after it has been installed.

For more information on the Register COM interfaces option and the System.RegisterActiveX action, please consult the Setup Factory help file.

Registering Fonts

There are two ways that you can register TrueType fonts in Setup Factory. The easiest way is to select the "Register as TrueType font" check box on the Advanced tab of the File Properties dialog for the file you want to register. The file must be a valid Windows TrueType font (.ttf) file.


Font

☑ Register as TrueType font

Font name:

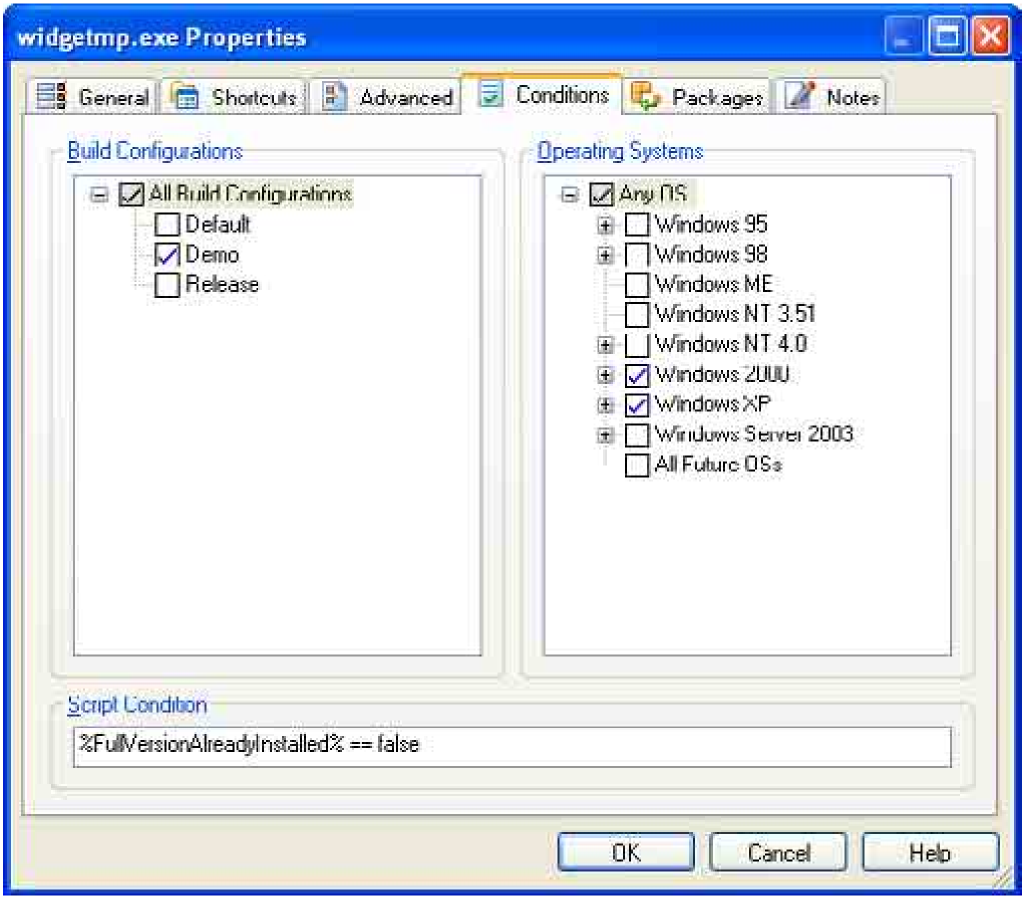
Funky Cold Moderna

The alternative method is to use a System.RegisterFont action to register a .ttf file after it has been installed.

For more information on the Register as TrueType font option and the System.RegisterFont action, please consult the Setup Factory help file.

Conditions

The Conditions tab is where you can edit settings that affect whether the file will be included in the setup executable, and whether it will be installed. In other words, it is used to conditionally include or exclude a file from the setup at build time or run time.



Build Configurations

- ☑ All Build Configurations
  - □ Default
  - ☑ Demo
  - □ Release

Operating Systems

- ☑ Any OS
  - □ Windows 95
  - □ Windows 98
  - □ Windows ME
  - □ Windows NT 3.51
  - □ Windows NT 4.0
  - ☑ Windows 2000
  - ☑ Windows XP
  - □ Windows Server 2003
  - □ All Future OSs

Script Condition

```
%FullVersionAlreadyInstalled% == false
```

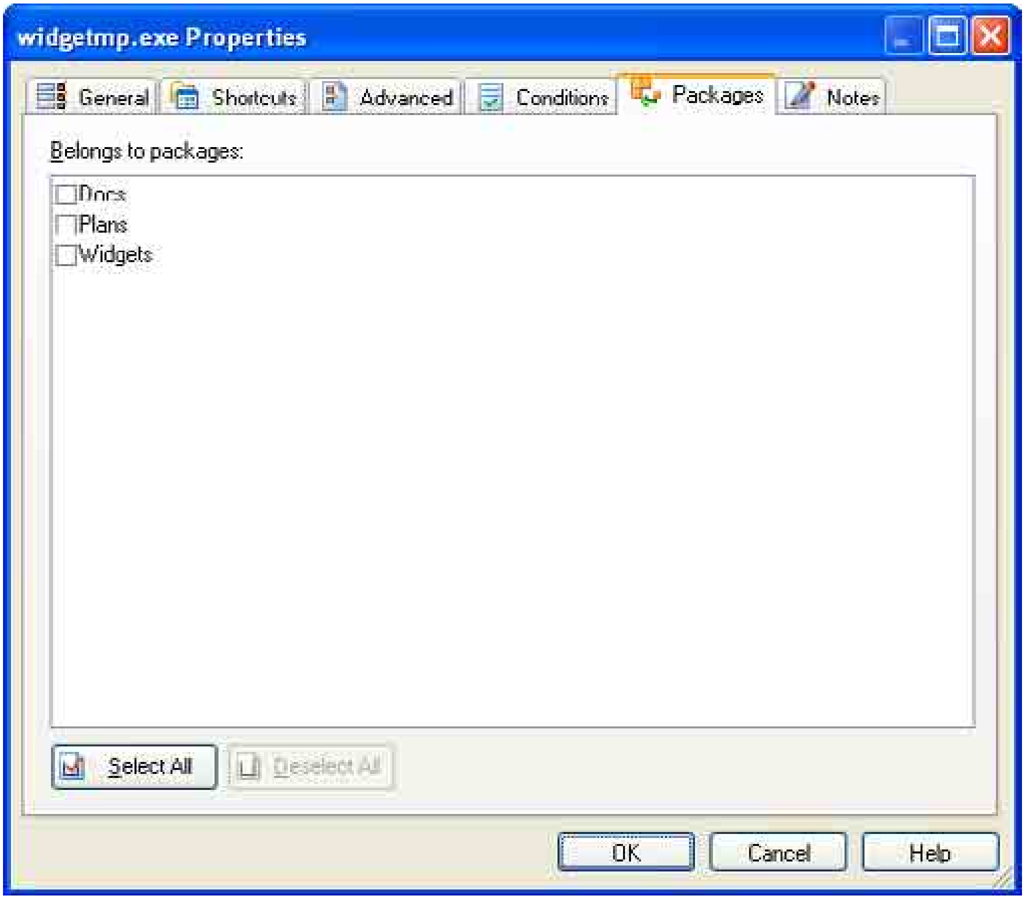
The list of build configurations allows you to select the build configurations that the file will be included in. The file will only be included in the setup executable if it is included in the build configuration that is in effect when you build the project.

The list of operating systems allows you to specify which operating systems the file will be installed on. The file will only be installed on an operating system if it is marked with a check in this list. So, for example, if you uncheck Windows 95 and Windows 98 for a file, it will not be installed on Windows 95 or Windows 98.

The script condition is a short expression written in Lua script for the file. It must evaluate to a Boolean value of either true or false. At run time this result will determine whether or not the file will be installed.

Packages

The Packages tab is where you can select the packages that you want the selected file to belong to. The Packages tab lists all of the packages that have been configured in your project. You can add the currently selected file to one or more packages by clicking on the check boxes in this list.



Belongs to packages:

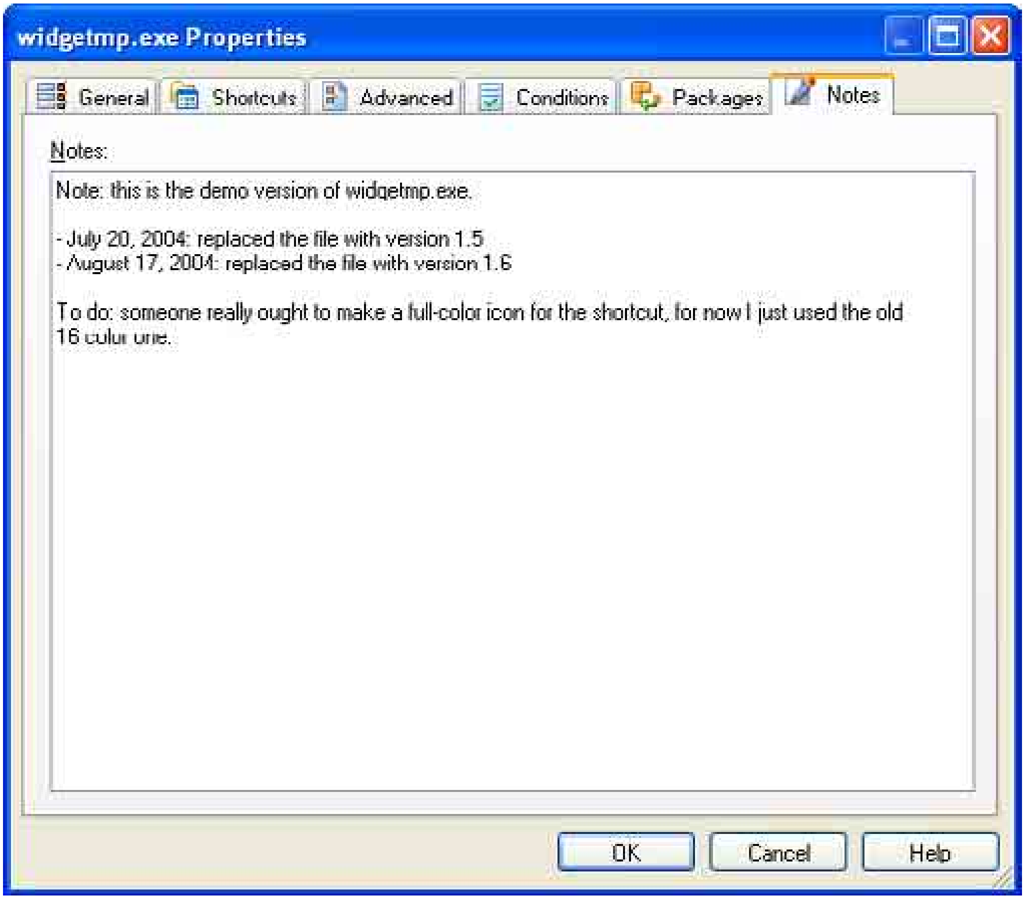
□ Docs
□ Plans
□ Widgets

The file will be assigned to the packages that have check marks beside them.

For more information on packages, see Chapter 6.

Notes

The Notes tab is where you can type any notes that you want to keep about the file. These are not used by the installer, and are not included in the setup executable at all. The Notes tab is simply a convenient place for you to keep notes about the file for your own purposes. For example, you could list any special instructions involved in preparing the file for the build process, or you could keep track of who last edited the file's properties.



Notes:

Note: this is the demo version of widgetmp.exe.

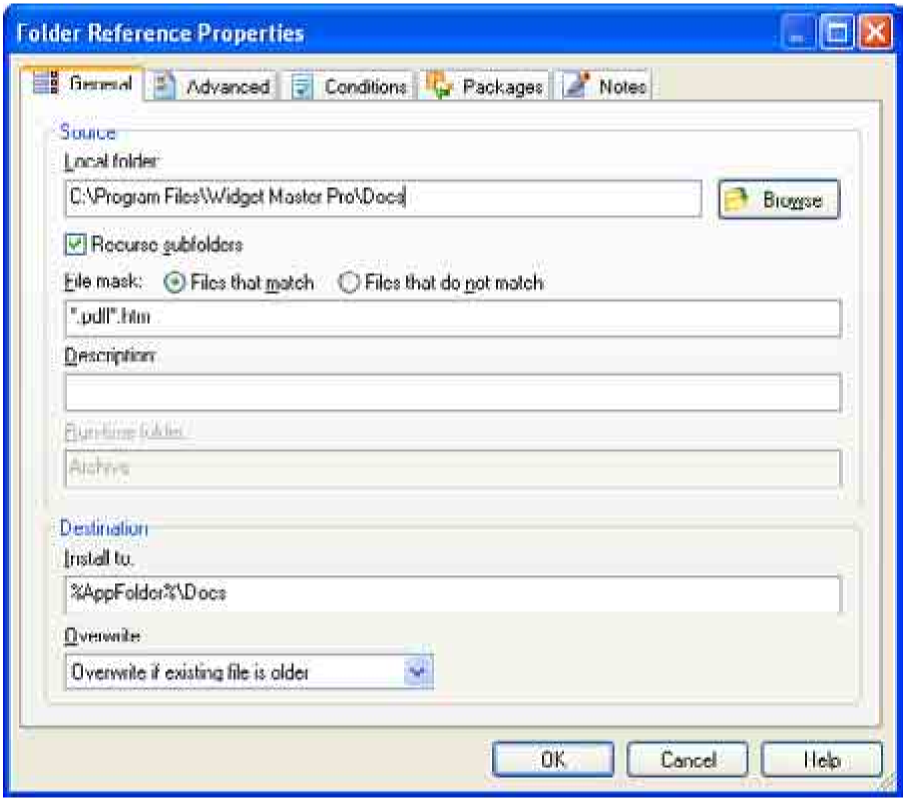
- July 20, 2004: replaced the file with version 1.5
- August 17, 2004: replaced the file with version 1.6

To do: someone really ought to make a full-color icon for the shortcut, for now I just used the old 16 color one.

Folder Reference Properties

For the most part, Folder References have almost exactly the same properties as files do. In fact, the Conditions, Packages and Notes tabs are exactly the same. Here are some differences worth mentioning:

- On the General tab, folder references have a File Mask setting that allows you to include or exclude a range of files that match a filename pattern, e.g. “*.exe”
- On the General tab, folder references have a “Recurse subfolders” option that controls whether files in subfolders (and their subfolders) should be included as well.
- There is no Shortcuts tab for folder references.
- On the Advanced tab, some of the OLE/ActiveX and Font options do not exist because they cannot be applied to multiple files.



Local folder: C:\Program Files\Widget Master Pro\Docs

☑ Recurse subfolders

File mask: ● Files that match ○ Files that do not match

.pdf|.htm

Description:

Run-time folder: Archive

Install to: %AppFolder%\Docs

Overwrite: Overwrite if existing file is older

Working with Multiple Files

You can use the Multiple File Properties dialog to edit the properties of more than one file (or folder reference) in your project at a time.

To access the Multiple File Properties dialog:

1. Select more than one file on the project window. (You can select multiple files by pressing and holding the Ctrl key or the Shift key while you click on the files.)

2. Choose Project > File Properties from the menu.

   (You can also click the File Properties button, press the Ctrl+Enter key, or right-click on the files and select File Properties from the right-click menu.)

   This will open the Multiple File Properties dialog, where you can view and edit the properties of the files you selected.

The Multiple File Properties dialog is similar to the File Properties dialog, but there are some important differences:

- There is no Notes tab.

- You cannot enter text in a field directly; instead, you must click on the Edit button to the right of the field, and use the Edit Multiple Values dialog to change the text. Note that the Edit Multiple Values dialog allows you to completely replace the text for all selected files, append (or prepend) some text to the value for each selected file, or search and replace among the values for all selected files.

- Check boxes on the Multiple File Properties dialog have three states: enabled (☑), disabled (□), and mixed (▨). The mixed state preserves the settings for that check box in all of the selected files.

- The Overwrite setting includes an option in the drop-down list to use the original values for each selected file.

For more information on using the Multiple File Properties dialog, please consult the Setup Factory help file.

Missing Files

Since Setup Factory only records an informational link to each file, and doesn't actually maintain a copy of the file itself, it's possible for files to go "missing" from Setup Factory's point of view. For example, if a file in your project is moved to another folder, Setup Factory will no longer be able to find it at its original location. The same thing happens when a file is renamed or deleted. Setup Factory only knows to look for a file at the place where it was when you "showed" the file to Setup Factory by adding it to your project.

Although Setup Factory might not know where a missing file has ended up, it definitely knows when a file is missing. Whenever a file in your project can't be found at its original location, Setup Factory displays the file's information on the project window in red instead of black. The red color makes it easy for you to see which files in your project are missing.

The file's status will also show as "Missing" instead of "OK" in the Status column on the list view.

Tip: A quick way to determine which files are missing in your project is to click on the Status column header to sort the files in the list view according to their status.

If you find that your files are suddenly playing hide-and-seek with Setup Factory, try to remember if you've made any changes to your files recently. If you moved the files, you can try moving them back. If you renamed the files, you can restore the original names. If you deleted the files, you'll have to replace them.

If Setup Factory still shows the files in red after you've corrected the situation, you probably just need to refresh the display. To do so, simply choose View > Refresh from the menu, or press the F5 key. Refreshing the display causes Setup Factory to re-examine the location of every file in your project.

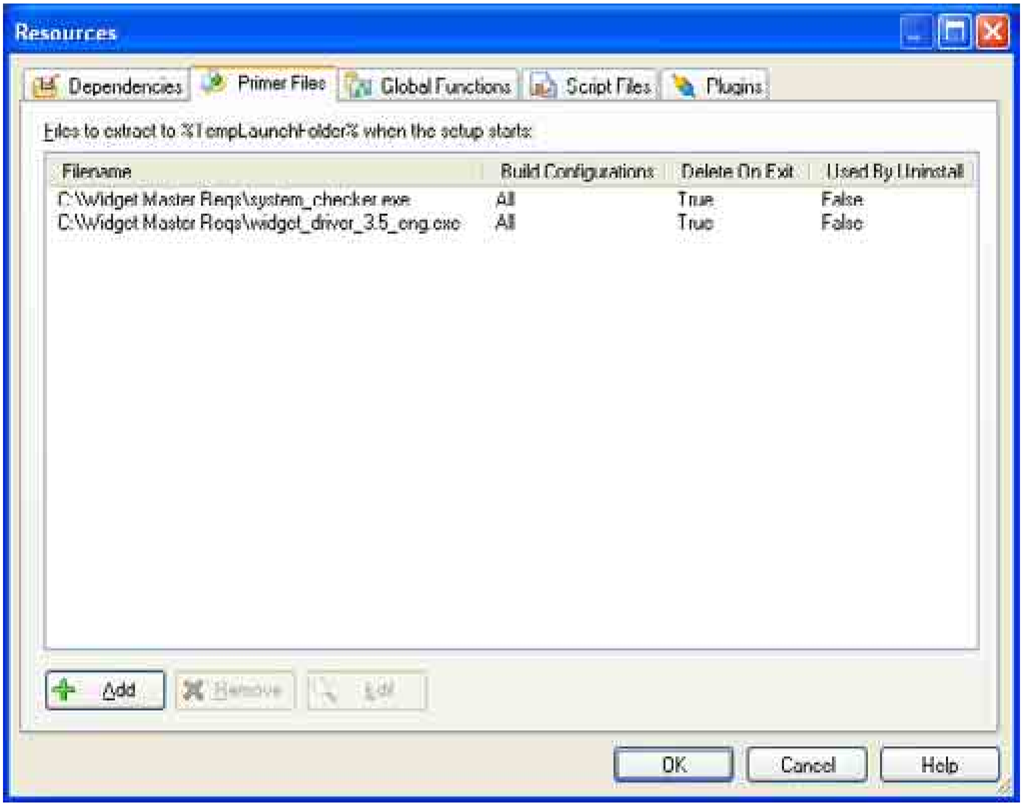
Note: Setup Factory automatically refreshes the display for you when you open a project or initiate the build process.

If you moved, renamed, or deleted the files on purpose, and you want Setup Factory to use the files at their new locations, remember the files by their new names, or just forget about the past and move on, you'll need to change the local source path on the File Properties dialog for each file. This can easily be done by selecting all the files and then modifying their properties using the Multiple File Properties dialog.

Primer Files

Primer files are just regular files that get extracted from the setup executable *before* the installation process begins. This means you can use primer files at the very start of the installation process, right after the user runs the setup executable.

You can make any file a primer file simply by adding it to the Primer Files tab on the Resources dialog. The files on this tab will be included in the setup executable when you build the installer.



Files to extract to %TempLaunchFolder% when the setup starts:

Filename	Build Configurations	Delete On Exit	Used By Uninstall
C:\Widget Master Reqs\system_checker.exe	All	True	False
C:\Widget Master Reqs\widget_driver_3.5_eng.exe	All	True	False

Primer files are automatically extracted to a temporary folder at run time. The path to this folder is automatically stored in a session variable named %TempLaunchFolder%. You should use this session variable in actions when you need to access your primer files.

Primer files make it easy to run a program on the user's system before the rest of your files are installed. Just add the program to the list of primer files, and execute it with a File.Run action early in the installation process.

For example, you might need to execute a custom program or DLL function before your software is installed—perhaps to perform some product-specific, low-level pre-installation tests on the user's hardware, and write the results to the Registry. By adding your custom program or .dll to the list of primer files, you could distribute it "inside" your setup executable and still be able to run it before any of the "normal" files in your project are installed.

Tip: Another way to access files at the start of the installation process is to distribute them "alongside" your installer. For instance, you could store the files "externally" on the same CD-ROM, and access them directly; or, you could download the files from your web site using an HTTP.Download action in the On Startup project event.

Chapter 3:

Creating the User Interface

Creating the user interface (or UI) is one of the most important aspects of any setup since the UI is the first thing your end users will see when they run your installer. Making a great first impression with a good-looking UI is something that all users of Setup Factory 7.0 should be concerned about.

This chapter will introduce you to the user interface and get you well on your way to creating a sharp and consistent look and feel for all of your projects.

3

In This Chapter

In this chapter, you'll learn about:

- The user interface
- The background window
- Screens
- Themes
- Taskbar settings
- Dialog and Status Dialog actions

The User Interface

You can think of the user interface (UI) as any part of the setup that the end user will see. When a screen is displayed, the end user is seeing part of the user interface. When the end user clicks the Next button, the end user is interacting with the user interface.

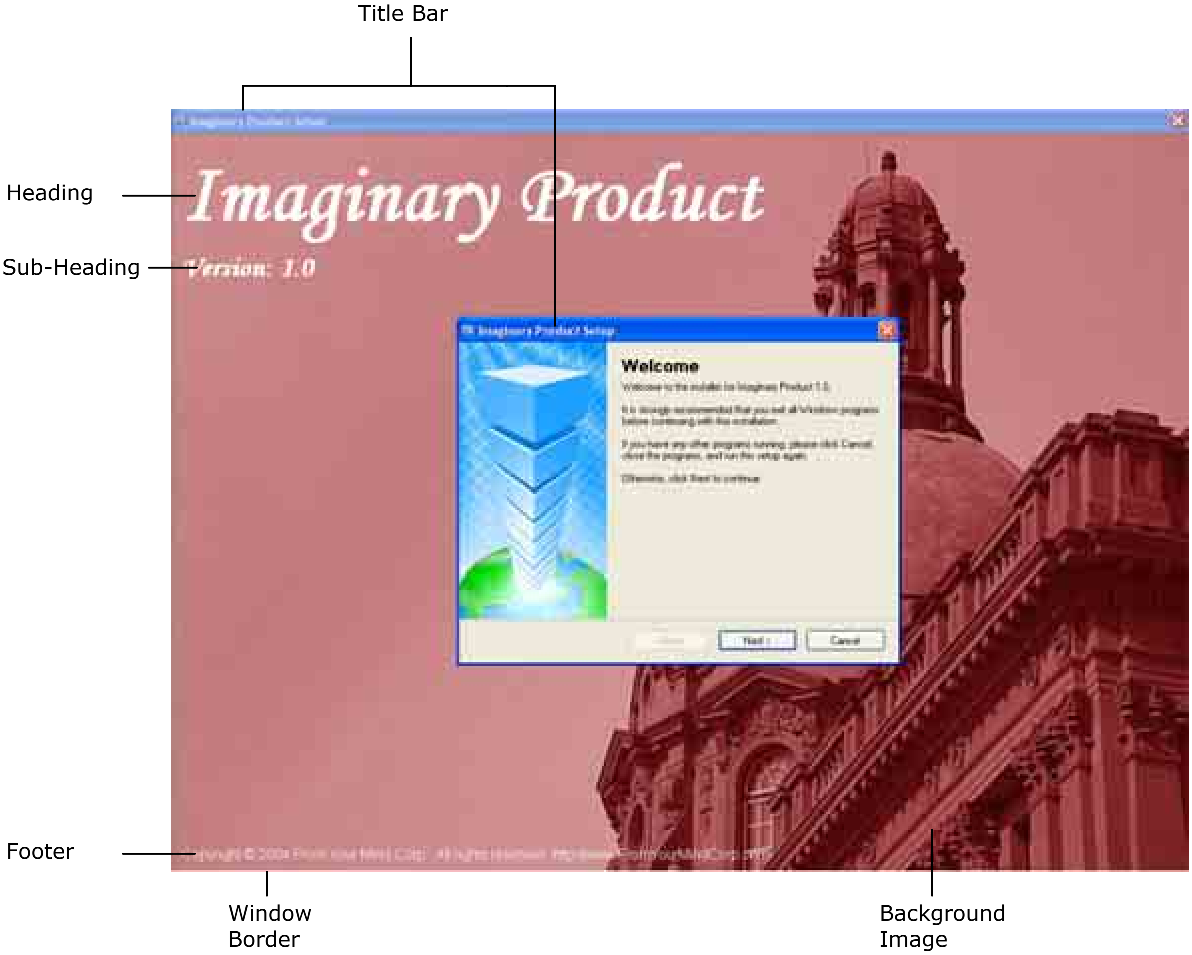
The basic elements of the user interface are:

- The Background Window
- Screens
- Themes
- Taskbar visibility
- Any actions that generate a UI element (e.g. Dialog.Message, StatusDlg.Show)

The Background Window

The background window is a maximized window that will appear behind all screens in your installation. Its main use is to focus the end user's attention on your setup by blocking out the rest of the desktop.

The settings for the background window can be found in the menu under Project > Background Window. The next few sections will explain all the settings available for the background window.



Welcome

Welcome to the installer for Imaginary Product 1.0.

It is strongly recommended that you exit all Windows programs before continuing with this installation.

If you have any other programs running, please click Cancel, close the programs, and run the setup again.

Otherwise, click Next to continue.

Copyright © 2004 From Your Mind Corp. All rights reserved. http://www.FromYourMindCorp.com

Window Style

The window style options control the title bar and border styles that will be applied to the background window. All background windows will be full screen and will cover the task bar regardless of the style you choose.

There are three different window styles:

- Standard
- Bordered
- Kiosk

Standard

The standard window style will display a background window with the features of a normal window (i.e., a title bar with a close button and a standard window border).

You should probably choose the standard window style if you want to present your setup and background window as a standard application. The full background window allows you to display extra information, but the border makes it known to the end user that this is just a standard application.

The standard window style also lets the end user move the background window by dragging the window via the title bar.

Note: The text on the window title bar is controlled by the %WindowTitle% session variable.

Bordered

The bordered window style has the same border as the standard window style but does not have a title bar.

The bordered style is a good choice if you do not want the end user to be able to move your background window, but you still want them to know that your installation is just another application (made clear by the border).

Kiosk

The kiosk window style will create a background window without a title bar or border.

This is an effective style to choose if you want an unmovable background window that thoroughly engages the end user's attention. Because there is no border or title bar to differentiate your installation from the rest of the operating system, there is nothing to compete with your background window.

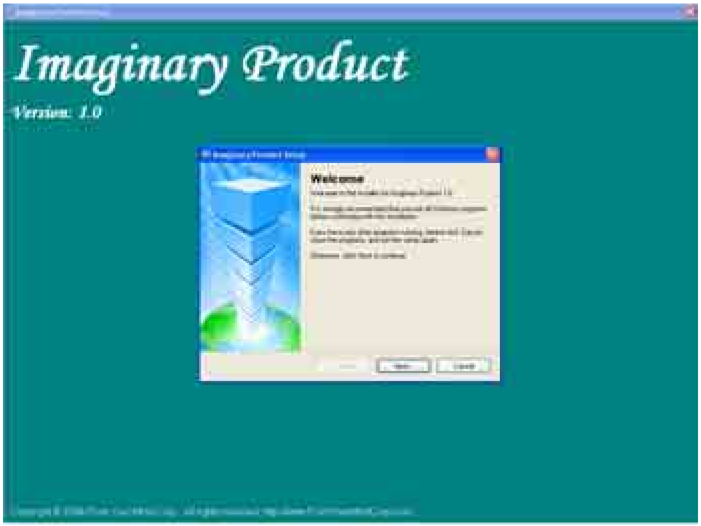
Window Appearance

The window appearance settings control the appearance of the background window. These options control what the interior of the background window will look like. You can think of the interior of the background window as everything but the title bar and the border.

There are three appearance options: solid color, gradient, and image.

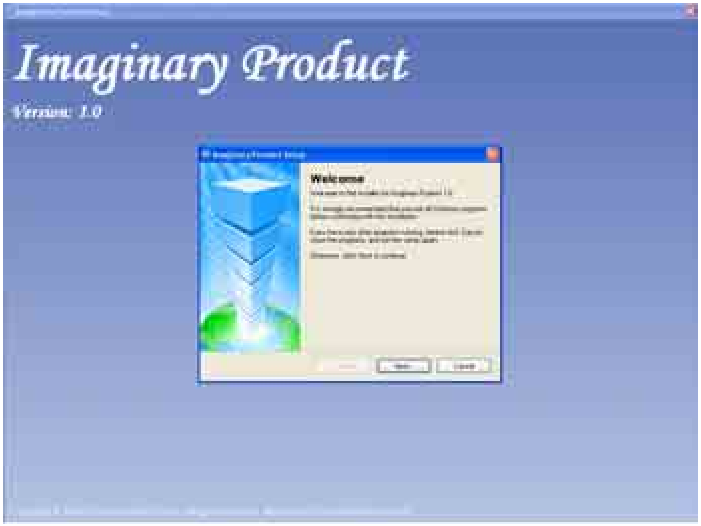
Solid Color

The solid color option will color the interior of the background window a single color.



Gradient

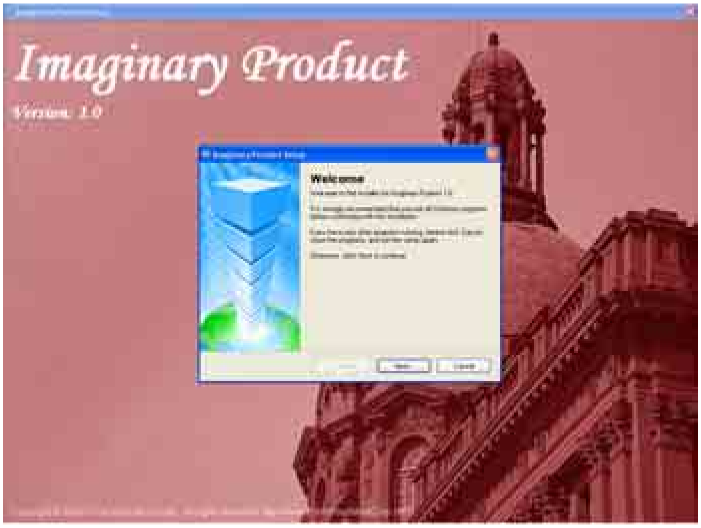
The gradient option will create a vertical gradient on the interior of the background window. The left color picker selects the color for the top of the background window and the right color picker selects the bottom color.



Image

The image option will use the image that you select to fill the interior of the background window.

The image you select will be stretched to fill the interior of the background window regardless of its size. This means that your image will vary in appearance on computers with different screen resolutions. It is very important to select an image that will look good at different sizes.



Background Window Text

The background window text settings control the look and position of the text that can be displayed on the interior of the background window.

Three separate text elements can be displayed on the background window: a heading, a subheading, and a footer. Each of these has similar formatting options, including the font type, the color of the text, whether to draw the text in a 3D style, the 3D color (if enabled), and the alignment of the text.

Heading

The heading text will be displayed at the top of the interior of the background window. It is usually used to display important information such as the name of your product or your computer name.

Subheading

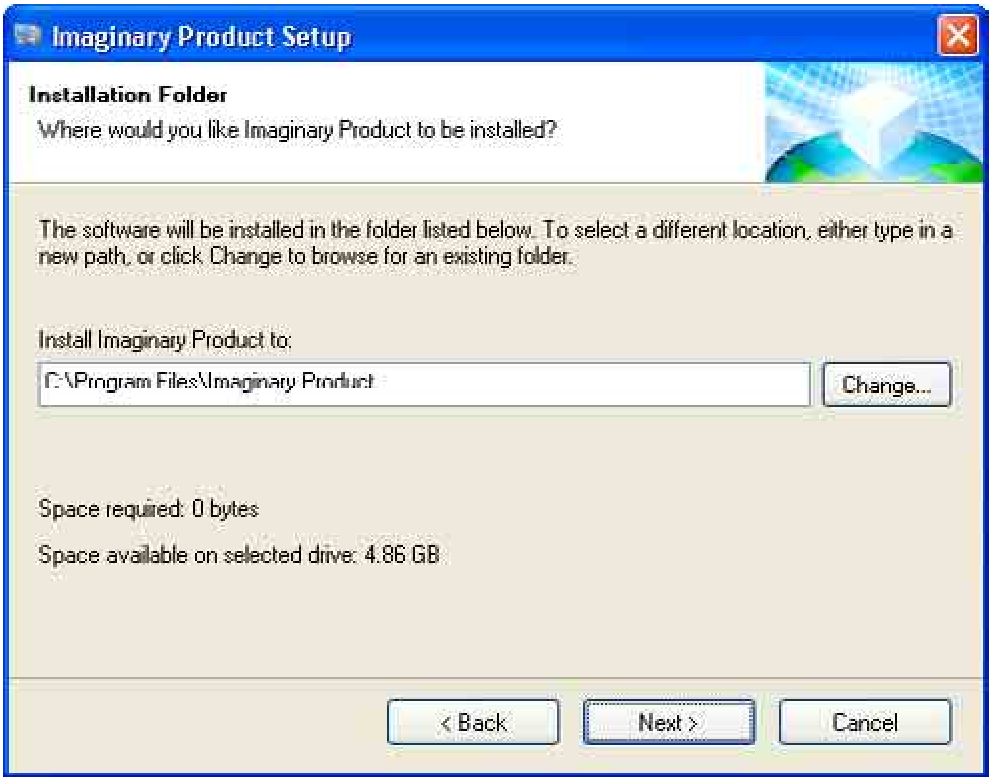
The subheading text will appear directly below the heading text and can contain your product’s version number, a marketing message, or any other message.

Footer

The footer text will appear at the bottom of the interior of the background window and is useful for displaying copyright information or website addresses.

Screens

The most important aspects of your user interface are the screens that you choose to display, since this is where your end user will actually interact with the setup. Screens allow you to provide important information (such as your license agreement) and allow your user to make decisions (such as where to install your product).



Imaginary Product Setup

Installation Folder

Where would you like Imaginary Product to be installed?

The software will be installed in the folder listed below. To select a different location, either type in a new path, or click Change to browse for an existing folder.

Install Imaginary Product to:

C:\Program Files\Imaginary Product

Space required: 0 bytes

Space available on selected drive: 4.86 GB

Screens are the individual windows that make up your installation. When you navigate through an installation by clicking the Next and Back buttons, you are navigating from screen to screen.

Tip: You can think of the screens in your project as steps in a wizard, walking your user through the process of installing your software.

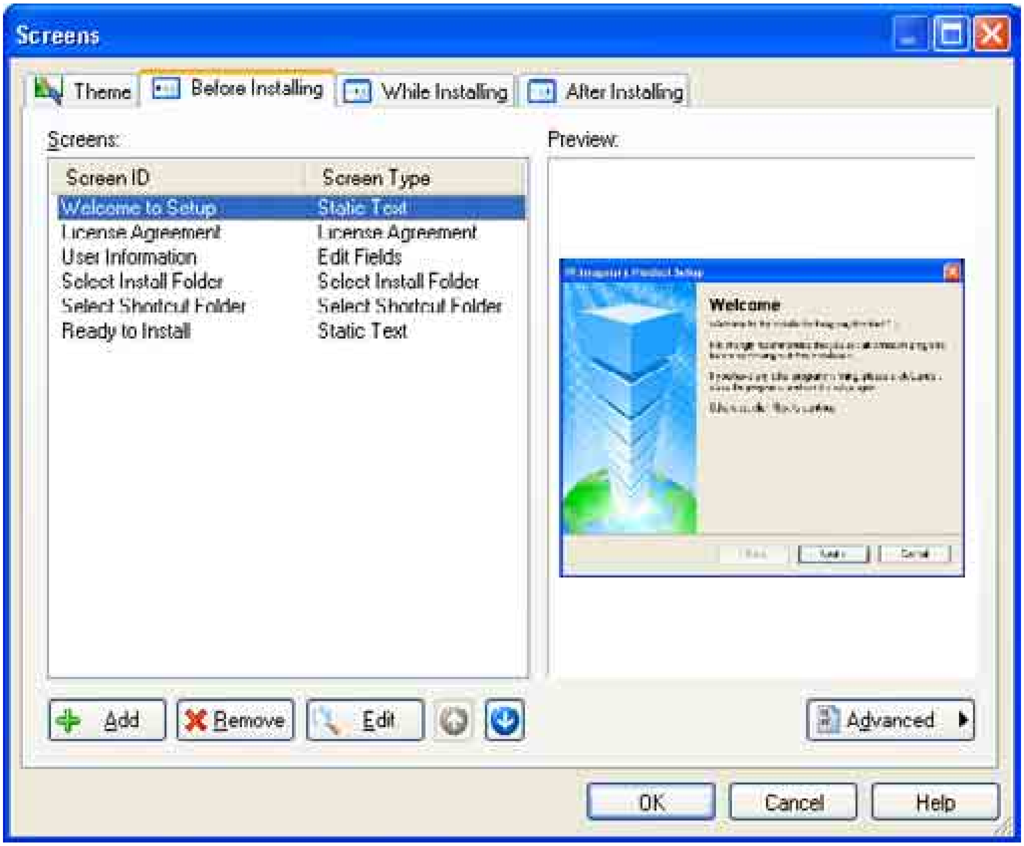
In general, each screen performs a single task, like showing a welcome message, allowing the end user to select which packages they want to install, or letting the end user select the name of a shortcut folder. If you want to perform a major setup task, chances are you will need a screen to do so.

The Screen Manager

The screen manager is where you will configure all of your screens. There are two screen managers in Setup Factory 7.0: one for the installer (Project > Screens) and one for the uninstaller (Uninstall > Screens).

Aside from when the screens will be displayed, the only difference between the two screen managers is that the uninstaller's screen manager does not allow you to configure the project theme.

Note: The project theme chosen in the installation screen manager affects your entire project and will be applied to screens in both the installer and uninstaller.



Screen ID	Screen Type
Welcome to Setup	Static Text
License Agreement	License Agreement
User Information	Edit Fields
Select Install Folder	Select Install Folder
Select Shortcut Folder	Select Shortcut Folder
Ready to Install	Static Text

Screen Stages

There are six stages at which screens can be displayed in Setup Factory 7.0. The first three stages are related to the installation, and the last three are related to the uninstallation.

The three installation stages at which screens can be displayed are:

- Before Installing
- While Installing
- After Installing

The three uninstallation stages at which screens can be displayed are:

- Before Uninstalling
- While Uninstalling
- After Uninstalling

Each screen stage is represented by a tab on the appropriate screen manager.

You will notice that the installation and uninstallation stages are very similar. In fact, the only difference between them is *when* they occur, i.e. during the installation or during the uninstallation.

Once you understand how the installation stages work, you can apply the same knowledge to the uninstallation stages. That being the case, the following section describes the installation stages only; however, the information is easily applicable to the uninstallation stages by substituting the word “uninstall” wherever you read “install.”

Before Installing

The Before Installing screens are displayed before the actual installation of any files occurs. In general, if the end user cancels the setup during this stage, no files will be installed on their computer.

This stage is used to perform such tasks as displaying your license agreement and allowing the end user to select where your product will be installed.

While Installing

The While Installing stage occurs while the files are being installed onto the end user’s computer. This stage differs from both the Before Installing and After Installing stages in that only one screen can be displayed (as opposed to a sequence of screens), and it must be a progress screen.

This stage is used to display the file installation progress as it occurs. In other words, while your files are being installed onto the end user’s computer, the screen you chose for the While Installing stage will display information about the progress of that installation, such as what file is being installed, what percentage of all files has been installed so far, and what percentage of the current file has been installed.

After Installing

The After Installing stage is the final screen stage and occurs after all of the files have been installed onto the end user's computer.

This stage is typically used to show a reboot advisement (if needed), provide any post-installation instructions, and inform the user that the installation has finished successfully.

Adding Screens

Adding a screen to your setup is as easy…you just click the Add button at the bottom of the screen manager.

Note: The While Installing screen stage is different since a maximum of one screen is allowed. For this stage, the button is labeled "Change" instead of "Add."

Clicking the Add button brings up a screen gallery where you can select from a variety of screen types. Once you've selected the type of screen you want, simply click the OK button to add it to the list of screens for the stage that you're currently working on.

Removing Screens

To remove a screen from your installation, simply select it in the screen list and click the Remove button.

Tip: If you remove a screen from your installation by accident, you can undo the deletion by pressing CTRL + Z.

Ordering Screens

In general, the order in which screens appear in the screen manager will be the order in which they appear during that screen stage. The screen that is at the top of the list will appear first and the screen that is at the bottom of the list will be the final screen of that stage.

To change the order of your screens, simply select a screen in the list and click the Up or Down button until it's in the desired location. Or, if you prefer, you can simply drag the screen from one position to another.

Tip: You can also use the cursor keys to move a screen up or down in the list. Simply select the screen you want to move, and press Alt+Up or Alt+Down.

Editing Screens

To edit a screen’s properties, just select it in the list and click the Edit button.

Clicking the Edit button opens the Screen Properties dialog where you can edit and customize all of the settings for that screen.

Tip: You can also edit a screen by double-clicking on it in the list.

Screen Properties

The Screen Properties dialog is where you can edit the properties of a specific screen. All Screen Properties dialogs have the same four tabs (although the specific content on these tabs may differ): Settings, Attributes, Style and Actions.

Settings

The Settings tab allows you to edit properties that are specific to the selected screen. Each screen type has different settings that are specific to that type of screen.

For example, a Check Boxes screen will have settings that apply to check boxes on its Settings tab.

For more information on the specific screen settings, please see the Setup Factory 7.0 help file.

Attributes

The Attributes tab contains settings that are common to all screens. The only differences that you will find between Attributes tabs are that progress screens lack options for the Next, Back, and Help buttons. (Those buttons don’t exist on progress screens.)

The Attributes tab is where you can configure which banner style to use, the name of the screen, and the navigation button settings.

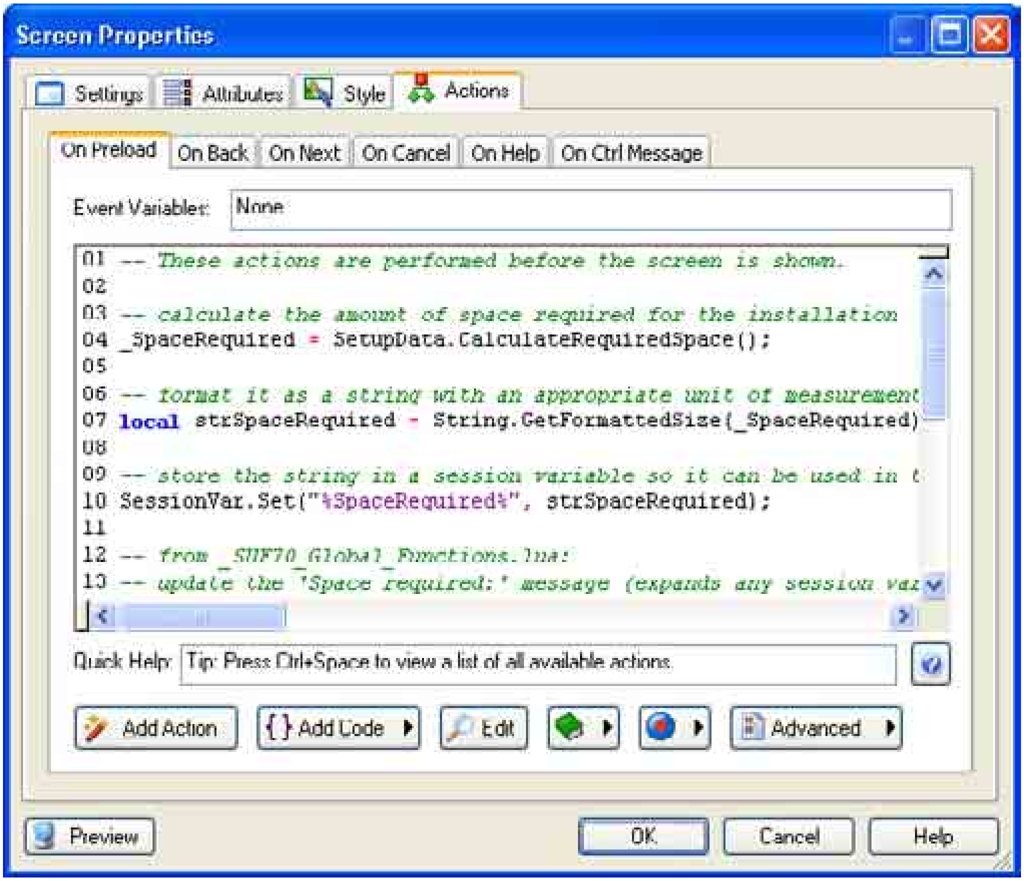
Style

The Style tab is where you can override the project theme on a per-screen basis. By default the project theme is applied to all screens throughout your project, however there might be some instances where you feel a certain screen needs something a bit different in order to stand out. You can use the Style tab to override any of the theme settings on a specific screen. (The changes will only be applied to that screen.)

Actions

The Actions tab is where you can edit the actions associated with the screen's events.

For more information on actions and events, please see the help file and Chapter 4.



Event Variables: None

```
-- These actions are performed before the screen is shown.

-- calculate the amount of space required for the installation
_SpaceRequired = SetupData.CalculateRequiredSpace();

-- format it as a string with an appropriate unit of measurement
local strSpaceRequired = String.GetFormattedSize(_SpaceRequired)

-- store the string in a session variable so it can be used in t
SessionVar.Set("%SpaceRequired%", strSpaceRequired);

-- from _SUF70_Global_Functions.lua:
-- update the 'Space required:' message (expands any session var
```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

The Language Selector

The Settings and Attributes tabs both have a language selector in the bottom right corner. The language selector is a drop-down list containing all of the languages that are currently enabled in the project. It is used for creating multilingual installations.

Selecting a language in the list allows you to edit the text that will be used on the screen when that language is detected.

Note: Unlike previous versions of Setup Factory, the language selector allows you to display different text on a screen for specific languages without having to create a separate screen for each language.

Session Variables

Session variables play a large part in the way that screens work and how they display their text. Anytime you see something in Setup Factory 7.0 that looks like %ProductName%, what you are looking at is a session variable.

Note: A session variable is essentially just a name (with no spaces) that begins and ends with %.

Session variables are very similar to normal variables in that they serve as "containers" for values that may change. We say that values are "assigned to" or "stored in" session variables. When you use a session variable, its name (i.e. %ProductName%) is replaced at run time by its value (i.e. "Setup Factory 7.0"). Session variables are basically *placeholders* for text that gets inserted later.

Session variables are often used in the default text for screens. They are automatically expanded the first time the screen is displayed, so instead of seeing %ProductName% on the screen, the end user will actually see the name of your product that you entered in the session variable editor: Project > Session variables.

Session variables are also used to store return values when screens or controls need them. For example, the Select Install Folder screen stores the installation folder path in a session variable named %AppFolder% by default.

Tip: Session variables can be created and changed at run time using actions like SessionVar.Expand, SessionVar.Get, SessionVar.Remove, and SessionVar.Set.

For more information please see Chapter 5, which discusses session variables in more detail.

Screen Navigation

Screen navigation can be thought of as the path that the end user takes through your installation. The end user navigates forward through various screens by clicking the Next button, and backward through the screens by clicking the Back button.

Screen navigation is basically a linear path from the top of your screen list in the screen manager to the bottom. Generally, the order of your screen list in the screen manager is exactly the order in which the navigation will proceed.

Although there are other ways to control the path through the screens (e.g. using actions to create a “branching” path), in most cases the default behavior is all that is needed.

How Screen Navigation Works

In its simplest form, screen navigation is when the end user moves forward or backward through your installation by clicking the Next and Back buttons. By default, this moves the end user down or up through the list of screens on the screen manager.

This is actually accomplished using actions. Each screen has Screen.Next and Screen.Back actions on its On Next and On Back events which are performed when the Next and Back buttons are clicked. If you ever need to, you can modify or override the default behavior of any screen by editing or replacing the default actions. Most of the time, however, you will not even need to know that the actions are there.

Navigation Buttons

Navigation buttons are the Back, Next, and Cancel buttons that are usually visible along the bottom (or “footer”) of each screen. The Next button moves the end user down the screen list from the top to the bottom, the Back button moves the end user up through the screen list, and the Cancel button stops the user’s navigation by canceling the installation.

The settings for these buttons can be found on the Attributes tab of the screen properties dialog for each screen. There you can change the text, enabled state and visible state of these buttons.

The two options for the visibility state are self-explanatory; they make the button either visible or invisible. The two options for the enabled state make the button enabled or disabled. If a button is in the enabled state, it looks and functions like a normal button; it will depress when the user clicks on it, and the text is displayed in its normal color (usually black). When a button is in the disabled state, however, it will not respond to the user's mouse, and is typically drawn in less prominent gray colors (also known as being "ghosted" or "grayed out").

Each navigation button has an event that will be fired when the button is clicked. These events can be found on the Actions tab of the screen properties dialog.

Note: A Help button is also available on the footer of each screen but is generally not considered a navigation button.

Navigation Events

An event is something that can happen during the installation. When an event is triggered (or "fired"), any actions that are associated with that event are performed. Note that an event must be triggered in order for its actions to be performed…if an event is not triggered, the actions associated with it will not be performed.

Each event represents something that can happen while your installation is running. For example, all screens have an On Preload event, which is triggered just before the screen is displayed. To make something happen before a screen is displayed, you simply add an action to its On Preload event.

All of the three navigation buttons have an event that will be fired when they are clicked. The events are "On Back" for the Back button, "On Next" for the Next button and "On Cancel" for the Cancel button.

In the case of the three navigation buttons, navigation actions are executed when their respective events are fired. This allows the end user to navigate through the installation from the beginning to the end.

There are other events that are associated with screens but aren't necessarily related to screen navigation:

- On Preload – just before the screen is displayed.
- On Help – when the help button is selected.
- On Ctrl Message – triggered by a control on the screen.

Navigation Actions

There are six navigation actions available to you in Setup Factory 7.0: Screen.Back, Screen.End, Screen.Jump, Screen.Next, Screen.Previous, and Application.Exit. The most common of the six actions are Screen.Next and Screen.Back.

When the Next button is clicked, the end user is attempting to navigate from the current screen to the next screen or phase of the installation. The easiest way to implement this behavior is to insert the Screen.Next action on the On Next event. This is done by default for all screens.

The same holds true for the Back button; when the Back button is clicked, the end user is attempting to move backwards in the installation to the previous screen. To implement this behavior, a Screen.Back action needs to be executed when the On Back event is fired.

Note: The Screen.Back action moves backward in the installation's history in the same way that a Back button does in a web browser. To move up (back) one screen in the screen list, use the Screen.Previous action.

In certain situations, simply moving down the screen list is not the appropriate behavior; instead, jumping to a specific screen in the screen stage is necessary. You can accomplish this by using a Screen.Jump action. If the goal is to jump to the next phase in the installation—i.e., to end the current screen stage—a Screen.End action can be used to jump past all of the screens in the current screen stage.

To interrupt screen navigation—which usually occurs when the Cancel button is clicked—you can use an Application.Exit action. The Application.Exit action causes your installation to stop as soon as it is performed.

Note: For more information on the specifics of screen actions, please see the help file.

Screen Layout

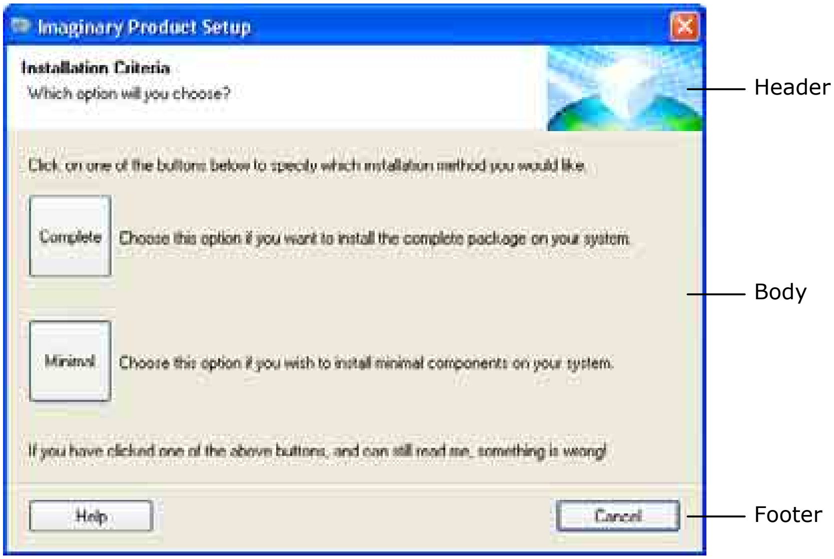
In Setup Factory 7.0 choosing a layout for your screens and the controls on them is now easier then ever. You can switch between all three banner styles (top, side, and none) on any screen that you like, and the controls on your screens will dynamically position themselves ensuring that all of your information is visible.

Note: A control can be thought of as the visible elements on a screen, from edit fields, to radio buttons, to static text controls. However, when the term "control" is used, it does not generally refer to the navigation buttons or banner text.

Header, Body, Footer

Screens in Setup Factory 7.0 are divided into three basic parts: the header, the body and the footer.

The header runs across the top of each screen and can be thought of as the area that the top banner fills.



Installation Criteria

Which option will you choose?

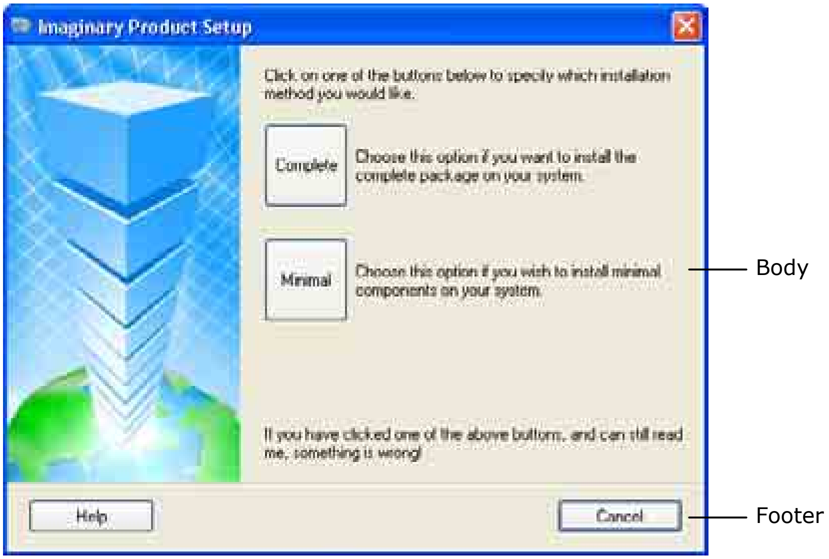
Click on one of the buttons below to specify which installation method you would like.

Choose this option if you want to install the complete package on your system.

Choose this option if you wish to install minimal components on your system.

If you have clicked one of the above buttons, and can still read me, something is wrong!

The footer is similar to the header area except that it runs along the bottom of each screen. This is the area of the screen where the navigation buttons are placed.



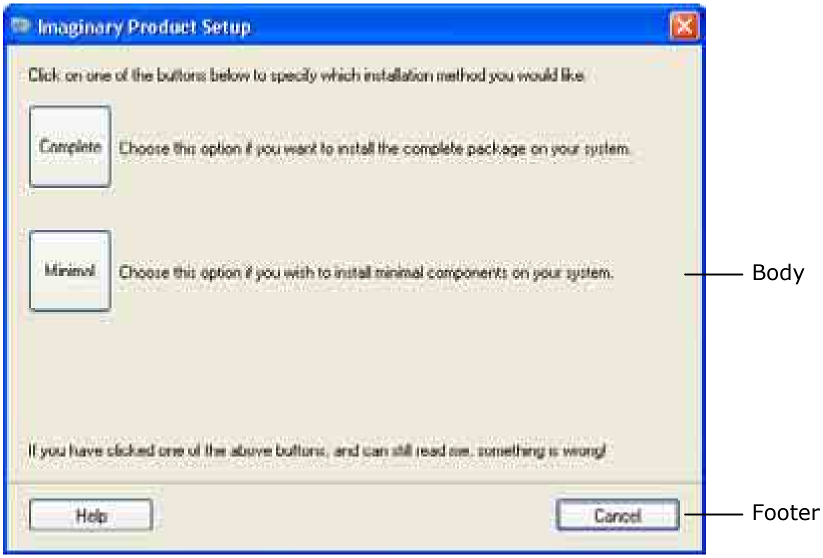
Click on one of the buttons below to specify which installation method you would like.

Choose this option if you want to install the complete package on your system.

Choose this option if you wish to install minimal components on your system.

If you have clicked one of the above buttons, and can still read me, something is wrong!

Now that you know what the header and the footer are, you can think of the body as the rest of the screen. The body of a screen takes up the majority of each screen and will contain most of the screen's information.


Imaginary Product Setup

Click on one of the buttons below to specify which installation method you would like.

Choose this option if you want to install the complete package on your system.

Choose this option if you wish to install minimal components on your system.

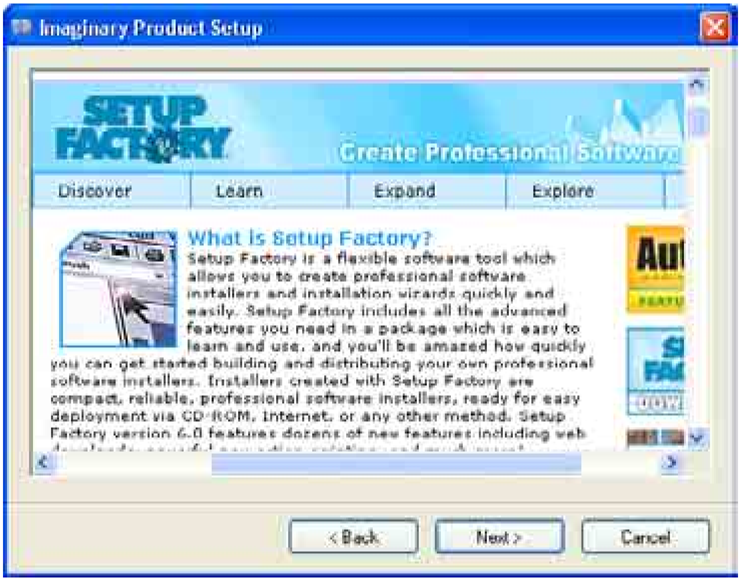
If you have clicked one of the above buttons, and can still read me, something is wrong!

Banner Style

In Setup Factory 7.0 the term banner refers to an area of the screen that is special and somewhat separate from the rest of the screen. You can use the banner area to display some descriptive text, an image, or both.

There are three different types of banner styles available in Setup Factory 7.0: none, top, and side.

The none banner style is the easiest of all three styles to understand since it means that there will be no banner displayed on the screen.



What is Setup Factory?

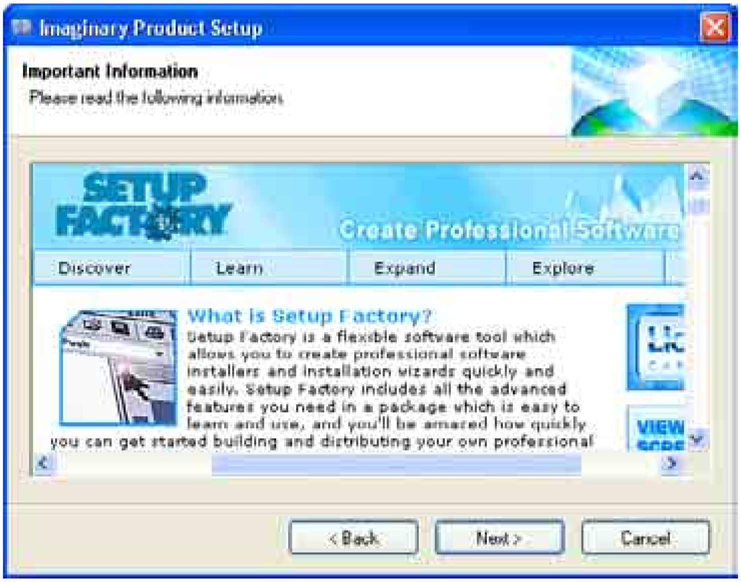
Setup Factory is a flexible software tool which allows you to create professional software installers and installation wizards quickly and easily. Setup Factory includes all the advanced features you need in a package which is easy to learn and use, and you'll be amazed how quickly you can get started building and distributing your own professional software installers. Installers created with Setup Factory are compact, reliable, professional software installers, ready for easy deployment via CD-ROM, Internet, or any other method. Setup Factory version 6.0 features dozens of new features including web

The none banner style

The top banner style will give you a long thin banner across the top of your screen (the header). This is the style that you will probably apply to the majority of the screens in your project. The top banner style supports two lines of text referred to as the heading text and the subheading text. This text is usually used to describe the current screen and/or provide some information about what is required of the end user.

The top banner style also supports an image that will be placed on the right hand side of the banner. Setup Factory 7.0 will resize this image proportionally so that its height matches the height of the top banner. The width of the image can be as wide as you want and can take up the entire banner if necessary.

Any area of the top banner that is not covered by an image will be painted with a color according to the project theme.



Imaginary Product Setup

Important Information

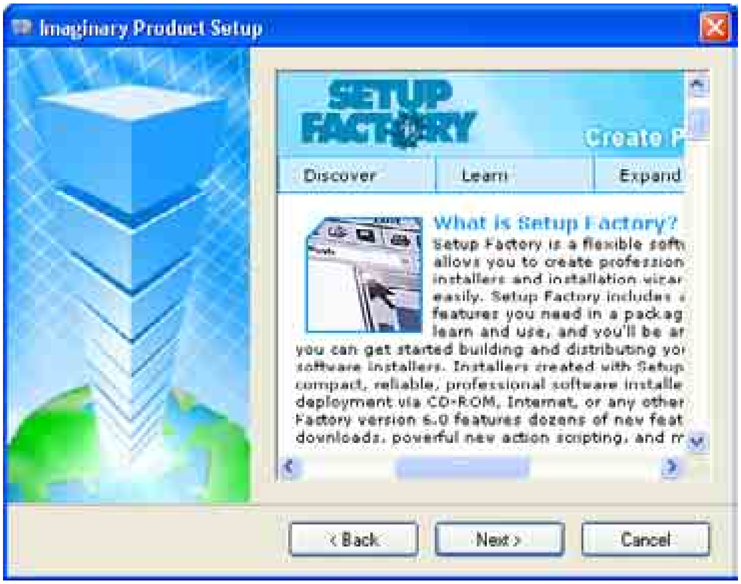
Please read the following information

What is Setup Factory?

Setup Factory is a flexible software tool which allows you to create professional software installers and installation wizards quickly and easily. Setup Factory includes all the advanced features you need in a package which is easy to learn and use, and you'll be amazed how quickly you can get started building and distributing your own professional

The top banner style

The side banner style will give you a thick banner that runs down the left side of your screen. The side banner will start at the top of the screen and then stop at the top of the footer of your screen. The side banner style supports an image that will automatically be stretched or resized by Setup Factory 7.0 to fill the entire side banner area.



Imaginary Product Setup

What is Setup Factory?

Setup Factory is a flexible softw
allows you to create profession
installers and installation wizar
easily. Setup Factory includes a
features you need in a packag
learn and use, and you'll be ar
you can get started building and distributing yo
software installers. Installers created with Setup
compact, reliable, professional software installe
deployment via CD-ROM, Internet, or any other
Factory version 6.0 features dozens of new feat
downloads, powerful new action scripting, and m

The side banner style

Dynamic Control Layout

One of the best new features in Setup Factory 7.0 is the dynamic control layout ability of screens. Setup Factory 7.0 will dynamically reposition the controls on your screen so that the maximum amount of information stays visible.

Dynamic control layout means that controls will resize and layouts will adjust automatically as you type. You no longer have to manually place your controls and you won’t find yourself locked into a pre-determined amount of space; dynamic control layout means that the layout will be automatically chosen based upon your chosen control settings.

Furthermore, in Setup Factory 7.0 you no longer have to worry about fitting your descriptive text within three lines. If you want a fourth (or fifth) line just type it, and all of the controls on the screen will resize to fit the new lines of text.

The dynamic repositioning of controls takes place within an area called the control area. The control area of a screen occupies a sub-section of the body of a screen; its size is controlled by the global theme.

The best part of the dynamic control layout feature is that it works without any effort on your part. Simply fill your screens with all of the information and controls that you desire, and Setup Factory will re-position all the controls so that everything is visible and a visually appealing look is achieved.

This is not to say that you do not have any control over how controls will be displayed on your screen; in fact, it’s just the opposite. Many screens (Edit Fields, Checkboxes, etc.) allow you to add as many as 32 controls to your screen, which Setup Factory 7.0 will dynamically position. You have the ability to choose how many columns you want the controls displayed in, whether they are distributed horizontally or vertically and, in the case of the Edit Fields screen, how many columns each control spans!

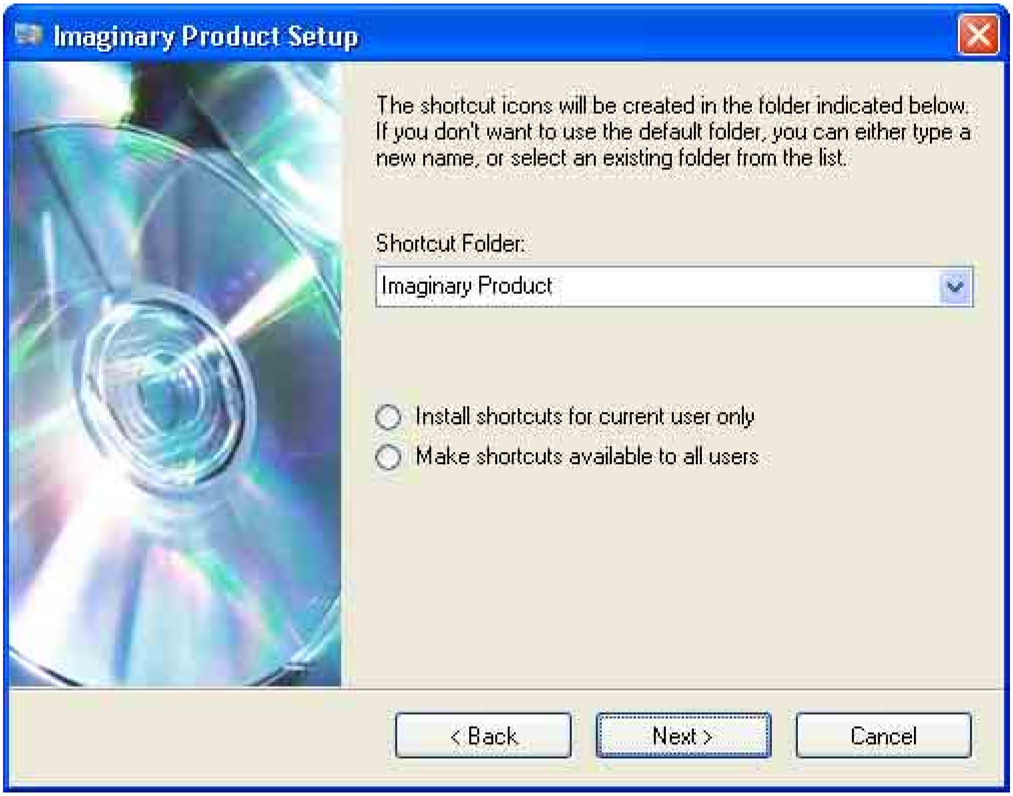
If this seems a bit confusing, don’t worry about it. The best way to understand the dynamic control layout feature is to use it. Try playing around with the settings of the Check Boxes or Select Install folder screens and observe how Setup Factory 7.0 positions your controls to achieve the best look possible.

Themes

A theme is a group of settings and images that control the way your installation looks. You've probably encountered themes before when using other applications or even Windows XP. Themes do not change *what* is displayed; instead, they change *how* it is displayed.

Themes in Setup Factory 7.0 control the general appearance of your screens and the controls they contain. Rather then controlling the position of screen controls or the banner style used, themes control the color and font of screens and controls. Themes are project-wide and affect all screens in the installation and uninstall unless intentionally overridden on the style tab of the screen's properties.

Themes provide an easy way to change the look and feel of your screens and controls without having to go to each screen and update it every time you wish to apply a new visual style.



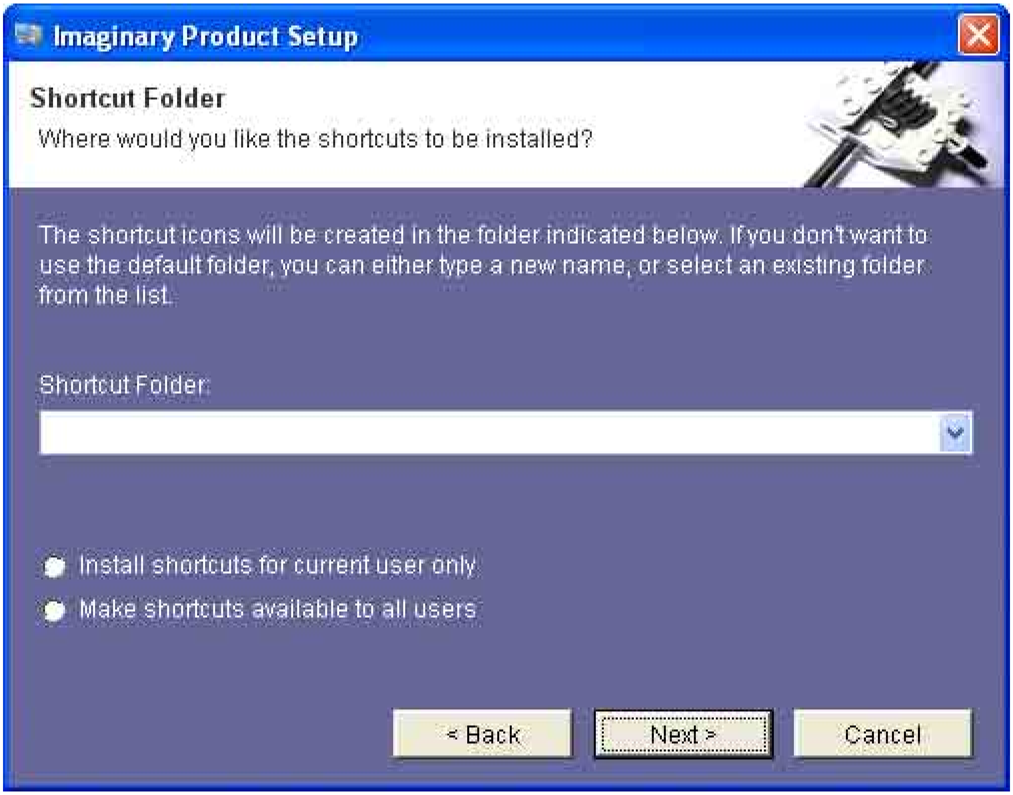
The shortcut icons will be created in the folder indicated below. If you don't want to use the default folder, you can either type a new name, or select an existing folder from the list.

Shortcut Folder: Imaginary Product

○ Install shortcuts for current user only

○ Make shortcuts available to all users

A Select Shortcut Folder screen with the "CD-ROM" theme applied



Imaginary Product Setup

Shortcut Folder

Where would you like the shortcuts to be installed?

The shortcut icons will be created in the folder indicated below. If you don't want to use the default folder, you can either type a new name, or select an existing folder from the list.

Shortcut Folder:

○ Install shortcuts for current user only

○ Make shortcuts available to all users

The same screen with a custom theme applied

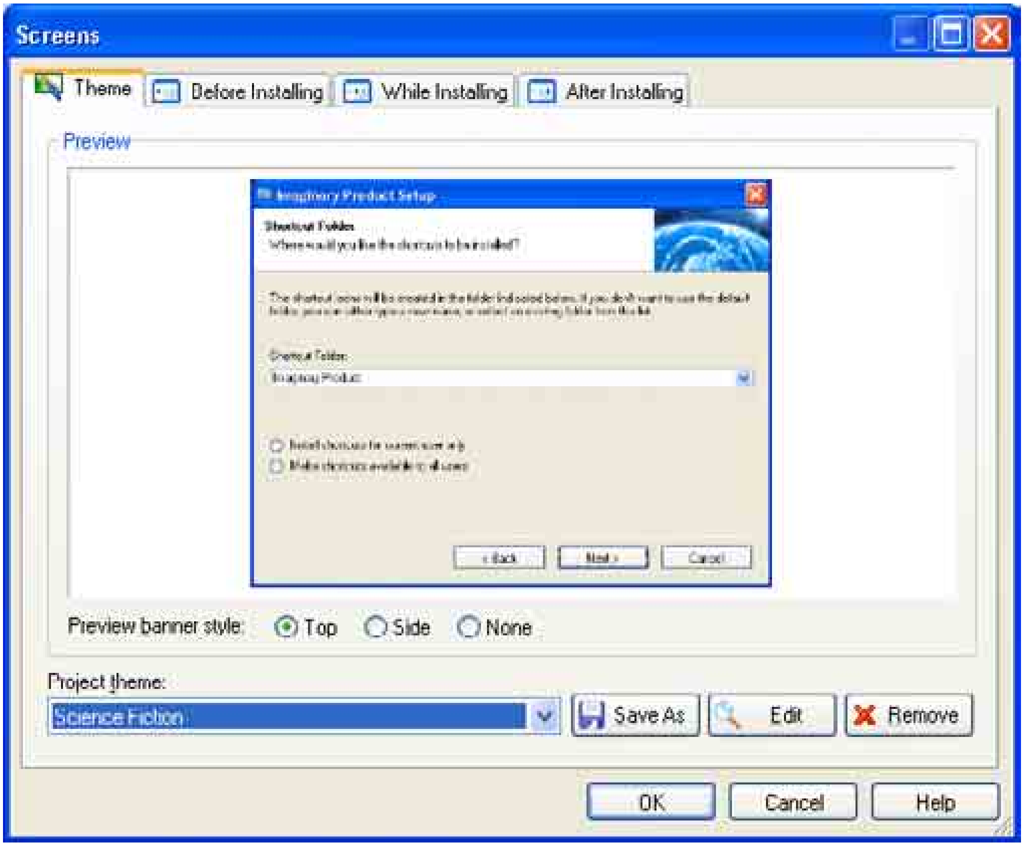
Choosing a Theme

You can choose a theme for your project on the theme tab of the installation screen manager, which you can access by choosing Project > Screens.

A list of the available themes will be found in the project theme drop down. Selecting a theme in this drop-down list will apply the theme to all screens in your installation and uninstallation.

For your convenience Setup Factory 7.0 provides a handy preview of the theme on the theme tab so that you will know what the theme looks like before you choose it.

As introduced above, themes affect the appearance of screens and their controls. For example, choosing a theme that colors all static text controls purple will result in all static text controls being purple, and choosing a theme that colors static text controls black will result in all static text controls being colored black.



Preview banner style: ● Top ○ Side ○ None

Project theme: Science Fiction

Creating a Custom Theme

Setup Factory 7.0 allows you to create your own custom themes. This provides you with an easy way to share the same custom look and feel between multiple projects.

Here is a brief step-by-step guide to help you in the creation of a custom theme.

1) Start a new project and save a copy of a pre-existing theme.

Start a new project by choosing File > New Project from the menu, then open up the theme settings by choosing Project > Screens making sure that the theme tab is selected.

The first step in creating a custom theme is to select a theme from the project theme drop down that you will use to base your new theme upon. If you cannot find a suitable theme, your best choice will be the default theme.

Once you have selected the theme that will be the basis for your new theme, use the Save As button in order to save a copy of the current theme.

2) Edit your new theme in the Theme Properties dialog and then press Ok to save your changes.

Make sure that your new theme is selected and press the edit button to bring up the theme properties dialog. Here you will be able to edit all of the properties of your theme. Once you have made all of your changes, simply press the Ok button and the changes to your theme will automatically be saved.

Now you have a working theme that will be available to you in all your Setup Factory 7.0 projects.

Note: If you are not happy with the changes made while editing your theme, simply click the Cancel button and your changes will not be saved.

Overriding Themes

As stated earlier, project themes affect every screen in your installation and uninstall. While in the vast majority of situations this is the desired effect, there may be a few instances where this is not exactly what you want. Fortunately, Setup Factory 7.0 allows you to override any or all of the theme settings on any of your screens.

As mentioned in the Screen Properties section, each screen has a style tab associated with its screen properties dialog. If you look at the Style tab you will notice that it looks identical to the Style tab on the theme properties dialog except that it has a checkbox in the top left corner labeled override project theme.

Enabling the override project theme option will enable the theme settings area and allow you to change any or all of the theme settings without affecting any of the other screens in your project.

Note: If you decide that you want to go back to the project theme on a screen where you have overridden the project theme, there is no need to re-create the screen. Simply go to the Style tab and uncheck the override project theme checkbox.

Other Options

There are a few other options in Setup Factory that relate to the user interface of your installation. These options may not be as important as screens or themes but just might provide the elements necessary to perfect your project's look and feel.

Taskbar Settings

The taskbar is the bar that runs across the bottom of all modern Windows operating systems beginning with the START button on the left. When a program is running, its icon and name will generally appear in a button in the taskbar.

Setup Factory 7.0 allows you to choose whether or not to show an icon in the task bar and to choose what that icon will be. Both of these settings can be found on the Advanced tab of the project settings: Project > Advanced Settings.

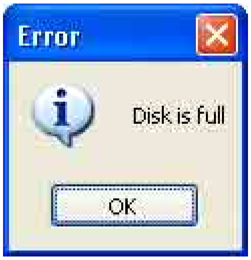
To hide the taskbar icon simply check the hide taskbar icon checkbox on the Advanced tab. To choose a custom icon simply check the use custom icon checkbox to turn on the custom icon setting and then click the browse button to locate the icon of your choice.

Note: If you configure your installation to be a silent install in the advanced settings then no taskbar entry will appear.

Actions

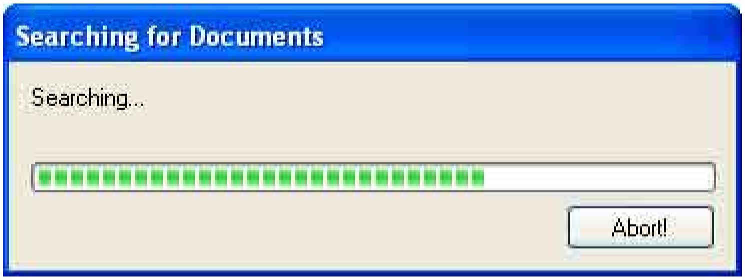
Some of the actions available to you in Setup Factory 7.0 are capable of showing user interface elements. These actions can be divided up into two main categories: Dialog actions and Status Dialog actions.

Dialog actions are used to show pop up dialogs to the end user. Examples include the Dialog.Message action that lets you display a message in a dialog, and the Dialog.TimedMessage action that lets you show a dialog with a message for specific amount of time.



A typical message dialog

Status dialogs are the other main user interface elements that are available to you through scripting. Status dialogs are mainly used to show progress or status during a lengthy event like an HTTP.Download action or a File.Find action.



A status dialog

Status dialogs are shown and configured using actions like StatusDlg.SetMessage, StatusDlg.ShowProgressMeter, and StatusDlg.Show.

Note: It is generally recommended that progress be shown in a more integrated manner using a progress screen, however there are situations where a status dialog may be more appropriate.

Note: For more information on the Dialog and StatusDlg actions, please see the Setup Factory 7.0 help file.

Chapter 4:

Actions, Scripts and Plugins

Actions are one of the most important features in Setup Factory 7.0. They are what you use to accomplish the specific tasks that make your installation unique. Each action is a specific command that tells your installer to do something, such as retrieving a value from the Registry.

Actions also allow your installation to react to different situations in different ways. Does the user already have the evaluation version of your software installed? Is an Internet connection available? You can use actions to answer these types of questions and have your installer respond accordingly.

Scripts are essentially sequences of actions that work together to perform a specific task. Plugins allow you to extend the built-in actions with additional libraries of commands.

Together, actions, scripts and plugins allow you to extend the default functionality of a Setup Factory 7.0 installation with virtually limitless possibilities.

4

In This Chapter

In this chapter, you'll learn about:

- Actions—what they are and what they're good for
- The action editor (including features such as syntax highlighting, intellisense, quickhelp, and context-sensitive help)
- Project, screen, and uninstall events
- Adding and removing actions
- Scripting basics, such as using a variable, adding an if statement, testing a numeric value, using a for loop, and creating functions
- Global functions
- Plugins
- External script files

What are Actions?

Actions are what you use in Setup Factory 7.0 when you want to get something done. Actions are specialized commands that your installer can perform at runtime. Each action is a short text instruction that tells the installer to do something—whether it's to open a document, download a file from the web, create a shortcut, or modify a registry key.

Actions are grouped into categories like "File" and "Registry." The category and the name of the command are joined by a decimal point, or "dot," like so: File.Run, Registry.GetValue. The text "File.Run" essentially tells Setup Factory that you want to perform a "Run" command from the "File" category…a.k.a. the "File.Run" action.

Tip: The period in an action name is either pronounced "dot," as in "File-dot-Open," or it isn't pronounced at all, as in "File Open."

By default Setup Factory 7.0 handles the most common installation needs without you ever having to add additional actions. As a result you will only need to use actions when you want to supplement the basic behavior and handle advanced installation tasks. Actions let you customize what Setup Factory 7.0 does for you automatically so that you can meet your installation needs exactly.

In other words, actions let you customize your installation to perform any task that it does not already perform by default. You can use actions to call functions from DLLs, submit data to a website, start and stop programs, get the amount of free space on a drive, and much more.

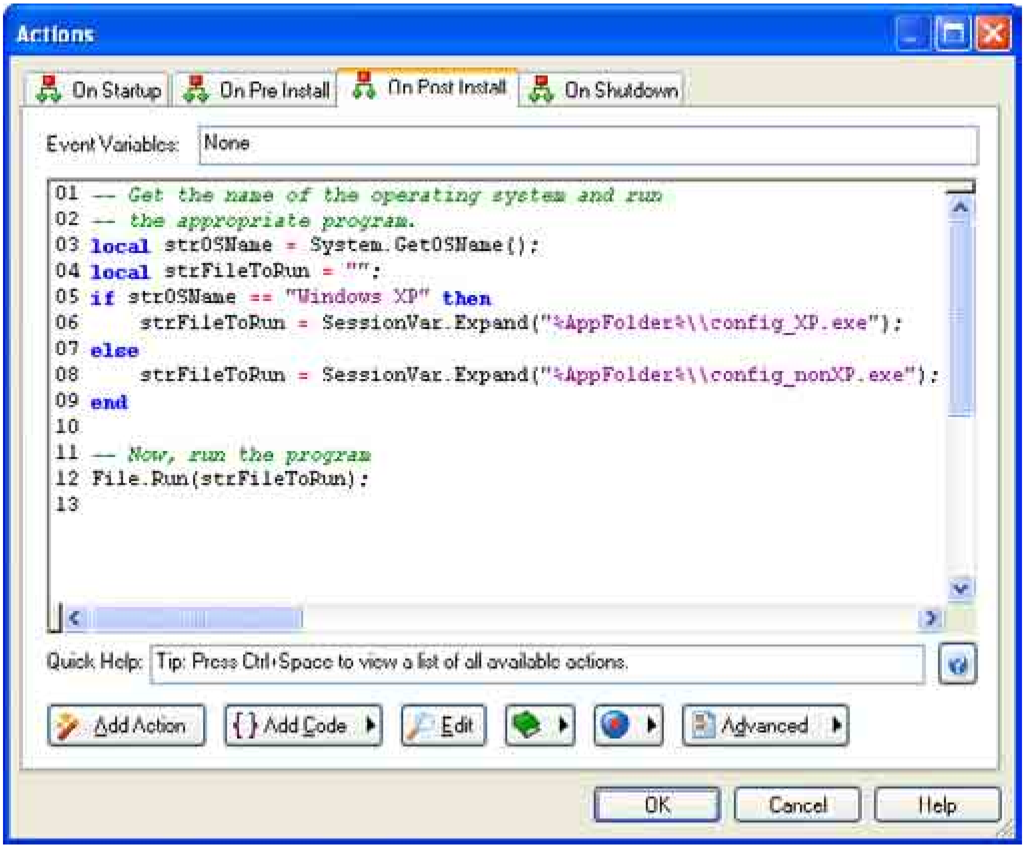
Note: You can find a complete list of actions in the Setup Factory help file under Program Reference > Actions.

It's worth noting that the pre-made screens in Setup Factory 7.0 actually use actions to accomplish their tasks! As a result, if a screen does not accomplish some additional task and you would like to modify its build-in functionality, you can do so simply by editing the screen's actions. The power is in your hands!

The Action Editor

The action editor is where you create and edit your actions in the Setup Factory 7.0 design environment. In general you can expect to find an action editor for every event in Setup Factory.

Essentially the action editor functions like a text editor by allowing you to type the actions that you want to use.



Event Variables: None

```
-- Get the name of the operating system and run
-- the appropriate program.
local strOSName = System.GetOSName();
local strFileToRun = "";
if strOSName == "Windows XP" then
    strFileToRun = SessionVar.Expand("%AppFolder%\\config_XP.exe");
else
    strFileToRun = SessionVar.Expand("%AppFolder%\\config_nonXP.exe");
end

-- Now, run the program
File.Run(strFileToRun);

```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

However, while it may function like a text editor, the action editor has powerful features that make it closer to a full-fledged programming environment. Some of these features include syntax highlighting, intellisense, quickhelp and context-sensitive help.

One of the most important features of the action editor is the *action wizard*. The action wizard provides an easy dialog-based way to select, create and edit actions without ever having to type a line of script.

The action editor gives you the best of both worlds: pure scripting capabilities for advanced users and programmers, and the easy-to-use action wizard interface for those who'd rather not use free form scripting.

Programming Features

As mentioned briefly above, the action editor provides some powerful features that make it a useful and accessible tool for programmers and non-programmers alike. Along with the action wizard (covered later under *Adding Actions*), the four most important features of the action editor are: syntax highlighting, intellisense, quickhelp and context-sensitive help.

Syntax Highlighting

Syntax highlighting colors text differently depending upon syntax. This allows you to identify script in the action editor as an operator, keyword, or comment with a quick glance.

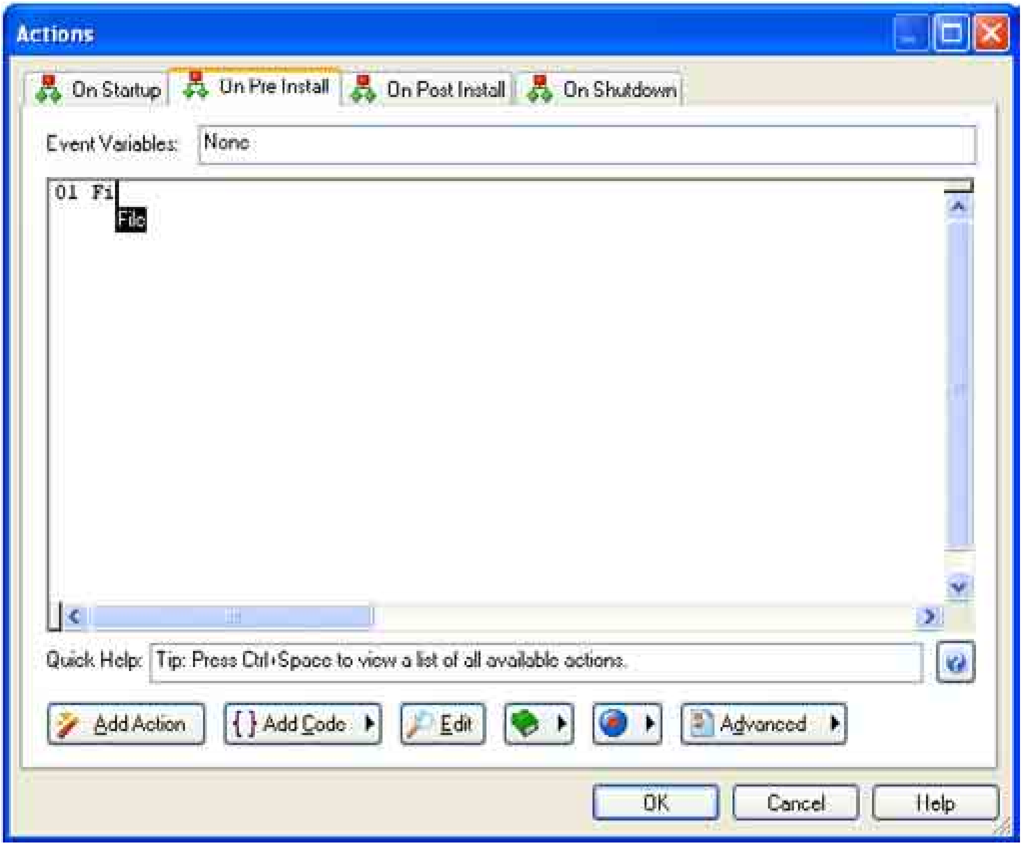
Note: You can customize the colors used for syntax highlighting via the action editor settings. The action editor settings are accessed via the advanced button: Advanced > Editor Settings.

Intellisense

Intellisense is a feature of advanced programming environments that refers to the surrounding script and the position of the cursor at a given moment to provide intelligent, contextual help to the programmer.

Intellisense is a term that has been used differently by different programs. In Setup Factory 7.0, intellisense manifests itself as two features: autocomplete, and the autocomplete dropdown.

Autocomplete is the editor's ability to automatically complete keywords for you when you press Tab. As you type the first few letters of a keyword into the action editor, a black tooltip will appear nearby displaying the whole keyword. This is the intellisense feature at work. Whenever you type something that the intellisense recognizes as a keyword, it will display its best guess at what you are typing in one of those little black tooltips. Whenever one of these tooltips is visible, you can press the Tab key to automatically type the rest of the word.

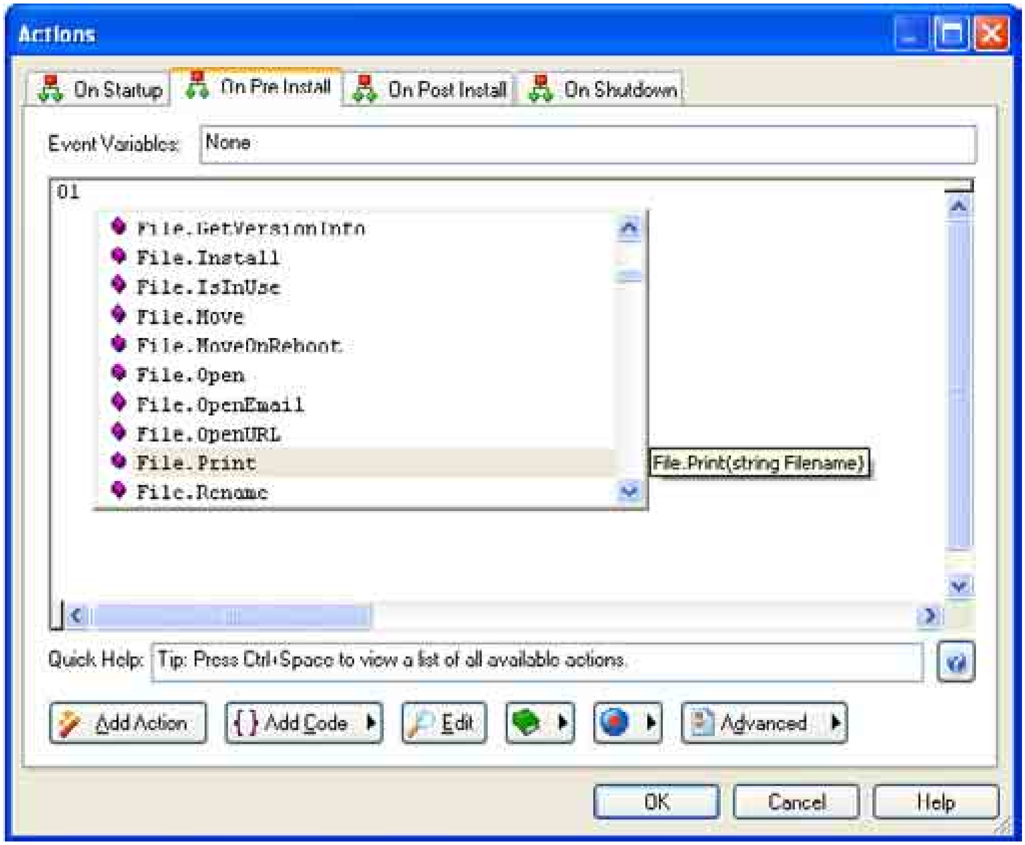


Event Variables: None

```
Fi
```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

Another feature of intellisense is the autocomplete dropdown. By pressing Ctrl+Space while your cursor is in the code window, a drop-down list will appear containing the names of all of actions, constants and global variables. You can choose one of the listed items and then press Tab or Enter to have it automatically typed.



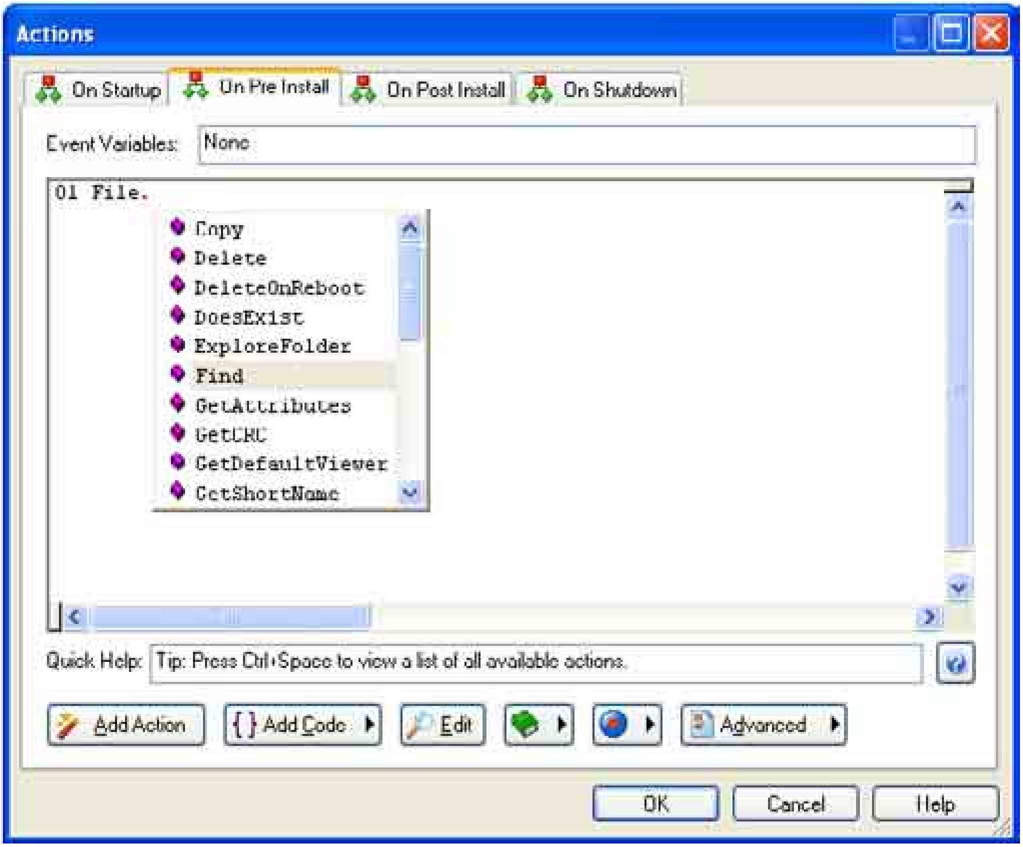
Event Variables: None

File.Print(string Filename)

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

Note: This dropdown cannot be accessed if your cursor is inside a set of quotes (a string).

The autocomplete dropdown is also available for completing action names after the category has been typed. For example, when you type a period after the word File, the intellisense recognizes what you've typed as the beginning of an action name and presents you with a drop-down list of all the actions that begin with "File."



Event Variables: None

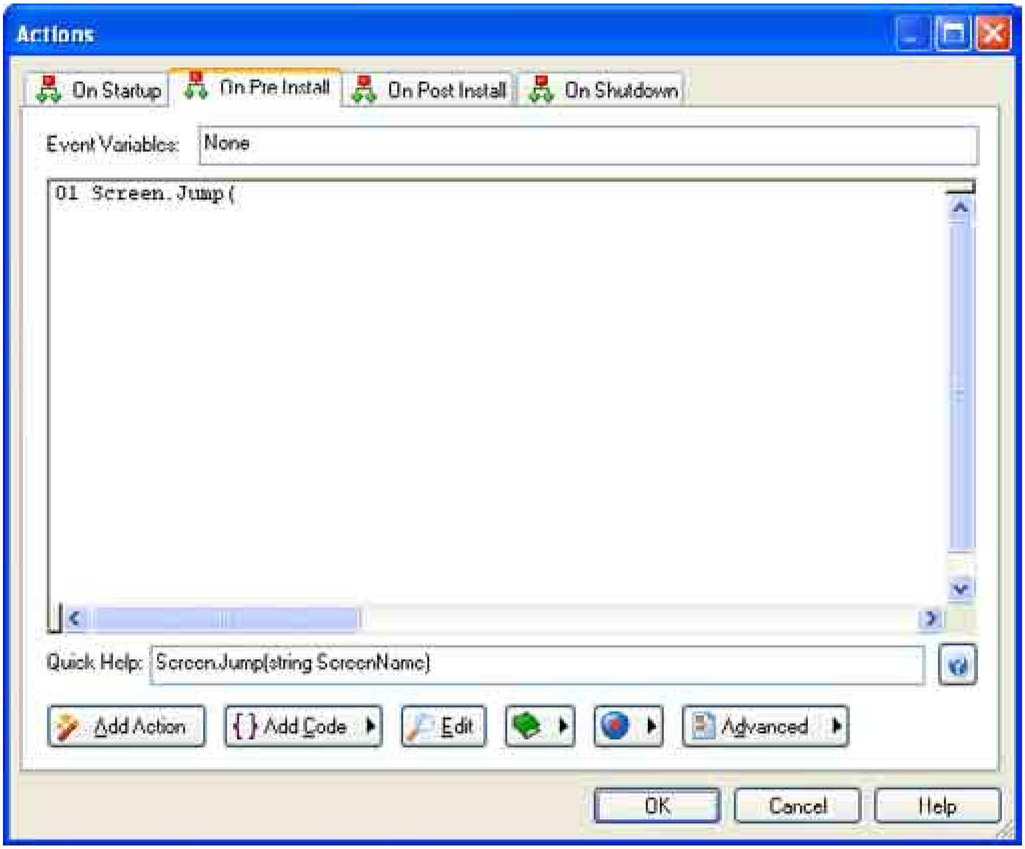
```
01 File.
```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

The word will automatically be typed for you if you choose it and then press Tab or Enter. However, you don’t have to make use of the dropdown list; if you prefer, you can continue typing the rest of the action manually.

Quickhelp

Once you’ve typed something that the action editor recognizes as the name of an action, the quickhelp feature appears near the bottom of the window.



```
Screen.Jump(
```

Quickhelp is essentially a “blueprint” for the action that lists the names of the action’s parameters and indicates what type of value is expected for each one. In the case of our Screen.Jump action, the quickhelp looks like this:

```
Screen.Jump(string ScreenName)
```

The quickhelp above indicates that the action Screen.Jump takes a single parameter called ScreenName, and that this parameter needs to be a string.

Remember, strings need to be quoted, so if we wanted to jump to a screen named Finish, the full action would have to be typed exactly like this:

```
Screen.Jump("Finish");
```

From this brief example, you can see how useful the quickhelp feature will be when you are working on your scripts.

Context Sensitive Help

Context sensitive help, as its name suggests, provides help for you based upon what you are currently doing. In the action editor, the context sensitive help lets you jump directly to the current action's topic help file.

For instance, if you are typing an action into the action editor and the quickhelp feature isn't giving you enough information (perhaps you would like to see an example), press the F1 key and the help file will open directly to that action's help topic.

Note: The context sensitive feature is only available when the action editor recognizes the action that the cursor is on. It is easy to know when this is the case since the action will appear in the quickhelp.

Tip: You can also click the Help button to trigger the context sensitive help feature.

Events

Events are simply things that can happen when your installation is running. For example, all screens have an On Preload event, which is triggered just before the screen is displayed. To make something happen just before the screen is displayed, you simply add an action to its On Preload event.

When an event is triggered, the actions (or "script") associated with that event will be executed. By adding actions to different events, you can control when those actions occur during the installation.

In Setup Factory 7.0 there is an action editor for every event. When you add an action in an action editor, you are adding that action to the event. At run time, when that event is triggered, the action that you added will be performed.

Every installation has four main project events: On Startup, On Pre Install, On Post Install, and On Shutdown. These events can be thought of as bookends to the main phases of the install. There are also project uninstall events, which are identical to the project events except that they occur during the uninstall. The remainder of the events in Setup Factory 7.0 are triggered by the screens.

Project Events

There are four project events. They are the main events or your installation and will be triggered before or after important stages of the installation. The project events can be found under Project > Actions.

The four project events are:

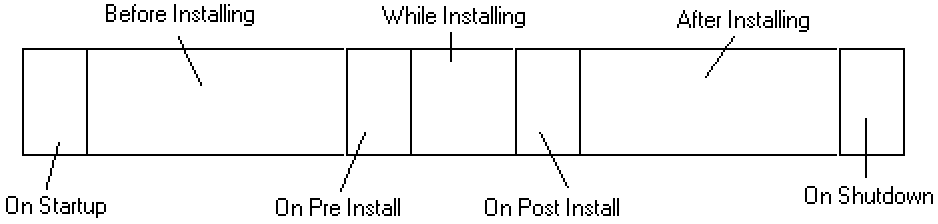
- On Startup
- On Pre Install
- On Post Install
- On Shutdown

The On Startup event is the first event triggered in Setup Factory 7.0 and thus occurs before any screens are displayed. This is a good place to perform pre-installation tasks, such as making sure the end user has a previous version of your installation installed, or that they haven't already run the installation once before.

The On Pre Install event is triggered just before the installation enters the install phase and begins installing files onto the end user's computer. This event is fired right after the last screen in the Before Installing screen stage is displayed.

The On Post Install event is triggered after the install phase of the installation has been completed, right before the first screen of the After Installing screen stage is displayed.

The On Shutdown event is the last event that will be triggered in your installation. It occurs after all screens in your installation have been displayed. It is your last chance to accomplish any action in your installation.



Screen Events

As covered in Chapter 3, the settings for each screen can be configured in the screen properties dialog. The screen events can be found on the Actions tab of the screen properties dialog.

Screen events are events that are triggered either by the controls on the screen or by the screen itself. Screen events are used for screen navigation (e.g. moving to the next screen), to change what is displayed on the screen (e.g. updating progress text), and to perform the task required of the screen (e.g. set the installation folder).

There are two main types of screens in Setup Factory 7.0: progress screens and non-progress screens. The majority of screens are non-progress screens, which basically means that these screens are not used to show progress.

Progress screens are used to show progress in your installation. They can be further broken down into two types of progress screens: those used in the Before Installing and After Installing screen stages, and those used in the While Installing stage. In general progress screens used in the Before and After Installing screen stages show progress that arises from actions. Progress screens that are used during the While Installing screen stage are used to show the progress of the main installation phase.

There are six events associated with non-progress screens:

- On Preload – triggered just before the screen is going to be displayed.
- On Back – when the back button is clicked.
- On Next – when the next button is clicked.
- On Cancel – when the cancel button is clicked.
- On Help – when the help button is clicked.
- On Ctrl Message – Triggered when a control on the screen fires a control message. Every time a control message event is fired, there will be event variables passed to the event to describe the message and which control fired it.

The On Ctrl Message event is where you will interact with the controls on the screens via script. For example, if you choose to add a Button screen to your setup, the On Ctrl Message event is where you will specify what each button does.

On the Buttons screen, each button will fire a control message when it has been clicked, basically saying “Hey I’ve been clicked!” You will then have a script in the

On Ctrl Message event that will check the event variables to see which button has been clicked and the corresponding actions will be performed.

There are four events associated with progress screens used in the Before and After Installing screen stages:

- On Preload – triggered just before the screen is going to be displayed.
- On Start – fired as soon as the screen is displayed. This is where you will put all of the actions that are to occur while the progress screen is visible. The actions placed here are what will cause the progress that the progress screen is going to display.
- On Finish – triggered as soon as all of the On Start's event actions have finished executing. This will usually be used to move to the next screen.
- On Cancel – triggered when the cancel button is pressed.

There are three events associated with progress screens used in the While Installing screen stage:

- On Preload – triggered just before the screen is going to be displayed.
- On Progress – fired as progress is made during the installation phase. Event variables are passed to this event to describe which stage is triggering the progress event and other information.
- On Cancel – triggered when the cancel button is pressed.

Uninstall Events

Uninstall events are identical to the project events except that they occur during the uninstallation. You can find the uninstall events in the Setup Factory 7.0 design environment under Uninstall > Actions.

The four uninstall events are:

- On Startup – the first event in the uninstallation, triggered when the uninstallation starts.
- On Pre Uninstall –right before your files are uninstalled from the end user's computer.

- On Post Uninstall – triggered right after all your files have been uninstalled from the end user's computer.
- On Shutdown – the final event in the uninstallation, fired right before the uninstallation finishes. This is your last chance to perform any actions during the uninstallation.

Note: For more information on the uninstall events, please read the project events section as though it were describing events occurring during the uninstall.

Adding Actions

In order to have an action performed when an event is triggered, you must add it to that event using the action editor.

Actions can either be typed directly, or you can use the Add Action button to start the action wizard. The action wizard is a dialog-based way for you to add actions in the action editor. It will guide you through the process of selecting your action and configuring its parameters.

Note: As you type an action, you can use the intellisense and code completion features to simplify this process.

Here is a brief example that shows how easy it is to add an action. It will explain how to add a Dialog.Message action to the On Startup event of your installation. The Dialog.Message action pops up a dialog with a message on it.

1) Start a new project and open the Actions dialog to the On Startup tab.

Start a new project and then bring up the Actions dialog by choosing Project > Actions. On the actions dialog select the On Startup tab so that you are editing the On Startup event.

2) Click the Add Action button. When the action wizard appears, switch to the Dialog application category and then click on the action called Dialog.Message.

The action wizard will walk you through the process of adding an action to the action editor. The first step is to choose a category using the drop-down list.

When you choose the "Dialog" category from the drop-down list, all of the actions in that category will appear in the list below.

To select an action from the list, just click on it. When you select an action in the list, a short description appears in the area below the list. In this description, the name of the action will appear in blue. You can click on this blue text to get more information about the action from the online help.

3) Click the Next button and configure the parameters for the Dialog.Message action.

Parameters are just values that get “passed” to an action. They tell the action what to do. For instance, in the case of our Dialog.Message action, the action needs to know *what* the title of the dialog and the message should be. The first parameter lets you specify the title of the dialog and the second parameter lets you specify the text that will be displayed on the dialog. For now the other parameters are not important but you should take some time to look at them and their options. By default Setup Factory 7.0 will fill most parameters with defaults.

For now change the title to:

```
"Setup Factory 7.0"
```

and the text to:

```
"Message from Chapter 4!"
```

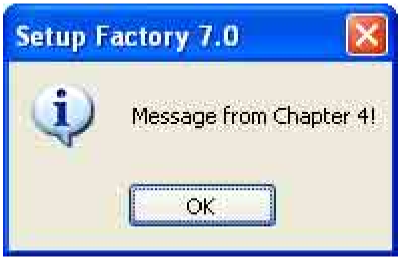
Note: Be sure to include the quotes on either side of both parameters. These are string parameters and the quotes are needed for Setup Factory 7.0 to properly interpret them.

Once you’ve set the action’s parameters, click the Finish button to close the action wizard. The Dialog.Message action will appear in the action editor. Note that the parameters you provided are listed between parentheses after the action’s name, in the same order they appeared in the Action wizard, separated by a comma.

4) Build and run your project. When the project starts you should see the dialog created by the Dialog.Message action.

Start the publish wizard by selecting Publish > Build from the menu, choose Web/Email/LAN as your distribution method and click the Next button. Build your installation to whichever folder you want and using whichever name you like. Once the build has completed successfully, make sure that you check the open output folder checkbox and then click the Finish button.

Once the folder where you built your installation appears, double-click on your setup executable to launch it. You should see the following dialog message appear:



Setup Factory 7.0

Message from Chapter 4!

Editing Actions

Often you will want to change a few of your actions' settings after you add them; to do this you need to edit the action. You can edit the action by typing directly into the action editor or by using the actions properties dialog.

The easiest way to bring up the actions properties dialog is by double-clicking on the action. If you prefer you can bring it up by placing the cursor in the action and then pressing the edit action button. (You can tell that the cursor is in an action when the actions function prototype appears in the quickhelp.)

Here is a quick example illustrating how to edit the Dialog.Message action that we created in the previous section.

1) Open the Actions dialog to the On Startup tab and bring up the Action Properties dialog.

Open the Actions dialog by choosing Project > Actions. Make sure that you are on the On Startup tab and that you see the Dialog.Message action created in the previous topic.

To edit the action, just double-click on it. Double-clicking on the action opens the Action Properties dialog, where you can modify the action's current parameters.

Tip: You can also edit the action's text directly in the action editor, just like you would edit text in a word processor. For example, you can select the text that you want to replace, and then simply type some new text in.

2) Change the Type parameter to be MB_OKCANCEL and the Icon parameter to MB_ICONNONE.

Click on the Type parameter, click the select button, and choose MB_OKCANCEL from the drop-down list.

Then click on the Icon parameter, click the select button, and choose MB_ICONNONE from the drop-down list.

This will add a cancel button to the dialog (MB_OKCANCEL) and get rid of the icon (MB_ICONNONE).

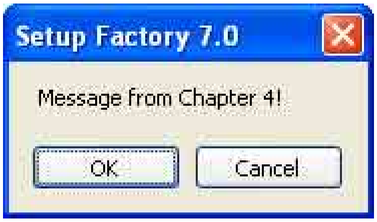
Finally, click OK to finish editing the action. Notice that the changes that you made now appear in the action editor.

Constants

SW_SHOWNORMAL is a *constant*. A constant is a name that represents a value, essentially becoming an "alias" for that value. Constants are often used to represent numeric values in parameters. It's easier to remember what effect SW_MAXIMIZE has than it is to remember what happens when you pass the number 3 to the action.

3) Build and run your project. When the project starts, you should see the dialog created by the Dialog.Message action.

When you run your setup, a dialog will pop up. The dialog is created by the Dialog.Message action on the On Startup event of your installation. Notice that there is no icon and that a cancel button is now part of the dialog.



Setup Factory 7.0

Message from Chapter 4!

Getting Help on Actions

You can get help on actions in a variety of different ways in Setup Factory 7.0. As previously mentioned, when using the action wizard some text will be displayed in the bottom of the dialog describing the current action or parameter selected. You can also select the text displayed in blue text in the action wizard to receive context sensitive help.

The same holds true for the action editor itself. The quickhelp will display the current action's function prototype. The help button to the right of the quickhelp will open the Help file directly to that action's help topic if there is a current action (as with the F1 key). The current action is simply the action that the cursor is in. If the cursor is not in an action, then there is no current action.

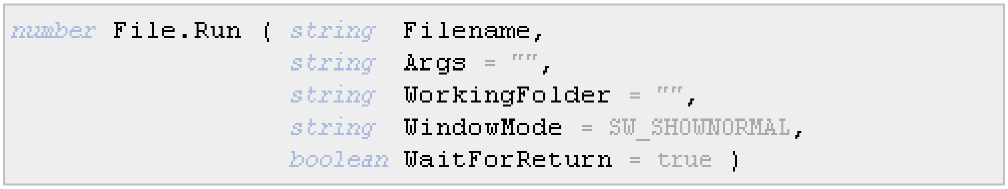
Note: For more information on the quickhelp, please see the Action Editor Features section above.

The Help button on the action editor can be clicked at any time to open the Setup Factory 7.0 help file. Here you can view more information on the event that the actions are being added to.

The help file contains detailed information for each action available to you in Setup Factory 7.0. It is divided into two topics for each action: Overview and Examples. The overview section provides detailed information about the action while the examples section provides one or more working examples of that action.

In the overview topic, the help file will provide you with the function prototype, which serves as a definition of the action, showing what (if anything) the action returns, and the parameters and their types.

A function prototype is the definition of the action. It defines the types of all of the parameters, the type of the return value (if any), and whether or not any of the parameters have defaults.



```
number File.Run ( string  Filename,
                  string  Args = "",
                  string  WorkingFolder = "",
                  string  WindowMode = SW_SHOWNORMAL,
                  boolean WaitForReturn = true )
```

The help file also describes each parameter in detail and what its purpose is.

If the action returns a value, then the help file will describe the return value and what it might be.

Scripting Basics

A script can be thought of as a sequence of actions. Scripting, therefore, is basically the creation of a sequence of actions. A script can contain one action or as many actions as you can type.

Although you can accomplish a lot in Setup Factory 7.0 without any scripting knowledge, even a little bit of scripting practice can make a big difference. You can accomplish far more in a project with a little bit of scripting than you ever could without it. Scripting opens the door to all sorts of advanced techniques, from actions that are only performed when specific conditions are met, to functions that you can define, name and then call from somewhere else.

Using a Variable

One of the most powerful features of scripting is the ability to make use of variables. Variables are essentially just "nicknames" or "placeholders" for values that may need to be modified or re-used in the future. Each variable is given a name that you can use to access its current value in your script.

We say that values are "assigned to" or "stored in" variables. If you picture a variable as a container that can hold a value, assigning a value to a variable is like "placing" that value into a container.

You place a value into a variable by assigning the value to it with an equals sign. For example, the following script assigns the value 10 to a variable called "amount."

```
amount = 10;
```

Note: The semi-colon at the end of the line tells Setup Factory where the end of the statement is. It acts as a *terminator*. (No relation to Arnold, though.) Although technically it's optional—Setup Factory can usually figure out where the end of the statement is on its own—it's a good idea to get in the habit of including it, to avoid any potential confusion.

You can change the value in a variable at any time by assigning a different value to it. (The new value simply replaces the old one.) For example, the following script assigns 45 to the amount variable, replacing the number 10:

```
amount = 45;
```

...and the following script assigns the string "Woohoo!" to the variable, replacing the number 45:

```
amount = "Woohoo!";
```

Note that you can easily replace a numeric value with a string in a variable. Having a number or a string in a variable doesn't “lock” it into only accepting that type of value—variables don’t care what kind of data they hold.

This ability to hold changeable information is what makes variables so useful.

Here is an example that demonstrates how variables work:

1) Start a new project and follow the steps from the Adding an action example on page 131.

Start a new Setup Factory 7.0 project by selecting File > New Project from the menu. If you have any unsaved changes in your current project, Setup Factory 7.0 will prompt you to save them. If the project wizard dialog comes up, simply hit cancel to get rid of it for now.

Once you have a brand new project, follow along with the three steps in the Adding an Action example.

2) In the action editor, replace the “Message from Chapter 4!” string with a variable named strMsg.

Just edit the script on the On Startup event so that it looks like this instead:

```
Dialog.Message("Setup Factory 7.0", strMsg);
```

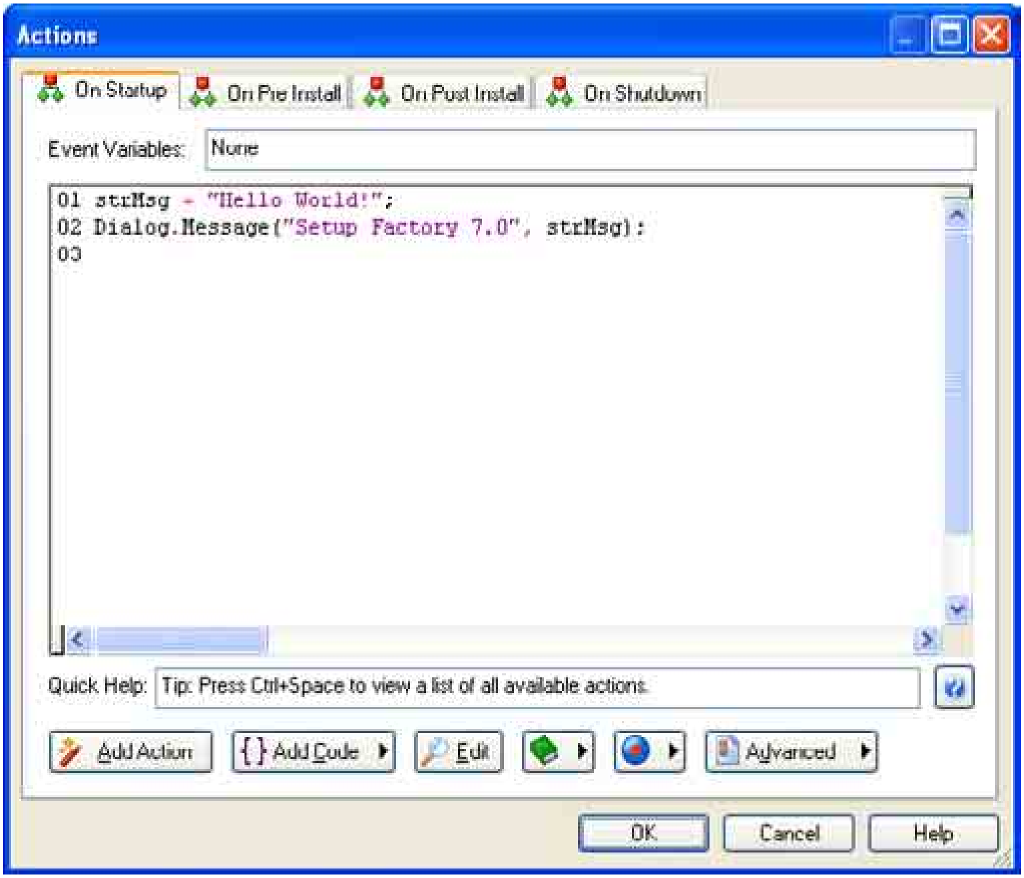
This will make the Dialog.Message action display the current value of the strMsg variable when it is performed. Before we try it out, though, we need to assign a value to that variable somewhere.

Note: A common practice among programmers is to give their variables names with prefixes that help them remember what the variables are supposed to contain. This practice is often referred to as Hungarian notation. One such prefix is “str,” which is used to indicate a variable that contains a string. Other common prefixes are “n” for a numeric value (e.g. nCount, nTotal) and “b” for a Boolean true/false value (e.g. bLoaded, bReadyToStart).

3) Insert the following text on the first line, before the Dialog.Message action:

```
strMsg = "Hello World!";
```

Remember to press Enter at the end of the line to move the Dialog.Message action to the next line. The script should look like this when you’re done:



Event Variables: None

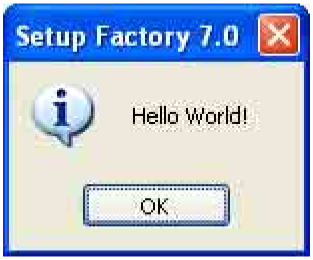
```
strMsg = "Hello World!";
Dialog.Message("Setup Factory 7.0", strMsg);
```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

This will assign the string “Hello World!” to the variable named strMsg before the Dialog.Message action is executed.

The On Startup event is triggered as soon as the installation begins, just before the first screen is displayed. This makes it a good place to put any initialization script, such as setting up default values or preparing variables that will be used in the installation. Another good location is the Global Functions section that will be described later.

4) Build and run your project. When the project starts, you should see the dialog created by the Dialog.Message action.



Hello World!

Note that the variable's name, strMsg, is nowhere to be found. Instead, the value that is currently in the variable is displayed. In this case, it's the value that was assigned to the variable in the page's On Preload event.

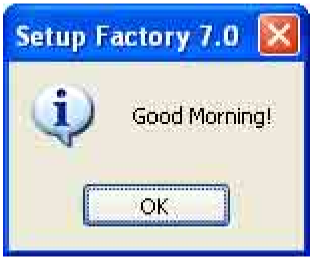
5) Exit the installation. In the action editor, change the value that is assigned to the strMsg variable to *"Good Morning!"*.

Edit the On Startup script so it looks like this instead:

```
strMsg = "Good Morning!";
Dialog.Message("Setup Factory 7.0", strMsg);
```

6) Build and run your project. When the project starts, you should see the dialog created by the Dialog.Message action.

This time, the message looks like this:



Good Morning!

As you can see, you've just changed the message without even touching the Dialog.Message action.

Now imagine if you had the same or similar Dialog.Message actions on fifty different events throughout your project, and you decided you wanted to change the text. With variables and a little planning, such changes are a piece of cake.

Adding an If Statement

The if statement gives your scripts the ability to make decisions, and do one thing or another in different circumstances.

Each if statement consists of a *condition* (or “test”), followed by a *block* of script that will only be performed if the condition is found to be true.

The basic syntax is:

```
if condition then
        do something here
end
```

For example:

```
if age < 18 then
    Dialog.Message("Sorry!", "You must be 18 to access this CD.");
    Application.Exit();
end
```

The above script checks to see if the value in a variable named “age” is less than 18. If it is, it puts up a message saying "You must be 18 to access this CD," and then immediately exits from the application.

For example, if we set age to 17 first:

```
age = 17;
if age < 18 then
    Dialog.Message("Sorry!", "You must be 18 to access this CD.");
    Application.Exit();
end
```

...the block of script between the “then” and “end” keywords will be performed, because 17 is less than 18. In this case, we say that the if statement’s condition “passed.”

However, if we set age to 20:

```
age = 20;
if age < 18 then
    Dialog.Message("Sorry!", "You must be 18 to access this CD.");
    Application.Exit();
end
```

...the block of script between the “then” and the “end” isn’t performed, because 20 isn’t less than 18. This time, we say that the if statement’s condition “failed.”

Note: An if statement's condition can be any expression that evaluates to true or false.

Let's modify the script on our project’s On Startup event to only display the message if it is “Hello world!”

1) Open the On Startup event and edit the script to enclose the Dialog.Message action in an if statement, like this:

```
strMsg = "Good Morning!";
if strMsg == "Hello World!" then
    Dialog.Message("Setup Factory 7.0", strMsg);
end
```

The double equals compares for absolute equality, so this if statement's condition will only be true if strMsg contains "Hello World!" with the exact same capitalization and spelling.

Tip: An easy way to add an if statement on the action editor is to highlight the line of text that you want to put “inside” the if, click the Add Code button, and then choose “if statement” from the menu. An if statement “template” will be added around the line that you highlighted. You can then edit the template to fit your needs.

2) Press F7 to build your project. When the build is complete, launch the setup to trigger the script.

This time nothing happens because strMsg is still set to "Good Morning!" in the first line of the On Startup event. "Good Morning!" doesn’t equal "Hello World!" so the if condition fails, and the block of code between the “then” and “end” keywords is skipped entirely.

3) Exit from the setup. Edit the On Startup script to change the == to ~=.

The script should look like this when it’s done:

```
strMsg = "Good Morning!";
if strMsg ~= "Hello World!" then
    Dialog.Message("Setup Factory 7.0", strMsg);
end
```

The tilde equals (~=) compares for inequality (does not equal), so this if statement’s condition will only be true if strMsg contains anything *but* "Hello World!" with the exact same capitalization and spelling.

4) Build and execute the project.

This time, because strMsg contains "Good Morning!", which is definitely not equal to "Hello World!", the message will appear.

The == and ~= operators are fine when you want to check strings for an exact match. But what if you’re not sure of the capitalization? What if the variable contains a string that the user typed in, and you don’t care if they capitalized everything correctly?

One solution is to use an old programmer’s trick: just convert the contents of the unknown string to all lowercase (or all uppercase), and do the same to the string you want to match.

Let's modify our script to ask the user for a message, and then display what they typed, but only if they typed the words "hello world" followed by an exclamation mark.

5) Exit from the installation. Edit the project’s On Startup script to look like this:

```
strMsg = Dialog.Input("", "Enter your message:");
if String.Upper(strMsg) == "HELLO WORLD!" then
    Dialog.Message("Setup Factory 7.0", strMsg);
else
    Dialog.Message("Um...", "You didn't type Hello World!");
end
```

The first line uses a Dialog.Input action to pop up a message dialog with an input field that the user can type into. Whatever the user types is then assigned to the strMsg variable.

In the if statement's condition, we used a String.Upper action to convert the contents of strMsg to all uppercase characters, and then compare that to "HELLO WORLD!".

We've added an "else" keyword after the first Dialog.Message action. It basically divides the if statement into two parts: the "then" part, that only happens if the condition is true; and the "else" part, that only happens if the condition is false. In this case, the else part uses a Dialog.Message action to tell the user that they didn't type the right message.

6) Build and run the setup. Try typing different messages into the input field by closing and re-running the setup.

Depending on what you type, you'll either see the "hello world!" message, or "You didn't type Hello World!".

Note that if you type some variation of "hello world!", such as "hello world!", or "hEllO WorlD!", the capitalization that you type will be preserved in the message that appears. Even though we used a String.Upper action to convert the message string to all uppercase letters in the if statement's condition, the actual contents of the variable remain unchanged. When the String.Upper action converts a value to all uppercase letters, it doesn't change the value in the variable...it just retrieves it, converts it, and passes it along.

Now you know how to compare two strings without being picky about capitalization.

Note: Programmers call this a *case-insensitive* comparison, whereas a comparison that only matches perfect capitalization is considered *case-sensitive*.

Tip: You can also use a String.CompareNoCase action to perform a case-insensitive comparison.

Testing a Numeric Value

Another common use of the if statement is to test a numeric value to see if it has reached a certain amount. To demonstrate this, let's create another script that will stop your installation from running after it has been run on the same system more then 5 times.

1) Add the following script to a new project’s On Startup event:

```
nRuns = Application.LoadValue("SUF70UsersGuide", "Chapter4");
if nRuns == "" then
    nRuns = 0;
end

nRuns = nRuns + 1;

Application.SaveValue("SUF70UsersGuide", "Chapter4", nRuns);

if nRuns>5 then
    Dialog.Message("Sorry!", "Setup has been run: ".. nRuns
                   .. " times and will exit");
    Application.Exit();
end
```

The first line tries to read an application value using the Application.LoadValue action. It stores the result in a variable named nRuns.

The second line tests the return value of the Application.LoadValue action. If a blank string has been returned ("") then this is the first time that the setup has been executed on this computer. As a result the third line then assigns a value of 0 to nRuns. The fourth line simply completes the if statement.

The fifth line adds 1 to the current value contained in the variable nRuns, and the sixth line saves that number using the Application.SaveValue action.

The rest of the script is just an if statement that tests whether the value of nRuns is greater than 5, displays a message when that’s true, and then exits the installation.

We used the concatenation operator to join the current value of nRuns to the end of the string "Setup has been run: " and then used them again to add the string " times and will exit" to the end. Note that this string has a space at the end of it, so there will be a space between the colon and the value of nRuns. (The resulting string just looks better that way.)

The concatenation operator consists of two periods, or dots (..). In fact, it’s often referred to as the “dot-dot” operator. It is used to join two strings together to form a new, combined string. It’s kind of like string glue.

Note: Technically, the value inside nRuns isn't a string—it's a number. That doesn't really matter, though. When you're doing concatenation, Setup Factory 7.0 will automatically convert a number into the equivalent string, as required.

2) Build the project and then run it six times.

Each time you run the project the value of nRuns will be loaded, incremented by one, and then saved again. After you have run the setup six times (and all times after that) the message will appear and the installation will stop.

There you go. You've just created a pretty sophisticated little program.

Using a For Loop

Sometimes it's helpful to be able to repeat a bunch of actions several times, without having to type them over and over and over again. One way to accomplish this is by using a *for* loop.

The basic syntax of the for loop is:

```
for variable = start,end,step do
        do something here
end
```

For example:

```
for n = 10,100,10 do
    Dialog.Message("", n);
end
```

The above script simply counts from 10 to 100, going up by 10 at a time, displaying the current value of the variable n in a dialog message box. This accomplishes the same thing as typing:

```
Dialog.Message("", 10);
Dialog.Message("", 20);
Dialog.Message("", 30);
Dialog.Message("", 40);
Dialog.Message("", 50);
Dialog.Message("", 60);
Dialog.Message("", 70);
Dialog.Message("", 80);
Dialog.Message("", 90);
Dialog.Message("", 100);
```

Obviously, the for loop makes repeating similar actions much easier.

Note: If the step is missing from the for loop, it will default to 1.

Let's use a simple for loop to add all of the digits between 1 and 100, and display the result.

1) Start a new project and add the following script to the project's On Startup event:

```
n = 0;
for i = 1, 100 do
    n = n + i;
end

Dialog.Message("", "The sum of all the digits is: " .. n);
```

The first line creates a variable called n and sets it to 0. The next line tells the for loop to count from 1 to 100, storing the current "count" in a variable named i.

During each "pass" through the loop, the script between the "do" and the "end" will be performed. In this case, this consists of a single line that adds the current values of n and i together, and then stores the result back into n. In other words, it adds i to the current value of n.

This for loop is the same as typing out:

```
n = n + 1;
n = n + 2;
n = n + 3;
n = n + 4;
```

...all the way up to 100.

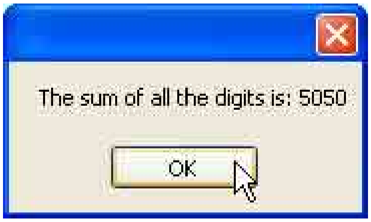
The last line of our script displays only the result of the calculation to the user in a dialog message box.

Tip: You can use the Add Code button to insert an example for loop, complete with comments explaining the syntax. You can then edit the example to fit your own needs.

2) Build and run the setup.

When you run the setup, the for loop will blaze through the calculation 100 times, and then the Dialog.Message action will display the result.

It all happens *very* fast.



The sum of all the digits is: 5050

3) Exit the setup.

You probably won't find yourself using for loops as often as you'll use if statements, but it's definitely worth knowing how to use them. When you need to repeat steps, they can save you a lot of effort.

Of course, when it comes to saving you effort, the real champions are functions.

Creating Functions

A function is just a portion of script that you can define, name and then call from somewhere else.

You can use functions to avoid repeating the same script in different places, essentially creating your own custom "actions"—complete with parameters and return values if you want.

In general, functions are defined like this:

```
function function_name (arguments)
        function script here
        return return_value;
end
```

The "function" keyword tells Setup Factory 7.0 that what follows is a function definition. The *function_name* is simply a unique name for the function. The *arguments* part is the list of parameters that can be passed to the function when it is called. It's essentially a list of variable names that will receive the values that are passed. (The resulting variables are local to the function, and only have meaning within it.) A function can have as many arguments as you want, even none at all.

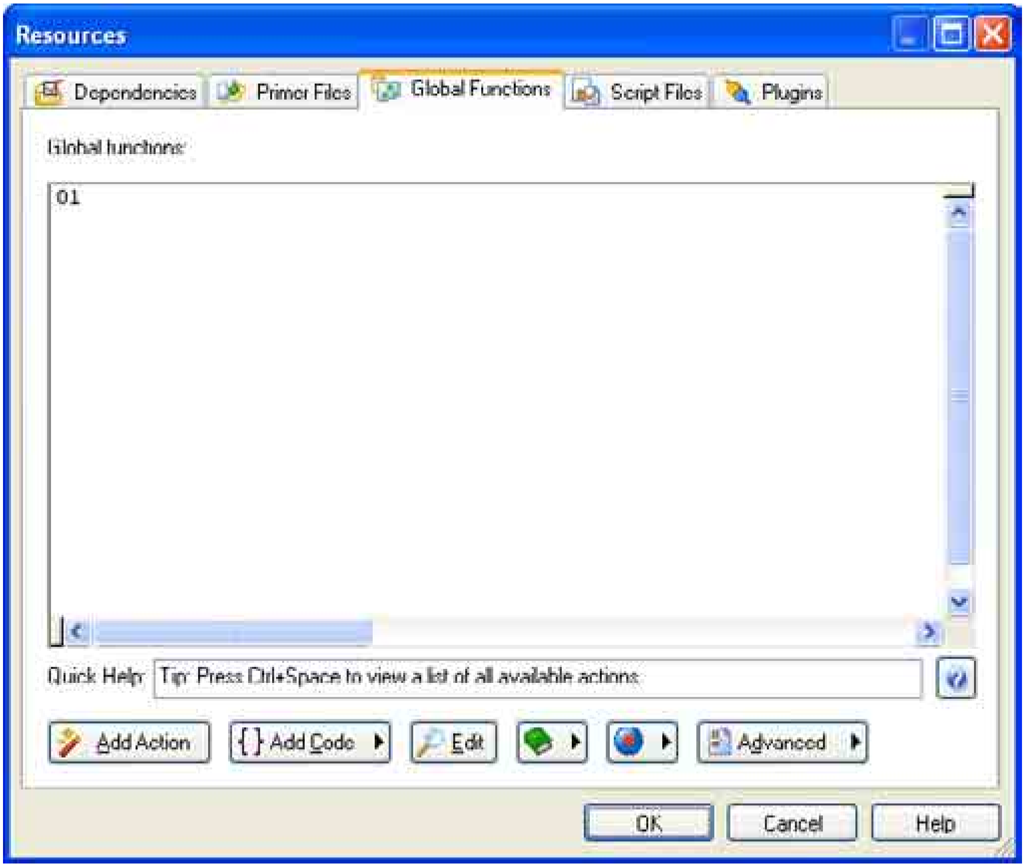
The "return" keyword tells the function to return one or more values back to the script that called it.

The easiest way to learn about functions is to try some examples, so let's dive right in.

1) Start a new project and then choose Resources > Global Functions.

This opens the resources dialog to the global functions tab.

The global functions section is a convenient place to put any functions or variables that you want to make available throughout your project. Any script that you add to this section will be performed when your application is launched, right before the project's On Startup event is triggered.



2) Type the following script into the Global Functions tab, then click OK to close the dialog:

```
function SayHello(name)
    Dialog.Message("", "Hello " .. name);
end
```

This script defines a function named SayHello that takes a single argument (which we've named "name") and displays a simple message.

Note that this only *defines* the function. When this script is performed, it will "load" the function into memory, but it won't actually display the message until the function is called.

Once you've entered the function definition, click OK to close the Global Functions dialog.

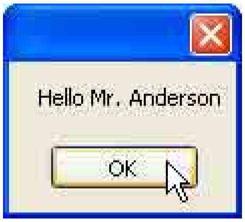
3) On the project's On Startup event add the following script:

```
SayHello("Mr. Anderson");
```

This script *calls* the SayHello function that we defined on the global functions section, passing the string "Mr. Anderson" as the value for the function's "name" parameter.

4) Build and run the setup.

When you run the setup, the script on the On Startup event calls the SayHello function, which then displays its message.



Hello Mr. Anderson

Note that the SayHello function was able to use the string that we passed to it as the message it displayed.

5) Exit the setup. Choose Resources > Global Functions, and add the following script below the SayHello function:

```
function GetName()
    local name = Dialog.Input("", "What is your name:");
    return name;
end
```

When you’re done, click OK to close the dialog.

This script defines a function called GetName that does not take any parameters. The first line inside the function uses a Dialog.Input action to display a message dialog with an input field on it asking the user to type in their name. The value returned from this action (i.e., the text that the user entered) is then stored in a local variable called name.

The “local” keyword makes the variable only exist inside this function. It’s essentially like saying, “for the rest of this function, whenever I use ‘name’ I’m referring to a temporary local variable, and not any global one.” Using local variables inside functions is a good idea—it prevents you from changing the value of a global variable without meaning to. Of course, there are times when you *want* to change the value of a global variable, in which case you just won’t use the “local” keyword the first time you assign anything to the variable.

The second line inside the function returns the current value of the local “name” variable to the script that called the function.

Tip: We could actually make this function’s script fit on a single line, by getting rid of the variable completely. Instead of storing the return value from the Dialog.Input action in a variable, and then returning the contents of that variable, we could just put those two statements together, like so:

```
function GetName()
    return Dialog.Input("", "What is your name:");
end
```

This would make the GetName function return the value that was returned from the Dialog.Input action, without storing it in a variable first.

6) Edit the script in the project's On Startup event so it looks like this instead:

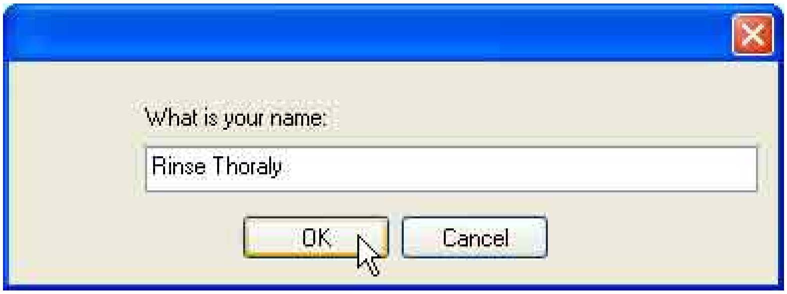
```
strName = GetName();
SayHello(strName);
```

The first line calls our GetName function to ask the user for their name, and then stores the value returned from GetName in a variable called strName.

The second line passes the value in strName to our SayHello function.

7) Build and launch the setup.

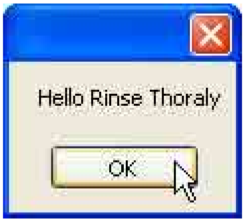
When the setup begins the On Startup script will automatically be executed and the input dialog will appear, asking you to enter your name.



What is your name:

Rinse Thoraly

After you type in your name and click OK (or press Enter), a second dialog box will appear, greeting you by the name you entered.



Hello Rinse Thoraly

Pretty neat, huh?

8) Exit the setup. Edit the script in the project's On Startup event so it looks like this:

```
SayHello(GetName());
```

This version of the script does away with the strName variable altogether. Instead, it uses the return value from our GetName function as the argument for the SayHello function.

In other words, it passes the GetName function's return value directly to the SayHello function.

Whenever a function returns a value, you can use a call to the function in the same way you would use the value, or a variable containing the value. This allows you to use the return value from a function without having to come up with a unique name for a temporary variable.

9) Build and launch the setup again to try out the script again. When you're done, exit the setup.

The script should work exactly the same as before: you'll be asked for your name, and then greeted with it.

This is just a simple example, but it should give you an idea of what an incredibly powerful feature functions are. With them, you can condense large pieces of script into simple function calls that are much easier to type and give you a single, central location to make changes to that script. They also let you create flexible "subroutines" that accept different parameters and return results, just like the built-in Setup Factory 7.0 actions.

And despite all that power, they are really quite simple to use.

Action Resources

There are three additional resources at your disposal that can be useful when you are working with actions in Setup Factory 7.0: Global Functions, Plugins, and Script Files.

Global Functions

The Global Functions resource in Setup Factory 7.0 can be found in the design environment under Resources > Global Functions. The Global Functions resource is an action editor that will let you enter script.

The script in the global functions section will be loaded into the installation before the On Startup event is executed. This makes it a great place to create any functions that you want to have available during your uninstall and a great place to initialize any global variables.

Note that any script in the Global Functions resource will be executed, so it is generally best to only use it for variable initialization and function declarations. It is a good idea to place all other scripts on events within your setup.

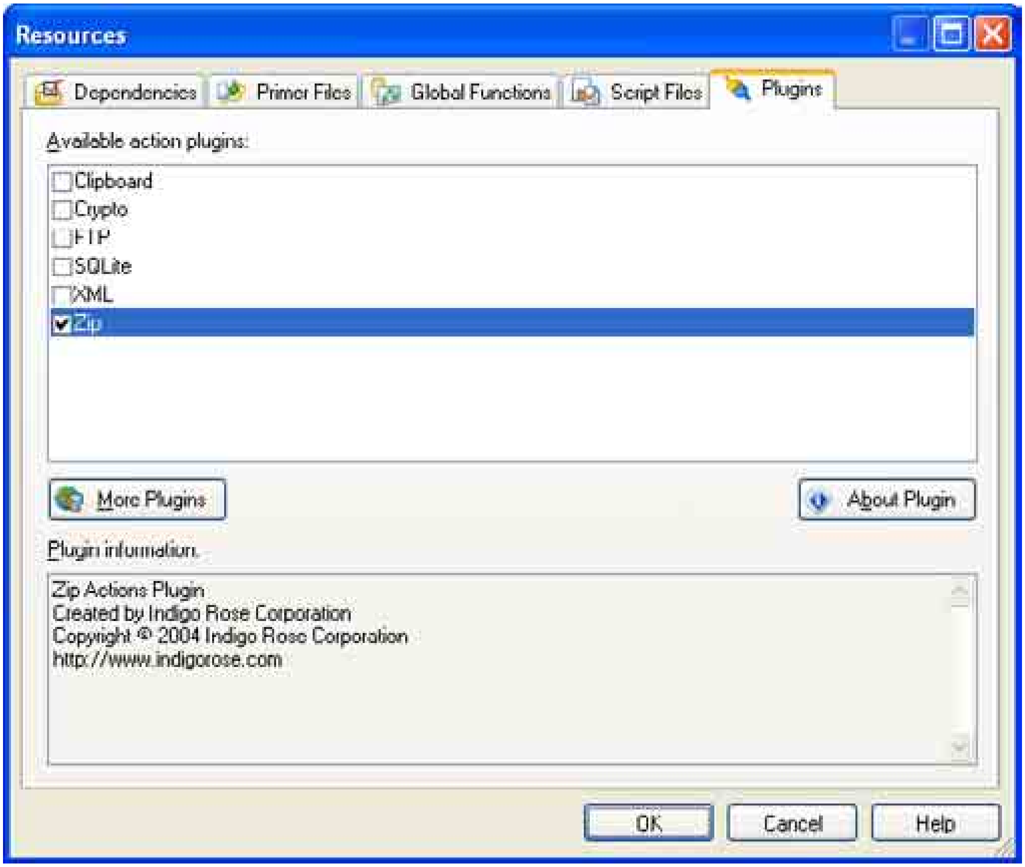
Note: Since the global function section is normally used to create global functions and initialize global variables, the script that is contained there is sometimes referred to as global script.

Plugins

Plugins are actions that are external to the Setup Factory 7.0 program. They are independently developed and distributed and can be integrated into your projects to extend their functionality. Some plugins may be developed by Indigo Rose, while others may be developed by third parties.

You can refer to the plugin's documentation for information on what features it adds and how to use them (click the About Plugin button to bring up the About Plugin dialog and then click the Plugin Help button).

Plugins can be added or removed from your project on the Plugins tab, which can be accessed by choosing Resources > Plugins in the Setup Factory 7.0 design environment.



□ Clipboard
□ Crypto
□ FTP
□ SQLite
□ XML
☑ Zip

Zip Actions Plugin
Created by Indigo Rose Corporation
Copyright © 2004 Indigo Rose Corporation
http://www.indigorose.com

Note: Any action plugins that are located in the Includes\Plugins folder within the Setup Factory 7.0 program directory will be available on the Plugins tab.

The More Plugins button is an easy way to see what plugins are currently available on the IndigoRose website. Pressing this button will bring you to the Setup Factory 7.0 plugins area of the IndigoRose website.

If a plugin has a check in the checkbox beside its name on the plugins tab, it will be included in your setup. Since there is some overhead in terms of file size, it is recommended that you only include plugins that are needed by your installer. If you do not check the checkbox beside a plugin, it will not be included in the installation and will not take up any extra space.

Once you have included the plugin in your project all of its actions will be available to you in the action editor and action wizard.

Note: Help for plugin actions can be accessed on the action editor in the same way that you would access help for built-in actions.

Script Files

Script files are external text files that contain Lua script usually with the .lua file extension. Script files can be added to your project on the script files tab of the resources dialog which can be found under Resources > Script Files.

Note: All action editors in Setup Factory 7.0 have the ability to export their script via the advanced button. Select Advanced > Save to save the script as an external file.

Script files are very similar to the global functions section of Setup Factory 7.0 except that instead of the script being kept in the Global Functions resource, it is stored externally to your project on a text file.

Script files are very useful if you share important and complex code between a variety of different projects.

Here is a quick comparison of the differences between sharing script between projects using the Global Functions resource versus script files.

If you have some script that you want to share between many projects using the Global Functions resource, you have to copy and paste the script into each setup that needs it. Now each project has an exact copy of the script.

The method works fine until you discover a bug or want to change some of the code. Since each project contains a copy of the script, you will have to edit the script in each project in order to be sure that it is using the correct code.

If you were to use an external script file, you need to develop a working piece of script and then get that script into a text file. Then, include the script file in each project that needs it. Using this script file method, each project does not contain a *copy* of the script; rather, each project *references* the *same* piece of script.

If you find an error or want to change any of the script, you do not have to edit the script in each project; you simply have to edit the script in the text file. Since each project references the same script file, you know that the next time you build a project, it will be using the correct script.

Essentially, with the Global Functions resource, you have to maintain the script for each project that uses it. On the other hand, the script file method allows you to maintain your script in a single location, the script file. For this reason, it's a good idea to use script files when you want to share global script between several projects.

Chapter 5:

Session Variables

When designing an installer, it’s often desirable to make parts of it dynamic. During the development and maintenance stages, there is usually some information that will need to change, such as your product's version number. Being able to quickly propagate such changes throughout the project can greatly decrease development time and costs. If you can predict such changes ahead of time, you can minimize their impact by using session variables as “placeholders” for the data that is likely to change.

There are also some things that are almost guaranteed to be different from one system to the next, such as the path to the user’s My Documents folder, or what drive they installed Windows on. Your installer must be flexible enough to handle all the ways a user can customize a system.

Having a dynamic installation at run time is equally important. For example, if you want to allow the user to choose the destination folder, this means that all files must be dynamically installed to the chosen location.

Session variables are designed to handle such dynamic data during the installation. In this chapter you will learn everything there is to know about session variables in Setup Factory.

5

In This Chapter

In this chapter, you’ll learn about:

- Built-in and custom session variables
- Setting session variables
- Adding session variables
- Removing session variables
- Using session variables on screens
- Expanding session variables

What Are Session Variables?

Session variables are special named “containers” for values that may change during the installation process. At run time these variables are automatically replaced with the string of text that is assigned to them. These variables are used when displaying dynamic text on screens or within the paths of files you are installing.

Even though all session variables behave the same, there are two categories of session variables that can be used in Setup Factory: built-in session variables, and custom session variables.

Built-in Session Variables

For convenience, Setup Factory contains a series of built-in session variables that are commonly used in projects. These variables are automatically expanded every time they’re used.

Many of the built-in session variables hold information that has been gathered from the user’s system. For example, since the path to the Windows folder can differ between systems, a session variable named %WindowsFolder% is provided, and it automatically contains the correct path. The list of built-in variables is too long to outline here, but some other examples include %ProgramFilesFolder% and %MyDocumentsFolder%.

While many of the built-in variables gather information from the target system, there are other built-in session variables that are available for managing information about the product you are installing. A few examples of these product-related variables are:

- %ProductName% - The name of the product you are installing.
- %CompanyName% - The name of the company that developed the product.
- %ProductVer% - The version of the product you are installing.
- %AppFolder% - The default directory that the product will be installed to.

Custom Session Variables

Since the built-in session variables were designed to be generic, there may be situations where the available built-in session variables do not accomplish all of your design needs. For these occasions, you can still take advantage of session variable technology by defining custom session variables in your install. These can be named anything you wish and can contain any textual data you require.

Setting Session Variables

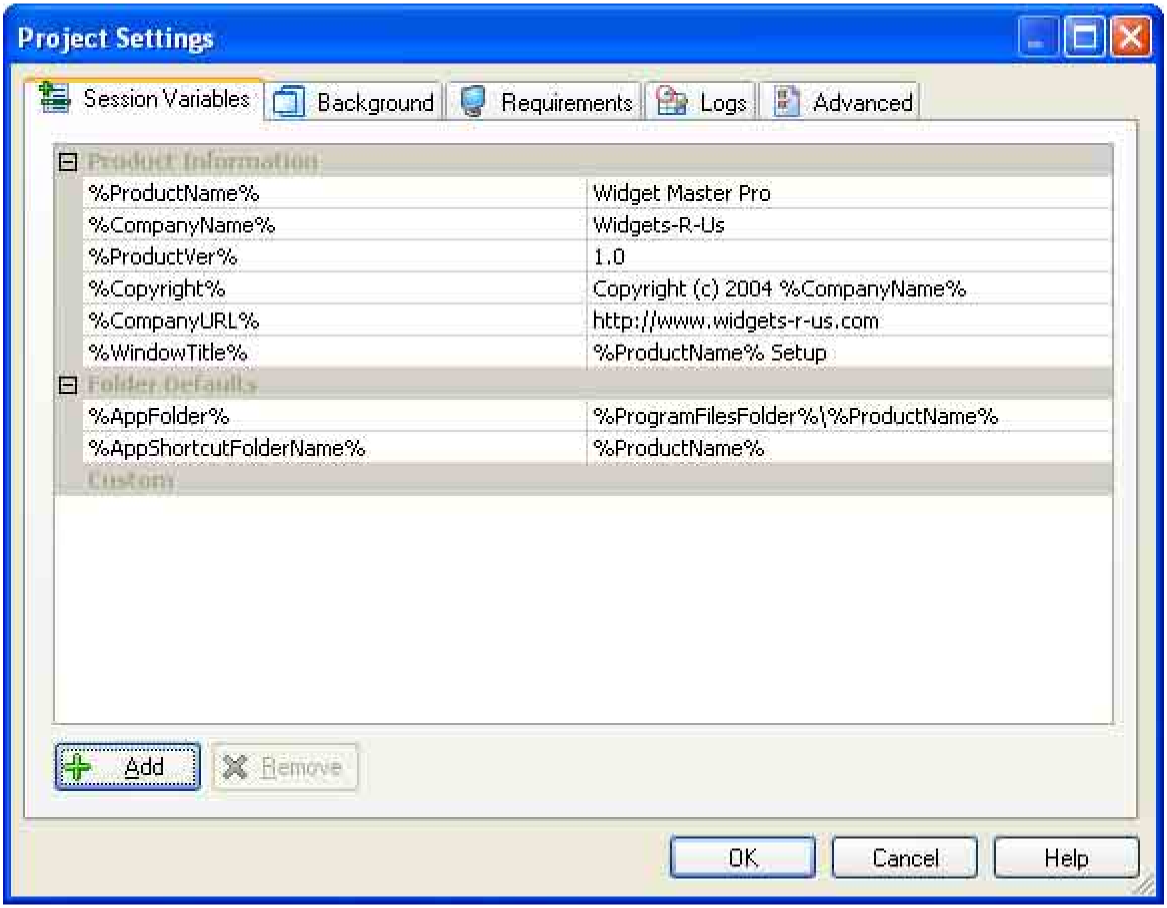
Each session variable consists of a name, e.g. "%NickName%," and a value that you assign to it, e.g. "Pookie." When a session variable is expanded at run time, its name is replaced by the value that is currently assigned to it. (For example, "I love you, %NickName%" would become "I love you, Pookie.")

There are two ways you can assign a value to a session variable: you can set its initial value on the Session Variables tab, or you can use an action to set its value anywhere in your installation.

Using the Session Variables Tab

Generally the Session Variables tab is used for setting the initial value of session variables at startup and for values that won't be changing at run time using actions.

The Session Variables tab can be found on the Project Settings dialog. To display this tab, choose Project > Session Variables from the program menu. An image of this dialog can be seen below:



Product Information	
%ProductName%	Widget Master Pro
%CompanyName%	Widgets-R-Us
%ProductVer%	1.0
%Copyright%	Copyright (c) 2004 %CompanyName%
%CompanyURL%	http://www.widgets-r-us.com
%WindowTitle%	%ProductName% Setup
Folder Defaults	
%AppFolder%	%ProgramFilesFolder%\%ProductName%
%AppShortcutFolderName%	%ProductName%
Custom	

As seen in the image above, the Session Variables dialog contains three categories: Product Information, Folder Defaults, and Custom.

The Product Information category contains a series of built-in session variables whose values can be set. They consist of informational values that pertain to the product you are installing. To set the value of one of these variables, click on the value in the right-hand column that is beside the session variable whose value you want to set. The session variable will become highlighted and the cursor will flash in the variable’s value field. This means it is ready for editing. Once you have finished editing its value, you can press the Enter key, or click on the next variable you wish to edit.

The Folder Defaults category contains two built-in variables that are used as defaults, both relating to the files you are installing. For example, the variable %AppFolder%

contains the default path that will be used as the destination for your files (if they are being installed to that location).

The Custom category contains any custom session variables that have been created. Variables of this type will appear in the same way as variables in the other two categories.

Tip: As the session variable list grows, it may help to hide portions of the list. Each category can be expanded or collapsed by clicking the "+" icon on the left hand side of the category text.

Using Actions

An action is also available to set the value of a session variable. This action is called SessionVar.Set and can be found in the "SessionVar" action category. It allows you to set the value of an existing session variable that was defined on the Session Variables tab, or to create a new one.

The SessionVar.Set action can be used with any event (i.e. on any Actions tab) throughout the project. The function prototype for this action is:

```
SessionVar.Set(string VariableName, string Value)
```

For example, if you want to assign the value "My Value" to a session variable named %MyVar%, the action would look like this:

```
SessionVar.Set("%MyVar%", "My Value");
```

After the above action is performed, all occurrences of the text %MyVar% on future screens will be replaced with the text "My Value."

Note: The SessionVar.Set action works with *all* session variables, i.e. both custom and built-in session variables. It is possible to overwrite a built-in session variable's value using the SessionVar.Set action, so be very careful when setting session variables with actions. Under normal circumstances, there should be no reason to modify the values of built-in variables.

Adding Session Variables

When you add a new session variable to your project and assign it a value, it is ready to be used on any screen or file path in your project.

Adding a new session variable to your project means that its name is now a special placeholder for the string that was assigned to it. As with built-in variables, custom session variables can be used on any install screen or within the path of any file in your project's file list.

There are two methods you can use to add custom session variables to your project: using the Session Variables tab, or using an action.

Using the Session Variables Tab

To add a new session variable to your project, choose Project > Session Variables from the program menu. This will display the Session Variables tab on the Project Settings dialog.

When setting a session variable from this tab, its value is available as soon as the install starts and throughout the install. The only difference between a custom session variable and a built-in one is that you can remove custom session variables from the Session Variables tab, but you cannot remove built-in ones.

To add a new session variable to your project, you can also click the Add button. This will display the New Session Variable dialog where you can define its name and assign it a value. When you click the OK button on the dialog, the newly defined session variable will appear in the Custom category of the list.

Using Actions

An action can also be used to create new session variables. As mentioned earlier in the chapter, this action is called SessionVar.Set and can be found in the "SessionVar" action category. This allows you to create a new session variable at any point during the install. The function prototype for this action can be found below:

```
SessionVar.Set(string VariableName, string Value)
```

For example, if you want to create a new session variable called %MyVar% and assign it the value "My Value," the action script would look like the following:

```
SessionVar.Set("%MyVar%", "My Value");
```

Note: Generally it is good practice to create new session variables on the Session Variables tab rather than using an action. Following these guidelines helps keep track of which variables exist in the install.

Removing Session Variables

When you remove a session variable from your project, Setup Factory will no longer recognize the variable's name as special placeholder text. For example, removing the session variable %ProductName% causes the name to revert back to its actual characters. In other words, the text "%ProductName%" ceases to be anything other than the letters %, P, r, o, d, u, c…and so on. After the session variable is "removed," there is no longer a value associated with the name, and no expansion occurs.

There are two methods for removing session variables from your project: using the Session Variables tab, or using actions.

Using the Session Variables Tab

Similar to adding session variables, removal of session variables can also be accomplished from the Session Variables tab. However, only those within the Custom category can be removed from your project. To remove a custom session variable, click on the desired session variable name in the list to highlight it, and then click on the Remove button. The session variable will be removed from the list.

Using Actions

Session variables can also be removed at any point during your install with an action. The action used to remove a session variable is called SessionVar.Remove and can be found in the "SessionVar" action category. The function prototype for this action can be seen below:

```
SessionVar.Remove(string VariableName)
```

For example, if you want to remove a session variable called %MyVar%, the action script would look like the following:

```
SessionVar.Remove("%MyVar%");
```

Note: Since both custom and built-in session variables behave the same, it is possible to remove a built-in session variable using the SessionVar.Remove action. For this reason, extra care should be taken when removing session variables with actions.

Using Session Variables on Screens

One of the main uses of session variables is for the dynamic expansion of text strings on screens in the install. One example of a valuable use of session variables is when you need to use a value on multiple screens, such as a product version number. While you can certainly enter the text directly for each screen, if that string changes in the future, it would require finding every location it is used in order to change its value. Using a session variable in place of that text would only require the modification of the session variable's value in one location.

Another valuable use of session variables is for gathering data that needs to be displayed on other install screens. In this case, values are often not known until some point during the install, and therefore could not be directly entered at design time.

Tip: Session variables can also be useful in multilingual installs for custom messages that you wish to display depending on the language detected or chosen.

When Are Session Variables Expanded?

When an installation starts, the session variables that are defined on the Session Variables tab are recognized and ready to be expanded. Any session variable that is used on a screen will automatically be expanded before that screen is shown and before the screen's On Preload event.

If a session variable is set to a different value, it will reflect that change the next time it is used on a screen, or if it is explicitly expanded using the SessionVar.Expand action.

Expanding Session Variables

As discussed in earlier sections, session variables are automatically replaced with their contents on screens. However, since these variables are often used to gather information from the user, the resulting values stored in these variables may need to be accessed and worked with in action script. In order to retrieve the contents of a session variable while in action script, the SessionVar.Expand action must be used. The function prototype for this action can be seen below:

```
string SessionVar.Expand(string Text);
```

For example, if there is an existing session variable called %MyVar% containing the string "These are the contents of my variable," you can access this string using the following action script:

```
strContents = SessionVar.Expand("%MyVar%");
```

In the above line of script, the Lua variable strContents would contain the value within the session variable, "These are the contents of my variable".

However, SessionVar.Expand does not restrict its Text parameter to only a session variable. It can also be used to expand a string that *contains* a session variable. For example, passing the string "The contents of my variable are: %MyVar%" as the parameter of SessionVar.Expand would return the entire string with the expanded contents of the session variable %MyVar%.

This action not only expands the session variable, but performs a recursive expansion. This means that if the session variable being passed to the action has an assigned value that contains a session variable, that session variable will be expanded as well. This type of recursion can be any number of levels deep. Yes, this sounds confusing, but an example should help:

Given the following two existing session variables:

%Inside% - whose value is "this is the inside string"
%Outside% - whose value is "The value of inside is: %Inside%"

If the following action script was executed:

```
strContents = SessionVar.Expand("%Outside%");
```

The contents of the Lua variable strContent" would be:

"The value of inside is: this is the inside string"

Setup Factory also contains an action that will prevent the recursive expansion of session variables. This action is called SessionVar.Get and can also be found in the "SessionVar" actions category. The function prototype can be seen below:

```
string SessionVar.Get(string Text);
```

Using the previous example, let's say we only wanted the expanded contents of %Outside%, without expanding %Inside%. In that case, the following action script could be executed:

```
strContents SessionVar.Get("%Outside%");
```

…and the contents of the Lua variable strContents would be:

"The value of inside is: %Inside%"

Tip: If you would like to see an advanced example of session variables in use, examine some of Setup Factory's built-in screens. For example, the Select Install Folder screen uses actions to set, update, and display a pair of session variables named %SpaceAvailable% and %SpaceRequired%.

Chapter 6:

Packages

Applications can often contain a series of optional components that extend the functionality of the main application. Since these components are not always required, it's often desirable to allow the user the choice of what components they would like to install. While this type of design can be accomplished with a series of file conditions, it can be very tedious. Packages are designed as a grouping mechanism that allows components to be managed in an easier and more efficient way, both at design time and run time.

6

In This Chapter

In this chapter, you'll learn about:

- What packages are and how they can be used
- Defining packages
- Assigning files to packages
- Adding a Select Packages screen to your project
- Advanced package actions

What Are Packages?

Packages are special categories that you can define in order to group related files together. They're usually used to give users the ability to choose which parts of an application they want to install. Here’s a basic overview of packages:

- Each package consists of a unique package ID, a localized display name, and localized description.
- You can assign file or folder references to packages by using the File Properties or Folder Reference Properties dialog.
- You can have as many packages in a Setup Factory project as you want.
- You can assign as many files to a package as you want.
- You can assign the same file to more than one package.
- You can let your users select packages by adding a Select Package screen to your installer.
- Selecting a package sets its “install” property to true. Deselecting a package sets its “install” property to false. The state of this property affects whether or not all of the files assigned to the package will be installed.
- The “install” property of the package is what determines whether files are installed.
- Files assigned to multiple packages are installed when at least one of the packages’ “install” properties is set to true.
- Setup Factory automatically calculates the total size of the files in each package. These package sizes can optionally be displayed on the Select Packages screen. You can also set an optional value for each package that will be added to the package size calculation.
- The name and description of a package can be localized so they will appear in the user’s chosen language.
- Packages can be disabled so that they cannot be modified if used on the Select Packages screen.

Using Packages

Using packages in your Setup Factory project involves four simple steps:

1. Create the packages.
2. Localize (translate) the package's properties (only if creating a multilingual installation).
3. Assign files to the packages.
4. Add a Select Packages screen to your project so users can select the packages.

For example, let's say you include a set of tutorials with your application in the form of very large video files. Because these files are so large, you want your users to be able to choose whether or not to install them at run time. This is a situation where packages should be used.

Any files in your installation that you know will always be installed should not be assigned to a package. Packages should only be used for groups of files that are conditionally installed.

Defining Packages

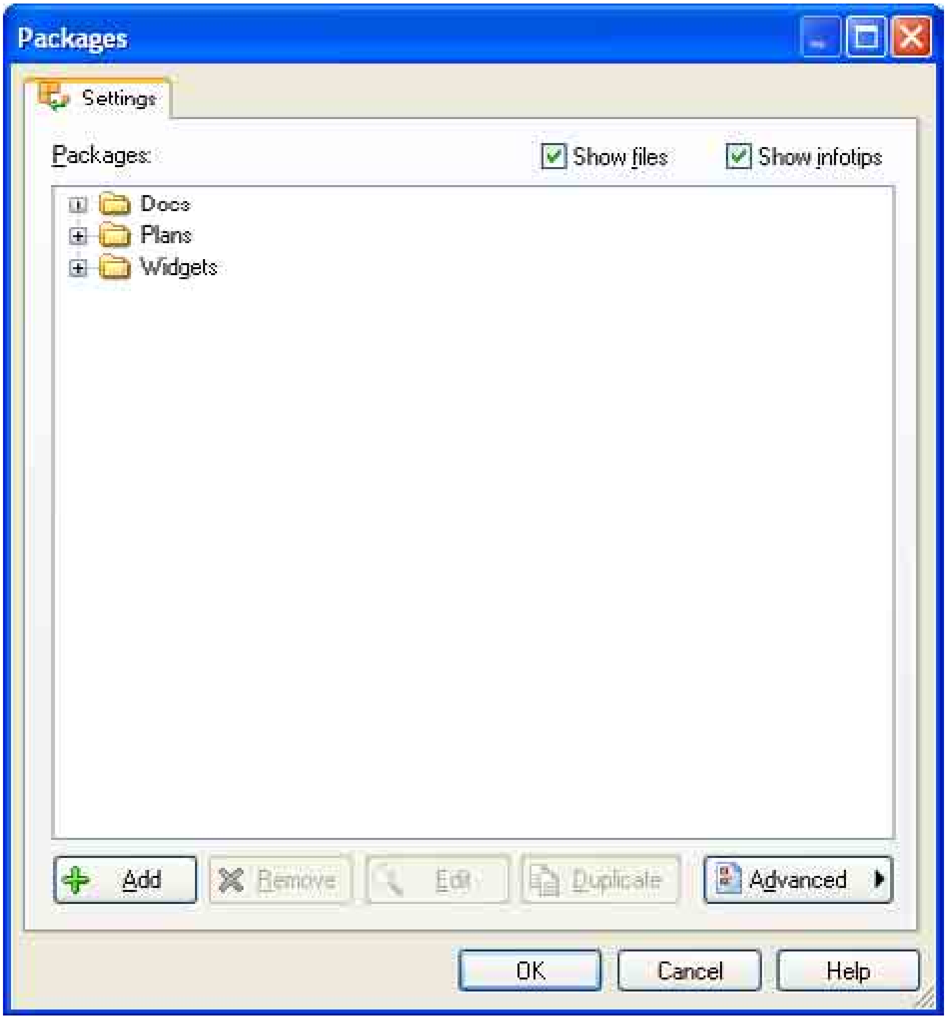
In order to group files into packages, they must first be added and configured in the project. This is done using the package manager.

The Package Manager

The management of each package in your project can be done through the package manager. This configuration tool is located on the Packages dialog and can be accessed by choosing Project > Packages from the program menu.

The Packages dialog contains a visual representation of the packages currently in the project, as well as their contents. Packages can be added, removed, or edited from this dialog.

Here is an example of the Packages dialog for a project containing three packages:



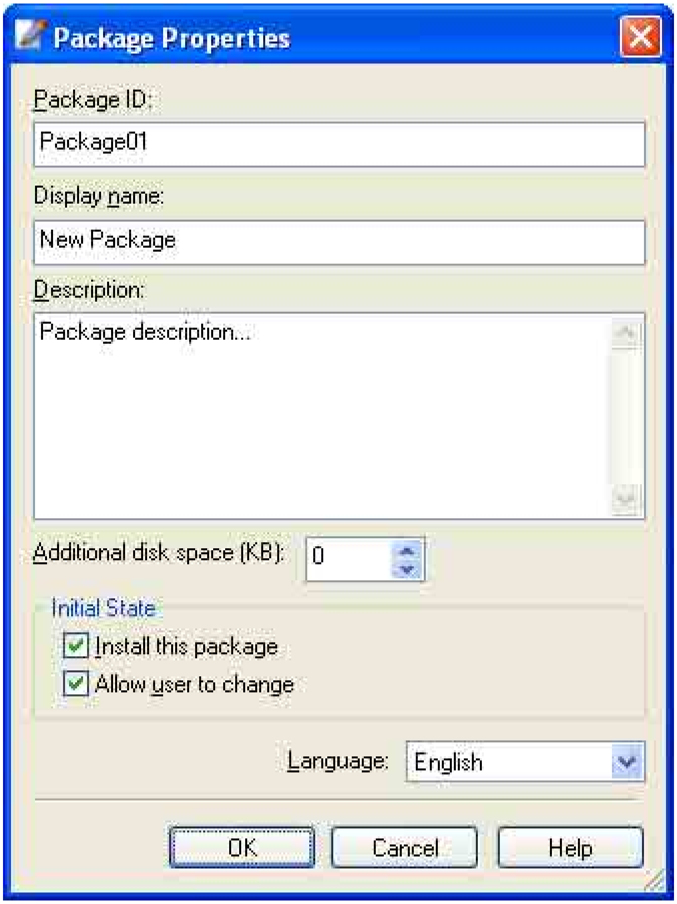
☑ Show files

☑ Show infotips

Other features of the dialog include the “Show files” and “Show infotips” check boxes, both located in the top right hand corner of the dialog. The Show files option is used to show or hide the contents of each package in the list. The Show infotips option controls whether or not to display a pop-up tooltip showing the properties of the package under the mouse pointer. When this option is checked, a summary of the properties will appear whenever the mouse pointer hovers over a package ID on the Packages dialog.

Adding Packages

Adding a package to your project provides you with a category that can later be used to group any related files that you wish to optionally install. Adding packages can be accomplished through the Packages dialog. As mentioned earlier, the Packages dialog can be opened by choosing Project > Packages from the program menu. To add a new package to your project, click the Add button. This will open the Package Properties dialog as seen below:



Package ID: Package01

Display name: New Package

Description: Package description...

Additional disk space (KB): 0

Initial State

☑ Install this package

☑ Allow user to change

Language: English

The Packages dialog contains all of the necessary settings to create a new package in your project.

1. The first step is to enter the Package ID value. The package ID is a unique string that is used to identify the current package. It is best to use an ID that is descriptive, for example, “Bonus_Images.”

2. The second piece of information to enter is the package’s display name. This is the name that will be seen by the user if the package is available on the Select Packages screen.

3. The Description field can be filled with additional information to help describe the contents of the package. This information is also displayed on the Select Packages screen and is designed to help the user decide whether or not it is wanted.

4. There may be occasions when the contents of a package are within another setup, or not even within the install itself. In this situation, Setup Factory would not have access to the amount of disk space required for the files. In these cases, the Additional disk space setting is available so you can manually populate the anticipated size of the content. When each package’s size is calculated at run time, this value will be added to the current size value.

5. The next step is to configure the initial states of the package. Two settings are available in this category: “Install this package” and “Allow user to change.” Checking the Install this package check box tells Setup Factory that its “install” property should be initially set to true and that all of its files should be installed. Unchecking this box sets the property to false, saying that none of the package’s files should be installed. The Allow user to change check box controls how this package’s check box will appear in the packages tree on the Select Packages screen. If this option is checked, the user will be able to select or deselect the package. Unchecking this option greys out the package, preventing the user from changing its install state.

6. The next step is to localize the package’s display name and description. This is only necessary if you are creating a multilingual installation. See the Localizing Packages topic for further information on this feature.

After the above properties have been set and you click the OK button, the package will be added to the list on the dialog.

Removing Packages

Similar to adding packages to your project, packages can also be removed. Removing a package from your project simply removes that category. No files in your project will be removed.

To remove a package, locate the one you want to remove and click on its package ID so it is highlighted. Once highlighted, you can either click the Remove button or press the Delete key. This results in the removal of the package from the list.

Editing Packages

After creating your package, you may need to update or edit some of its properties. The editing of packages can be done on the Packages dialog. To edit a package's properties, locate the desired package ID in the list and click on it so it is highlighted. Next click the Edit button to display its settings on the Package Properties dialog.

Tip: Instead of clicking the Edit button, you can double click on the package ID to open its Package Properties dialog.

Localizing Packages

When creating multilingual installations containing packages, it's not only necessary to localize screens. Since the properties of a package such as its display name and description are visible on the Select Packages screen, those properties must also be localized so that their text is displayed in the language detected on that machine. These two properties can be localized from the Packages dialog. By localizing we mean translating the text into a specific language that is supported by the install.

In order for a package to be localized, the languages must first be configured in the project. (See Chapter 7 for more information on adding languages.) Each package can be localized from its Properties dialog. To open the package's properties, click on the package's ID in the list and click the Edit button.

At the bottom of the Package Properties dialog you will find a drop-down list containing all of the available languages that the current package's properties can be translated to. This is the language selector. When you change the value in the language selector, the Display Name and Description text values can be translated for the chosen language. For example, if English, French and German are supported by your setup, you would first select English from the dropdown and translate the two strings. Then you could select French in the dropdown and translate the two strings to their equivalent French translation. The same procedure would be used to translate the package to German.

When the package is displayed on the Select Packages screen, its display name and description will appear in the correct language—the appropriate text will be chosen according to the language that is running on the user's system.

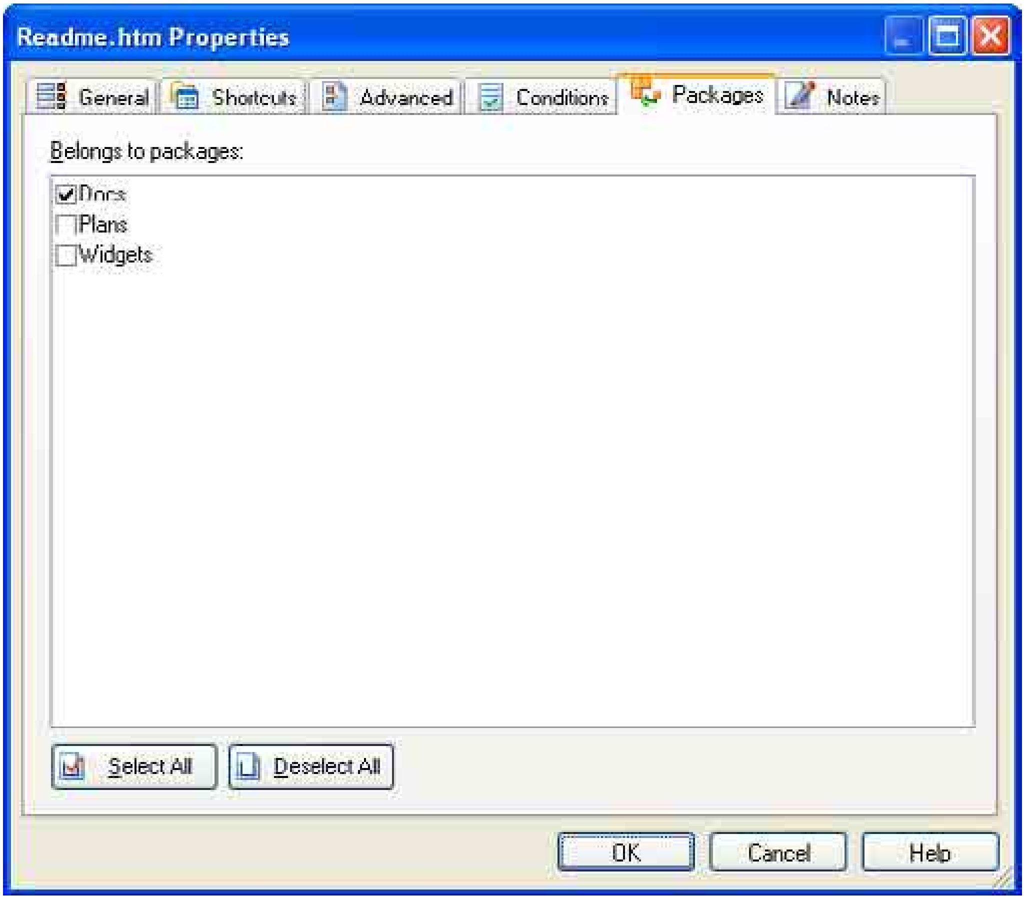
Assigning Files to Packages

When a file is assigned to a package it means that it is now part of a collection of files that can be selected at run time. If that package is selected, all files that have been assigned to it will be installed.

Assigning a file to a package can be done through each file's Properties dialog. The following steps can be taken to add a file to a package:

1. Find a file in the main file list that you want to assign to a package.
2. Right click on the target file and select "File Properties…" from the context menu to open the file's Properties dialog. You can also double click the file to accomplish the same result.
3. Click on the Packages tab.

The packages tab will look similar to the example below:



Belongs to packages:

☑ Docs
□ Plans
□ Widgets

The packages tab lists all of the current packages defined in the project, each by their package ID. Each package will also contain a check box used to determine whether or not the current file belongs to that package. A file can belong to zero or more packages.

Tip: Multiple files can be assigned to a package at once using the Multiple File Properties dialog. Once a group of files is selected, choosing File Properties from the right-click context menu will open the Multiple File Properties dialog. When checking a package on the Packages tab, each file that has been initially selected will belong to the selected package.

If you were to go back to the package manager, each package will optionally display the files that have been assigned to them.

Adding a Select Packages Screen

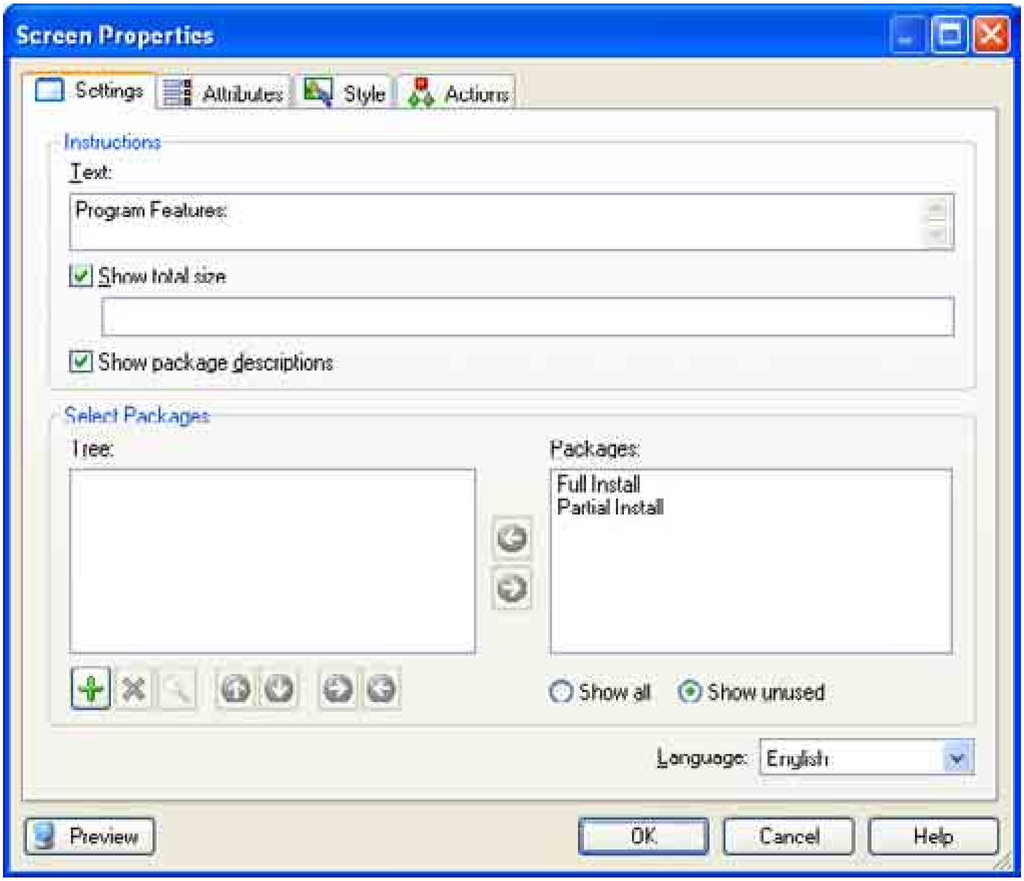
After each package has been created and all of the desired files have been assigned to them, it is time to add a Select Packages screen to the install. While there are other methods of interacting with packages, the select packages screen was designed to present a user-friendly interface for selecting packages.

The Select Packages screen is normally placed before the Select Install Folder screen, located on the Before Installing screen stage. This will allow the selected package to be part of the file size that is calculated on the Select Install Folder screen. The following steps can be taken to add a Select Packages screen:

1. Choose Project > Screens from the program menu to open the Screens dialog.
2. Click on the Before Installing tab.
3. Click the Add button to open the Screen Gallery dialog.
4. The screen gallery contains an alphabetic list of all of the screens that can be added to your install. Locate the Select Packages screen, highlight it, and click the Ok button. This will add the selected screen to the screen list.
5. While the Select Packages screen is highlighted, keep clicking the up arrow button to move the screen one position up in the list until it is before the Select Install Folder screen.

Screen Properties

The Select Package screen's Properties dialog can be opened by highlighting it in the Before Installing list and clicking the Edit button. The settings that are used to configure the displaying of packages can be found on the Settings tab. An example of the dialog can be found below:



Text: Program Features:

☑ Show total size

☑ Show package descriptions

Packages:
Full Install
Partial Install

○ Show all ● Show unused

Language: English

The Show total size check box controls whether or not Setup Factory will automatically calculate and display the amount of disk space the installation requires for the selected package, along with a text description.

Note: By default the text used is “Space required by setup: %SpaceRequired%.” %SpaceRequired% is a custom session variable defined on this screen which will be replaced by the actual required space value at run time.

The Select Packages section of the dialog contains the settings that are used to determine what the user will see at run time. The area in the bottom right labeled “Packages:” contains the list of packages that are currently being used in the install; you will most likely want to make this visible to the user.

On the left is a section labeled “Tree:” This section is the framework for a selectable tree structure that will be used to represent the packages in your install.

Tip: You can also open the screen’s Properties dialog by double-clicking it.

Adding Packages to the Tree

Any packages that you want to make available on the screen at run time must first be added to the tree. To add a package to the tree, highlight its package ID in the Packages section on the right-hand side and then click the Add button (represented by an arrow pointing to the left). This will add the package to the tree on the left.

This procedure can be replicated for each package that you want to display in the tree.

Grouping Packages into Categories

Since packages can only contain files, there must be another element if the goal is to present a hierarchy of selectable options. The element that accomplishes this type of design is called a “category.”

A category is designed to group a number of packages and/or other categories together in a select package tree. If a parent category is selected by the user, all of the packages and categories within it will be selected as well.

Creating Categories

To add a category to the package tree, click the Add button to display the Category Properties dialog. The Category Properties dialog contains several settings that control its appearance and functionality in the tree, such as the category’s display name and description.

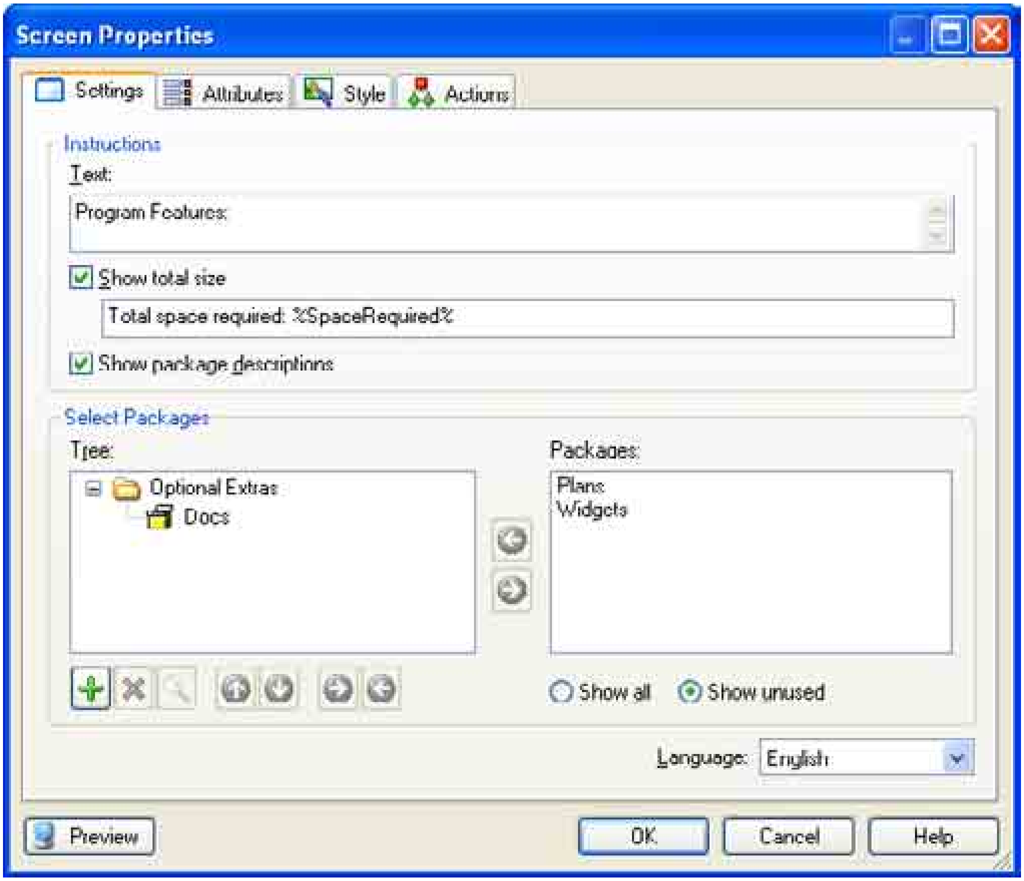
The defaults section of the properties is similar to those of a package’s properties. The Install check box controls its initial state. If it is checked, all of its contents will be checked and if unchecked, all of its contents will be unchecked. The Enabled check box controls whether or not it can be changed by the user. The Expanded check box controls whether or not the category’s contents will initially appear expanded.

Organizing the Tree

Once the desired packages and categories have been added to the tree, you will then need to organize them into the structure you want your users to see. As mentioned earlier, a category can contain packages or other categories. Both packages and categories can be moved up and down using the up arrow and down arrow buttons.

In order to add a package to a category, you would move the package so it is below the target category. Then while it is still highlighted, click the button with the arrow pointing right. To remove a package from a category, you would instead click the button with the arrow pointing to the left.

Below is a sample of how the tree will look after a package has been added to a category:



Instructions

Text:

Program Features:

☑ Show total size

Total space required: %SpaceRequired%

☑ Show package descriptions

Select Packages

Tree:

Optional Extras
Docs

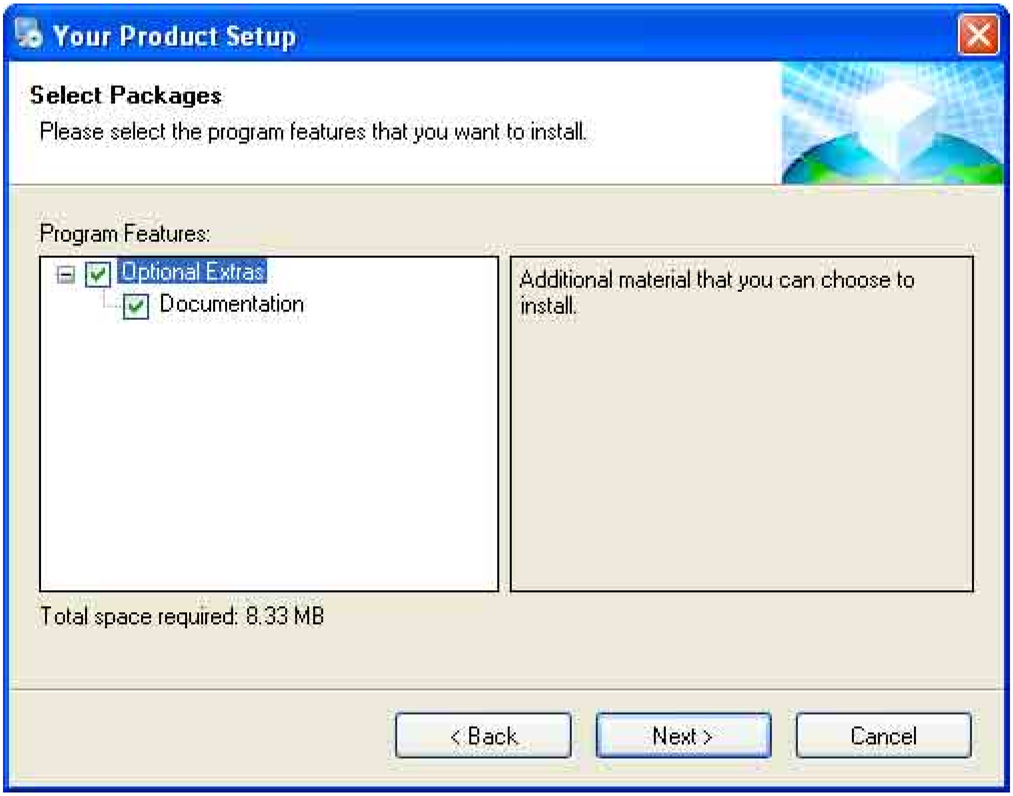
Packages:

Plans
Widgets

○ Show all ● Show unused

Language: English

Here is a sample of how the above settings would appear to the user at run time:


Your Product Setup
Select Packages
Please select the program features that you want to install.
Program Features:
☑ Optional Extras
☑ Documentation
Additional material that you can choose to install.
Total space required: 8.33 MB
< Back
Next >
Cancel

You can remove a package or category from the tree by clicking the Remove button. Removing a package from the tree only removes its representation in the tree. The package itself will not be removed from your install.

Advanced Package Actions

While the Select Packages screen handles the normal situation while working with packages, there may be occasions where you wish to handle things in a highly customized way. For this reason there are a few actions that are available for getting and setting package properties.

SetupData.GetPackageList

This action enables you to get a list of all package IDs in the setup. When called, it returns a numerically indexed table of the package IDs.

SetupData.GetPackageProperties

This action allows you to get all of the available information about a particular package, which is stored in a table. For example, let's say that you need to know whether a certain package is set to be installed at a certain point during the install. The following action script could be used to gather this information:

```
tbMyTable = SetupData.GetPackageProperties(My_PackageID);
bIsInstalled = My_PackageID.Install;
```

In the above script, the variable bIsInstalled will contain true if the package is set to be installed, or false if it is not. Other properties that are available are its display name, description, additional disk space value and enabled state.

SetupData.SetPackageProperties

This action is similar to Setup.GetPackageProperties. However instead of retrieving information about a package, you can set its properties at runtime. For example, if you wanted to explicitly set a package so its contents will be installed, you could use the following action script:

```
tbMyTable = {Install=true};
SetupData.SetPackageProperties(My_PackageID, tbMyTable);
```

The above action script first creates a table with the desired index value. The next action takes that table and merges its values with the specified package's current settings.

These types of actions may be useful in cases where the install states of packages are not controlled by user interaction, but rather a series of queries of the user's system.

Tip: Another use for these actions could be to control the tree in a custom way when the user selects a particular option. For example, you could automatically turn a specific package on whenever another specific package was turned off.

Chapter 7:

Languages

The Internet has opened many new markets to software developers whose past products would have usually only supported their own local language. Today, even smaller software development and data distribution companies have realized the need to provide multilingual software to meet existing and newly emerging international software markets.

One of Setup Factory's strongest features is its support for creating foreign language installations. You can use Setup Factory to create an installer that will automatically display messages and prompts in your user's native language.

Setup Factory ships with display messages and prompts in many of the world’s most commonly used languages. It also makes it very easy to modify existing translations at any time.

7

In This Chapter

In this chapter, you'll learn about:

- Internationalizing your installer
- How run-time language detection works
- The language manager
- Language files
- Setting the default language
- Adding and removing languages
- The language selector
- Localizing screens, packages, the uninstaller, and actions
- Customizing error messages and prompts
- Advanced techniques, such as using actions to determine the current language and "changing" the current language for testing purposes

Internationalizing Your Installer

Setup Factory has the ability to automatically detect the user's system's language and to display messages and screens in that language. The developer has full control over what languages are supported in their setup and the content that they would like to present to the user.

Language text is mainly used for messages generated throughout the install and on screens, both of which can be easily translated for multilingual installs.

Setup Factory allows you to localize your setup application in three areas:

- Error messages, status messages and prompts that are common to all setups.
- Screens that appear as the main GUI of the installation that are usually application-specific.
- Packages used by the setup to organize files into logical units.

The following sections of this chapter will look more closely at how to provide international support to all of these areas of the install.

Run-time Language Detection

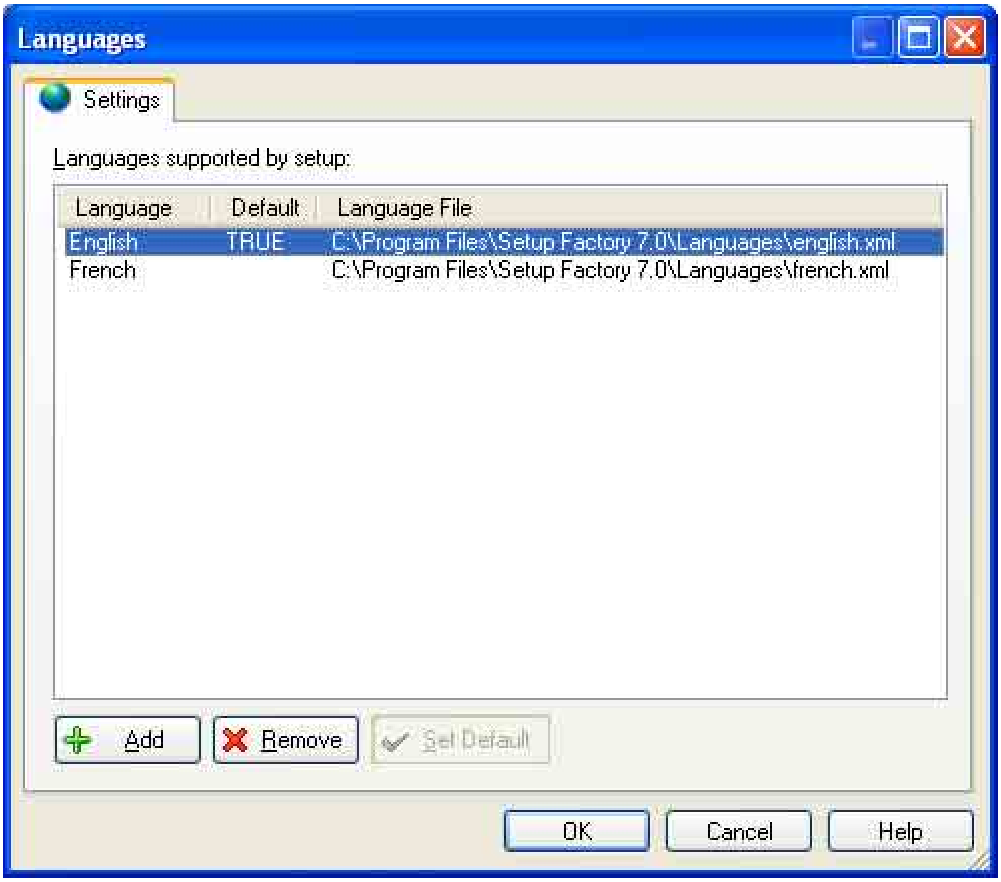
At run time, the language that Setup Factory detects is based on the user's regional and language settings. These settings allow Windows users to configure which language is displayed, which input locale is used and which keyboard layout is supported in the Windows operating system environment. These settings are usually configured when Windows is installed and can usually be changed from the Windows Control Panel. For example, in Windows XP, a user can select Start > Settings > Control Panel and start the Regional and Language Settings control panel application.

Each language in Windows has a constant language identifier. A language identifier is a standard international numeric abbreviation for the language in a country or geographical region made up of a primary and secondary language ID (there is a complete list of primary and secondary language IDs in the Setup Factory help file). Setup Factory maps all known languages and sub-languages according to the language identifier used for that language by Windows.

Tip: Setup Factory maps language identifiers to language names in a file called language_map.xml located in Setup Factory's Language folder (usually "C:\Program Files\Setup Factory 7.0\Languages"). You can look at this file to see which primary and secondary language IDs are mapped to which languages. It is NOT recommended that you modify this file unless you have a very specific reason to do so.

The Language Manager

The languages that are supported by your setup are all configured from the Language Manager. You can access the language manager by selecting Project > Languages from the menu.



Language	Default	Language File
English	TRUE	C:\Program Files\Setup Factory 7.0\Languages\english.xml
French		C:\Program Files\Setup Factory 7.0\Languages\french.xml

Although you can localize messages in several areas of the design environment, the Language Manager is the only place where you can control the setup's default language as well as which languages are or are not supported by your setup. For example, if you are editing one of your screens and decide that you would like to add German support to your setup, you will have to use the Language Manager to do so. Once you add German support here, it will also be available to all other areas of the setup.

When you add a language to your project, you are indicating that you want the setup to recognize that particular language identifier on a user's system and use specific messages for that language when identified. Conversely, if a language is not represented in the languages list, it does not mean that the setup will not run on a system that uses that language; it simply means that it will use the default language in that case.

Default Language

Every setup must have a default language. The default language is the one that will be used whether the user happens to be using the default language or any other language not represented in the languages list.

For example, let's suppose that you have English, French and German support in your setup with English as the default language. If your user runs the setup on a Greek system, the user will see the English messages since you did not specifically include support for the Greek language.

Note that the default language must be one that has a corresponding language file (see the next section).

Language Files

A language file is an XML file that contains all error messages, status messages and prompts that are used internally by the setup. Language files do not contain setup-specific messages such as those used on screens and packages. Note that not all languages have a pre-configured language file.

Language files are located in Setup Factory's Languages folder (usually "C:\Program Files\Setup Factory 7.0\Languages"). They are named according to the English name for the language they represent. Each file contains a language map that identifies which language the file is responsible for and all of the built-in messages that will be used for that language.

If you add a language to your project that does not have a language file, that language will use the same messages as the default language.

Getting Additional Language Files

If you need a language file that is not shipped with Setup Factory, please visit the Indigo Rose Web site (www.indigorose.com) and user forums (www.indigorose.com/forums/) where new language files are made available from time to time.

Making Your Own Language File

If after consulting the Indigo Rose web site you still can't find the language file you need, you can always make one yourself. To make a new language file, simply make a copy of the existing language file that you want to translate from, rename it to the new language name, change the language map in the file accordingly and then translate the messages.

To clarify this process, here is an example of how to create a French language file (this is for demonstration purposes only and is unnecessary since French already ships with Setup Factory):

1. Open Windows Explorer to Setup Factory's Languages folder (usually "C:\Program Files\Setup Factory 7.0\Languages".)

2. Make a copy of English.xml and name it French.xml.

3. Open French.xml in a text editor such as Notepad.

4. Open language_map.xml from the Languages folder in a text editor such as Notepad.

5. Locate the section that maps French in the language_map.xml file. It should look like this:

```
<Language>
  <Name>French</Name>
  <Primary>12</Primary>
  <Secondary>
        <ID>1</ID>
        <ID>2</ID>
        <ID>3</ID>
        <ID>4</ID>
        <ID>5</ID>
```

```
      <ID>6</ID>
  </Secondary>
</Language>
```

6. Copy the above section from language_map.xml and paste it in place of the <Language></Language> section of the French.xml file. This will allow Setup Factory to recognize this file as a language file for the French language.

7. Translate all messages in the <Messages></Messages> section to French. Do not change any actual XML tags. For example:

   ```
   <MSG_SUCCESS>Success</MSG_SUCCESS>
   ```

   becomes:

   ```
   <MSG_SUCCESS>Succès</MSG_SUCCESS>
   ```

8. Save the file and re-open Setup Factory. The new language will now be available.

9. Share the file with others! Go to http://support.indigorose.com and open a support ticket saying that you have made a language that you would like to share with other Setup Factory users. We will take it from there.

Adding Languages

To add a new language to your setup, click the Add button in the Language Manager. This will open the Add New Language dialog. Simply select the language that you want to add and click OK. You will then see the language appear in the languages list.

What Happens When You Add a New Language

When you add a new language to your setup, the following happens automatically:

- Setup Factory searches the Languages folder for an appropriate language file for the language. If one is not found, the new language is set to use the default language's language file.

- The newly added language is added to all screens. That is, all screens have messages added to them for the new language. If a translated language file for that particular screen already exists, it will be used. Otherwise, the messages from the default language will be replicated for the new language.

- The newly added language is added to all packages. That is, all packages have names and descriptions added to them for the new language. The default messages for the new language will be replicated from the project's default language.

- The language is added to the list of languages that you can select from in the language selectors throughout the design environment.

Of course, you will still need to go into the Packages and Screens dialogs to verify and/or translate the text for the new language.

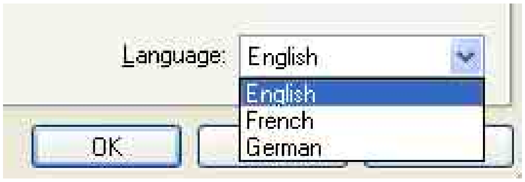
Removing Languages

To remove a language from the setup, select the language in the Language Manager's list and click the Remove button. Note that the default language cannot be removed. You will have to make a different language the default language before deleting the original default language.

When you remove a language from the setup, all translations in that language are removed from all screens and packages in your project. Therefore, use caution when removing languages.

The Language Selector

Once a language is added to your setup project, it will be available for translation when editing screens and packages via the language selector. The language selector is simply a combo box that lets you choose the language you will be entering text for in the screen or package.


Language: English

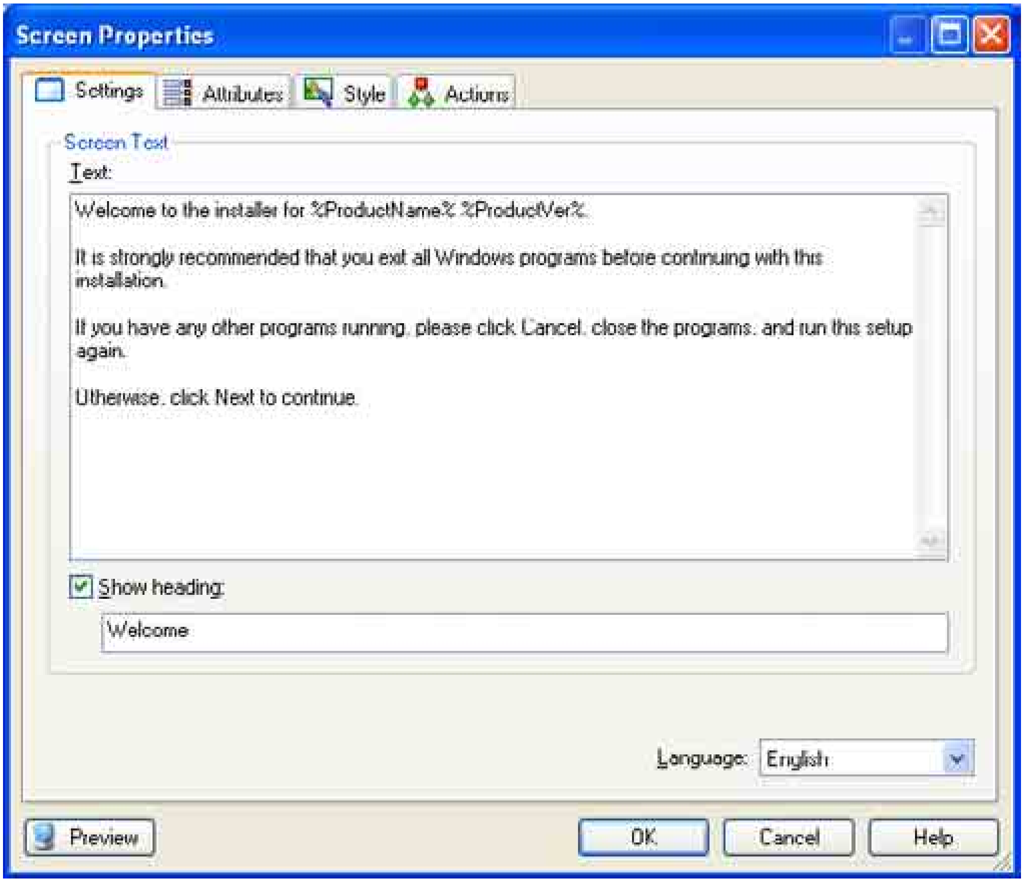
If you want to work on a language that is not in the list, you will first have to add it through the Language Manager.

Localizing Screens

To localize a screen, open the Screens dialog (Project > Screens) and double-click a screen to open the screen's properties. Next, select the language that you want to enter text for in the language selector. Then, simply type the text that you want for the various fields in the language that you are currently working in.

Note that the **Screen ID** field on the Attributes tab cannot be translated. The screen ID is a unique identifier for the screen and is never displayed to the end user.

Here is an example of a Welcome screen with English text:



Text:

Welcome to the installer for %ProductName% %ProductVer%.

It is strongly recommended that you exit all Windows programs before continuing with this installation.

If you have any other programs running, please click Cancel, close the programs, and run this setup again.

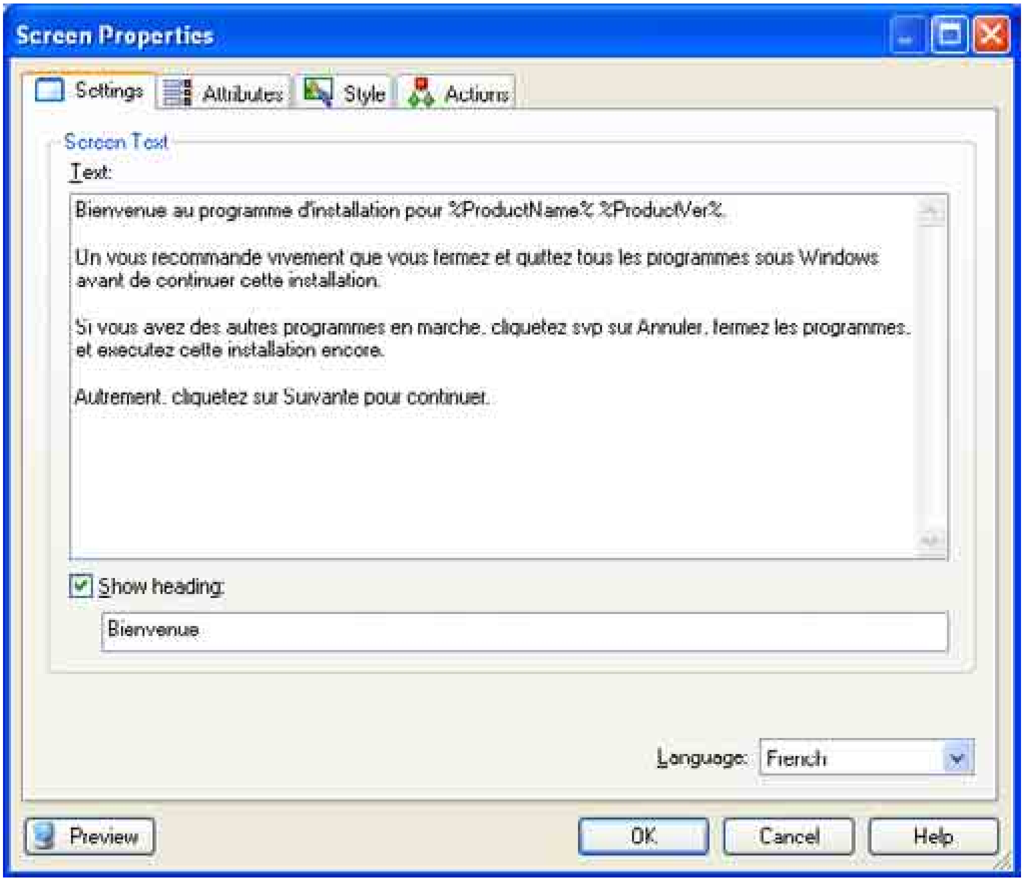
Otherwise, click Next to continue.

☑ Show heading:

Welcome

Language: English

And here is the same screen with French text:



Text:

Bienvenue au programme d'installation pour %ProductName% %ProductVer%.

On vous recommande vivement que vous fermez et quittez tous les programmes sous Windows avant de continuer cette installation.

Si vous avez des autres programmes en marche, cliquetez svp sur Annuler, fermez les programmes, et executez cette installation encore.

Autrement, cliquetez sur Suivante pour continuer.

☑ Show heading: Bienvenue

Language: French

Notice that in the second screenshot, “French” has been selected in the language selector near the bottom of the dialog. The text that you enter always corresponds to the language that is selected in the language selector.

Tip: If the language that you want to translate to doesn’t appear in the language selector, you need to add support for that language to your project. This is done by adding the language in the language manager. For more information, see *The Language Manager* beginning on page 189 and *Adding Languages* on page 192.

Importing and Exporting Screen Translations

There may be times when you want to have your screens translated by a third party translator. If the translator owns a copy of Setup Factory 7.0, you can simply send them your project file, have them translate the screens using the method explained above, and then have them send the project file back to you.

However, it may be that the translator does not own Setup Factory or that you need to work with the project in other ways while the translation is taking place. Setup Factory has a solution for this situation. You must:

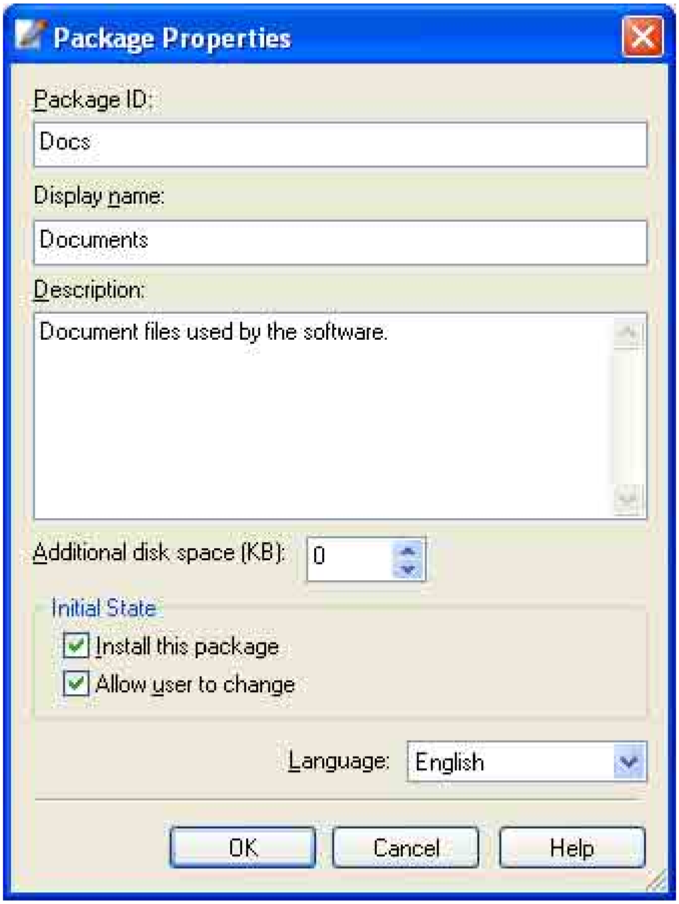
1. Open the Screens dialog (Project > Screens).
2. Select the screen that you want to have translated in the Screens list.
3. Click the Advanced button (this will open a popup menu).
4. Select Export Language and then the language that you want to export to.
5. Choose a location to save the file to.
6. Send the exported file to your translator.
7. When you receive the file back, select the screen in the Screens list.
8. Click the Advanced button and then choose Import Language from the popup menu.
9. Locate the translated file and select Open.
10. You will now have the new translated strings in your screen.

Localizing Packages

To localize a package, open the Packages dialog (Project > Packages) and double-click a package to open the package's properties. Next, select the language that you want to enter text for in the language selector. Then, simply type the text that you want for the various fields in the language that you are currently working in.

Note that the **Package ID** field cannot be translated. The package ID is a unique identifier for the package and is never displayed to the end user.

Here is an example showing the properties of a package that has been localized in English:



Package Properties

Package ID: Docs

Display name: Documents

Description: Document files used by the software.

Additional disk space (KB): 0

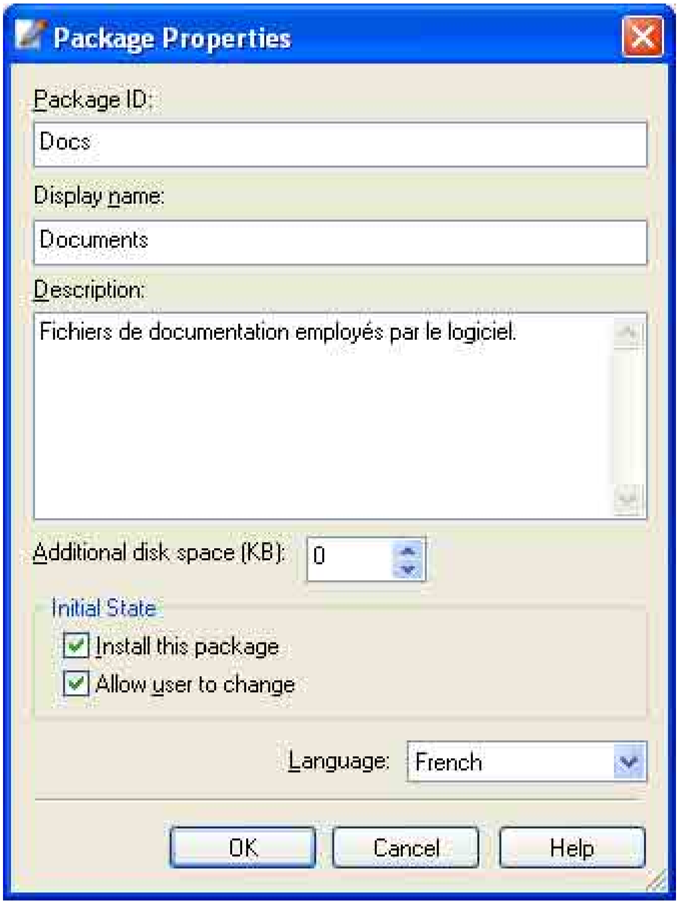
Initial State

☑ Install this package

☑ Allow user to change

Language: English

And here are the properties of the same package translated into French:



Package ID: Docs

Display name: Documents

Description: Fichiers de documentation employés par le logiciel.

Additional disk space (KB): 0

Initial State

☑ Install this package

☑ Allow user to change

Language: French

Importing and Exporting Package Translations

Packages can be exported and imported for translation in the same manner that screens are. Please refer to “Importing and Exporting Screen Translations” above for specific instructions on how to do this.

Localizing the Uninstall

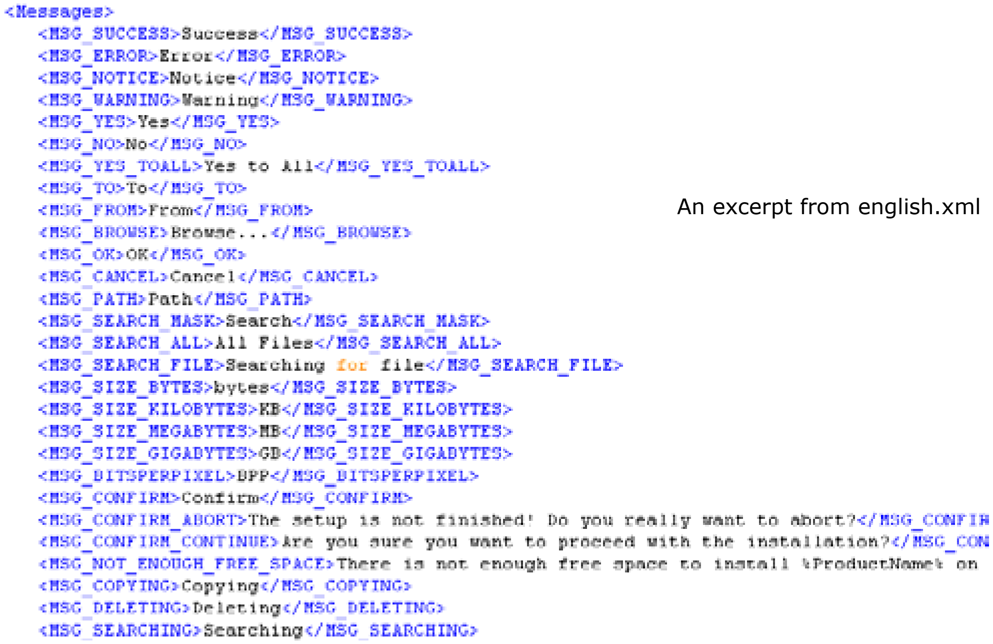
The uninstaller uses the same language file that was used by the installer. Setup Factory language files contain all messages needed by the installer and the uninstaller.

The uninstall screens can be translated in the same manner as the installation screens. To work on the uninstall screens, choose Uninstall > Screens from the menu.

Customizing Error Messages and Prompts

If you want to change the default error messages, prompts or status messages used in the setup, you can simply edit the appropriate XML file in the Languages folder using a text editor such as Notepad. Note however that this is not generally recommended; the messages that you change will be propagated to all projects that are built after the changes are made. However, if this is the desired effect, you can certainly do this.

Note also that if you choose to change default messages, the file may be overwritten by a future update to Setup Factory 7.0. To avoid this problem, you may want to modify your XML file's name. For example, from english.xml to my_english.xml.



```
<Messages>
   <MSG_SUCCESS>Success</MSG_SUCCESS>
   <MSG_ERROR>Error</MSG_ERROR>
   <MSG_NOTICE>Notice</MSG_NOTICE>
   <MSG_WARNING>Warning</MSG_WARNING>
   <MSG_YES>Yes</MSG_YES>
   <MSG_NO>No</MSG_NO>
   <MSG_YES_TOALL>Yes to All</MSG_YES_TOALL>
   <MSG_TO>To</MSG_TO>
   <MSG_FROM>From</MSG_FROM>
   <MSG_BROWSE>Browse...</MSG_BROWSE>
   <MSG_OK>OK</MSG_OK>
   <MSG_CANCEL>Cancel</MSG_CANCEL>
   <MSG_PATH>Path</MSG_PATH>
   <MSG_SEARCH_MASK>Search</MSG_SEARCH_MASK>
   <MSG_SEARCH_ALL>All Files</MSG_SEARCH_ALL>
   <MSG_SEARCH_FILE>Searching for file</MSG_SEARCH_FILE>
   <MSG_SIZE_BYTES>bytes</MSG_SIZE_BYTES>
   <MSG_SIZE_KILOBYTES>KB</MSG_SIZE_KILOBYTES>
   <MSG_SIZE_MEGABYTES>MB</MSG_SIZE_MEGABYTES>
   <MSG_SIZE_GIGABYTES>GB</MSG_SIZE_GIGABYTES>
   <MSG_BITSPERPIXEL>BPP</MSG_BITSPERPIXEL>
   <MSG_CONFIRM>Confirm</MSG_CONFIRM>
   <MSG_CONFIRM_ABORT>The setup is not finished! Do you really want to abort?</MSG_CONFIR
   <MSG_CONFIRM_CONTINUE>Are you sure you want to proceed with the installation?</MSG_CON
   <MSG_NOT_ENOUGH_FREE_SPACE>There is not enough free space to install %ProductName% on
   <MSG_COPYING>Copying</MSG_COPYING>
   <MSG_DELETING>Deleting</MSG_DELETING>
   <MSG_SEARCHING>Searching</MSG_SEARCHING>
```

An excerpt from english.xml

Advanced Techniques

There are a number of advanced techniques that you can use to manipulate the language and the translated language strings in your setup at run time. Most of these methods are accomplished using actions. This section covers a few of these advanced techniques.

Determining the Current Language

There are two actions that can be used to retrieve information about the user's language ID: System.GetDefaultLangID and Application.GetInstallLanguage. Although both actions return a table of information containing language IDs, it is important to know the difference between the two.

System.GetDefaultLangID

System.GetDefaultLangID is used to get the primary and secondary language ID that the user employs in Windows. This is absolute and cannot be changed with any other actions. For example, if this action returns 10 as the primary ID and 13 as the secondary ID you will know that the user's Windows system is configured for Spanish (Chile). There is a complete list of primary and secondary language IDs in the Setup Factory help file.

The information returned by this action can be used in cases where you want to make specific choices about what to do based on the user's absolute system language. For example, if you have a Web site that is translated into several different languages, you might want to make a series of File.OpenURL actions surrounded by if…then statements to jump to the appropriate site:

```
-- Determine absolute system language
local tblSysLang = System.GetDefaultLangID();

-- Set a default
local strURL = "http://www.yourcompany.com/english");

-- see if we should jump to a different site
if (tblSysLang.Primary == 7) then
    strURL = "http://www.yourcompany.com/german");
elseif (tblSysLang.Primary == 10) then
    strURL = "http://www.yourcompany.com/spanish");
elseif(tblSysLang.Primary == 16) then
    strURL = "http://www.yourcompany.com/italian");
end
```

```
-- jump to the Web site
File.OpenURL(strURL);
```

Application.GetInstallLanguage

Application.GetInstallLanguage is used to retrieve the primary and secondary language ID that is actually being used by the setup. Note that Application.GetInstallLanguage may return a different result than System.GetDefaultLangID if the language being used by the setup differs from the language your user employs in Windows.

For example, let’s say that you have three languages in your setup: English (default language), French and German. Suppose a user from Chile runs the setup on their system. Even though their system uses primary ID 10 and secondary ID 13 (which would be returned by System.GetDefaultLangID), Application.GetInstallLanguage will return a primary ID of 9 and a secondary ID of 1 which is the setup’s default language (English) since Spanish was not added to the setup at design time.

The value returned by Application.GetInstallLanguage will always be a value for a language that you added to your project using the Language Manager. It will never contain values for languages that were not explicitly included in the setup.

Changing the Current Language

You can change the setup’s language at run time at any point using the action Application.SetInstallLanguage. This action will allow you to set the primary and secondary language ID used by the setup. When you call this action, it will actually change all error and status messages as well as the language used by screens and packages “on-the-fly.”

One common reason to use this action might be because you want to use a language other than the default if certain other languages cannot be located. Being able to change the language that your setup uses from script allows you can handle certain specific cases in any way you want.

For example, let’s say that your setup supports two languages: English (which is the default language) and Simplified Chinese. Since English is the default language, it will be used whenever the setup is run on anything other than Simplified Chinese.

However, you might prefer that the setup use Simplified Chinese if the user runs the setup on a system configured to use Traditional Chinese.

In other words, you want to override the default language rule and force your installer to use Simplified Chinese whenever Traditional Chinese is detected. You can do so easily by placing the following short script in your project's On Startup event:

```
-- Determine absolute system language
local tblSysLang = System.GetDefaultLangID();
if (tblSysLang.Primary == 4) and (tblSysLang.Secondary == 1) then
    -- Traditional Chinese on user's system,
    -- so use Simplified instead
    Application.SetInstallLanguage(4, 2);
end
```

Testing Different Languages

You may also want to use the Application.SetInstallLanguage action for testing purposes. For example, if you are running an English version of Windows, you might want to see how your setup will look on an Italian system. Because your system is running in English, the setup will always choose English as the language to display whenever you run it on your system. However, you could force the setup to use Italian by putting the following script in your On Startup event:

```
Application.SetInstallLanguage(16, 1);
```

You could even do the above in response to a custom command line option that you handle so that your setup can be forced to use a different language at any time.

Localizing Actions

You may have noticed that there is no language selector when editing scripts in Setup Factory. This means that any text you enter into a script will not be translated for you. However, by using the Application.GetInstallLanguage action you can use if…then statements to do something different based on the language being used.

For example, let's say that your setup supports English and French and you want to show a dialog box using actions that will be localized according to those languages. The following script first determines the language that the setup is using, and then displays one of two possible greetings:

```
local tblSysLang = Application.GetInstallLanguage();

if (tblSysLang.Primary == 9) then
    Dialog.Message("Welcome",
                   "Welcome to the installer");
end

if (tblSysLang.Primary == 12) then
    Dialog.Message("Bienvenue",
                   "Bienvenue au programme d’installation");
end
```

Working with Existing Translated Messages

Setup Factory allows you to get and set translated messages for general messages (from the language files), screens, and packages at run time. This is done through several actions.

SetupData.GetLocalizedString

This action allows you to retrieve the localized text for a general message from the language files at run time. The message will be returned in the current install language. For example, the default language files provide messages to confirm if the user wants to abort the setup. Here is a way to show a dialog that confirms if the user wants to abort the setup in the current language:

```
local strTitle = SetupData.GetLocalizedString("MSG_CONFIRM");
local strPrompt = SetupData.GetLocalizedString("MSG_CONFIRM_ABORT");
local nResult = Dialog.Message( strTitle
                              , strPrompt
                              , MB_YESNO
                              , MB_ICONQUESTION
                              , MB_DEFBUTTON2 );

if(nResult == IDYES)then
    Application.Exit();
end
```

SetupData.SetLocalizedString

This action allows you to change the value of a localized string at run time. This could be useful if you want to override the default value of an error message at run time so that you don’t have to permanently change your language file at design time. For example, let’s say that you want to change the message that is displayed if the user tries to cancel the setup. By default it is “The setup is not finished! Do you really want to abort?”, but you want to change it to: “Stopping now is not a good idea. Are you sure?”

```
SetupData.SetLocalizedString("MSG_CONFIRM_ABORT",
    "Stopping now is not a good idea. Are you sure?");
```

Screen.GetLocalizedString and Screen.SetLocalizedString

These actions are used to get and set the localized string for the current language of the currently displayed screen at run time. They work the same way that SetupData.GetLocalizedString and SetupData.SetLocalizedString do except that they will only access strings used on a screen.

SetupData.GetPackageProperties and SetupData.SetPackageProperties

These two actions can be used to get and set the translated string for a package’s display name and description at run time. In this case, you will want to work with the table’s DisplayName and Description indexes to get and set the text. For example, to display a package named “Documents” display name and description in a dialog box:

```
local tblPackageProps = SetupData.GetPackageProperties("Documents");

Dialog.Message( tblPackageProps.DisplayName
              , tblPackageProps.Description );
```

Sharing Your Translations

If you do translate a language file or even screens for common use, please feel free to share them in the Indigo Rose forums (www.indigorose.com/forums/). There is a forum in the Setup Factory section where you can find, share and collaborate on translations for setups. You might even be able to trade translation services with other users.

Chapter 8:

Security and Expiration

Setup Factory 7.0 comes with installation security options that add another valuable layer of security to your application without sacrificing ease of use. Setup Factory 7.0's easy-to-implement security features make unauthorized use prohibitive without creating complications for authorized users.

This chapter will introduce you to the security options available in Setup Factory 7.0 and get you one step closer to a secure yet easy-to-use installation.

8

In This Chapter

In this chapter, you'll learn about:

- Security in Setup Factory 7.0
- Serial numbers
- Serial number lists
- The Verify Serial Number screen
- Serial number actions
- Date expiration
- Usage count expiration
- Log files, data integrity, and file encryption
- Internet authentication, post-installation security, and other important considerations

Security in Setup Factory 7.0

With the ever-increasing threat of software piracy, many developers are choosing to secure not only their software product, but also their installation. This extra layer of security prevents unauthorized users from installing the software and can assist in controlling the distribution of the installer.

In Setup Factory 7.0, there are two main ways to secure your installation. The first method involves utilizing product-specific serial numbers. From within Setup Factory 7.0, you can create and manage serial number lists. The serial number lists are encrypted once your setup is built. During your installation, you can prompt the user for a serial number using a Verify Serial Number screen or by creating your own custom screen. If the provided serial number is not found on any of your serial number lists, the installation will not continue.

The second method for securing your installation is to set expiration criteria. Your installation can be made to expire (cease to run) under certain situations. For example, your installation could expire after it has been run twice. Or instead, your installation could be made valid for a period of thirty days after which time your customer must procure the most recent version of your software. If the installation has expired, the user is alerted and the installation exits. The alert is fully customizable.

Serial Numbers

Think of a serial number as a pass phrase used by your customers to access the contents of the installation. Each user has a unique pass phrase. During the installation process, the user is prompted for this pass phrase to continue. The serial number provided by the user is then compared to the serial numbers in your list(s). If the serial matches, the installation continues. If the serial does not match, the installation exits.

Serial Number Lists

In Setup Factory 7.0, serial numbers in your installation are grouped into ‘lists.’ A serial number list is an inventory of every valid serial number in your installation.

Setup Factory 7.0 allows the creation of multiple (separate) serial number lists. One use of multiple lists would be if you wanted to distribute only one setup containing both a standard and enhanced version of your software. In this case, you would create two separate serial number lists, and based on the serial number that your customer provides to the installation, either the standard or enhanced version of your software would be installed. This method of separation is also valid for distributing both a demo and full version of your software in one installation.

Serial Number Lists Are Secure

In Setup Factory 7.0, your actual serial number lists are not stored in the installation. Rather, an MD5 hash (or “digest”) is calculated for each serial number within each of your lists. These MD5 hashes are then stored in the installer instead of the original values. This ensures that your list of serial numbers is not vulnerable.

When the user enters a serial number, the installer calculates the MD5 hash for that value and then compares it to the list of MD5 hashes that were created from the original serial numbers. If a matching MD5 hash is found, the installer knows that the user’s serial number is valid. (Since every MD5 hash is unique, the only way that the MD5 hashes for two serial numbers can match is if the serial numbers are the same.)

This allows the installer to detect a valid serial number without actually knowing what the original serial number was. Since the MD5 hash calculation is irreversible, there is no way for a hacker to retrieve the original serial numbers from the installer. The hacker is reduced to using brute force tactics, i.e. trying a series of serial numbers at random until one works…which, if your serial numbers are sufficiently complex, would be highly prohibitive.

How MD5 Security Works

An MD5 hash is a digital signature that can be used to uniquely identify any piece of data. It is the result of a special calculation that can be applied to any kind of data. This calculation process is strictly one-way. Although you can calculate the MD5 hash for any value, you *cannot* later retrieve the value from the MD5 hash. It is literally impossible to reverse the process.

This makes MD5 extremely useful for validating serial numbers. Storing the MD5 hash for a serial number allows you to determine whether a matching serial number has been entered without storing any information that could be used to determine what the original serial number was.

Build Configurations

Build configurations are collections of settings that you can switch between when you build the project. They essentially allow you to have different configurations of your software within a single project.

Every serial number list in a Setup Factory 7.0 project can be associated with one or more build configurations. In other words, you can specify which build configurations a specific serial number list will be included in. This allows you to generate multiple editions of your product from a single project while still allowing you to include different lists of serial numbers in each edition.

For example, let's say you have two build configurations in your project: one for a Professional version, and one for a Standard version of your software. You also have two serial number lists, one for your Professional version, and one for your Standard version. By assigning each serial number list to the appropriate build configuration, you can ensure that each edition has the correct list of serial numbers in it without having to create and maintain two separate projects. You only have to worry about one Setup Factory 7.0 project file; yet, as you build each separate configuration into its own separate installer, only the corresponding serial number list will be included.

Tip: More information on build configurations can be found in Chapter 11.

Generating Serial Numbers

Once the decision has been made to use serial numbers to secure your installation, a list of serial numbers must be generated. Setup Factory 7.0 has a built-in tool to generate lists of serial numbers using criteria provided by you.

Serial numbers generated with this tool are created using a serial number "mask" to specify the pattern for all of the serial numbers. In the mask, placeholder characters are used to describe the structure and format of the desired serial numbers. The built-in tool then replaces these placeholder characters with static and random letters or numbers automatically.

This tool is a huge timesaving feature. In mere seconds, you can generate a list of thousands of serial numbers formatted to your specifications.

There are two types of serial number lists that can be generated: random, and sequential. Generally, it is good practice to generate random serial number lists, unless you have a legitimate reason to generate a sequential list.

When considering what format your serial numbers should take, there are a couple of things to keep in mind. First, if you deal with more than one product, you may want to design serial numbers that are product specific (e.g. #####-#####-PRO-##### and #####-#####-STD-#####). Otherwise, if your two separate products generate enough serials, the chance exists that the same number may be generated for both products. As well, differentiating serial numbers according to product type ensures that human beings can differentiate between the numbers.

The second thing to consider is the length of the serial numbers. There is a delicate balance between the security of a serial number and the ease of entering it. A user will not want to enter a hundred-character alphanumeric string. Also, with increased length comes the increased chance the user will mess up a character in the serial number and call your support number unnecessarily.

Importing Serial Numbers

If you already have a list of serial numbers that you wish to use with your installer, you can easily import them into Setup Factory 7.0. This is accomplished by choosing Commands > Import on the Serial Number List Properties dialog. Setup Factory 7.0 can import serial numbers from a text file with one serial number per line.

Alternatively, if your serial numbers are in a spreadsheet program such as Microsoft Excel, it is possible to copy and paste the serials directly into the list.

Exporting Serial Numbers

Once you have serial number lists created, there are many things you can do with them. For example, you could import the lists into a label-printing program to create serial number labels to ship with your product. Or, you could import them into a database to keep track of your customers.

To import your serial number lists into an external program, you first have to export them from Setup Factory 7.0. This is accomplished by choosing Commands > Export from the Serial Number List Properties dialog. Setup Factory 7.0 exports the serial numbers into a text file, with one serial number per line.

As another option, you can copy and paste the serial numbers right from the Serial Number List properties dialog into a spreadsheet program such as Microsoft Excel.

The Verify Serial Number Screen

There are two steps required when adding serial number verification to your installation. The first is to create the serial number lists to be included with your installer. The second step is to prompt the user for a serial number during the installation process. This is done by using a Verify Serial Number screen. The Verify Serial Number screen is a ready-made screen template that is included with Setup Factory 7.0. It can be added into your project by clicking the Add button on the Before Installing screens dialog.

The Verify Serial Number screen is a modified Edit Field screen with pre-made actions on its On Next event. Right out of the box, this screen will take any serial number provided by the user and compare it to every list in the installation. If you want, the actions can be easily customized to compare the provided serial number to only specific lists in your installation.

Note: You can create your own Verify Serial Number screen; using the ready-to-use solution included with Setup Factory 7.0 is only one option.

When a new screen is inserted into your installation project, it is inserted at the end of the list of screens. In Setup Factory 7.0, the order in which screens appear in the screen list is the order that screens will be displayed in your installation. Since we are treating a serial number screen as a ‘locked door’ to pass through before the installation can continue, it is good practice to place the serial number screen before any screens that you want only authorized users to see. A typical screen order would be: Welcome screen, License Agreement screen, Verify Serial Number screen, etc.

Serial Number Actions

The same actions that are used by the Verify Serial Number screen to validate the user’s serial number can be used to perform the same task elsewhere. You don’t even need to use a screen for this purpose; for example, you could use a Dialog.Input action to ask the user for their serial number in a popup dialog box. Or if the user’s serial number is stored in a registry key, you could retrieve that value using Registry actions and then use a SetupData.IsValidSerialNumber action to determine whether the serial number is valid for the add-on you’re installing.

The SetupData.IsValidSerialNumber action is incredibly easy to use…you just give it the name of the serial number list you want to search in, and the serial number you want to search for, and it returns ‘true’ or ‘false’ according to whether the serial number is valid.

Here's an example that demonstrates how easy it is to validate a serial number:

```
bValid = SetupData.IsValidSerialNumber("Main", strSerial);
if not bValid then
    Dialog.Message("Sorry!", Invalid serial number.");
    Application.Exit();
end
```

In this example, we validate the serial number that was previously stored in a variable named strSerial against a serial number list named “Main.” If the serial number is invalid, we display a quick message to the user and then exit the installer.

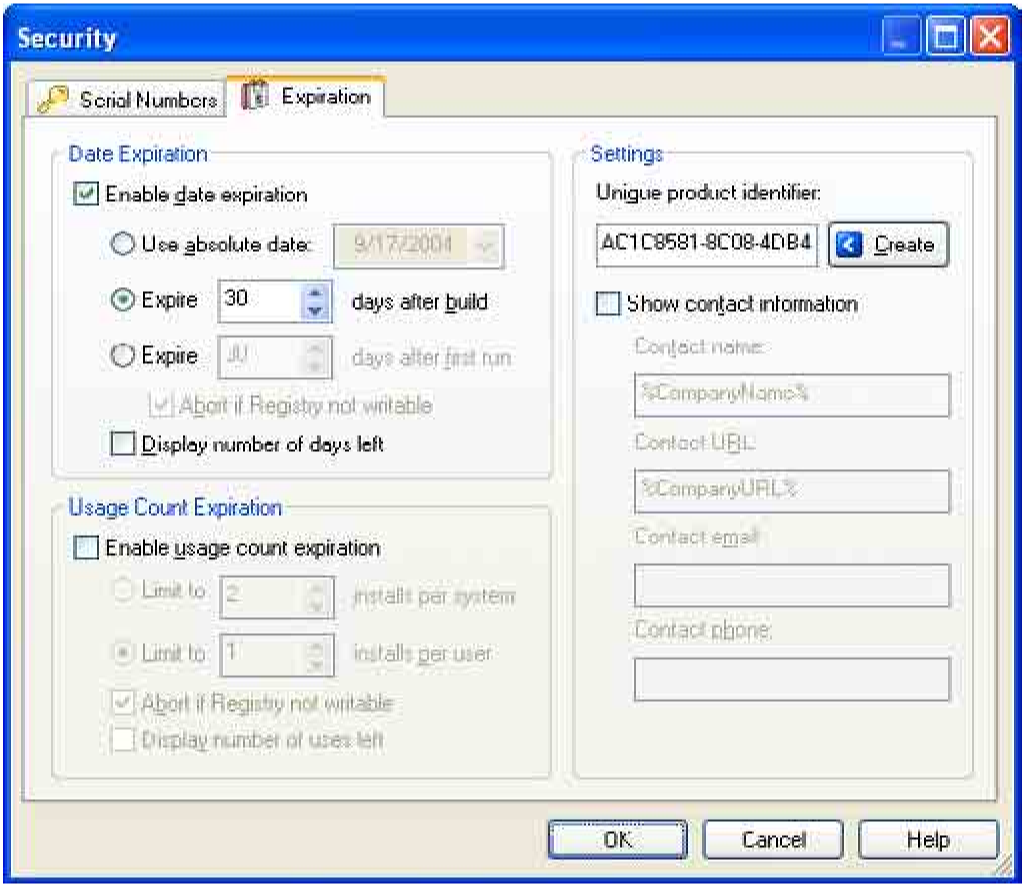
Date Expiration

One of the expiration criteria available in Setup Factory 7.0 is date expiration. When this security measure is implemented, your installer functions normally until the expiration date is reached, at which point it will cease to function. If the user runs the installer after the expiration date, they are presented with an error dialog, and the installer exits. (The message presented to the user can be customized.)

Note: Date expiration applies to the installer only and not to the software itself once it has been installed.

As you can see in the image on the next page, there are quite a few options for date expiration. Your installer can be made to expire at a specific date, such as the first of the month, or at the end of the current quarter. Or, it can be set to expire at a relative date, i.e., x days after the installer is built, or x days after the installation is first performed. As well, you have the option of displaying the number of days left to the user, or simply having the installer exit with no prior notice.

Note: The absolute date option, as well as the ‘expire x days after build’ option, can be used to protect the installer on more than one system. The ‘expire x days after first run’ option, however, is *system specific*. If the user runs the installer on one system, and then 50 days later runs it on another system, the installer will still function normally, even if it was set to expire 10 days after first run.



Date Expiration

☑ Enable date expiration

○ Use absolute date: 9/17/2004

● Expire 30 days after build

○ Expire 30 days after first run

☑ Abort if Registry not writable

☐ Display number of days left

Usage Count Expiration

☐ Enable usage count expiration

○ Limit to 2 installs per system

● Limit to 1 installs per user

☑ Abort if Registry not writable

☐ Display number of uses left

Settings

Unique product identifier: AC1C8581-8C08-4DB4

☐ Show contact information

Contact name: %CompanyName%

Contact URL: %CompanyURL%

Contact email:

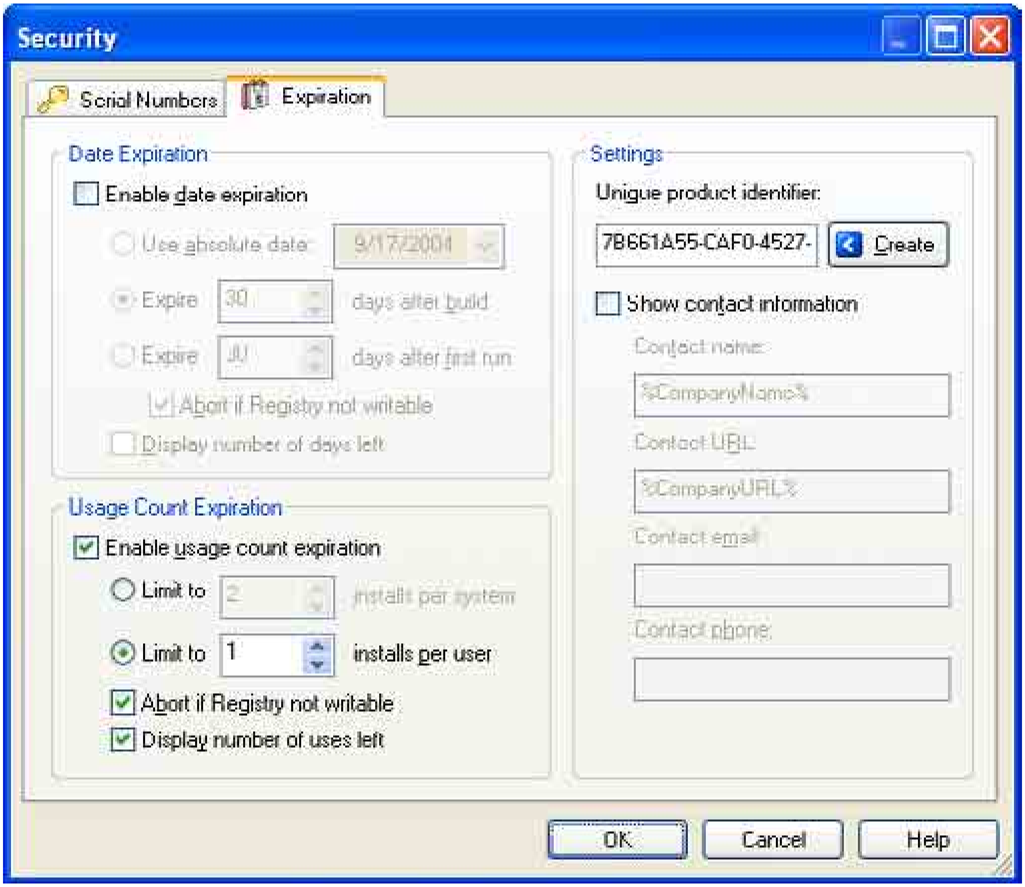
Contact phone:

Usage Count Expiration

The second expiration option available to you is usage count expiration. Your installer can be set to expire after x number of uses either on a per-user basis (NT systems only) or on a system-wide basis. This is useful when you want your users to have the most current version of your installer whenever they run the setup, or when you want to allow only one installation on a system, regardless of the number of users.

Note: Usage count expiration can be used by itself, or in combination with date expiration.

Here are some typical usage count expiration settings:



Security

Serial Numbers | Expiration

Date Expiration

☐ Enable date expiration

○ Use absolute date 9/17/2004

● Expire 30 days after build

○ Expire 30 days after first run

☑ Abort if Registry not writable

☐ Display number of days left

Usage Count Expiration

☑ Enable usage count expiration

○ Limit to 2 installs per system

● Limit to 1 installs per user

☑ Abort if Registry not writable

☑ Display number of uses left

Settings

Unique product identifier: 7B661A55-CAF0-4527- Create

☐ Show contact information

Contact name: %CompanyName%

Contact URL: %CompanyURL%

Contact email:

Contact phone:

You have the option of limiting the number of uses either by user or by system. As well, you can have the installation abort if the user does not have permission to write to the registry, since the usage information is stored there. Lastly, you can either display the number of uses left to the user, or have the installation expire with no prior warning.

Note: Usage count expiration is system specific. If a user runs an installer that is set to expire after one use, and then copies this installer to another system and runs it there, the installer will function normally on that other system.

In other words, this form of security does not expire the actual installation file; rather, it expires the installation on the current system by writing a value to the Registry. This process does not modify the setup file at all—the setup file will be the same after 100 uses as it was when it was first run.

Log Files

Setup Factory 7.0’s built-in logging function can quickly become a developer’s best friend. It is enabled by default, and contains all the information that you as a developer will need to diagnose a setup gone bad, should that situation arise. Everything that happens during the setup is logged, from which screens were displayed, to what files were installed and where.

Beyond its basic components, the installation log file is fully customizable. You as the developer can choose what information is logged, and what information is not. If you require even more detail than what the built-in logging capabilities provide, you can add custom lines to the log at any point during your installation.

Using log files is invaluable to developers for diagnosing problems. And let’s face it, no matter how much testing is done, there will always be one end user who hits a snag. Utilizing log files, you can find out exactly where that snag occurred, and fix the problem. As a result, you save on support costs and keep your customer happy.

Since all this information is (or can be) written to the log file, you may be wondering if log files pose a security risk. In general, no sensitive information is written to the log files by default. Unless you specifically write such information out using actions, the log files should be pretty safe. (Naturally the definition of what could be considered sensitive information may vary from one developer to the next. The best course of action, of course, is to test your installer with log files enabled, and see what kinds of information gets logged—both when the setup completes successfully, and when it encounters errors. If any information that you consider to be sensitive is revealed in the logs, you can adjust the default logging behavior, or turn the logging feature off.) Overall, since so little sensitive information is written to the log files, the risk is typically very low.

Furthermore, nothing is stored in the log file that the user couldn’t access using third party setup-monitoring software. In most cases, the usefulness of the log file far outweighs any perceived ‘security dangers’ that may exist. The log file contains information vital to the diagnosing of installation issues, and by default does not contain any information that is not available to a user by other means, should they desire it. Since you can specify where the log file is created, and can control what information is included in it, there is no reason not to use the logging feature.

Ultimately, the choice of whether or not to use the log file, and how much information should be in that log file, is up to you.

Data Integrity

Whenever using a process that deals with files, whether compression or simply copying, it is important to verify the integrity of the source files. After all, the quality of a copy depends on the quality of the original, and if the original file is ‘broken,’ it is not possible for the copied file to work.

Setup Factory 7.0 handles file integrity verification by using CRC32 checksums. At runtime, the setup checks each file as it is installed onto the user’s system. If it fails the CRC32 check, an error message is displayed to the user.

Note: CRC values are calculated using an algorithm known as the Cyclic Redundancy Check, or “CRC” for short. Basically, this involves generating a 32-bit number (or “CRC value”) from the contents of a file. If the contents of a file change, its CRC value changes as well. This allows the CRC number to be used as a “checksum” in order to identify whether or not the file has changed. It also allows you to distinguish between different versions of a file by comparing its CRC value to the CRC values of the originals.

File Encryption

By default, the files that get stored in Setup Factory 7.0’s data archive are compressed using a proprietary compression method. Although the proprietary nature of the compression scheme offers some security (e.g. a user can’t just extract the files using a third-party tool like WinZip), it is not intended as a security measure.

If you need to fully encrypt any files, you can add Blowfish encryption to your projects using the Crypto plugin from Indigo Rose. For more information on the Crypto plugin, please visit the Indigo Rose website.

Note that it is equally (if not more) important to secure your software than it is to secure the installer. If you only secure the installation, all it takes is one user to install the files “legitimately” for the protection to be removed, leaving the door wide open for that user to share the files against your wishes—or for an intruder to steal those files from the user’s unsecured system.

When planning your deployment, it is wise to view the installer as an open vehicle for the files, and to focus on protecting the software after it’s installed.

Internet Authentication

A very advanced method of securing an installer is to query a web database using HTTP.Submit actions. For instance, you could use a server-side script (e.g. a PHP script at your web site) to validate the user’s serial number by checking it against the list of serial numbers that have already been used. If everything checked out okay, the server-side script would send back a special “key” to unlock the rest of the installation. If the server-side script rejected the user’s serial number, the installer would abort the installation.

Note: Traditionally, software installers don’t use such elaborate protection methods; usually when such authentication is in place it is implemented in the software itself, in order to continue protecting the software from being shared after the installation. However, this feature of Setup Factory can be used to securely distribute non-executable files, where the distribution itself is the only part that you can secure.

Tip: Setup Factory even has full support for HTTPS transactions. Simply use the HTTP.SubmitSecure and HTTP.DownloadSecure actions which communicate with the server using Secure Sockets Layer (SSL) instead of regular unencrypted HTTP.

Post-Installation Security

Once your software is installed, any security measures taken to protect the setup do not protect the installed software product. Once your software is installed, Setup Factory 7.0’s job is done. To ensure the security of your software product after installation, you may want to consider a third party security solution. There are many solutions on the market designed to protect your software product beyond the installation stage.

Important Considerations

When planning out product security, it is important to keep in mind that given enough time and resources, even the best security measures can be circumvented. This fact can be illustrated with real-world, billion-dollar security attempts, such as DVD encryption and secure government communications. Given enough time, even these security measures can be 'cracked.'

In addition to securing your installation, and perhaps securing your installed application, you may want to consider controlling the distribution of your installation files. After all, a software thief can't steal something he or she can't find.

Serial numbers can be shared, computer clocks can be set back, and encryption can be broken. The advantage of imposing installation security checks is to make the task of stealing your software more difficult. Making a setup expire, by whatever criteria you choose, is a way to help control your source files. Using serial numbers is a way to track your users, and can be used for such purposes as limiting product support to registered users only.

One final point to keep in mind is that *too* much security can be a bad thing. While it is important to secure your application to prevent against unauthorized use, it is equally important to allow legitimate users a painless install.

Chapter 9:

Dependencies

There are countless software creation tools available to developers today, a number rivaled only by the amount of software dependencies that exist. Whether it’s the Visual Basic runtime files, or a complex, proprietary set of database libraries, software created to use various technologies will not function without them, and are thus dependant on these technologies.

Enter Setup Factory 7.0’s dependency modules. These self-contained wonders of programming science seamlessly integrate with Setup Factory 7.0, allowing any developer to ensure that their end users have the required dependencies. If they don’t, Setup Factory’s dependency modules will make sure the technologies are installed before the setup is allowed to continue.

9

In This Chapter

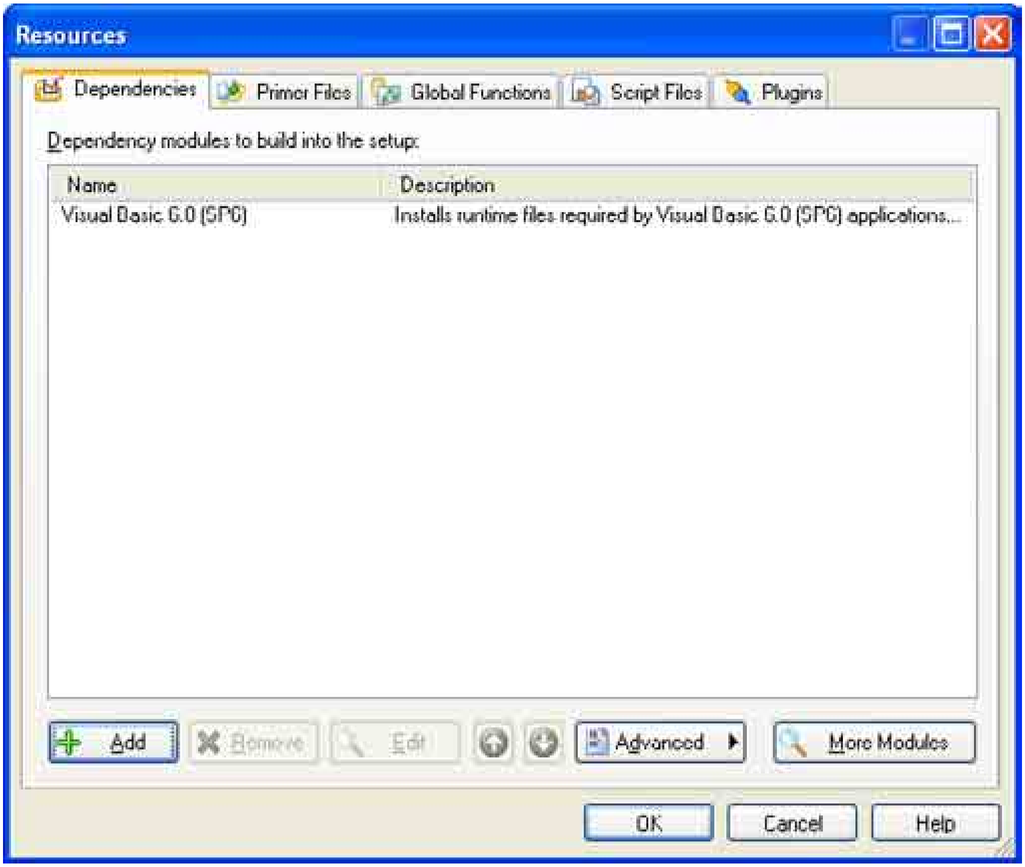
In this chapter, you'll learn about:

- Dependencies
- Dependency modules
- Why you should *always* read the dependency module information
- Detection scripts and installation scripts
- Special early-extracted dependency files

What are Dependencies?

Many applications created with development tools such as Visual Basic require certain technologies to be present on the system in order for the application to run. These technologies are usually files that must be installed alongside the application in order for the application to function. Dependencies range in scope from including small runtime files to requiring the presence of a properly configured local database server. Basically, anything outside of your actual application that your application depends on is a dependency.

Note: Dependencies as they relate to Setup Factory 7.0 are sometimes referred to as runtime modules.



Dependency modules to build into the setup:

Name	Description
Visual Basic 6.0 (SP6)	Installs runtime files required by Visual Basic 6.0 (SP6) applications...

Dependency Modules

Setup Factory 7.0 was built with application dependencies in mind, and includes support for dependency detection and installation in the form of dependency modules. Dependency modules are self-contained modules for installing technologies needed by the main application being installed. They consist of a script that detects whether the technology is already installed, and a script (and optional files) that will install the technology if it isn't already installed or needs to be updated. This detection and installation occurs at the beginning of the installation process, before the installation of the main product takes place.

Adding Dependency Modules

Adding a dependency module to your installer is as easy as clicking your left mouse button five times. First, choose Resources > Dependencies (that's 2 clicks). Now, click on Add. Lastly, click the dependency module you want to add, and click OK. You have just added a dependency module to your setup, which will be installed before your actual application files are.

Removing Dependency Modules

Removing a dependency module is even easier than adding one! Instead of five left clicks, this task requires only four. First, choose Resources > Dependencies, as you did to add the module you now wish to remove. Second, click on the dependency module to be eliminated, and click Remove. Yes, it's that easy!

Note: Because Setup Factory's dependency modules are self-contained, they can be added and removed as described. There are no files to worry about, nor actions to create. Simply point, choose, and click.

Adjusting the Order of Dependency Modules

While most dependencies are completely standalone, some dependencies may have dependencies of their own, and will not function unless another dependency is already installed. If this is the case, you will want to adjust the order of the dependency modules within your Setup Factory project to ensure that the modules are installed in the correct order. In other words, if dependency A requires dependency B, you will want to make sure that Setup Factory checks for dependency B first.

Tip: Dependency modules are detected and installed in the order they are listed in on the Dependencies tab of the Resources dialog. You can adjust this order by selecting a dependency module in the list and clicking the Up or Down button to move it up or down.

Getting More Modules

A plethora of dependency modules come with Setup Factory 7.0, but if you require a dependency that was not included with Setup Factory, give the More Modules button a try. Clicking the More Modules button will open the Indigo Rose website in your default browser and take you directly to the Setup Factory dependency module page. As more dependency modules are developed, they will be posted there.

Note: It is possible to create your own dependency modules, too. To start a new module, click the Advanced button and choose Create New Module. Name the new module accordingly, and edit its properties to configure the scripts and files that the module will use to detect and install the technology.

Dependency Module Properties

Once you have added a dependency module to your project, you can view and change its settings through the Dependency Module Properties dialog. You can get to this dialog by selecting a module in your project and clicking the Edit button, or by double-clicking on a dependency module within your project.

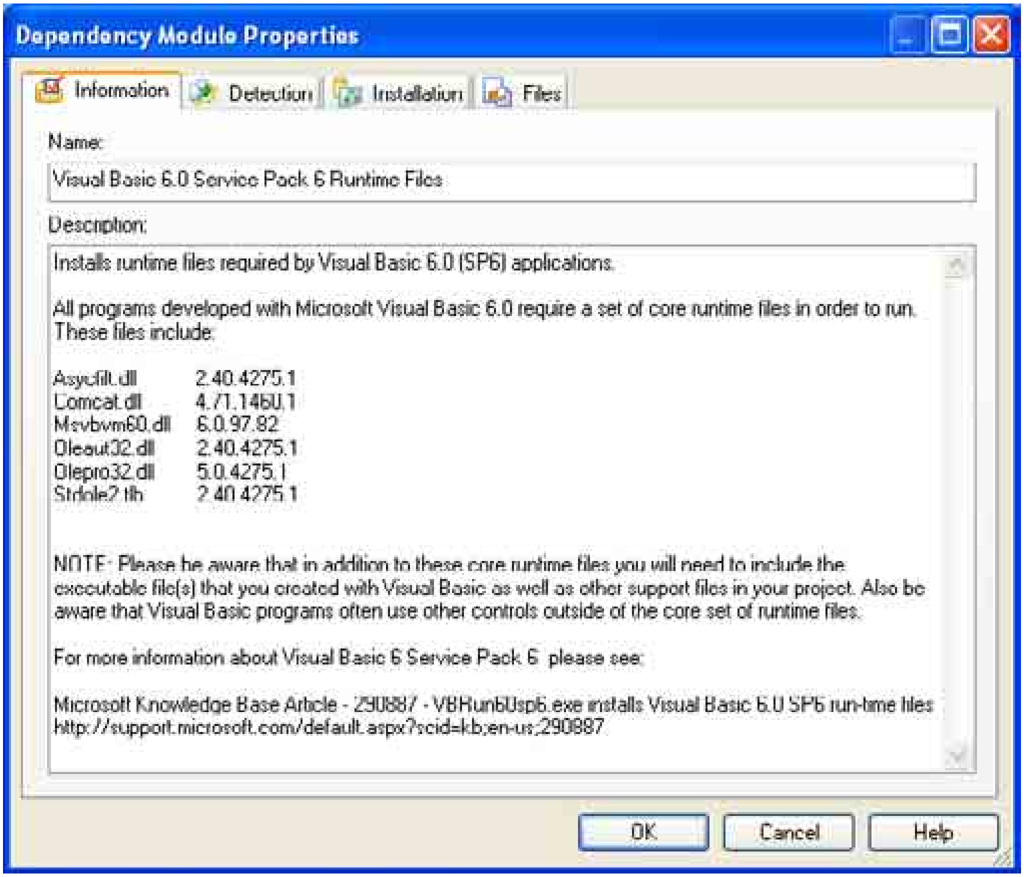
The Dependency Module Properties dialog has four tabs:

- Information
- Detection
- Installation
- Files

Information

The Information tab contains information about the current dependency module and often contains important instructions that you need to follow in order for the module to function as expected. If there are any related articles and documents that you should review they will be referenced here as well.

Note: Some dependency modules make use of files or settings that you must acquire or configure in order for the module to work. Be sure to read all of the information on this tab whenever you add a dependency module to your project!



Name:

Visual Basic 6.0 Service Pack 6 Runtime Files

Description:

Installs runtime files required by Visual Basic 6.0 (SP6) applications.

All programs developed with Microsoft Visual Basic 6.0 require a set of core runtime files in order to run. These files include:

Asycfilt.dll	2.40.4275.1
Comcat.dll	4.71.1460.1
Msvbvm60.dll	6.0.97.82
Oleaut32.dll	2.40.4275.1
Olepro32.dll	5.0.4275.1
Stdole2.tlb	2.40.4275.1

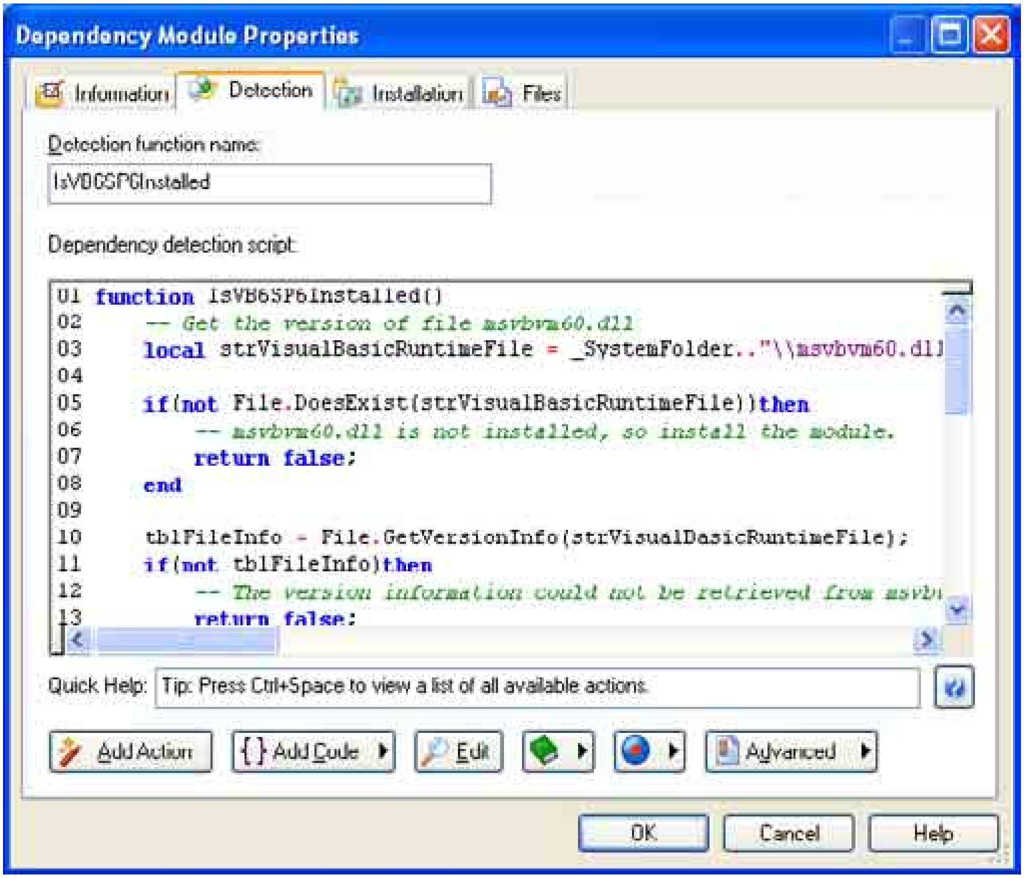
NOTE: Please be aware that in addition to these core runtime files you will need to include the executable file(s) that you created with Visual Basic as well as other support files in your project. Also be aware that Visual Basic programs often use other controls outside of the core set of runtime files.

For more information about Visual Basic 6 Service Pack 6 please see:

Microsoft Knowledge Base Article - 290887 - VBRun60sp6.exe installs Visual Basic 6.0 SP6 run-time files
http://support.microsoft.com/default.aspx?scid=kb;en-us;290887

Detection

The Detection tab is home to the detection script. This script is written in Lua, and is designed to determine whether the required technology already exists on the user's system. Only if the technology does not exist (or is determined to be an older version that should be updated) will the installation of the dependency continue.



Detection function name: IsVB65P6Installed

Dependency detection script:

```
function IsVB65P6Installed()
    -- Get the version of file msvbvm60.dll
    local strVisualBasicRuntimeFile = _SystemFolder.."\\msvbvm60.dll

    if(not File.DoesExist(strVisualBasicRuntimeFile))then
        -- msvbvm60.dll is not installed, so install the module.
        return false;
    end

    tblFileInfo = File.GetVersionInfo(strVisualBasicRuntimeFile);
    if(not tblFileInfo)then
        -- The version information could not be retrieved from msvb
        return false;
```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

The detection script must include at least one function that will be called by the installer. This detection function will determine what version (if any) of the required technology is installed, and return either true or false: true if the technology is "detected" (i.e. it is already installed and of a sufficient version) and false if the technology is "not detected" (it is either too old or missing altogether).

You can specify which function will be used by typing its name into the edit field at the top of the Detection tab. Some detection scripts may come with multiple versions of a detection function that you can choose from in this way, for instance to perform the detection using alternate methods.

Tip: You can have more than one function in the detection script. This is useful if, for example, you wanted to run different scripts based on a certain condition, such as the user's operating system. In this case, you would create one function that is called by the installer, which would then decide which specific dependency function to run.

The value that the detection function returns is what determines whether the dependency module's installation script is performed. The installation script will only be performed if the detection function returns false, indicating that the required technology was not detected. If the function returns true, the installer will proceed to the next dependency module in the list and run *its* dependency script.

Detection Happens Early

The detection script is run at the beginning of the installation, before the actual installation of your product takes place. At this stage of the installation, the only files that are available to the installer are the files that already exist on the user's system, any external files that were distributed alongside the setup, and any files that were included as primer files (which get extracted at the *very* beginning of the installation). None of the files that are in the Archive file list are available yet, since they are still packed inside the setup archive waiting to be installed.

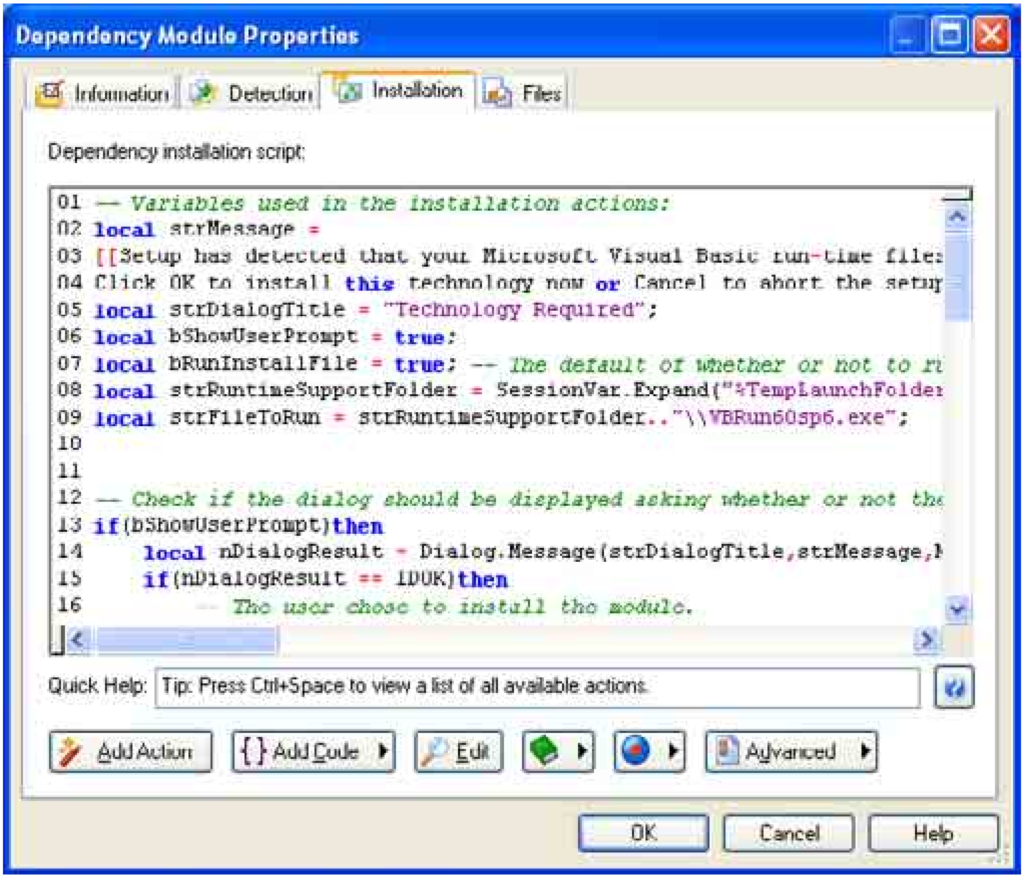
When modifying or creating a detection script, it's important to remember what files will be available at that stage of the installation.

Installation

The Installation tab contains the script that will be executed if (and only if) the detection function returns false—in other words, if the detection script determines that the required technology is not installed on the user's system.

If the detection script is the brains of the operation, the installation script is the brawn. Once given the green light by the detection script, the installation script charges ahead, performing all the tasks that are required in order to get the missing technology installed on the user's system.

Often this will consist of simply launching another self-packaged installer that will handle the installation of the dependency files. Of course, you can also use any of Setup Factory's built-in actions to perform whatever tasks you choose.



Dependency installation script:

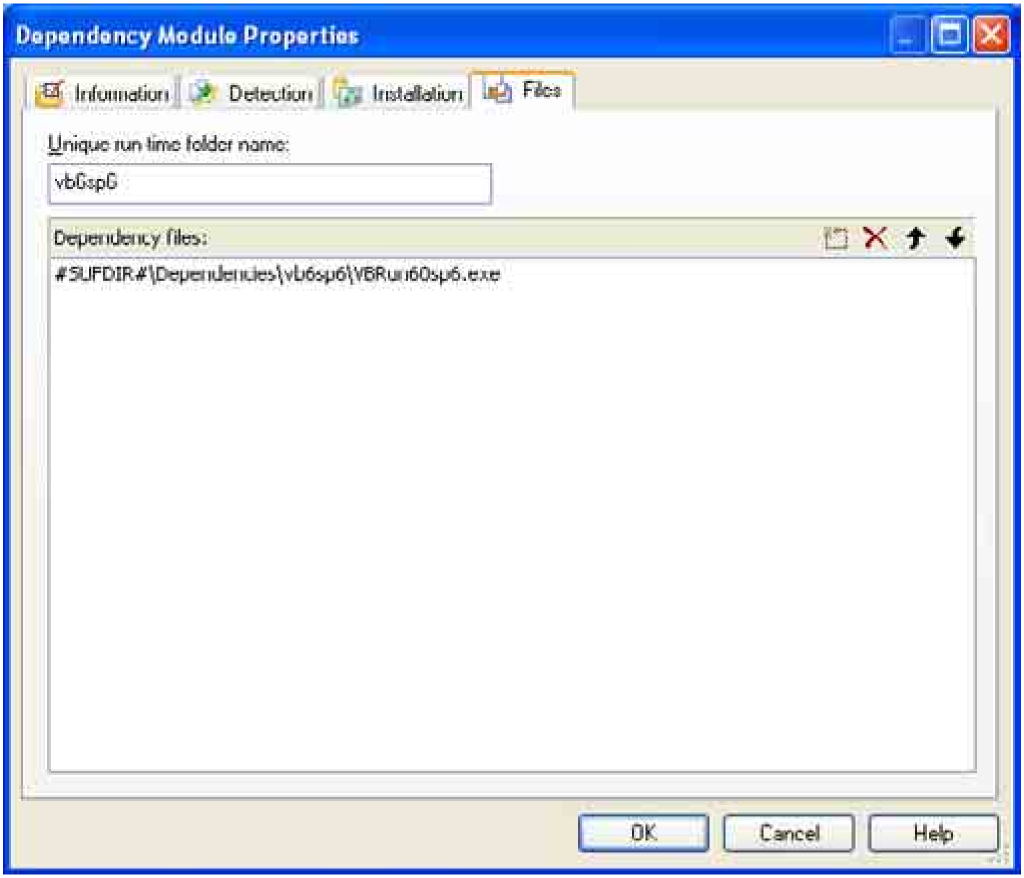
```
-- Variables used in the installation actions:
local strMessage =
[[Setup has detected that your Microsoft Visual Basic run-time file:
Click OK to install this technology now or Cancel to abort the setup
local strDialogTitle = "Technology Required";
local bShowUserPrompt = true;
local bRunInstallFile = true; -- The default of whether or not to ru
local strRuntimeSupportFolder = SessionVar.Expand("%TempLaunchFolder
local strFileToRun = strRuntimeSupportFolder.."\\VBRun60sp6.exe";


-- Check if the dialog should be displayed asking whether or not the
if(bShowUserPrompt)then
    local nDialogResult = Dialog.Message(strDialogTitle,strMessage,M
    if(nDialogResult == IDOK)then
        -- The user chose to install the module.
```

Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

Like the detection script, when the dependency module's installation script runs, it runs at the beginning of the installation, before the actual installation of your product takes place…long before any of the files in the Archive file list are installed. For this reason, most dependency modules are designed to include their own "dependency files" that will be extracted before the installation script is run. These dependency files are listed on the Files tab of the Dependency Module Properties dialog.

Files

The Files tab contains a list of files that are required by the installation script. These files will be extracted if (and only if) the detection script determines that the required technology is not installed. This essentially allows the dependency module to bring its own files along for installation script to use.



Dependency Module Properties

Unique run time folder name: vb6sp6

Dependency files:
#SUFDIR#\Dependencies\vb6sp6\VBRun60sp6.exe

The files listed on the Files tab typically consist of a setup executable that will install the required technology and any additional files needed by the installation script.

Note: If the detection script determines that the technology is already installed on the user’s system, the files listed on the Files tab will *not* be extracted.

Files on the Files tab are extracted to the user’s temp folder in a uniquely named subfolder. This is a subfolder within the installer’s own unique folder in the user’s temp folder. (Each installer creates a uniquely-named subfolder in the user’s temp folder if it needs a place to extract temporary files. This is done to avoid overwriting existing files that the user may have stored in the temp folder, or that might have been left behind by a previous installation.)

For example, if the unique run-time folder name was vb6sp6, the path that the files would be extracted to would be something similar to:

%TempFolder%\\_ir_sf7_temp_0\\vb6sp6

Constants in Dependency File Paths

The paths in the list of dependency files often contain constants like #SUFDIR#. #SUFDIR# is a built-in constant that gets replaced at build time with the path to the Setup Factory program folder on your development system—i.e., where the Setup Factory design environment is running from. This allows the built-in dependency modules to reference files that are located in the Setup Factory application folder, without knowing the exact location that you chose to install Setup Factory to.

Chapter 10:

Uninstall

Uninstallation: an inevitability that, though sad, is a fact of life. With Setup Factory 7.0, the task of removing your labor of love from a user's system is easy to configure, and in many cases, works with little to no involvement on your part.

Yet if you are performing many advanced tasks, the uninstallation is entirely customizable, from what screens are displayed, to what actions are performed, and when.

10

In This Chapter

In this chapter, you’ll learn about:

- What happens during an uninstallation
- What uninstallation tasks you need to implement yourself
- Configuring the uninstaller
- The uninstall configuration file
- Control panel options, uninstall shortcuts, and uninstall log files
- Customizing the uninstall interface
- Uninstall actions and events

What is an Uninstaller?

The uninstaller is the Ying to the installer's Yang. Everything created, installed, or otherwise modified by the installer is undone by the uninstaller. The best uninstaller leaves no trace of the original software product on the user's system and is easy to use.

Most programs list their uninstallers in the "Add or Remove Programs" control panel applet, which you access by choosing Start > Control Panel > Add or Remove Programs. Each item in Add or Remove Programs represents an already-installed program that can be uninstalled. (Some programs also provide a shortcut to their uninstaller in the same place where their program shortcuts are found on the Start menu.)

By default, Setup Factory 7.0 includes an uninstaller with every installation built. This default uninstaller is accessed through the Add or Remove Programs dialog.

What Happens During an Uninstallation

In a nutshell, during an uninstallation everything that was done automatically by the installer is undone. Files from your main file list are removed, shortcuts created by the installer are deleted, and folders that were automatically created during the installation are destroyed.

Note: The uninstaller does NOT remove files created or copied using actions, nor does it remove shortcuts, folders, or registry entries created via actions. By default the uninstaller only removes what the installer created automatically. Anything done with actions in the installation will need to be undone with actions in the uninstallation.

Tip: You can use UninstallData actions to add items to the list of files, folders, shortcuts, and registry entries that will be removed by the uninstaller.

By default, every file in your project is removed from the user's system by the uninstaller. However, you can bypass this on a file-by-file basis by adjusting either the shared file or never remove options in the file's properties. Any file for which the never remove option has been enabled will not be removed by the uninstaller.

If a file is specified as a shared file, then in addition to installing it, a usage count is set for the file. For example, if you have two separate products that share a DLL, when the user installs the second product, the usage count for that DLL is increased by one. When one of the products is removed, the usage count is decreased by one. The file is only removed from the system if it has a usage count of zero, which indicates that no other currently installed program needs it.

Going Beyond the Default Behavior

Unless you have made system changes using actions, the default behavior of the uninstaller should be sufficient. However, if you have made any changes to the user's system using actions, those changes must be reversed using actions during the uninstall.

For example, if you created a bunch of registry keys using Registry actions and made a couple of shortcuts using the Shell.CreateShortcut action, then you have to reverse these changes by using their counterpart actions (for example, Registry.DeleteKey and Shell.DeleteShortcut) in the uninstaller.

Of course, the uninstaller doesn't have to be limited to simply undoing things that were done by the installer. You can make it do anything you want; for example, you could present the user with an option to extend their license at a discount, or submit a message to an authentication system on your web site that will allow that user to install the software somewhere else.

One of the great things about Setup Factory 7.0 is how customizable it is. Installations can be created for almost any need. The same is true for uninstallations created with Setup Factory 7.0. The uninstallation user interface, combined with actions, can be customized to handle any situation.

Configuring the Uninstall

By default, Setup Factory 7.0 creates an uninstaller for your project that handles the basics automatically. Any files that are installed (without using actions) are marked to be uninstalled automatically, and there is a standard uninstall interface already set up.

If you need more functionality than Setup Factory 7.0's default features provide, then you can fully customize your uninstaller. The uninstaller is in essence a project within your actual project; it has its own startup and shutdown actions and its own screens, complete with all expected events.

The Uninstall Configuration File

Setup Factory 7.0 utilizes an XML configuration file to keep track of the changes that the installer makes to the user's system during the installation, such as what files were installed and what shortcuts were created. This configuration file is dynamically created by the installer as it proceeds through the installation.

Tip: Although the uninstall configuration file is created during the installation, it can be altered manually at any time during the installation or uninstallation by using UninstallData actions.

The uninstall configuration file contains the following information:

- Path to the uninstall data file
- Paths of all files that were installed
- Session variables and values
- All shortcuts created
- Any folders created by the installation
- Any additional scripts that should run during the uninstallation

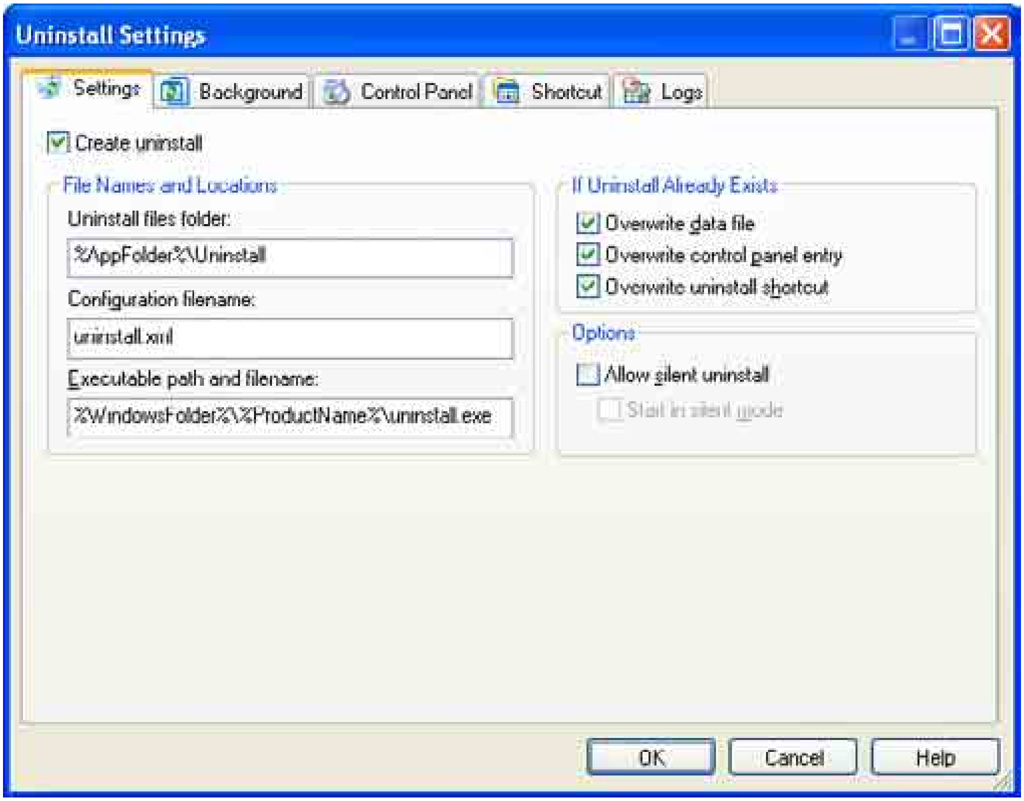
Note: The uninstall configuration file does **not** contain any information about tasks completed with actions. Any files, shortcuts, folders, session variables, etc. created with or modified by actions are not referenced in the uninstall configuration file unless you specifically add them to the file using UninstallData actions.

The uninstall configuration file is basically a list of steps for the uninstaller to follow in order to perform the uninstallation.

Note: The uninstall configuration file is different from the uninstall data file. The uninstall data file is an encrypted binary file containing all of your design time settings—screens, actions, etc.—and cannot be changed outside of the Setup Factory 7.0 design environment.

Uninstall Settings

The Uninstall Settings dialog can be accessed by choosing Uninstall > Settings from the menu. Typical uninstall settings can be seen here:



☑ Create uninstall

File Names and Locations

Uninstall files folder: %AppFolder%\Uninstall

Configuration filename: uninstall.xml

Executable path and filename: %WindowsFolder%\%ProductName%\uninstall.exe

If Uninstall Already Exists

☑ Overwrite data file

☑ Overwrite control panel entry

☑ Overwrite uninstall shortcut

Options

□ Allow silent uninstall

□ Start in silent mode

The Create Uninstall Option

The Create uninstall option basically toggles the uninstall functionality. When this option is checked, your installer will create an uninstaller at runtime. When it is unchecked, no uninstaller will be created.

File Names and Locations

The File Names and Locations section of the Uninstall Settings dialog allows you to specify: the folder where the uninstall data and configuration files will be stored, the name that should be used for the uninstall configuration file, and where to store the uninstaller's executable.

Overwrite Options

The If Uninstall Already Exists section of the Uninstall Settings dialog allows you to control how your installer behaves when it finds existing uninstall settings on the user's system from a previous installation. For example, you can choose whether to overwrite the control panel entry with the new uninstall settings, or to leave the existing control panel entry alone.

Other Options

The Allow silent uninstall option controls whether the uninstaller can be run in 'silent mode' (with no interface showing) using a command line option.

The Show background window option makes your uninstaller run with a window in the background (covering the desktop). The settings for this background window are the same as the background window settings for your installer.

Control Panel Options

Setup Factory 7.0 creates uninstalls that are extremely easy to use. One user friendly aspect is a control panel entry—specifically an entry in the Add or Remove Programs dialog—which makes uninstallation a breeze. The settings for this entry are found by choosing Uninstall > Control Panel, which opens the Uninstall Settings dialog directly to the Control Panel tab.

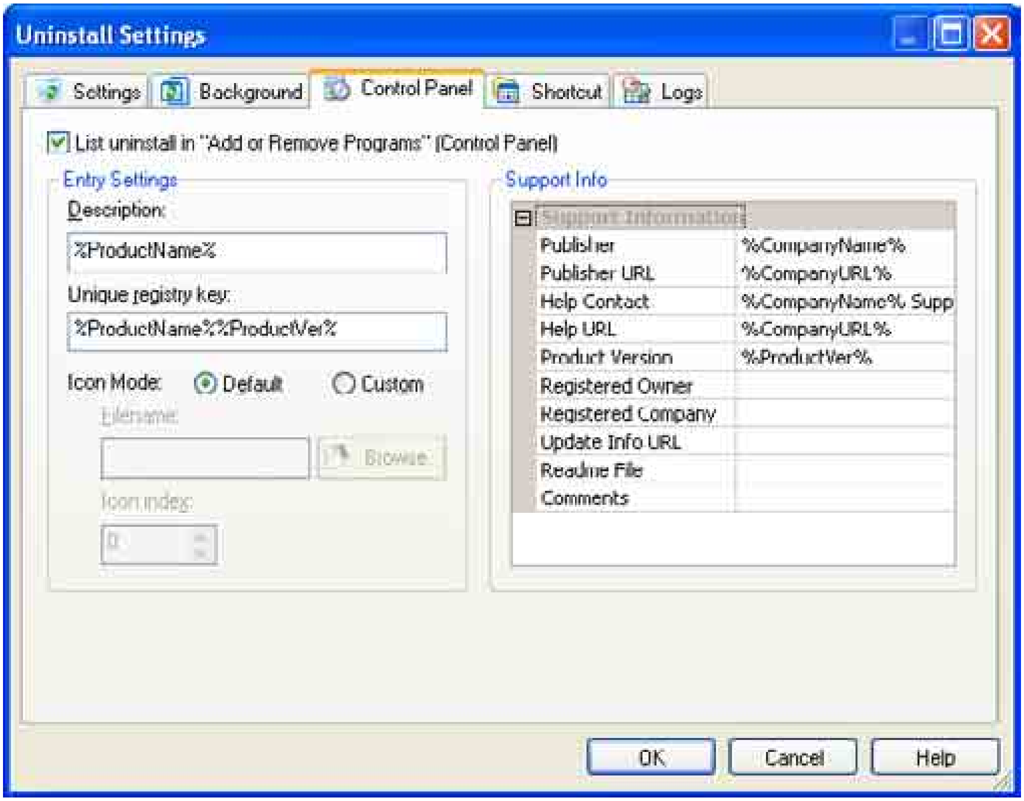
The first setting that you can configure on the Control Panel tab is whether or not to list the uninstall in the Add or Remove Programs dialog. Unchecking this option will prevent your installer from making this entry.

Assuming, however, that you choose to make this entry, the next settings to configure are the Entry Settings. This is where you can specify the description, Registry key, and icon to use for the control panel entry.

After the Entry Settings are configured, only the Support Info settings remain. The information you place here will be available from the Add or Remove Programs dialog and is often used to help the user identify your product.

Tip: For more information on any of these settings, see the Setup Factory help file.

Typical control panel options:



☑ List uninstall in "Add or Remove Programs" (Control Panel)

Entry Settings

Description: %ProductName%

Unique registry key: %ProductName%%ProductVer%

Icon Mode: ● Default ○ Custom

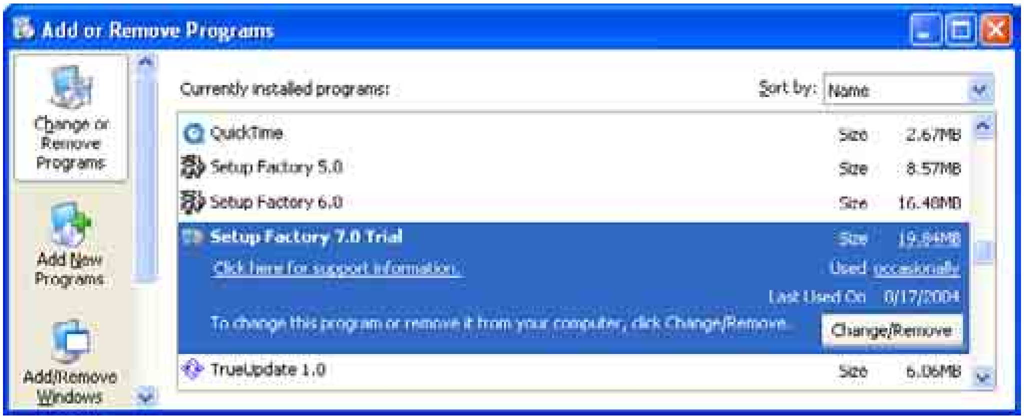
Filename:

Icon index: 0

Support Info

Support Information	
Publisher	%CompanyName%
Publisher URL	%CompanyURL%
Help Contact	%CompanyName% Supp
Help URL	%CompanyURL%
Product Version	%ProductVer%
Registered Owner	
Registered Company	
Update Info URL	
Readme File	
Comments	

A typical control panel entry:



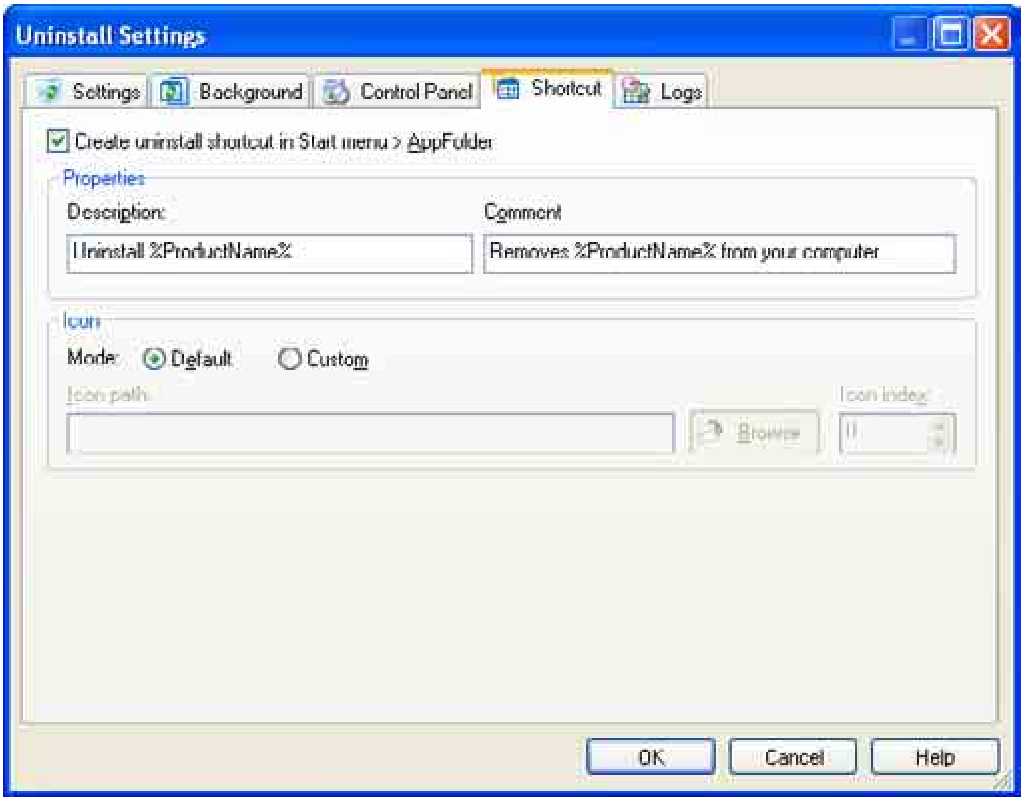
Currently installed programs: Sort by: Name

QuickTime	Size	2.67MB
Setup Factory 5.0	Size	8.57MB
Setup Factory 6.0	Size	16.48MB
Setup Factory 7.0 Trial	Size	19.84MB
Click here for support information.	Used	occasionally
	Last Used On	8/17/2004
To change this program or remove it from your computer, click Change/Remove.		
TrueUpdate 1.0	Size	6.06MB

Shortcut Icons

An alternative to creating an entry in the Add or Remove Programs dialog is to include an uninstall shortcut in the start menu. This option is found on the Shortcut tab of the Uninstall Settings dialog, and can be accessed from the design environment by choosing Uninstall > Shortcut. If enabled, this option will create an uninstall shortcut in the start menu alongside the other shortcuts created by your installation.

Typical shortcut options:



☑ Create uninstall shortcut in Start menu > AppFolder

Properties

Description: Uninstall %ProductName%

Comment: Removes %ProductName% from your computer

Icon

Mode: ● Default ○ Custom

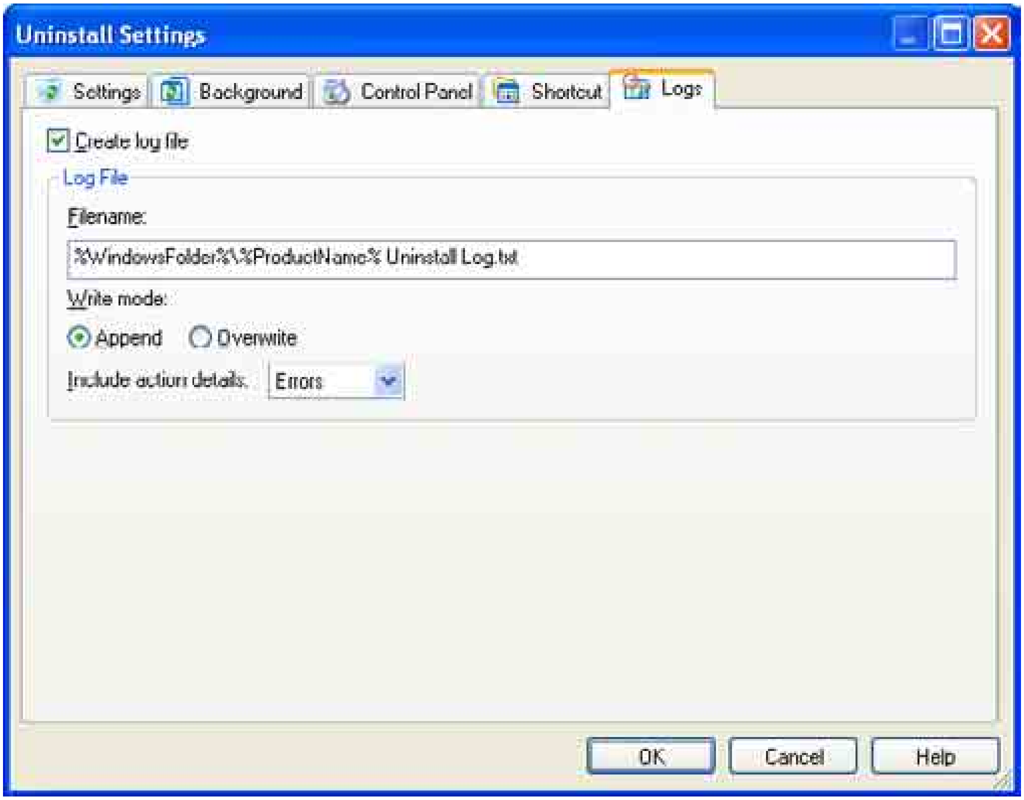
Icon path:

Icon index: 0

Log Files

In the event that your uninstallation fails, having a log file available to diagnose the problem is often extremely helpful. Knowing exactly where and why your uninstall failed are two very valuable pieces of information. Setup Factory 7.0 enables uninstallation logging by default.

You can configure the uninstall log file options or turn off this option entirely from the Logs tab of the Uninstall Settings dialog. You can access this dialog from the design environment by choosing Uninstall > Log Files.



☑ Create log file

Filename: %WindowsFolder%\%ProductName% Uninstall Log.txt

Write mode: ● Append ○ Overwrite

Include action details: Errors

The settings on the Logs tab allow you to configure where the log file is stored, how the data is written, and what data is written to the log file.

Note: Do not disable this feature unless you are absolutely positive that you will never need the log file. Regardless of how much testing you do, there is always a chance that your installer will produce unforeseen behavior on some user's system. The task of diagnosing such problems is made exponentially easier with a log file. If there is a security concern, keep in mind that you can set the log file to include no action details, and then use your own actions to output only the information you deem appropriate.

The Uninstall Interface

The uninstall user interface can be configured similarly to how the installer was configured. By default, the uninstaller will use the theme settings from the installer, but these can be overridden on a screen-by-screen basis if you desire.

You can control the look and feel of your uninstaller just as you created your installer by adding, removing and manipulating the screens to be included. The uninstaller uses its own screens, independent of the screens included in the installer. By manipulating the various properties of each respective screen, you can have an uninstaller that reflects the look and feel of the original installer, or you can create an entirely new appearance.

Tip: For more information on configuring screens, please see Chapter 3.

Note: Whether or not a background window is shown is controlled from the Uninstall Settings dialog. However, the actual settings for the background window are taken from the installer's background window settings. The uninstaller's background window settings cannot be configured independently.

Uninstall Actions

Using Setup Factory 7.0, you can quickly and easily create an installer to suit most general requirements. However the real power of Setup Factory 7.0 lies in the ability to use the free-form scripting engine, combined with a plethora of built-in actions, to perform whatever task you desire.

It is important to keep in mind that whatever tasks you perform using actions during your installation will not automatically be undone by the uninstaller. You must manually make use of the uninstall actions to undo whatever changes the installation actions made to the user's system.

With the above in mind, it is possible to automate some of the uninstallation tasks by making use of the UninstallData actions. These are a group of actions that allow you to interact with the uninstall configuration file directly, during both the installation and uninstallation. So instead of adding actions to the uninstall to reverse what your actions did during the installation, you can use UninstallData actions to modify the uninstall configuration file, and let the uninstaller take care of undoing the changes for you.

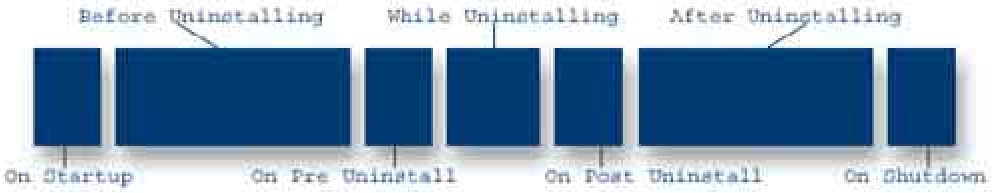
One advantage of this is that it allows you to put your “undo” actions in the installer, instead of in the uninstaller. This lets you keep the actions that undo your changes near the actions that made the changes in the first place, making it much less likely that you’ll forget to update the uninstall actions whenever the install actions change.

Tip: For more information on actions and how to work with them, see Chapter 4. The command reference is also an excellent resource for learning about actions and the action wizard.

Uninstall Events

As with the installer, there are four uninstall project events: On Startup, On Pre Uninstall, On Post Uninstall, and On Shutdown. These events are accessible by choosing Actions from the Uninstall menu.

It is up to you to decide where to place your actions based on the task that the actions are performing.



The On Startup event is the first event that is triggered in a Setup Factory 7.0 uninstallation. This event is one of the first things that occurs and is your first chance to make anything happen in your uninstallation.

The On Startup event occurs before any screens are displayed. This makes it a good place to perform pre-uninstallation tasks, such as making sure the end user is positive that they want to uninstall.

The On Pre Uninstall event is triggered just before the uninstallation enters the uninstall phase and begins uninstalling files from the end user’s computer. This event is fired right after the last screen in the Before Uninstalling screen stage is displayed.

The On Post Uninstall event is triggered after the uninstall phase of the uninstallation has completed, right before the first screen of the After Uninstalling screen stage is displayed.

The On Shutdown event is the last event that will be triggered in your uninstallation. It occurs after all screens in your uninstallation have been displayed and is your last chance to accomplish anything in your uninstallation. This is an excellent stage to complete any cleanup tasks, such as deleting temp files that you may have created earlier, or directing the user to your website.

Chapter 11:

Building and Distributing

Designing your installation is only half the battle. Once you have something tangible to work with, you must build and eventually distribute your installer. Luckily, Setup Factory 7.0 makes this process as seamless as the actual installer design process.

From choosing your distribution media to fully unattended build options, Setup Factory 7.0 makes building your project a pleasure, not a chore.

11

In This Chapter

In this chapter, you'll learn about:

- The build process
- The publish wizard
- Build configurations
- Constants
- Pre- and post-build steps
- Build-related preferences
- Testing and distributing your installer

The Build Process

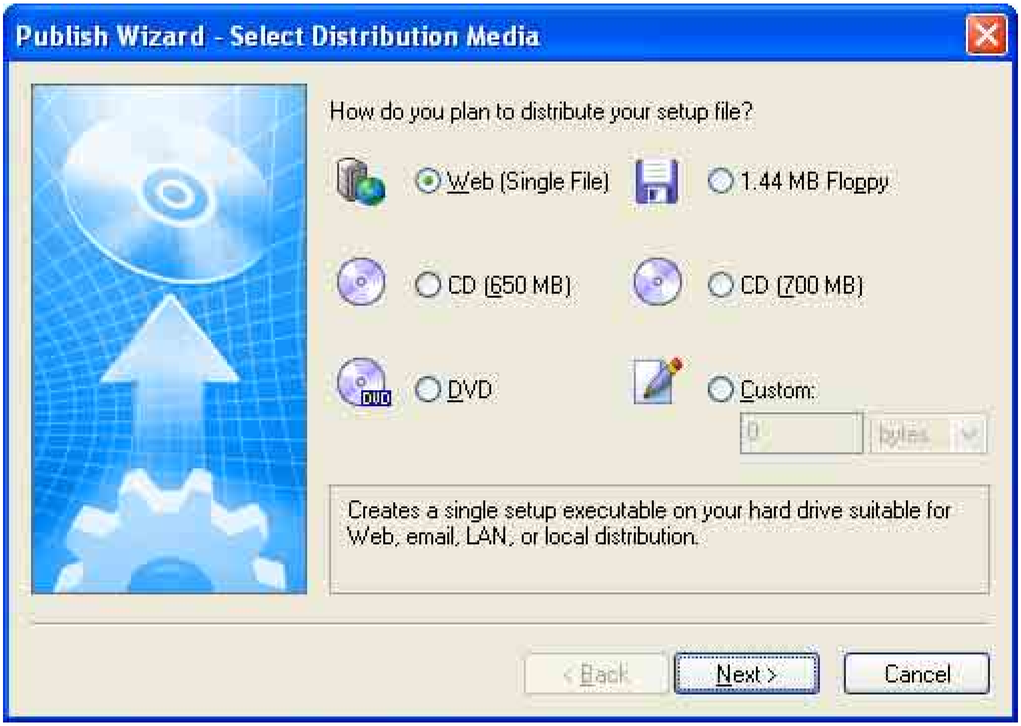
Once you have finished creating your Setup Factory 7.0 project, you must build it into the actual setup file (or files, if you're building a multi-part installer). This is similar to a programmer compiling code once the code is complete.

When you choose to build your project, Setup Factory 7.0 will first verify all files and folders that you have included. It will then convert any serial number lists you have into MD5 hash lists and encrypt them. It collects any fonts you have opted to install and converts any graphics into the setup file. Once the images have been converted, the language modules are processed, all the contained files are compressed, the configuration file is created, and the compressed setup is built.

The Publish Wizard

With the goal of making the build process as seamless as possible, Setup Factory 7.0 includes a Publish Wizard to assist you.

The first screen of the Publish Wizard allows you to specify which type of media you will be distributing your installation on. By identifying this, you are essentially telling Setup Factory 7.0 what the maximum size of any one created file should be. For example, if your setup is 1.15 gigs and you are distributing on 700mb CDs, Setup Factory 7.0 will split your installation into two files, ready to be burned.



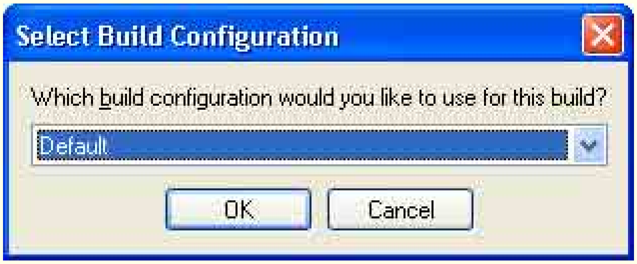
Publish Wizard - Select Distribution Media

How do you plan to distribute your setup file?

● Web (Single File)
○ 1.44 MB Floppy
○ CD (650 MB)
○ CD (700 MB)
○ DVD
○ Custom: 0 bytes

Creates a single setup executable on your hard drive suitable for Web, email, LAN, or local distribution.

Note: If your setup project contains more than one build configuration, the wizard will prompt you to choose which build configuration you want to use before it starts:

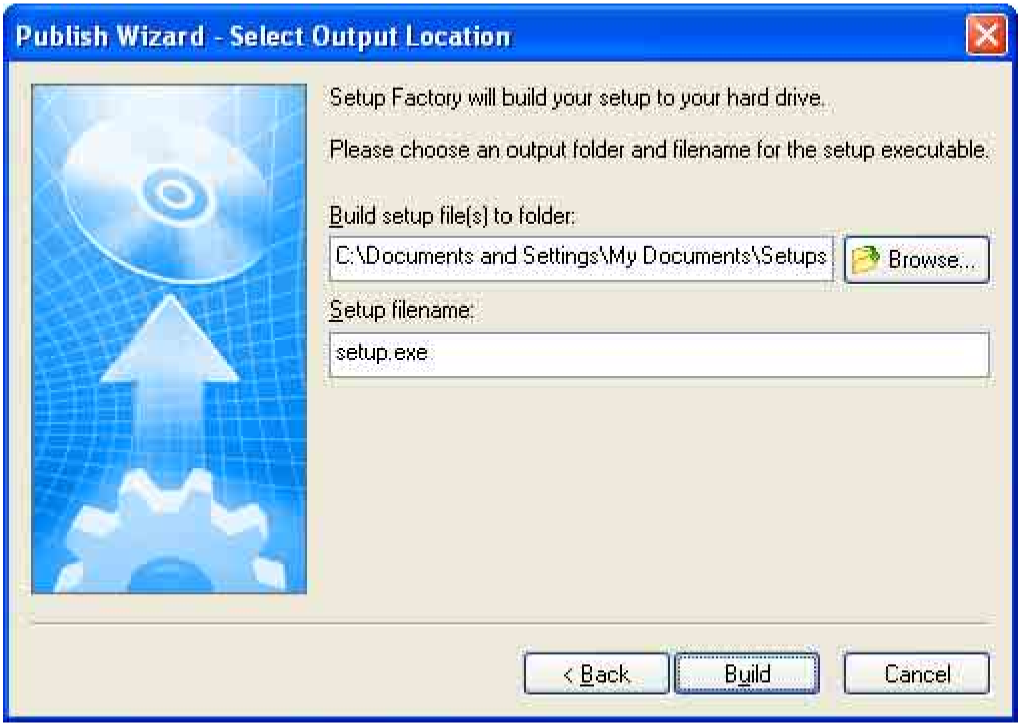


Select Build Configuration

Which build configuration would you like to use for this build?

Default

The second screen of the Publish Wizard prompts you for the location that you would like to publish to, as well as the name of the file that you would like to create.



Publish Wizard - Select Output Location

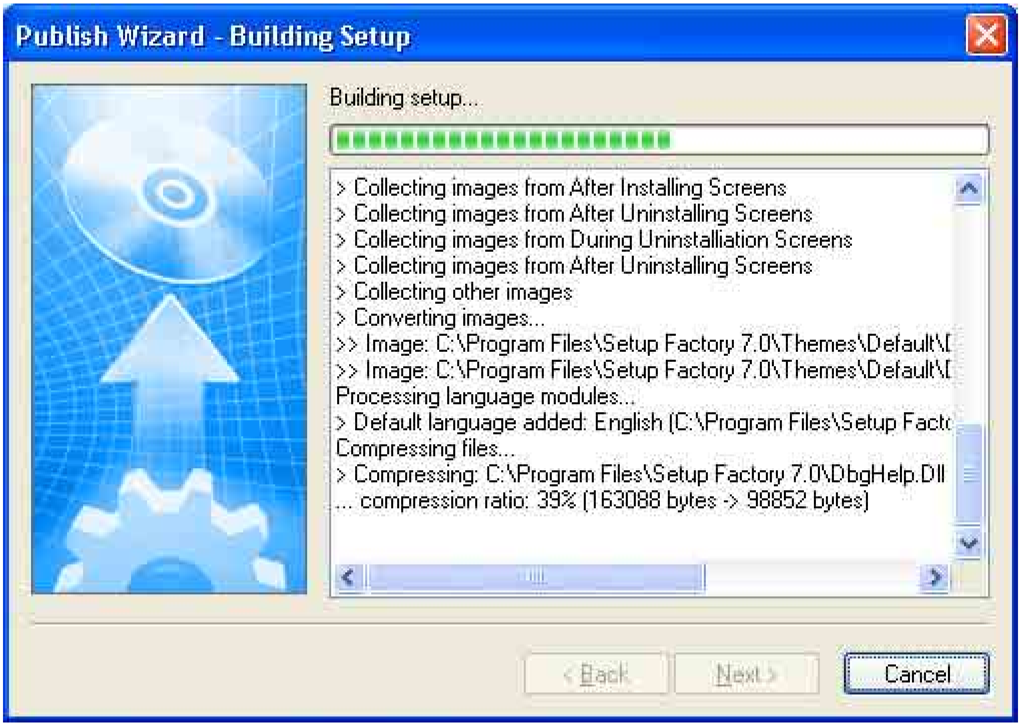
Setup Factory will build your setup to your hard drive.

Please choose an output folder and filename for the setup executable.

Build setup file(s) to folder: C:\Documents and Settings\My Documents\Setups

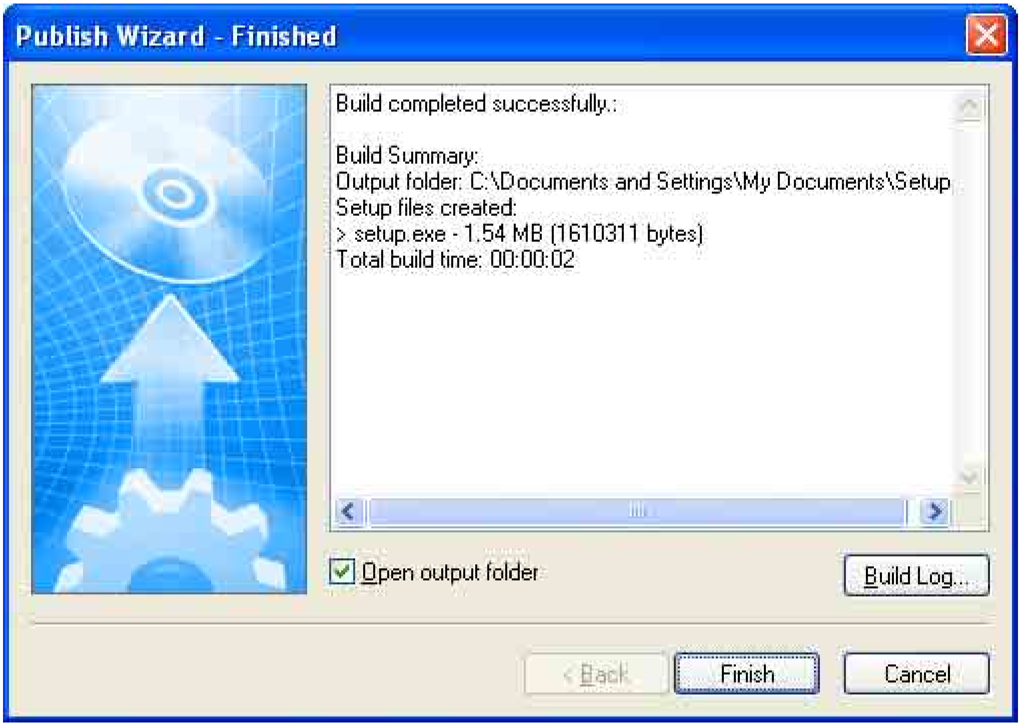
Setup filename: setup.exe

Once the Next button is clicked on the above screen, Setup Factory 7.0 begins to build the setup.



```
> Collecting images from After Installing Screens
> Collecting images from After Uninstalling Screens
> Collecting images from During Uninstalliation Screens
> Collecting images from After Uninstalling Screens
> Collecting other images
> Converting images...
>> Image: C:\Program Files\Setup Factory 7.0\Themes\Default\[
>> Image: C:\Program Files\Setup Factory 7.0\Themes\Default\[
Processing language modules...
> Default language added: English (C:\Program Files\Setup Facto
Compressing files...
> Compressing: C:\Program Files\Setup Factory 7.0\DbgHelp.Dll
... compression ratio: 39% (163088 bytes -> 98852 bytes)
```

Once the build process is complete, the following summary screen is displayed. This screen presents a summary of the build process and displays any errors. Once you are finished viewing this screen, you can simply click Finish. If the Open output folder option is checked, the output folder will be opened automatically when you click Finish.



```
Build completed successfully.:

Build Summary:
Output folder: C:\Documents and Settings\My Documents\Setup
Setup files created:
> setup.exe - 1.54 MB (1610311 bytes)
Total build time: 00:00:02
```

☑ Open output folder

Tip: If you need more information or want to trace an error, you can view the entire build log file by clicking the Build Log button.

Build Settings

The Build Settings dialog, accessible by choosing Settings from the Publish menu, allows you to set the defaults you would like to use for the build process. Basically, any setting affecting the final built file is found here.

When working with an installer that can be built into one file, changing the setup filename simply changes the name of the built file. However, when working with an installer that will be split into two or more segments, changing the setup filename also changes the names of each of the segment files.

The segment size setting allows you to specify the maximum file size that any installation file will have. If your complete setup is larger than the size specified, the final installation will be split into two or more files. This is especially useful if you are distributing your installation on fixed-size media such as floppy disks, CDs or DVDs.

Tip: Setup Factory 7.0 has pre-specified segment sizes to make distributing your installations on standard media such as 650mb and 700mb CDs, DVDs, and floppy disks a breeze.

Build Configurations

One of the many timesaving features of Setup Factory 7.0 is the ability to use build configurations within your setup project. When working with two similar installs, such as a Standard and Professional version of your software product, it is often easier to create one setup project and have only the required files and serial number lists for each respective installer included at build time.

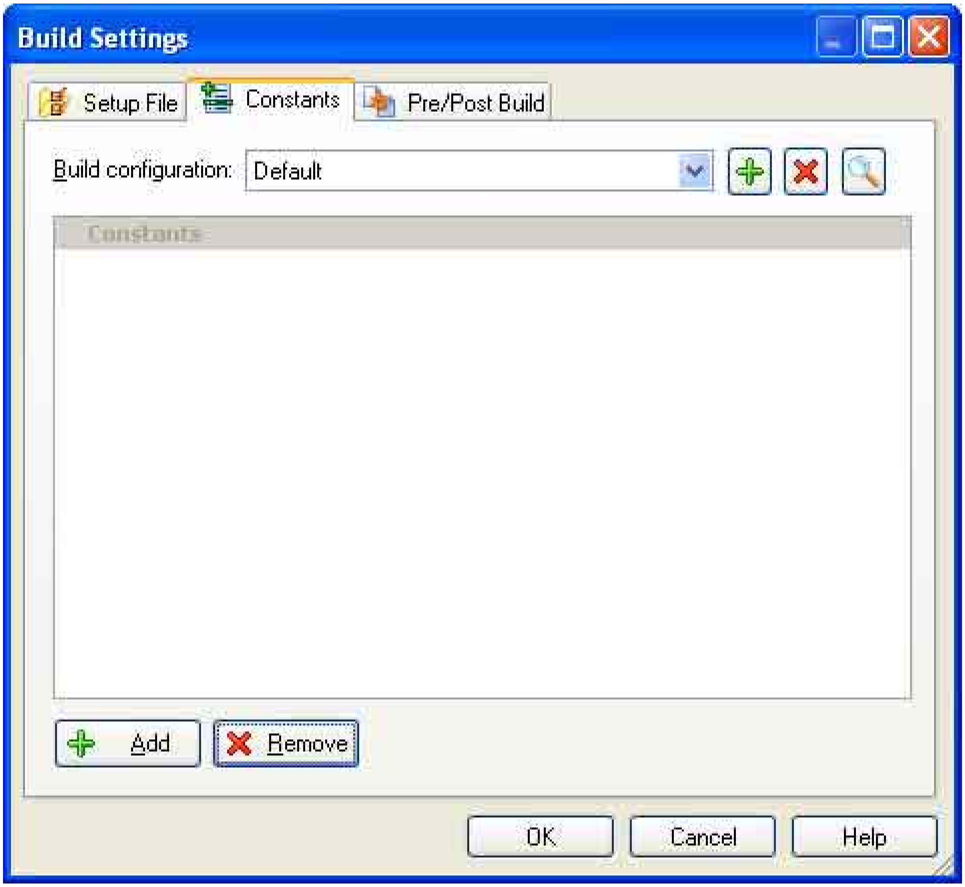
Build configurations can be added to the current project by clicking the Add button (➕) near the top of the Build Settings dialog. Once a build configuration is added, you can assign files, folders, and serial numbers to it from the main design window.

Files and folders can be associated with specific build configurations from their respective properties dialog, or they can be associated with 'All' build configurations. Serial numbers can be included as well with only specific build configurations by assigning the list to a build configuration in the Serial Numbers List Properties dialog.

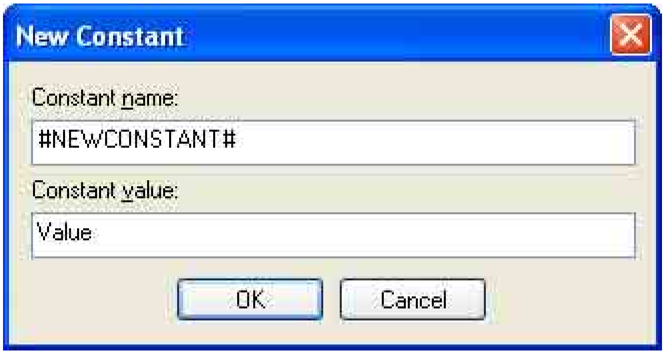
Constants

Constants are essentially design time variables. They are similar to session variables, but instead of being expanded as the user runs the installer, they are expanded when you build the project. In other words, when you build your project, all of the constants are automatically replaced by the values assigned to them.

You can define constants on the Constants tab of the Build Settings dialog, which you can access by choosing Publish > Settings.



Clicking the Add button displays a dialog where you can name the constant and give it a value.



Constant name:
#NEWCONSTANT#

Constant value:
Value

Each constant has a name that begins and ends with a # sign, and an associated value. The name will be replaced by this value throughout the project when the project is built. It’s exactly like a big search-and-replace operation that happens whenever you build the project.

Since each constant is essentially just a name that gets replaced with different text, you can use them just about anywhere. You can use them on screens, in file paths, in actions…pretty much anywhere that you can enter text.

Build Configurations

The values that you assign to constants are specific to each build configuration. In fact, this is what makes constants so useful. By assigning different values to the constants in each build configuration, you can make sweeping changes to your project by simply selecting a different build configuration when you build it.

Note: The build configuration determines which values are used when the project is built.

Here are some things you can do with constants and build configurations:

- use constants in the paths for your source files, and use different versions of the files for different build configurations
- use constants for the name of the product, so you can switch between building a full version and an evaluation version just by changing the build configuration
- use actions that test the value of a constant named #DEBUG#, and set it to true in a “Debug” build configuration and false in all other configurations. Then you can leave helpful debugging code in your scripts, and only have it performed when you build the Debug configuration. For example:

```
function MyFunc()
    nCount = nCount + 1;
    if #DEBUG# then
        Debug.ShowWindow();
        Debug.Print("Inside MyFunc() - nCount = " .. nCount);
    end
end
```

Unattended Builds

Constants are extremely useful for performing unattended builds because they can be changed using a command line argument. You can even use constants to specify the output location, and which build configuration to use. The possibilities are endless.

For example, you could create a batch file that would generate a unique installer for every one of your customers, with the customer's name showing up on one or all of the screens during the installation.

For more information on performing unattended builds of your project, search for "Unattended Build Options" in the Setup Factory help file.

Pre/Post Build Steps

There may be instances where you want to run a program either before or after your installer performs its tasks. Setup Factory 7.0 makes this process easy through Pre and Post Build Steps. On the Pre/Post Build tab of the Build Settings dialog, you can set a program to Run Before Build, and a program to Run After Build.

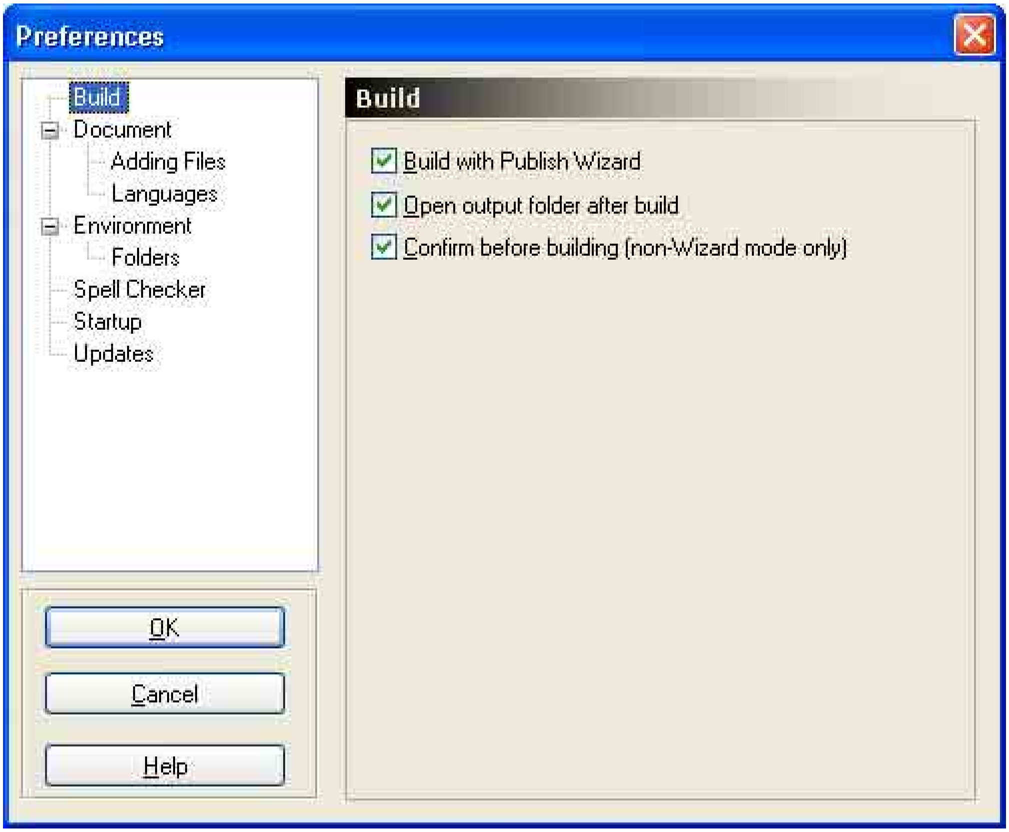
The uses of this feature are limited only by your imagination. You could, for example, chain several setups together by having one setup call another through the Run After Build option. Or you could have your setup launch a registration program that modifies a key in the registry without which your setup will not continue. The possibilities are limitless.

You can specify the path to the desired program in either of the Run Program fields. Any command line arguments that are needed should be specified in the Command Line arguments fields. If you want your installer to 'sleep' while the other program is open, check the Wait for program to finish running checkbox.

Preferences

To control how Setup Factory 7.0 handles the build process, you can modify the Build Preferences by choosing Preferences from the Edit menu. You can control whether Setup Factory 7.0 opens the output folder by default, disable the publish wizard and, when not using the wizard, control whether a confirmation dialog is displayed before the build process begins.

Typical build preferences can be seen here:



Build

☑ Build with Publish Wizard

☑ Open output folder after build

☑ Confirm before building (non-Wizard mode only)

Testing Your Installer

One of the most important and often overlooked steps when creating an installer is testing it after it has been built. You should test your installer on as many computers and operating systems as possible. Try it on every operating system that your software product supports. Windows 95, Windows 98, Windows 2000, Windows ME, Windows NT, and Windows XP all have both slight and major differences between them, and should be thoroughly tested. As well, test your installer on various OS configurations with different resolutions, color depth and font sizes, and on systems with a lot of hard-drive space and on those with a very limited amount of hard drive space.

If you have made use of Setup Factory 7.0’s multilingual support, be sure to test out each language in your install. If you included any dependency modules, test your installer on a virgin system to ensure that the dependency files install correctly.

Many problems with installers have simple causes, such as forgetting to include a file, or moving a file to an incorrect location. Compatibility issues can crop up when your software is not written for a particular operating system, or when the dependency files that are included in the installer do not function on a particular operating system.

If you do run into a problem with your installer, test it on as many systems as possible to narrow down the problem. You might discover a common factor between the systems that is causing the installer (or your software) to fail.

It is very important to test before you distribute. An ‘internal build’ of your installer is much easier to recall should a crippling problem arise than a public release is.

Tip: For information related to tracking down script-related issues, see *Debugging Your Scripts* in Chapter 12.

Distributing Your Installer

Once you have thoroughly tested your installer, it is time to distribute your software. After all, a beautifully created setup is an awful waste if your customers and clients cannot enjoy it! There are several ways to distribute your software, but the four main media are floppy disks, CD-ROMs, DVDs, and the Internet. Setup Factory 7.0 allows you to specify a segment size to effectively split your installer into smaller sized files. This is ideal for distributing an installer that is larger than the target media.

Preparing your installer for distribution is as easy as selecting your desired media type in the publish wizard, and publishing to a hard drive folder. Once the build operation has completed, simply copy each segment file (or just the setup executable if there is only one file) onto separate discs, and you are ready to go! Or, if you have opted to distribute via the Internet, simply take the generated setup file and upload it onto your website for your customers to download.

Chapter 12:

Scripting Guide

One of the powerful new features of Setup Factory 7.0 is its scripting engine. This chapter will introduce you to the new scripting environment and language.

Setup Factory scripting is very simple, with only a handful of concepts to learn. Here is what it looks like:

```
a = 5;
if a < 10 then
      Dialog.Message("Guess what?", "a is less than 10");
end
```

(Note: this script is only a demonstration. Don’t worry if you don’t understand it yet.)

The example above assigns a value to a variable, tests the contents of that variable, and if the value turns out to be less than 10, uses a Setup Factory action called “Dialog.Message” to display a message to the user.

New programmers and experienced coders alike will find that Setup Factory 7.0 is a powerful, flexible yet simple scripting environment to work in.

12

In This Chapter

In this chapter, you'll learn about:

- Important scripting concepts
- Variables
- Variable scope and variable naming
- Types and values
- Expressions and operators
- Control structures (if, while, repeat, and for)
- Tables (arrays)
- Functions
- String manipulation
- Debugging your scripts
- Syntax errors and functional errors
- Other scripting resources

A Quick Example of Scripting in Setup Factory

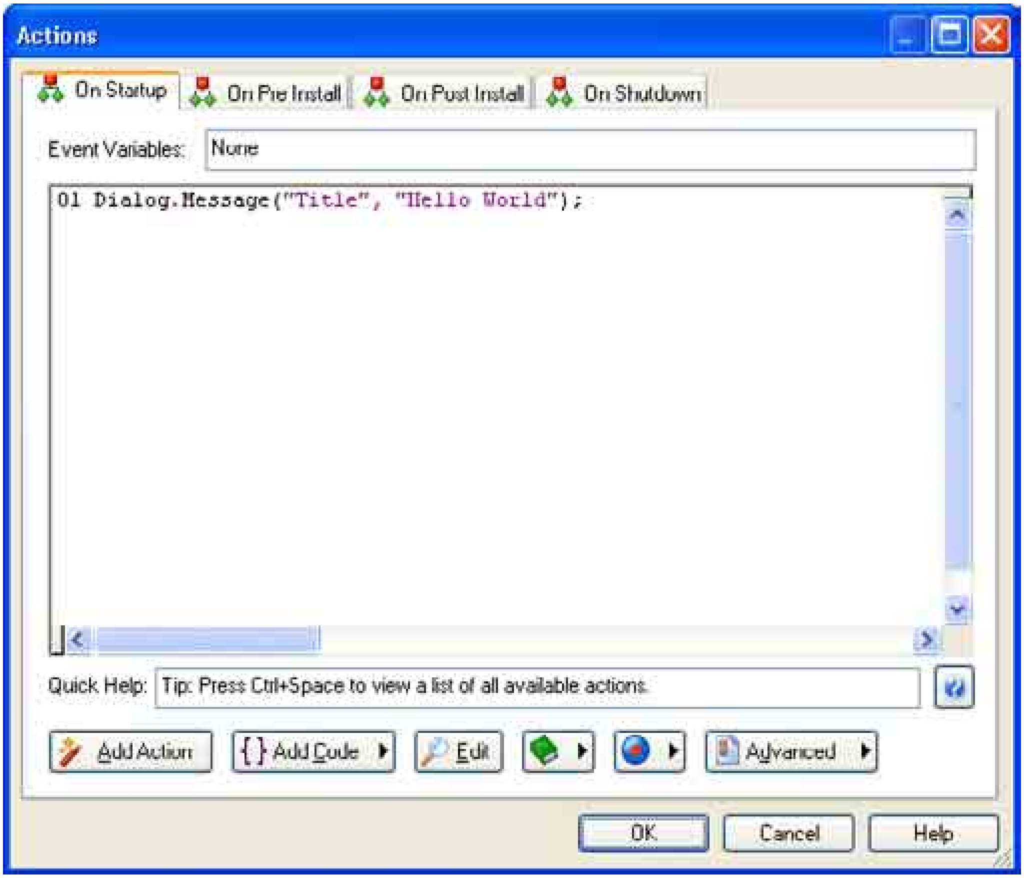
Here is a short tutorial showing you how to enter a script into Setup Factory and preview the results:

1. Start a new project.

2. In the On Startup event of your setup's actions, add the following code:

```
Dialog.Message("Title", "Hello World");
```

It should look like this when you're done:



Event Variables: None

```
Dialog.Message("Title", "Hello World");
```

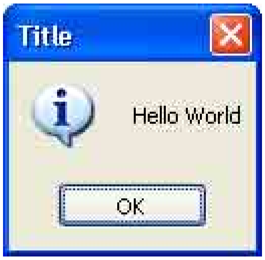
Quick Help: Tip: Press Ctrl+Space to view a list of all available actions.

3. Click OK to close the action editor.

4. Choose Publish > Build from the menu, and go through the publish wizard.

5. Once you have built your setup, run it so that the script you entered will be performed.

   You should see the following dialog appear:



Congratulations! You have just made your first script. Though this is a simple example, it shows you just how easy it is to make something happen in your Setup Factory installer. You can use the above method to try out any script you want in Setup Factory.

Note: If you are working with actions that interact with screens, you must perform these actions from a screen event.

Important Scripting Concepts

There are a few important things that you should know about the Setup Factory scripting language in general before we go on.

Script is Global

The scripting engine is global to the runtime environment. That means that all of your events will “know” about other variables and functions declared elsewhere in the product. For example, if you assign “myvar = 10;” in the project’s On Startup event, myvar will still equal 10 when the next event is triggered. There are ways around this global nature (see *Variable Scope* on page 266), but it is generally true of the scripting engine.

Script is Case-Sensitive

The scripting engine is case-sensitive. This means that upper and lower case characters are important for things like keywords, variable names and function names.

For example:

```
ABC = 10;
aBC = 7;
```

In the above script, ABC and aBC refer to two different variables, and can hold different values. The lowercase “a” in “aBC” makes it completely different from “ABC” as far as Setup Factory is concerned.

The same principle applies to function names as well. For example:

```
Dialog.Message("Hi", "Hello World");
```

...refers to a built-in Setup Factory function. However,

```
DIALOG.Message("Hi", "Hello World");
```

...will not be recognized as the built-in function, because DIALOG and Dialog are seen as two completely different names.

Note: It’s entirely possible to have two functions with the same spelling but different capitalization—for example, GreetUser and gREeTUSeR would be seen as two totally different functions. Although it’s definitely possible for such functions to coexist, it’s generally better to give functions completely different names to avoid any confusion.

Comments

You can insert non-executable comments into your scripts to explain and document your code. In a script, any text after two dashes (--) on a line will be ignored. For example:

```
-- Assign 10 to variable abc
abc = 10;
```

...or:

```
abc = 10; -- Assign 10 to abc
```

Both of the above examples do the exact same thing—the comments do not affect the script in any way.

You can also create multi-line comments by using --[[and]]-- on either side of the comment:

```
--[[ This is
a multi-line
comment ]]--
a = 10;
```

You should use comments to explain your scripts as much as possible in order to make them easier to understand by yourself and others.

Delimiting Statements

Each unique statement can either be on its own line and/or separated by a semi-colon (;). For example, all of the following scripts are valid:

Script 1:

```
a = 10
MyVar = a
```

Script 2:

```
a = 10; MyVar = a;
```

Script 3:

```
a = 10;
MyVar = a;
```

However, we recommend that you end all statements with a semi-colon (as in scripts 2 and 3 above).

Variables

What are Variables?

Variables are very important to scripting in Setup Factory. Variables are simply “nicknames” or “placeholders” for values that might need to be modified or re-used in the future. For example, the following script assigns the value 10 to a variable called “amount.”

```
amount = 10;
```

Note: We say that values are “assigned to” or “stored in” variables. If you picture a variable as a container that can hold a value, assigning a value to a variable is like “placing” that value into a container. You can change this value at any time by assigning a different value to the variable; the new value simply replaces the old one. This ability to hold changeable information is what makes variables so useful.

Here are a couple of examples demonstrating how you can operate on the “amount” variable:

```
amount = 10;
amount = amount + 20;
Dialog.Message("Value", amount);
```

This stores 10 in the variable named amount, then adds 20 to that value, and then finally makes a message box appear with the current value (which is now the number 30) in it.

You can also assign one variable to another:

```
a = 10;
b = a;
Dialog.Message("Value", b);
```

This will make a message box appear with the number 10 in it. The line “b = a;” assigns the value of “a” (which is 10) to “b.”

Variable Scope

As mentioned earlier in this document, all variables in Setup Factory are *global* by default. This just means that they exist project-wide, and hold their values from one script to the next. In other words, if a value is assigned to a variable in one script, the variable will still hold that value when the next script is executed.

For example, if you enter the script:

```
foo = 10;
```

...into the setup's On Startup event, and then enter:

```
Dialog.Message("The value is:", foo);
```

...into a screen's On Preload event, the second script will use the value that was assigned to “foo” in the first script. As a result, when the screen loads, a message box will appear with the number 10 in it.

Note that the order of execution is important...in order for one script to be able to use the value that was assigned to the variable in another script, that other script has to be executed first. In the above example, the On Startup event is triggered *before* the On Preload event, so the value 10 is already assigned to foo when the On Startup event's script is executed.

Local Variables

The global nature of the scripting engine means that a variable will retain its value throughout your entire project. You can, however, make variables that are non-global, by using the special keyword “local.” Putting the word “local” in front of a variable assignment creates a variable that is local to the current script or function.

For example, let's say you have the following three scripts in the same project:

Script 1:

```
-- assign 10 to x
x = 10;
```

Script 2:

```
local x = 500;
Dialog.Message("Local value of x is:", x);
x = 250; -- this changes the local x, not the global one
Dialog.Message("Local value of x is:", x);
```

Script 3:

```
-- display the global value of x
Dialog.Message("Global value of x is:", x);
```

Let's assume these three scripts are performed one after the other. The first script gives x the value 10. Since all variables are global by default, x will have this value inside all other scripts, too. The second script makes a *local* assignment to x, giving it the value of 500—but only inside that script. If anything else inside that script wants to access the value of x, it will see the local value instead of the global one. It's like the "x" variable has been temporarily replaced by another variable that looks just like it, but has a different value.

(This reminds me of those caper movies, where the bank robbers put a picture in front of the security cameras so the guards won't see that the vault is being emptied. Only in this case, it's like the bank robbers create a whole new working vault, just like the original, and then dismantle it when they leave.)

When told to display the contents of x, the first Dialog.Message action inside script #2 will display 500, since that is the local value of x when the action is performed. The next line assigns 250 to the local value of x—note that once you make a local variable, it completely replaces the global variable for the rest of the script.

Finally, the third script displays the global value of x, which is still 10.

Variable Naming

Variable names can be made up of any combination of letters, digits and underscores as long as they do not begin with a number and do not conflict with reserved keywords.

Examples of **valid** variables names:

a
strName
_My_Variable
data1
data_1_23
index
bReset
nCount

Examples of **invalid** variable names:

1
1data
%MyValue%
$strData
for
local
FirstName+LastName
User Name

Reserved Keywords

The following words are reserved and cannot be used for variable or function names:

and	break	do	else	elseif
end	false	for	function	if
in	local	nil	not	or
repeat	return	table	then	true
until	while			

Types and Values

Setup Factory's scripting language is dynamically typed. There are no type definitions—instead, each value carries its own type.

What this means is that you don't have to declare a variable to be of a certain type before using it. For example, in C++, if you want to use a number, you have to first declare the variable's type and then assign a value to it:

```
int j;
j = 10;
```

The above C++ example declares j as an integer, and then assigns 10 to it.

As we have seen, in Setup Factory you can just assign a value to a variable without declaring its type. Variables don't really have types; instead, it's the values inside them that are considered to be one type or another. For example:

```
j = 10;
```

...this automatically creates the variable named "j" and assigns the value 10 to it. Although this value has a type (it's a *number*), the variable itself is still typeless. This means that you can turn around and assign a different type of value to j, like so:

```
j = "Hello";
```

This replaces the number 10 that is stored in j with the string "Hello." The fact that a string is a different type of value doesn't matter; the variable j doesn't care what kind of value it holds, it just stores whatever you put in it.

There are six basic data types in Setup Factory: number, string, nil, Boolean, function, and table. The sections below will explain each data type in more detail.

Number

A number is exactly that: a numeric value. The number type represents real numbers—specifically, double-precision floating-point values. There is no distinction between integers and floating-point numbers (also known as "fractions")...all of them are just "numbers." Here are some examples of valid numbers:

4   4.   .4   0.4   4.57e-3   0.3e12

String

A string is simply a sequence of characters. For example, "Joe2" is a string of four characters, starting with a capital "J" and ending with the number "2." Strings can vary widely in length; a string can contain a single letter, or a single word, or the contents of an entire book.

Strings may contain spaces and even more exotic characters, such as carriage returns and line feeds. In fact, strings may contain any combination of valid 8-bit ASCII characters, including null characters ("\0"). Setup Factory automatically manages string memory, so you never have to worry about allocating or de-allocating memory for strings.

Strings can be used quite intuitively and naturally. They should be delimited by matching single quotes or double quotes. Here are some examples that use strings:

```
Name = "Joe Blow";
Dialog.Message("Title", "Hello, how are you?");
LastName = 'Blow';
```

Normally double quotes are used for strings, but single quotes can be useful if you have a string that contains double quotes. Whichever type of quotes you use, you can include the other kind inside the string without escaping it. For example:

```
doubles = "How's that again?";
singles = 'She said "Talk to the hand," and I was all like "Dude!"';
```

If we used double quotes for the second line, it would look like this:

```
escaped = "She said \"Talk to the hand,\" and I was all like \"Dude!\"";
```

Normally, the scripting engine sees double quotes as marking the beginning or end of a string. In order to include double quotes inside a double-quoted string, you need to *escape* them with backslashes. This tells the scripting engine that you want to include an actual quote character *in* the string.

The backslash and quote (\") is known as an *escape sequence*. An escape sequence is a special sequence of characters that gets converted or “translated” into something else by the script engine. Escape sequences allow you to include things that can’t be typed directly into a string.

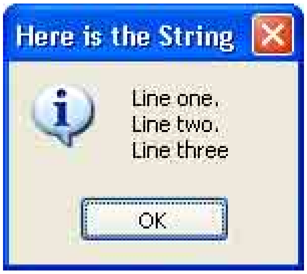
The escape sequences that you can use include:

\a	-	bell
\b	-	backspace
\f	-	form feed
\n	-	newline
\r	-	carriage return
\t	-	horizontal tab
\v	-	vertical tab
\\	-	backslash
\"	-	quotation mark
\'	-	apostrophe
\[	-	left square bracket
\]	-	right square bracket

So, for example, if you want to represent three lines of text in a single string, you would use the following:

```
Lines = "Line one.\nLine two.\nLine three";
Dialog.Message("Here is the String", Lines);
```

This assigns a string to a variable named Lines, and uses the newline escape sequence to start a new line after each sentence. The Dialog.Message function displays the contents of the Lines variable in a message box, like this:



Here is the String

Line one.
Line two.
Line three

Another common example is when you want to represent a path to a file such as C:\My Folder\My Data.txt. You just need to remember to escape the backslashes:

```
MyPath = "C:\\My Folder\\My Data.txt";
```

Each double-backslash represents a single backslash when used inside a string.

If you know your ASCII table, you can use a backslash character followed by a number with up to three digits to represent any character by its ASCII value. For example, the ASCII value for a newline character is 10, so the following two lines do the exact same thing:

```
Lines = "Line one.\nLine two.\nLine three";
Lines = "Line one.\10Line two.\10Line three";
```

However, you will not need to use this format very often, if ever.

You can also define strings on multiple lines by using double square brackets ([[and]]). A string between double square brackets does not need any escape characters. The double square brackets let you type special characters like backslashes, quotes and newlines right into the string.

For example:

```
Lines = [[Line one.
Line two.
Line three.]];
```

is equivalent to:

```
Lines = "Line one.\nLine two.\nLine three";
```

This can be useful if you have preformatted text that you want to use as a string, and you don’t want to have to convert all of the special characters into escape sequences.

The last important thing to know about strings is that the script engine provides automatic conversion between numbers and strings at run time. Whenever a numeric operation is applied to a string, the engine tries to convert the string to a number for the operation. Of course, this will only be successful if the string contains something that can be interpreted as a number.

For example, the following lines are both valid:

```
a = "10" + 1; -- Result is 11
b = "33" * 2; -- Result is 66
```

However, the following lines would not give you the same conversion result:

```
a = "10+1"; -- Result is the string "10+1"
b = "hello" + 1; -- ERROR, can’t convert "hello" to a number
```

For more information on working with strings, see page 298.

Nil

Nil is a special value type. It basically represents the absence of any other kind of value.

You can assign nil to a variable, just like any other value. Note that this isn’t the same as assigning the letters “nil” to a variable, as in a string. Like other keywords, nil must be left unquoted in order to be recognized. It should also be entered in all lowercase letters.

Nil will always evaluate to false when used in a condition:

```
a = nil;
if a then
    -- Any lines in here
    -- will not be executed
end
```

It can also be used to “delete” a variable:

```
y = "Joe Blow";
y = nil;
```

In the example above, “y” will no longer contain a value after the second line.

Boolean

Boolean variable types can have one of two values: true, or false. They can be used in conditions and to perform Boolean logic operations. For example:

```
boolybooly = true;
if boolybooly then
    -- Any script in here will be executed
end
```

This sets a variable named boolybooly to true, and then uses it in an if statement. Similarly:

```
a = true;
b = false;
if (a and b) then
    -- Any script here will not be executed because
    -- true and false is false.
end
```

This time, the if statement needs both “a” and “b” to be true in order for the lines inside it to be executed. In this case, that won’t happen because “b” has been set to false.

Function

The script engine allows you to define your own functions (or “sub-routines”), which are essentially small pieces of script that can be executed on demand. Each function has a name which is used to identify the function. You can actually use that function name as a special kind of value, in order to store a “reference” to that function in a variable, or to pass it to another function. This kind of reference is of the *function* type.

For more information on functions, see page 293.

Table

Tables are a very powerful way to store lists of indexed values under one name. Tables are actually associative arrays—that is, they are arrays which can be indexed not only with numbers, but with any kind of value (including strings).

Here are a few quick examples (we cover tables in more detail on page 285):

Example 1:

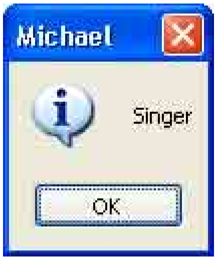
```
guys = {"Adam", "Brett", "Darryl"};
Dialog.Message("Second Name in the List", guys[2]);
```

This will display a message box with the word “Brett” in it.

Example 2:

```
t = {};
t.FirstName = "Michael";
t.LastName = "Jackson";
t.Occupation = "Singer";
Dialog.Message(t.FirstName, t.Occupation);
```

This will display the following message box:



Michael

Singer

You can assign tables to other variables as well. For example:

```
table_one = {};
table_one.FirstName = "Michael";
table_one.LastName = "Jackson";
table_one.Occupation = "Singer";
table_two = table_one;
occupation = table_two.Occupation;
Dialog.Message(b.FirstName, occupation);
```

Tables can be indexed using array notation (my_table[1]), or by dot notation if not indexed by numbers (my_table.LastName).

Note that when you assign one table to another, as in the following line:

```
table_two = table_one;
```

...this doesn’t actually copy table_two into table_one. Instead, table_two and table_one both refer to the *same* table.

This is because the name of a table actually refers to an address in memory where the data within the table is stored. So when you assign the contents of the variable table_one to the variable table_two, you’re copying the *address*, and not the actual data. You’re essentially making the two variables “point” to the same table of data.

In order to copy the contents of a table, you need to create a new table and then copy all of the data over one item at a time.

For more information on copying tables, see page 290.

Variable Assignment

Variables can have new values assigned to them by using the assignment operator (=). This includes copying the value of one variable into another. For example:

```
a = 10;
b = "I am happy";
c = b;
```

It is interesting to note that the script engine supports multiple assignment:

```
a, b = 1, 2;
```

After the script above, the variable “a” contains the number 1 and the variable “b” contains the number 2.

Tables and functions are a bit of a special case: when you use the assignment operator on a table or function, you create an alias that points to the same table or function as the variable being "copied." Programmers call this copying *by reference* as opposed to copying *by value*.

Expressions and Operators

An expression is anything that evaluates to a value. This can include a single value such as "6" or a compound value built with operators such as "1 + 3". You can use parentheses to "group" expressions and control the order in which they are evaluated. For example, the following lines will all evaluate to the same value:

```
a = 10;
a = (5 * 1) * 2;
a = 100 / 10;
a = 100 / (2 * 5);
```

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations on numbers. The following mathematical operators are supported:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)
unary -	(negation)

Here are some examples:

```
a = 5 + 2;
b = a * 100;
twentythreepercent = 23 / 100;
neg = -29;
pos = -neg;
```

Relational Operators

Relational operators allow you to compare how one value relates to another. The following relational operators are supported:

>	(greater-than)
<	(less-than)
<=	(less-than or equal to)
>=	(greater than or equal to)
~=	(not equal to)
==	(equal)

All of the relational operators can be applied to any two numbers or any two strings. All other values can only use the == operator to see if they are equal.

Relational operators return Boolean values (true or false). For example:

```
10 > 20; -- resolves to false

a = 10;
a > 300; -- false

(3 * 200) > 500; -- true
"Brett" ~= "Lorne" -- true
```

One important point to mention is that the == and ~= operators test for *complete equality*, which means that any string comparisons done with those operators are case sensitive. For example:

```
"Jojoba" == "Jojoba"; -- true
"Wildcat" == "wildcat"; -- false
"I like it a lot" == "I like it a LOT"; -- false

"happy" ~= "HaPPy"; -- true
```

Logical Operators

Logical operators are used to perform Boolean operations on Boolean values. The following logical operators are supported:

and	(only true if both values are true)
or	(true if either value is true)
not	(returns the opposite of the value)

For example:

```
a = true;
b = false;
c = a and b; -- false
d = a and nil; -- false
e = not b; -- true
```

Note that only nil and false are considered to be false, and all other values are true.

For example:

```
iaminvisible = nil;
if iaminvisible then
    -- any lines in here won't happen
    -- because iaminvisible is considered false
    Dialog.Message("You can't see me!", "I am invisible!!!!");
end

if "Brett" then
    -- any lines in here WILL happen, because only nil and false
    -- are considered false...anything else, including strings,
    -- is considered true
    Dialog.Message("What about strings?", "Strings are true.");
end
```

Concatenation

In Setup Factory scripting, the concatenation operator is two periods (..). It is used to combine two or more strings together. You don't have to put spaces before and after the periods, but you can if you want to.

For example:

```
name = "Joe".." Blow"; -- assigns "Joe Blow" to name
b = name .. " is number " .. 1; -- assigns "Joe Blow is number 1" to b
```

Operator Precedence

Operators are said to have *precedence*, which is a way of describing the rules that determine which operations in a series of expressions get performed first. A simple example would be the expression 1 + 2 * 3. The multiply (*) operator has higher precedence than the add (+) operator, so this expression is equivalent to 1 + (2 * 3). In other words, the expression 2 * 3 is performed first, and then 1 + 6 is performed, resulting in the final value 7.

You can override the natural order of precedence by using parentheses. For instance, the expression (1 + 2) * 3 resolves to 9. The parentheses make the whole sub-expression “1 + 2” the left value of the multiply (*) operator. Essentially, the sub-expression 1 + 2 is evaluated first, and the result is then used in the expression 3 * 3.

Operator precedence follows the following order, from lowest to highest priority:

and	or				
<	>	<=	>=	~=	==
..					
+	-				
*	/				
not	- (unary)				
^					

Operators are also said to have *associativity*, which is a way of describing which expressions are performed first when the operators have equal precedence. In the script engine, all binary operators are left associative, which means that whenever two operators have the same precedence, the operation on the left is performed first. The exception is the exponentiation operator (^), which is right-associative.

When in doubt, you can always use explicit parentheses to control precedence. For example:

```
a + 1 < b/2 + 1
```

...is the same as:

```
(a + 1) < ((b/2) + 1)
```

...and you can use parentheses to change the order of the calculations, too:

```
a + 1 < b/(2 + 1)
```

In this last example, instead of 1 being added to half of b, b is divided by 3.

Control Structures

The scripting engine supports the following control structures: if, while, repeat and for.

If

An if statement evaluates its condition and then only executes the “then” part if the condition is true. An if statement is terminated by the “end” keyword. The basic syntax is:

```
if *condition* then
    *do something here*
end
```

For example:

```
x = 50;
if x > 10 then
    Dialog.Message("result", "x is greater than 10");
end

y = 3;
if ((35 * y) < 100) then
    Dialog.Message("", "y times 35 is less than 100");
end
```

In the above script, only the first dialog message would be shown, because the second if condition isn't true...35 times 3 is 105, and 105 is not less than 100.

You can also use else and elseif to add more “branches” to the if statement:

```
x = 5;
if x > 10 then
    Dialog.Message("", "x is greater than 10");
else
    Dialog.Message("", "x is less than or equal to 10");
end
```

In the preceding example, the second dialog message would be shown, because 5 is not greater than 10.

```
x = 5;
if x == 10 then
    Dialog.Message("", "x is exactly 10");
elseif x == 11 then
    Dialog.Message("", "x is exactly 11");
elseif x == 12 then
    Dialog.Message("", "x is exactly 12");
else
    Dialog.Message("", "x is not 10, 11 or 12");
end
```

In that example, the last dialog message would be shown, because x is not equal to 10, or 11, or 12.

While

The while statement is used to execute the same “block” of script over and over until a condition is met. Like if statements, while statements are terminated with the “end” keyword. The basic syntax is:

```
while *condition* do
    *do something here*
end
```

The condition must be true in order for the actions inside the while statement (the “do something here” part above) to be performed. The while statement will continue to loop as long as this condition is true. Here's how it works:

If the condition is true, all of the actions between the “while” and the corresponding “end” will be performed. When the “end” is reached, the condition will be reevaluated, and if it's still true, the actions between the “while” and the “end” will be performed again. The actions will continue to loop like this until the condition evaluates to false.

For example:

```
a = 1;
while a < 10 do
    a = a + 1;
end
```

In the preceding example, the “a = a + 1;” line would be performed 9 times.

You can break out of a while loop at any time using the “break” keyword. For example:

```
count = 1;
while count < 100 do
    count = count + 1;
    if count == 50 then
        break;
    end
end
```

Although the while statement is willing to count from 1 to 99, the if statement would cause this loop to terminate as soon as count reached 50.

Repeat

The repeat statement is similar to the while statement, except that the condition is checked at the *end* of the structure instead of at the beginning. The basic syntax is:

```
repeat
    do something here
until condition
```

For example:

```
i = 1;
repeat
      i = i + 1;
until i > 10
```

This is similar to one of the while loops above, but this time, the loop is performed 10 times. The “i = i + 1;” part gets executed before the condition determines that a is now larger than 10.

You can break out of a repeat loop at any time using the “break” keyword. For example:

```
count = 1;
repeat
    count = count + 1;
    if count == 50 then
        break;
    end
until count > 100
```

Once again, this would exit from the loop as soon as count was equal to 50.

For

The for statement is used to repeat a block of script a specific number of times. The basic syntax is:

```
for variable = start,end,step do
    do something here
end
```

The *variable* can be named anything you want. It is used to “count” the number of trips through the for loop. It begins at the *start* value you specify, and then changes by the amount in *step* after each trip through the loop. In other words, the *step* gets added to the value in the *variable* after the lines between the for and end are performed. If the result is smaller than or equal to the *end* value, the loop continues from the beginning.

For example:

```
-- This loop counts from 1 to 10:
for x = 1, 10 do
    Dialog.Message("Number", x);
end
```

This displays 10 dialog messages in a row, counting from 1 to 10.

Note that the step is optional; if you don’t provide a value for the step, it defaults to 1.

Here's an example that uses a step of "-1" to make the for loop count backwards:

```
-- This loop counts from 10 down to 1:
for x = 10, 1, -1 do
    Dialog.Message("Number", x);
end
```

That example would display 10 dialog messages in a row, counting back from 10 and going all the way down to 1.

You can break out of a for loop at any time using the "break" keyword. For example:

```
for i = 1, 100 do
    if count == 50 then
        break;
    end
end
```

Once again, this would exit from the loop as soon as count was equal to 50.

There is also a variation on the for loop that operates on tables. For more information on that, see *Using For to Enumerate Tables* on page 288.

Tables (Arrays)

Tables are very useful. They can be used to store any type of value, including functions or even other tables.

Creating Tables

There are generally two ways to create a table from scratch. The first way uses curly braces to specify a list of values:

```
my_table = {"apple","orange","peach"};

associative_table = {fruit="apple", vegetable="carrot"}
```

The second way is to create a blank table and then add the values one at a time:

```
my_table = {};
my_table[1] = "apple";
my_table[2] = "orange";
my_table[3] = "peach";

associative_table = {};
associative_table.fruit = "apple";
associative_table.vegetable = "carrot";
```

Accessing Table Elements

Each "record" of information stored in a table is known as an *element*. Each element consists of a key, which serves as the index into the table, and a value that is associated with that key.

There are generally two ways to access an element: you can use array notation, or dot notation. Array notation is typically used with numeric arrays, which are simply tables where all of the keys are numbers. Dot notation is typically used with associative arrays, which are tables where the keys are strings.

Here is an example of array notation:

```
t = {"one", "two", "three"};
Dialog.Message("Element one contains:", t[1]);
```

Here is an example of dot notation:

```
t = {first="one", second="two", third="three"};
Dialog.Message("Element 'first' contains:", t.first);
```

Numeric Arrays

One of the most common uses of tables is as arrays. An array is a collection of values that are indexed by numeric keys. In the scripting engine, numeric arrays are one-based. That is, they start at index 1.

Here are some examples using numeric arrays:

Example 1:

```
myArray = {255,0,255};
Dialog.Message("First Number", myArray[1]);
```

This first example would display a dialog message containing the number "255."

Example 2:

```
alphabet = {"a","b","c","d","e","f","g","h","i","j","k",
"l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"};
Dialog.Message("Seventh Letter", alphabet[7]);
```

This would display a dialog message containing the letter “g.”

Example 3:

```
myArray = {};
myArray[1] = "Option One";
myArray[2] = "Option Two";
myArray[3] = "Option Three";
```

This is exactly the same as the following:

```
myArray = {"Option One", "Option Two", "Option Three"};
```

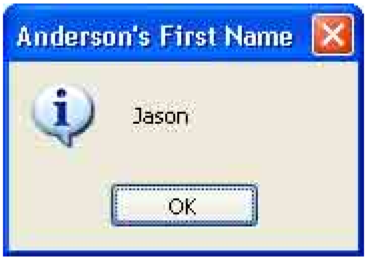
Associative Arrays

Associative arrays are the same as numeric arrays except that the indexes can be numbers, strings or even functions.

Here is an example of an associative array that uses a last name as an index and a first name as the value:

```
arrNames = {Anderson="Jason",
            Clemens="Roger",
            Contreras="Jose",
            Hammond="Chris",
            Hitchcock="Alfred"};

Dialog.Message("Anderson's First Name", arrNames.Anderson);
```

The resulting dialog message would look like this:



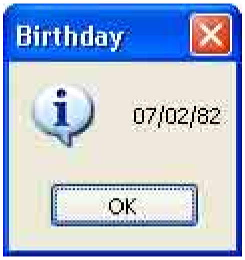
Anderson's First Name

Jason

Here is an example of a simple employee database that keeps track of employee names and birth dates indexed by employee numbers:

```
Employees = {}; -- Construct an empty table for the employee numbers

-- store each employee's information in its own table
Employee1 = {Name="Jason Anderson", Birthday="07/02/82"};
Employee2 = {Name="Roger Clemens", Birthday="12/25/79"};

-- store each employee's information table
-- at the appropriate number in the Employees table
Employees[100099] = Employee1;
Employees[137637] = Employee2;

-- now typing "Employees[100099]" is the same as typing "Employee1"
Dialog.Message("Birthday",Employees[100099].Birthday);
```

The resulting dialog message would look like this:


Birthday

07/02/82

Using For to Enumerate Tables

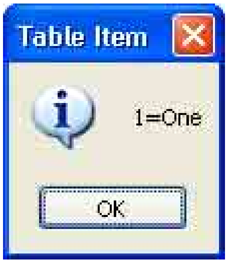
There is a special version of the for statement that allows you to quickly and easily enumerate the contents of an array. The syntax is:

```
for index,value in table do
    operate on index and value
end
```

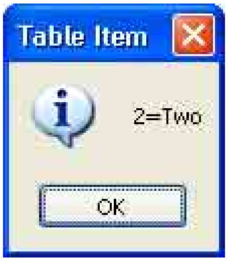
For example:

```
mytable = {"One","Two","Three"};

-- display a message for every table item
for j,k in mytable do
    Dialog.Message("Table Item", j .. "=" .. k);
end
```

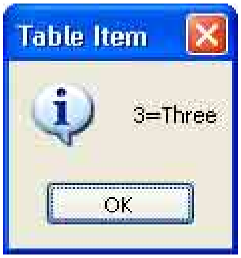
The result would be three dialog messages in a row, one for each of the elements in mytable, like so:



1=One



2=Two



3=Three

Remember the above for statement, because it is a quick and easy way to inspect the values in a table. If you just want the indexes of a table, you can leave out the *value* part of the for statement:

```
a = {One=1, Two=2, Three=3};

for k in a do
    Dialog.Message("Table Index", k);
end
```

The above script will display three message boxes in a row, with the text “One,” “Three,” and then “Two.”

Whoa there—why aren’t the table elements in order? The reason for this is that internally the scripting engine doesn’t store tables as arrays, but in a super-efficient structure known as a hash table. (Don’t worry, I get confused about hash tables too.) The important thing to know is that when you define table elements, they are not necessarily stored in the order that you define or add them, unless you use a numeric array (i.e. a table indexed with numbers from 1 to whatever).

Copying Tables:

Copying tables is a bit different from copying other types of values. Unlike variables, you can't just use the assignment operator to copy the contents of one table into another. This is because the name of a table actually refers to an address in memory where the data within the table is stored. If you try to copy one table to another using the assignment operator, you end up copying the address, and not the actual data.

For example, if you wanted to copy a table, and then modify the copy, you might try something like this:

```
table_one = { mood="Happy", temperature="Warm" };

-- create a copy
table_two = table_one;

-- modify the copy
table_two.temperature = "Cold";

Dialog.Message("Table one temperature is:", table_one.temperature);
Dialog.Message("Table two temperature is:", table_two.temperature);
```

If you ran this script, you would see the following two dialogs:

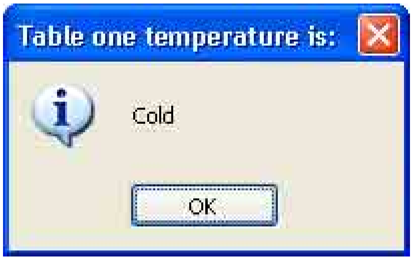


Table one temperature is:

Cold

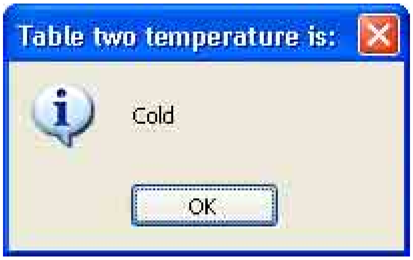


Table two temperature is:

Cold

Wait a minute...changing the “temperature” element in table_two also changed it in table_one. Why would they both change?

The answer is simply because the two are in fact the same table.

Internally, the name of a table just refers to a memory location. When table_one is created, a portion of memory is set aside to hold its contents. The location (or “address”) of this memory is what gets assigned to the variable named table_one.

Assigning table_one to table_two just copies that memory address—not the actual memory itself.

It’s like writing down the address of a library on a piece of paper, and then handing that paper to your friend. You aren’t handing the entire library over, shelves of books and all...only the location where it can be found.

If you wanted to actually copy the library, you would have to create a new building, photocopy each book individually, and then store the photocopies in the new location.

That’s pretty much how it is with tables, too. In order to create a full copy of a table, contents and all, you need to create a new table and then copy over all of the elements, one element at a time.

Luckily, the for statement makes this really easy to do. For example, here’s a modified version of our earlier example, that creates a “true” copy of table_one.

```
table_one = { mood="Happy", temperature="Warm" };

-- create a copy
table_two = {};
for index, value in table_one do
    table_two[index] = value;
end

-- modify the copy
table_two.temperature = "Cold";

Dialog.Message("Table one temperature is:", table_one.temperature);
Dialog.Message("Table two temperature is:", table_two.temperature);
```

This time, the dialogs show that modifying table_two doesn't affect table_one at all:

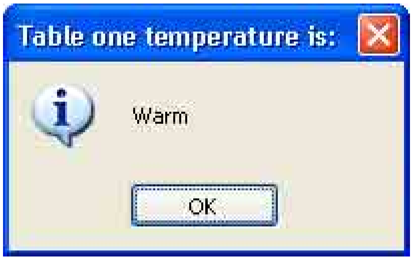


Table one temperature is:

Warm

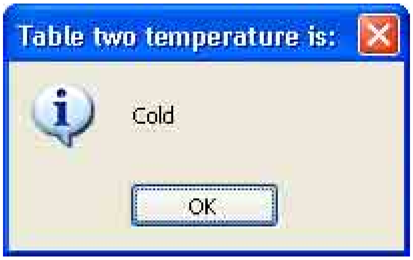


Table two temperature is:

Cold

Table Functions

There are a number of built-in table functions at your disposal, which you can use to do such things as inserting elements into a table, removing elements from a table, and counting the number of elements in a table. For more information on these table functions, please see *Program Reference / Actions / Table* in the online help.

Functions

By far the coolest and most powerful feature of the scripting engine is functions. You have already seen a lot of functions used throughout this document, such as "Dialog.Message." Functions are simply portions of script that you can define, name and then call from anywhere else.

Although there are a lot of built-in Setup Factory functions, you can also make your own custom functions to suit your specific needs. In general, functions are defined as follows:

```
function function_name (arguments)
    function script here
    return return_value;
end
```

The first part is the keyword "function." This tells the scripting engine that what follows is a function definition. The *function_name* is simply a unique name for your function. The *arguments* are parameters (or values) that will be passed to the function every time it is called. A function can receive any number of arguments from 0 to infinity (well, not infinity, but don't get technical on me). The "return" keyword tells the function to return one or more values back to the script that called it.

The easiest way to learn about functions is to look at some examples. In this first example, we will make a simple function that shows a message box. It does not take any arguments and does not return anything.

```
function HelloWorld()
    Dialog.Message("Welcome","Hello World");
end
```

Notice that if you put the above script into an event and build your install, nothing script related happens. Well, that is true and not true. It is true that nothing visible happens but the magic is in what you don't see. When the event is fired and the function script is executed, the function called "HelloWorld" becomes part of the scripting engine. That means it is now available to the rest of the install in any other script.

This brings up an important point about scripting in Setup Factory. When making a function, the function does not get "into" the engine until the script is executed. That means that if you define HelloWorld() in a screen's On Preload event, but that event never gets triggered (because the screen is never displayed), the HelloWorld()

function will never exist. That is, you will not be able to call it from anywhere else. That is why, in general, it is best to define your global functions in the global script of the project. (To access the global script, choose Resources > Global Functions from the menu.)

Now back to the good stuff. Let's add a line to actually call the function:

```
function HelloWorld()
    Dialog.Message("Welcome","Hello World");
end

HelloWorld();
```

The "HelloWorld();" line tells the scripting engine to "go perform the function named HelloWorld." When that line gets executed, you would see a welcome message with the text "Hello World" in it.

Function Arguments

Let's take this a bit further and tell the message box which text to display by adding an argument to the function.

```
function HelloWorld(Message)
    Dialog.Message("Welcome", Message);
end

HelloWorld("This is an argument");
```

Now the message box shows the text that was "passed" to the function.

In the function definition, "Message" is a variable that will automatically receive whatever argument is passed to the function. In the function call, we pass the string "This is an argument" as the first (and only) argument for the HelloWorld function.

Here is an example of using multiple arguments.

```
function HelloWorld(Title, Message)
    Dialog.Message(Title, Message);
end

HelloWorld("This is argument one", "This is argument two");
HelloWorld("Welcome", "Hi there");
```

This time, the function definition uses two variables, one for each of its two arguments...and each function call passes two strings to the HelloWorld function.

Note that by changing the content of those strings, you can send different arguments to the function, and achieve different results.

Returning Values

The next step is to make the function return values back to the calling script. Here is a function that accepts a number as its single argument, and then returns a string containing all of the numbers from one to that number.

```
function Count(n)

    -- start out with a blank return string
    ReturnString = "";

    for num = 1,n do
        -- add the current number (num) to the end of the return string
        ReturnString = ReturnString..num;

        -- if this isn't the last number, then add a comma and a space
        -- to separate the numbers a bit in the return string
        if (num ~= n) then
            ReturnString = ReturnString..", ";
        end
    end

    -- return the string that we built
    return ReturnString;
end

CountString = Count(10);
Dialog.Message("Count", CountString);
```

The last two lines of the above script uses the Count function to build a string counting from 1 to 10, stores it in a variable named CountString, and then displays the contents of that variable in a dialog message box.

Returning Multiple Values

You can return multiple values from functions as well:

```
function SortNumbers(Number1, Number2)
    if Number1 <= Number2 then
        return Number1, Number2
    else
        return Number2, Number1
    end
end

firstNum, secondNum = SortNumbers(102, 100);
Dialog.Message("Sorted", firstNum .. ", " .. secondNum);
```

The above script creates a function called SortNumbers that takes two arguments and then returns two values. The first value returned is the smaller number, and the second value returned is the larger one. Note that we specified two variables to receive the return values from the function call on the second last line. The last line of the script displays the two numbers in the order they were sorted into by the function.

Redefining Functions

Another interesting thing about functions is that you can override a previous function definition simply by re-defining it.

```
function HelloWorld()
    Dialog.Message("Message","Hello World");
end

function HelloWorld()
    Dialog.Message("Message","Hello Earth");
end

HelloWorld();
```

The script above shows a message box that says “Hello Earth,” and not “Hello World.” That is because the second version of the HelloWorld() function overrides the first one.

Putting Functions in Tables

One really powerful thing about tables is that they can be used to hold functions as well as other values. This is significant because it allows you to make sure that your functions have unique names and are logically grouped. (This is how all of the Setup Factory functions are implemented.) Here is an example:

```
-- Make the functions:
function HelloEarth()
    Dialog.Message("Message", "Hello Earth");
end

function HelloMoon()
    Dialog.Message("Message", "Hello Moon");
end

-- Define an empty table:
Hello = {};

-- Assign the functions to the table:
Hello.Earth = HelloEarth;
Hello.Moon = HelloMoon;

-- Now call the functions:
Hello.Earth();
Hello.Moon();
```

It is also interesting to note that you can define functions right in your table definition:

```
Hello = {
Earth = function () Dialog.Message("Message", "Hello Earth") end,
 Moon = function () Dialog.Message("Message", "Hello Moon") end
};

-- Now call the functions:
Hello.Earth();
Hello.Moon();
```

String Manipulation

In this section we will briefly cover some of the most common string manipulation techniques, such as string concatenation and comparisons.

(For more information on the string functions available to you in Setup Factory, see *Program Reference / Actions / String* in the online help.)

Concatenating Strings

We have already covered string concatenation, but it is well worth repeating. The string concatenation operator is two periods in a row (..). For example:

```
FullName = "Bo".." Derek";  -- FullName is now "Bo Derek"

-- You can also concatenate numbers into strings
DaysInYear = 365;
YearString = "There are "..DaysInYear.." days in a year.";
```

Note that you can put spaces on either side of the dots, or on one side, or not put any spaces at all. For example, the following four lines will accomplish the same thing:

```
foo = "Hello " .. user_name;
foo = "Hello ".. user_name;
foo = "Hello " ..user_name;
foo = "Hello "..user_name;
```

Comparing Strings

Next to concatenation, one of the most common things you will want to do with strings is compare one string to another. Depending on what constitutes a “match,” this can either be very simple, or just a bit tricky.

If you want to perform a case-sensitive comparison, then all you have to do is use the equals operator (==).

For example:

```
strOne = "Strongbad";
strTwo = "Strongbad";

if strOne == strTwo then
    Dialog.Message("Guess what?", "The two strings are equal!");
else
    Dialog.Message("Hmmm", "The two strings are different.");
end
```

Since the == operator performs a case-sensitive comparison when applied to strings, the above script will display a message box proclaiming that the two strings are equal.

If you want to perform a case-*insensitive* comparison, then you need to take advantage of either the String.Upper or String.Lower function, to ensure that both strings have the same case before you compare them. The String.Upper function returns an all-uppercase version of the string it is given, and the String.Lower function returns an all-lowercase version. Note that it doesn't matter which function you use in your comparison, so long as you use the same function on both sides of the == operator in your if statement.

For example:

```
strOne = "Mooohahahaha";
strTwo = "MOOohaHAHAha";

if String.Upper(strOne) == String.Upper(strTwo) then
    Dialog.Message("Guess what?", "The two strings are equal!");
else
    Dialog.Message("Hmmm", "The two strings are different.");
end
```

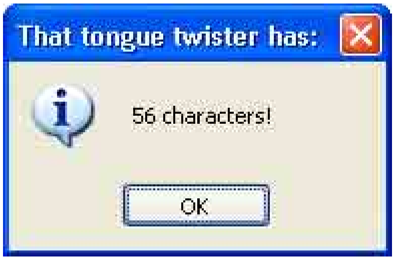
In the example above, the String.Upper function converts strOne to "MOOOHAHAHAHA" and strTwo to "MOOOHAHAHAHA" and then the if statement compares the results. (Note: the two original strings remain unchanged.) That way, it doesn't matter what case the original strings had; all that matters is whether the letters are the same.

Counting Characters

If you ever want to know how long a string is, you can easily count the number of characters it contains. Just use the String.Length function, like so:

```
twister = "If a wood chuck could chuck wood, how much would...um...";
num_chars = String.Length(twister);
Dialog.Message("That tongue twister has:", num_chars .. " characters!");
```

...which would produce the following dialog message:



That tongue twister has:

56 characters!

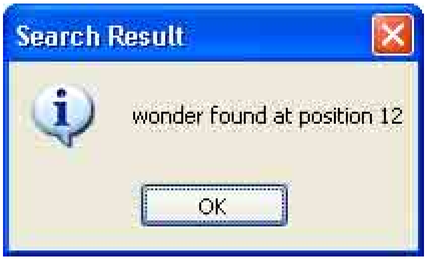
Finding Strings:

Another common thing you'll want to do with strings is to search for one string within another. This is very simple to do using the String.Find action.

For example:

```
strSearchIn = "Isn't it a wonderful day outside?";
strSearchFor = "wonder";

-- search for strSearchIn inside strSearchFor
nFoundPos = String.Find(strSearchIn, strSearchFor);

if nFoundPos ~= nil then
    -- found it!
    Dialog.Message("Search Result", strSearchFor ..
                   " found at position " .. nFoundPos);
else
    -- no luck
    Dialog.Message("Search Result", strSearchFor.." not found!");
end
```

...would cause the following message to be displayed:



Search Result

wonder found at position 12

Tip: Try experimenting with different values for strSearchFor and strSearchIn.

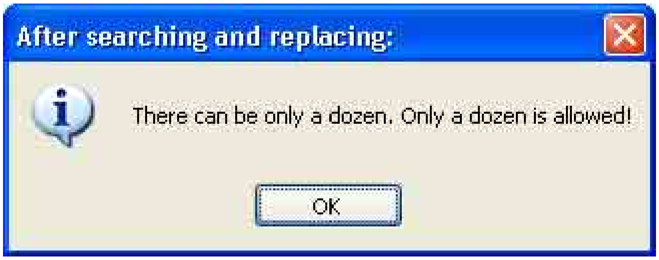
Replacing Strings:

One of the most powerful things you can do with strings is to perform a search and replace operation on them.

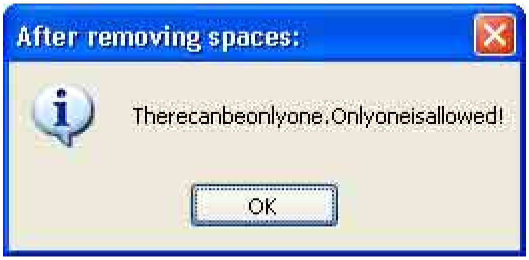
The following example shows how you can use the String.Replace action to replace every occurrence of a string with another inside a target string:

```
strTarget      = "There can be only one. Only one is allowed!";
strSearchFor   = "one";
strReplaceWith = "a dozen";
strNewString   = String.Replace( strTarget
                               , strSearchFor
                               , strReplaceWith );

Dialog.Message("After searching and replacing:", strNewString);

-- create a copy of the target string with no spaces in it
strNoSpaces = String.Replace(strTarget, " ", "");

Dialog.Message("After removing spaces:", strNoSpaces);
```

The above example would display the following two messages:


After searching and replacing:

There can be only a dozen. Only a dozen is allowed!


After removing spaces:

Therecanbeonlyone.Onlyoneisallowed!

Extracting Strings

There are three string functions that allow you to “extract” a portion of a string, rather than copying the entire string itself. These functions are String.Left, String.Right, and String.Mid.

String.Left copies a number of characters from the beginning of the string. String.Right does the same, but counting from the right end of the string instead. String.Mid allows you to copy a number of characters starting from any position in the string.

You can use these functions to perform all kinds of advanced operations on strings.

Here’s a basic example showing how they work:

```
strOriginal = "It really is good to see you again.";

-- copy the first 13 characters into strLeft
strLeft = String.Left(strOriginal, 13);

-- copy the last 18 characters into strRight
strRight = String.Right(strOriginal, 18);
```

```
-- create a new string with the two pieces
strNeo = String.Left .. "awesome" .. strRight .. " Whoa.";

-- copy the word "good" into strMiddle
strMiddle = String.Mid(strOriginal, 13, 4);
```

Converting Numeric Strings into Numbers

There may be times when you have a numeric string, and you need to convert it to a number.

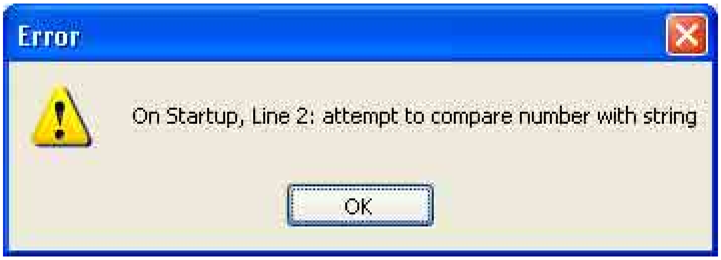
For example, if you have an input field where the user can enter their age, and you read in the text that they typed, you might get a value like “31”. Because they typed it in, though, this value is actually a string consisting of the characters “3” and “1”.

If you tried to compare this value to a number, you would get a syntax error saying that you attempted to compare a number with a string.

For example, the following script (when placed in the On Startup event):

```
age = "31";
if age > 18 then
    Dialog.Message("", "You're older than 18.");
end
```

...would produce the following error message:



On Startup, Line 2: attempt to compare number with string

The problem in this case is the line that compares the contents of the variable “age” with the number 18:

```
if age > 18 then
```

This generates an error because age contains a string, and not a number. The script engine doesn’t allow you to compare numbers with strings in this way. It has no way of knowing whether you wanted to treat age as a number, or treat 18 as a string.

The solution is simply to convert the value of age to a number before comparing it. There are two ways to do this. One way is to use the String.ToNumber function.

The String.ToNumber function translates a numeric string into the equivalent number, so it can be used in a numeric comparison.

```
age = "31";
if String.ToNumber(age) > 18 then
    Dialog.Message("", "You're older than 18.");
end
```

The other way takes advantage of the scripting engine’s ability to convert numbers into strings when it knows what your intentions are. For example, if you’re performing an arithmetic operation (such as adding two numbers), the engine will automatically convert any numeric strings to numbers for you:

```
age = "26" + 5; -- result is a numeric value
```

The above example would not generate any errors, because the scripting engine understands that the only way the statement makes sense is if you meant to use the numeric string as a number. As a result, the engine automatically converts the numeric string to a number so it can perform the calculation.

Knowing this, we can convert a numeric string to a number without changing its value by simply adding 0 to it, like so:

```
age = "31";
if (age + 0) > 18 then
    Dialog.Message("", "You're older than 18.");
end
```

In the preceding example, adding zero to the variable gets the engine to convert the value to a number, and the result is then compared with 18. No more error.

Other Built-in Functions

Script Functions

There are three other built-in functions that may prove useful to you: dofile, require, and type.

dofile

Loads and executes a script file. The contents of the file will be executed as though it was typed directly into the script. The syntax is:

dofile(*file_path*);

For example, say we typed the following script into a file called MyScript.lua (just a text file containing this script, created with notepad or some other text editor):

```
Dialog.Message("Hello", "World");
```

Now we include the file as a primer file in our Setup Factory isntaller. Wherever the following line of script is added:

```
dofile(SessionVar.Expand("%TempLaunchFolder%\\MyScript.lua"));
```

...that script file will be read in and executed immediately. In this case, you would see a message box with the friendly “hello world” message.

Tip: Use the dofile function to save yourself from having to re-type or re-paste a script into your projects over and over again.

require

Loads and runs a script file into the scripting engine. It is similar to dofile except that it will only load a given file once per session, whereas dofile will re-load and re-run the file each time it is used. The syntax is:

require(*file_path*);

So, for example, even if you do two requires in a row:

```
require("%AppFolder%\\foo.lua");
require("%AppFolder%\\foo.lua"); -- this line won't do anything
```

...only the first one will ever get executed. After that, the scripting engine knows that the file has been loaded and run, and future calls to require that file will have no effect.

Since require will only load a given script file once per session, it is best suited for loading scripts that contain only variables and functions. Since variables and functions are global by default, you only need to “load” them once; repeatedly loading the same function definition would just be a waste of time.

This makes the require function a great way to load external script libraries. Every script that needs a function from an external file can safely require() it, and the file will only actually be loaded the first time it’s needed.

type

This function will tell you the type of value contained in a variable. It returns the string name of the variable type. Valid return values are “nil,” “number,” “string,” “boolean,” “table,” or “function.” For example:

```
a = 989;
strType = type(a);  -- sets strType to "number"

a = "Hi there";
strType = type(a);  -- sets strType to "string"
```

The type function is especially useful when writing your own functions that need certain data types in order to operate. For example, the following function uses type() to make sure that both of its arguments are numbers:

```
-- find the maximum of two numbers
function Max(Number1, Number2)
    -- make sure both arguments are numeric
    if (type(Number1) ~= "number") or (type(Number2) ~= "number") then
        Dialog.Message("Error", "Please enter numbers");
        return nil; -- we're using nil to indicate an error condition
    else
        if Number1 >= Number2 then
            return Number1;
        else
            return Number2;
        end
    end
end
```

Actions

Setup Factory comes with a large number of built-in functions. In the program interface, these built-in functions are commonly referred to as *actions*. For scripting purposes, actions and functions are essentially the same; however, the term “actions” is generally reserved for those functions that are built into the program and are included in the alphabetical list of actions in the online help. When referring to functions that have been created by other users or yourself, the term “functions” is preferred.

Debugging Your Scripts

Scripting (or any kind of programming) is relatively easy once you get used to it. However, even the best programmers make mistakes, and need to iron the occasional wrinkle out of their code. Being good at debugging scripts will reduce the time to market for your projects and increase the amount of sleep you get at night. Please read this section for tips on using Setup Factory as smartly and effectively as possible!

This section will explain Setup Factory’s error handling methods as well as cover a number of debugging techniques.

Error Handling

All of the built-in Setup Factory actions use the same basic error handling techniques. However, this is not necessarily true of any third-party functions, modules, or scripts—even scripts developed by Indigo Rose Corporation that are not built into the product. Although these externally developed scripts can certainly make use of Setup Factory's error handling system, they may not necessarily do so. Therefore, you should always consult a script or module's author or documentation in order to find out how error handling is, well, handled.

There are two kinds of errors that you can have in your scripts when calling Setup Factory actions: syntax errors, and functional errors.

Syntax Errors

Syntax errors occur when the syntax (or “grammar”) of a script is incorrect, or a function receives arguments that are not appropriate. Some syntax errors are caught by Setup Factory when you build installer.

For example, consider the following script:

```
foo =
```

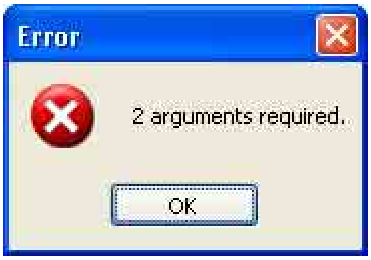
This is incorrect because we have not assigned anything to the variable foo—the script is incomplete. This is a pretty obvious syntax error, and would be caught by the scripting engine at build time (when you build your project).

Another type of syntax error is when you do not pass the correct type or number of arguments to a function. For example, if you try and run this script:

```
Dialog.Message("Hi There");
```

...the project will build fine, because there are no *obvious* syntax errors in the script. As far as the scripting engine can tell, the function call is well formed. The name is valid, the open and closed parentheses match, the quotes are in the right places, and there's even a terminating semi-colon at the end. Looks good!

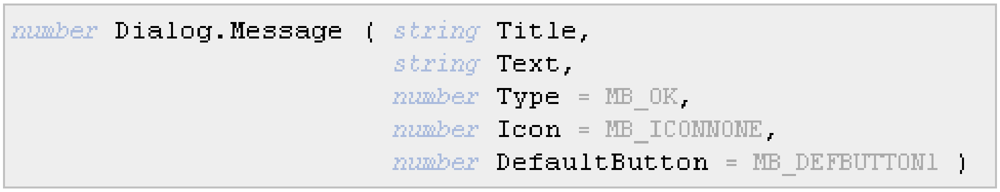
However, at run time you would see something like the following:



2 arguments required.

Looks like it wasn't so good after all. Note that the message says two arguments are required for the Dialog.Message action. Ah. Our script only provided one argument.

According to the function prototype for Dialog.Message, it looks like the action can actually accept up to *five* arguments:



```
number Dialog.Message ( string Title,
                        string Text,
                        number Type = MB_OK,
                        number Icon = MB_ICONNONE,
                        number DefaultButton = MB_DEFBUTTON1 )
```

Looking closely at the function prototype, we see that the last three arguments have default values that will be used if those arguments are omitted from the function call. The first two arguments—Title and Text—don't have default values, so they cannot be omitted without generating an error. To make a long story short, it's okay to call the Dialog.Message action with anywhere from 2 to 5 arguments...but 1 argument isn't enough.

Fortunately, syntax errors like these are usually caught at build time or when you test your installer. The error messages are usually quite clear, making it easy for you to locate and identify the problem.

Functional Errors

Functional errors are those that occur because the functionality of the action itself fails. They occur when an action is given incorrect information, such as the path to a file that doesn't exist. For example, the following code will produce a functional error:

```
filecontents = TextFile.ReadToString("this_file_don't exist.txt");
```

If you put that script into an event right now and try it, you will see that nothing appears to happen. This is because Setup Factory's functional errors are not automatically displayed the way syntax errors are. We leave it up to you to handle (or to not handle) such functional errors yourself.

The reason for this is that there may be times when you don't care if a function fails. In fact, you may expect it to. For example, the following code tries to remove a folder called C:\My Temp Folder:

```
Folder.Delete("C:\\My Temp Folder");
```

However, in this case you don't care if it really gets deleted, or if the folder didn't exist in the first place. You just want to make sure that if that particular folder exists, it will be removed. If the folder isn't there, the Folder.Delete action causes a functional error, because it can't find the folder you told it to delete...but since the end result is exactly what you wanted, you don't need to do anything about it. And you certainly don't want the user to see any error messages.

Conversely, there may be times when it is very important for you to know if an action fails. Say for instance that you want to copy a very important file:

```
File.Copy("C:\\Temp\\My File.dat","C:\\Temp\\My File.bak");
```

In this case, you really want to know if it fails and may even want to exit the program or inform the user. This is where the Debug actions come in handy. Read on.

Debug Actions

Setup Factory comes with some very useful functions for debugging your installs. This section will look at a number of them.

Application.GetLastError

This is the most important action to use when trying to find out if a problem has occurred. At run time there is always an internal value that stores the status of the last action that was executed. At the start of an action, this value is set to 0 (the number zero). This means that everything is OK. If a functional error occurs inside the action, the value is changed to some non-zero value instead.

This last error value can be accessed at any time by using the Application.GetLastError action.

The syntax is:

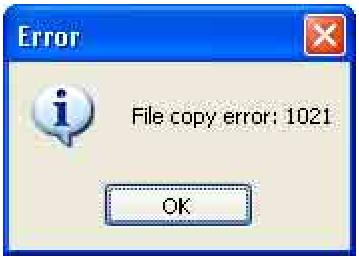
> *last_error_code* = Application.GetLastError();

Here is an example that uses this action:

```
File.Copy("C:\\Temp\\My File.dat","C:\\Temp\\My File.bak");

error_code = Application.GetLastError();
if (error_code ~= 0) then
    -- some kind of error has occurred!
    Dialog.Message("Error", "File copy error: "..error_code);
    Application.Exit();
end
```

The above script will inform the user that an error occurred and then exit the install. This is not necessarily how all errors should be handled, but it illustrates the point. You can do anything you want when an error occurs, like calling a different function or anything else you can dream up.

The above script has one possible problem. Imagine the user seeing a message like this:



Error

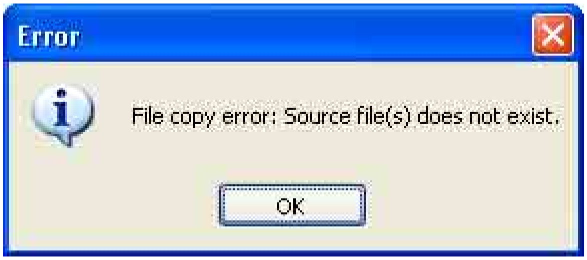
File copy error: 1021

It would be much nicer to actually tell them some information about the exact problem. Well, you are in luck! At run time there is a table called _tblErrorMessages that contains all of the possible error messages, indexed by the error codes. You can easily use the last error number to get an actual error message that will make more sense to the user than a number like “1021.”

For example, here is a modified script to show the actual error string:

```
File.Copy("C:\\Temp\\My File.dat","C:\\Temp\\My File.bak");

error_code = Application.GetLastError();

if (error_code ~= 0) then

    -- some kind of error has occurred!
    Dialog.Message( "Error", "File copy error: "
                    .. _tblErrorMessages[error_code] );

    Application.Exit();

end
```

Now the script will produce the following error message:



Error

File copy error: Source file(s) does not exist.

Much better information!

Just remember that the value of the last error gets reset every time an action is executed. For example, the following script would not produce an error message:

```
File.Copy("C:\\Temp\\My File.dat","C:\\Temp\\My File.bak");

-- At this point Application.GetLastError() could be non-zero, but...

Dialog.Message("Hi There", "Hello World");

-- Oops, now the last error number will be for the Dialog.Message action,
-- and not the File.Copy action. The Dialog.Message action will succeed,
-- resetting the last error number to 0, and the following lines will not
-- catch any error that happened in the File.Copy action.

error_code = Application.GetLastError();

if (error_code ~= 0) then

    -- some kind of error has occurred!
    Dialog.Message( "Error", "File copy error: "
                    .. _tblErrorMessages[error_code] );

    Application.Exit();

end
```

Debug.ShowWindow

The Setup Factory runtime has the ability to show a debug window that can be used to display debug messages. This window exists throughout the execution of your install, but is only visible when you tell it to be.

The syntax is:

Debug.ShowWindow(*show_window*);

...where *show_window* is a Boolean value. If true, the debug window is displayed, if false, the window is hidden. For example:

```
-- show the debug window
Debug.ShowWindow(true);
```

If you call this script, the debug window will appear on top of your installer, but nothing else will really happen. That’s where the following Debug actions come in.

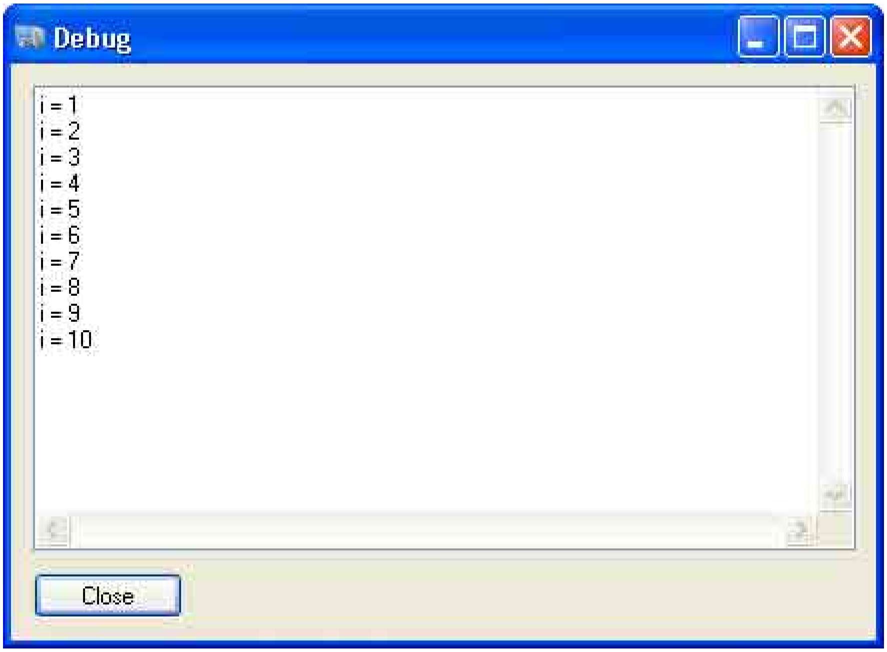
Debug.Print

This action prints the text of your choosing in the debug window. For example, try the following script:

```
Debug.ShowWindow(true);

for i = 1, 10 do
    Debug.Print("i = " .. i .. "\r\n");
end
```

The “\r\n” part is actually two escape sequences that are being used to start a new line. (This is technically called a “carriage return/linefeed” pair.) You can use \r\n in the debug window whenever you want to insert a new line.

The above script will produce the following output in the debug window:



```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

You can use this method to print all kinds of information to the debug window. Some typical uses are to print the contents of a variable so you can see what it contains at run time, or to print your own debug messages like “inside outer for loop” or “foo() function started.” Such messages form a trail like bread crumbs that you can trace in order to understand what's happening behind the scenes in your project. They can be invaluable when trying to debug your scripts or test your latest algorithm.

Debug.SetTraceMode

Setup Factory can run in a special "trace" mode at run time that will print information about every line of script that gets executed to the debug window, including the value of Application.GetLastError() if the line involves calling a built-in action. You can turn this trace mode on or off by using the Debug.SetTraceMode action:

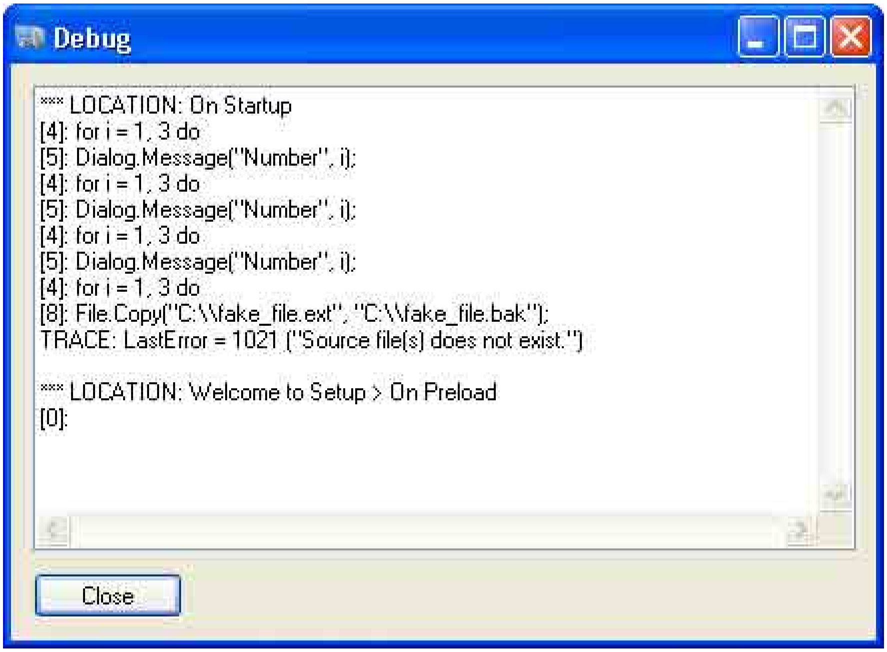
Debug.SetTraceMode(*turn_on*);

...where *turn_on* is a Boolean value that tells the program whether to turn the trace mode on or off.

Here is an example:

```
Debug.ShowWindow(true);
Debug.SetTraceMode(true);

for i = 1, 3 do
    Dialog.Message("Number", i);
end

File.Copy("C:\\fake_file.ext", "C:\\fake_file.bak");
```

Running that script will produce the following output in the debug window:



```
*** LOCATION: On Startup
[4]: for i = 1, 3 do
[5]: Dialog.Message("Number", i);
[4]: for i = 1, 3 do
[5]: Dialog.Message("Number", i);
[4]: for i = 1, 3 do
[5]: Dialog.Message("Number", i);
[4]: for i = 1, 3 do
[8]: File.Copy("C:\\fake_file.ext", "C:\\fake_file.bak");
TRACE: LastError = 1021 ("Source file(s) does not exist.")

*** LOCATION: Welcome to Setup > On Preload
[0]:
```

Notice that every line produced by the trace mode starts with "TRACE:" This is so you can tell them apart from any lines you send to the debug window with Debug.Print. The number after the "TRACE:" part is the line number that is currently being executed in the script.

Turning trace mode on is something that you will not likely want to do in your final, distributable installer, but it can really help find problems during development.

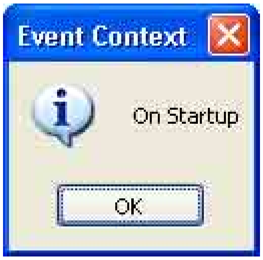
Debug.GetEventContext

The Debug.GetEventContext action is used to get a descriptive string about the event that is currently being executed. This can be useful if you define a function in one place but call it somewhere else, and you want to be able to tell where the function is being called from at any given time.

For example, if you execute this script from the On Startup event:

```
Dialog.Message("Event Context", Debug.GetEventContext());
```

...you will see something like this:



Event Context

On Startup

Dialog.Message

This brings us to good ole' Dialog.Message. You have seen this action used throughout this document, and for good reason. This is a great action to use throughout your code when you are trying to track down a problem.

For example, you can use it to display the current contents of a variable that you're working with:

```
Dialog.Message("The current value of nCats is: " .. nCats);
```

You can also use it to put up messages at specific points in a script, to break it into arbitrary stages. This can be helpful when you're not sure where in a script an error is occurring:

```
function foobar(arg1, arg2)

    Dialog.Message("Temporary Debug Msg", "In foobar()");

    -- bunch of script

    Dialog.Message("Temporary Debug Msg", "1");

    -- bunch of script

    Dialog.Message("Temporary Debug Msg", "2");

    -- bunch of script

    Dialog.Message("Temporary Debug Msg", "Leaving foobar()");

end
```

Final Thoughts

Hopefully this chapter has helped you to understand scripting in Setup Factory 7.0. Once you get the hang of it, it is a really fun, powerful way to get things done.

Other Resources

Here is a list of other places that you can go for help with scripting in Setup Factory.

Help File

The Setup Factory help file is packed with good reference material for all of the actions and events supported by Setup Factory, and for the design environment itself. You can access the help file at any time by choosing Help > Setup Factory Help from the menu.

Tip: If you are in the action editor and you want to learn more about an action, simply click on the action and press the F1 key on your keyboard.

Setup Factory Web Site

The Setup Factory web site is located at http://www.setupfactory.com. Be sure to check out the user forums where you can read questions and answers by fellow users and Indigo Rose staff as well as ask questions of your own.

Tip: A quick way to access the online forums is to choose Help > User Forums from the menu.

Indigo Rose Technical Support

If you need help with any scripting concepts or have a mental block to push through, feel free to open a support ticket at http://support.indigorose.com. Although we can't write scripts for you or debug your specific scripts, we will be happy to answer any general scripting questions that you have.

The Lua Web Site

Setup Factory's scripting engine is based on a popular scripting language called *Lua*. Lua is designed and implemented by a team at Tecgraf, the Computer Graphics Technology Group of PUC-Rio (the Pontifical Catholic University of Rio de Janeiro in Brazil). You can learn more about Lua and its history at the official Lua web site:

http://www.lua.org

The Lua website is is also where you can find the latest documentation on the Lua language, along with tutorials and a really friendly community of Lua developers.

Note that there may be other built-in functions that exist in Lua and in Setup Factory that are not officially supported in Setup Factory. These functions, if any, are documented in the Lua 5.0 Reference Manual.

Only the functions listed in the online help are supported by Indigo Rose Software. Any other “undocumented” functions that you may find in the Lua documentation are not supported. Although these functions may work, you must use them entirely on your own.

INDEX

Notes