# InstallAnywhere® 2008 Training Manual

# InstallAnywhere® 2008 Training Manual

*InstallAnywhere 2008 Training Manual*

Printed October 2007.

Printed in the USA.

Part No. TRNIA09-1007

## Course Introduction

- **An Introduction to Installation Issues**
- **Multiplatform Installation**

## *An Introduction to Installation Issues*

A well-planned installation and deployment strategy should be part of any serious software development project. When the installer is an afterthought, the result is usually a poorly prepared first impression for end users. In practice, however, this critical requirement is sometimes ignored until the software is complete. Why wait until the product is ready to be released before thinking about deployment? Regardless of whether the end user is a member of the general public, a consulting client, or another group within your own organization, it’s unlikely that the product will simply be checked out of a source control solution and the resources laid immediately into their final operational location.

Once you have made your Gold Master, how does the software make its way to your customers? Will it be enough to simply deliver an archive such as a ZIP or JAR file, or a Unix tarball? This method allows you to deliver a number of different types of files as a single unit. That’s certainly one way to do it and it is easier than having your end user login to a source control solution, or copy individual files. However, this method has inherent weaknesses. Rarely can a collection of files be simply laid into a file system, and be ready for one-click execution without some measure of configuration. What if the application requires installation into several locations? What if portions of the installation require configuration prior to use?

Today’s sophisticated software applications require complex configurations, and complex configurations require installation utilities. You can choose from a variety of methods and types of installation utilities. Windows and Macintosh users are familiar with the executable installer (e.g., InstallAnywhere®), while users of Unix systems are accustomed to deployment schemes that utilize complex scripts or native package managers. Installation utilities allow you to provide your end users with a familiar interface, assuring a positive product installation experience with a minimum of inconvenience.

Many ready-made solutions are available for specific target platforms. For example, RPM is a packaging system that generally functions only on Linux (though it has been introduced to other mainstream Unix and Unix-like systems). Such targeted solutions are not useful for multiplatform deployment. Multiplatform deployment, while once unusual, is no longer a fringe issue. More platform-agnostic software development is being done in languages such as Java, Perl, Python, PHP, and those outlined by the .NET standards. In order to keep pace with this new development landscape, you need a tool that deploys and configures your applications on many different platforms.

## *Multiplatform Installation*

If your product is intended for multiplatform deployment, you need InstallAnywhere. InstallAnywhere deploys your applications to many different systems, while you build and create only a single project. Using Java, InstallAnywhere installers run on nearly any platform for which a Java Virtual Machine is available, from the ubiquitous Windows desktop, to the high-end, headless Unix servers used in e-Business and Web services applications.

Complex application delivery requires an installer that allows complete configuration and precise control over uncounted variables. A multiplatform installer is preferable over a simple archive because you can dynamically configure your applications and deliver associated (or other necessary) applications along with your own packages. For example, if your application is a database-based tool, you may need to include the database engine necessary for your application to run. A single installer can be used as a “Master” installer and manage the entire installation process for your end users. This method is often referred to as a “Suite” installer or a “Software Stack” installation.

You never get a second chance at a first impression, and if an end user’s first experience with your product is difficult, unfamiliar, or time-consuming, you’re already “in the hole” in terms of credibility. End-user confidence is enhanced when you use a multiplatform installer. By providing your end users with a comfortable, easily accomplished installation of your product, your end users’ product experience is immediately positive.

Many of today’s end users “grew up” on Microsoft Windows or Macintosh operating systems. Even those for whom the primary platform is a commercial Unix (such as Solaris or AIX) are familiar with the modern Graphical User Interfaces implemented by these operating environments. The custom graphics features in InstallAnywhere provide instant familiarity without sacrificing the power behind the attractive front end. Additionally, InstallAnywhere provides powerful “silent” features allowing you to integrate your installation with any number of automation processes.

# Chapter 1: Introduction to InstallAnywhere

- **What Is InstallAnywhere?**
- **Requirements**
- **Editions**

## What Is InstallAnywhere?

InstallAnywhere is the most powerful multiplatform software installation solution available. InstallAnywhere deploys software onto any platform and configures applications for optimal performance. InstallAnywhere supports the platforms that run the enterprise, including the latest editions of Windows, Mac OS X, Solaris, Linux, NetWare, HP-UX, AIX, and many more.

Installation programs created with InstallAnywhere will run on systems with a compatible Java virtual machine. As described later in this course, your installer can either search for an existing appropriate JVM on a target system, or bundle a JVM for the use of the running installer. Because the installers are Java-based, they provide a consistent end-user interface and experience across the supported platforms, while supporting the use of native code to extend the Java-based installation functionality.

InstallAnywhere creates installers that meet the demands of diverse computing environments and that dynamically adapt to the systems on which they are deployed, making even the most complex software configuration easy. Its intuitive architecture brings intelligence to the process of installing any kind of software, including desktop software, enterprise software, or multi-tiered Web services, onto any client or server platform, configuring those applications for optimal performance. InstallAnywhere handles all installation details automatically, minimizing time-to-deployment, and increasing developer productivity.

By delivering an ideal mix of power, ease of use, and functionality, the award-winning InstallAnywhere family has become the preferred choice of multiplatform developers worldwide. Software innovators like Adobe, Borland, HP, i2, IBM, Intel, Iona, Lucent, Nortel, and Sun are just some of the software industry’s leaders who depend upon InstallAnywhere for fast, powerful, and intuitive installers.

## Requirements

When developing an installer, just as with developing software, you must think in terms of the authoring environment where you create the installer, and the target environments, the various operating systems and configurations where the installer will be deployed.

### Authoring Environment

#### Authoring Environment Requirements

- 128 MB free RAM
- Minimum of 8-bit color depth (256 colors)
- Minimum 1024 by 768 resolution

#### Operating Systems Supported

- Windows 2000, XP, 2003, and Vista (32-bit x86)
- Red Hat Enterprise Linux 4 and 5 (32-bit x86)
- SUSE Linux 9 and 10 (32-bit x86)
- Mac OS X 10.4 and 10.5 with Java 1.5 (Intel and PowerPC)
- Solaris 9 and 10 (SPARC)

- HP-UX 11i (PA-RISC)
- AIX 5.2 or higher

Language-localized versions of InstallAnywhere Enterprise Edition (for French, German, and Japanese developers) are available on the Windows platform only. Installers can be built from any platform to any other platform, locale, or language. Localizations for 31 languages are included.

## *Target Environment*

- 64 MB of free RAM
- Minimum of 8-bit color depth (256 colors)
- Minimum 640 by 480 screen resolution

### Operating Systems Supported

Installers run on any version of these operating systems, as long as the operating system supports Java 1.4 or newer:

- Windows XP and 2003 (32-bit and 64-bit x86, Itanium 2, x86, AMD64)
- Windows Vista (32-bit and 64-bit x86)
- Windows 2000 and NT (32-bit x86)
- Red Hat Enterprise Linux 4 and 5 (x86, Itanium 2, and AMD64)
- SUSE Linux 9 and 10 (x86 and PowerPC)
- z/Linux (s390)
- Mac OS X 10.2, 10.3, 10.4, and 10.5 (Intel and PowerPC)
- Solaris 9 and 10 (SPARC, x86, and AMD64)
- HP-UX 11i (Itanium 2 and PA-RISC)
- AIX 5.x
- FreeBSD
- Other Linux and Unix operating systems (POSIX-compliant shell required)

### Supported Java Virtual Machines

- Sun: 1.4.x, 1.5.x (Java 5), 1.6.x (Java 6)
- IBM: 1.4.x, 1.5.x
- Apple: 1.4.x, 1.5.x
- HP: 1.4.x, 1.5.x

The InstallAnywhere installer bundles Java 1.5 by default for all platforms. Any Java virtual machine can be bundled with an installer, ensuring that the target system meets the minimum requirements for both the installers and your applications. To download these bundled VM packs, see InstallAnywhere downloads at

http://www.macrovision.com/downloads/flexnet_installshield/installanywhere/VM_packs.shtml

**NOTE:** InstallAnywhere installers are not supported on beta or on early-access operating system or Java releases.

## Editions

There are two editions of InstallAnywhere: Standard and Enterprise. Each edition is designed to meet product deployment needs for the different types of customers. This manual describes the features available in the Enterprise Edition.

### Enterprise Edition

Enterprise Edition provides the ultimate in configuration options, user interaction, and client/server features. It simplifies complex installations and provides maximum developer customization. The Enterprise Edition is available in English, French, German, or Japanese. Each Enterprise Edition has full international support to create installers in 31 different languages.

Some InstallAnywhere features available only or primarily in the Enterprise Edition are:

- A localized development environment (English, French, German, or Japanese)
- Collaboration functionality using InstallAnywhere Collaboration
- A services layer for adding advanced functionality in custom code actions
- Creation of silent and console installations, and use of response files
- Use of developer-defined locations to represent destinations on a user’s system
- Creation of Application Server hosts to which you can assign WAR/EAR deployment actions to a J2EE container, as well as Database Server hosts to which you can assign SQL instruction actions that run SQL scripts during installation

### Standard Edition

Standard Edition offers more features and customizability than any other product in its class. It is ideal for desktop application deployment and has international support for 9 languages.

For a detailed list of features available in each edition, see

http://www.macrovision.com/products/installation/installanywhere.htm.

# Chapter 2: The InstallAnywhere End-User Experience

- The End-User Experience
- The Client-Side Installer Experience
- The Server-Side Installer Experience

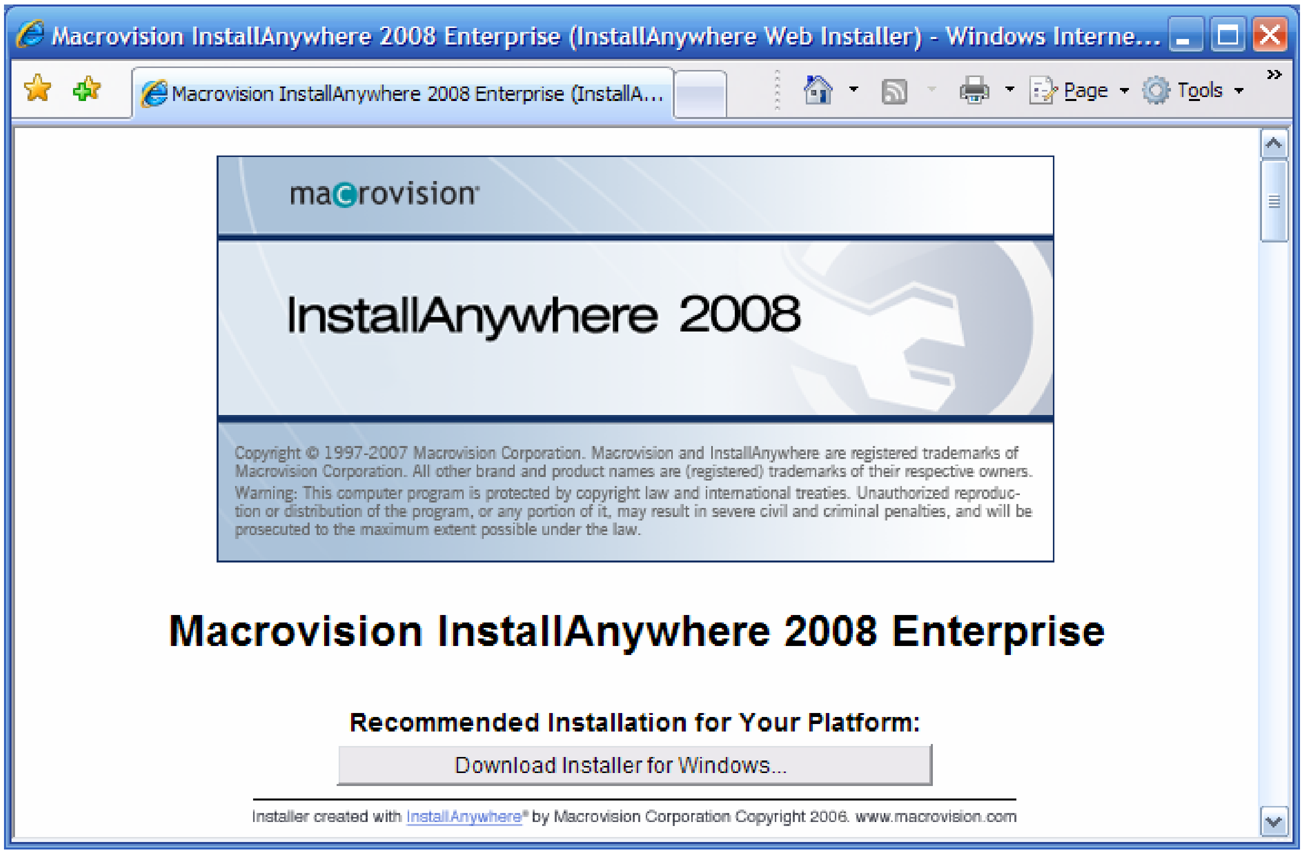
## The End-User Experience

There is no difference in InstallAnywhere’s usability, whether creating client-side or server-side application installers. The needs of installations are different; server-side installations generally require more choices, configuration options, and more experience on the person performing the installation rather than the installer. The following examples show some of the options InstallAnywhere offers for these different types of installers.

## The Client-Side Installer Experience

The InstallAnywhere application is itself installed using an installer built with InstallAnywhere. (We’ll call this installer the “InstallAnywhere installer,” just as you might call your installer “*MyProduct* installer.”) Going through the process of installing InstallAnywhere is a useful exercise in observing the end-user experience of installing a client-side application, and that’s what we are going to do next. The InstallAnywhere installer is built completely with InstallAnywhere and makes use of many of the InstallAnywhere features we’ll cover in our exercises.

The first objective is simplicity. When your end user visits your download page, you want to present a “single-click install.” This is accomplished by the InstallAnywhere Web Installer Applet. Since InstallAnywhere automatically creates HTML pages configured with the InstallAnywhere Web Installer Applet, not only is the end user’s experience simple, but your work is done for you (we’ll talk more about that later).


Macrovision InstallAnywhere 2008 Enterprise (InstallAnywhere Web Installer) - Windows Interne...
Macrovision InstallAnywhere 2008 Enterprise (InstallA...
Page
Tools
macrovision
InstallAnywhere 2008

Copyright © 1997-2007 Macrovision Corporation. Macrovision and InstallAnywhere are registered trademarks of Macrovision Corporation. All other brand and product names are (registered) trademarks of their respective owners. Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of the program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

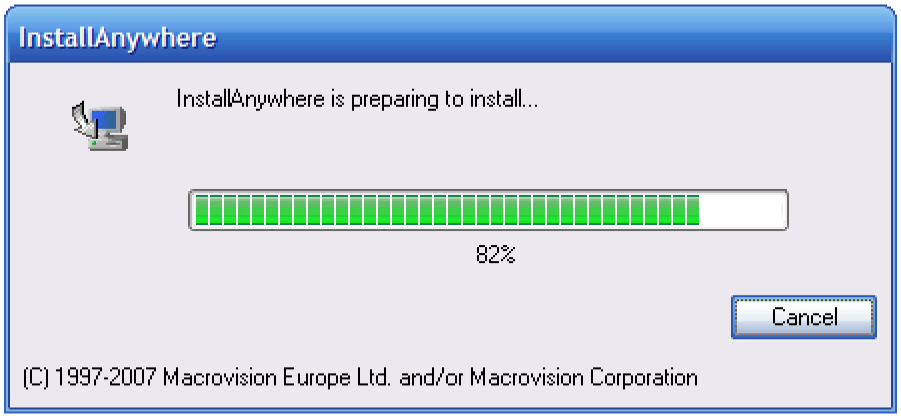
Macrovision InstallAnywhere 2008 Enterprise

Recommended Installation for Your Platform:

Installer created with InstallAnywhere® by Macrovision Corporation Copyright 2006. www.macrovision.com

**Figure 2-1: The InstallAnywhere Web Install Applet**

When the user selects the “Download Installer for [Operating System]” button, the applet will check for disk space, download the installer, and then execute the installer.

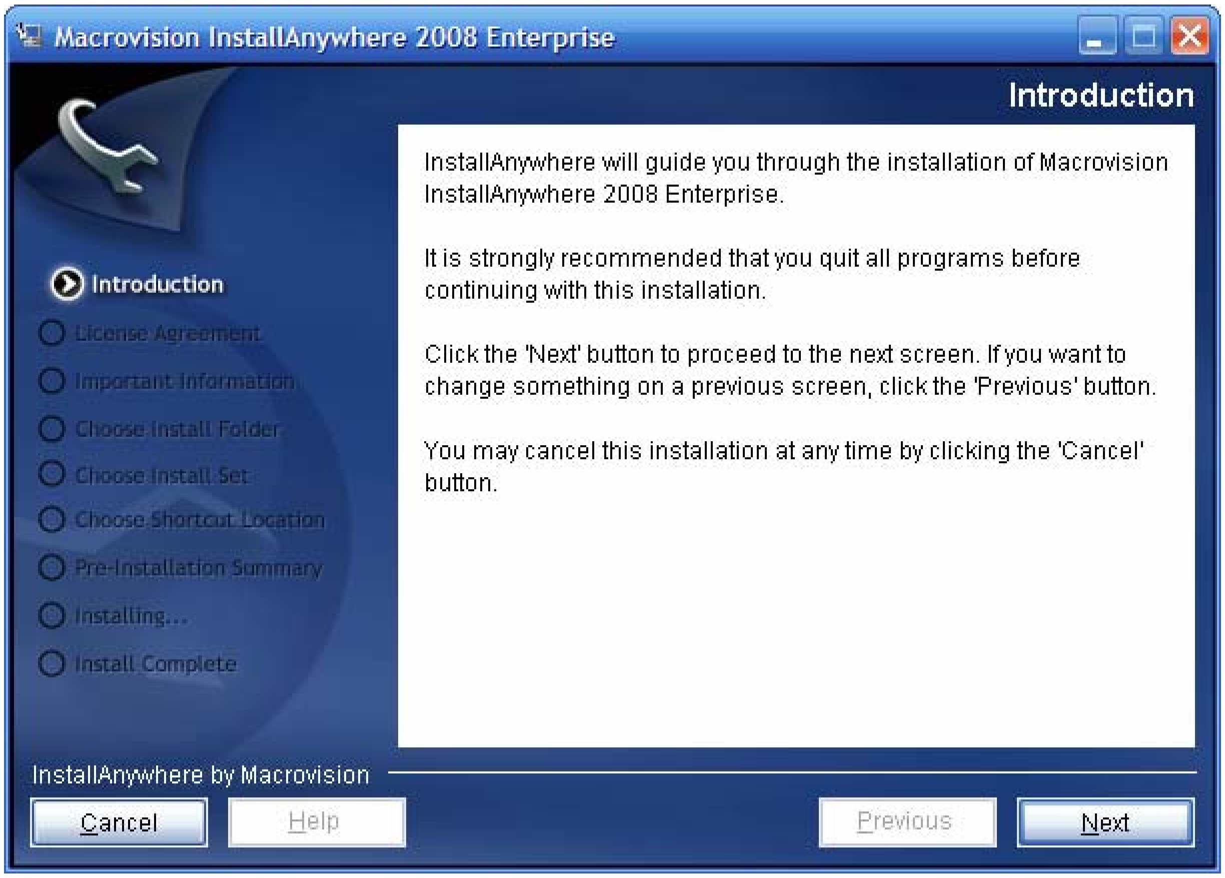

InstallAnywhere

InstallAnywhere is preparing to install...

82%

(C) 1997-2007 Macrovision Europe Ltd. and/or Macrovision Corporation

**Figure 2-2: Running the InstallAnywhere Installer**
InstallAnywhere's installer utilizes some of InstallAnywhere's advanced user interface (UI) customization options. For example, note the background images and the dynamic list of installation steps used in the installer. These features are available when you use InstallAnywhere's advanced GUI user interface.


Macrovision InstallAnywhere 2008 Enterprise

Introduction

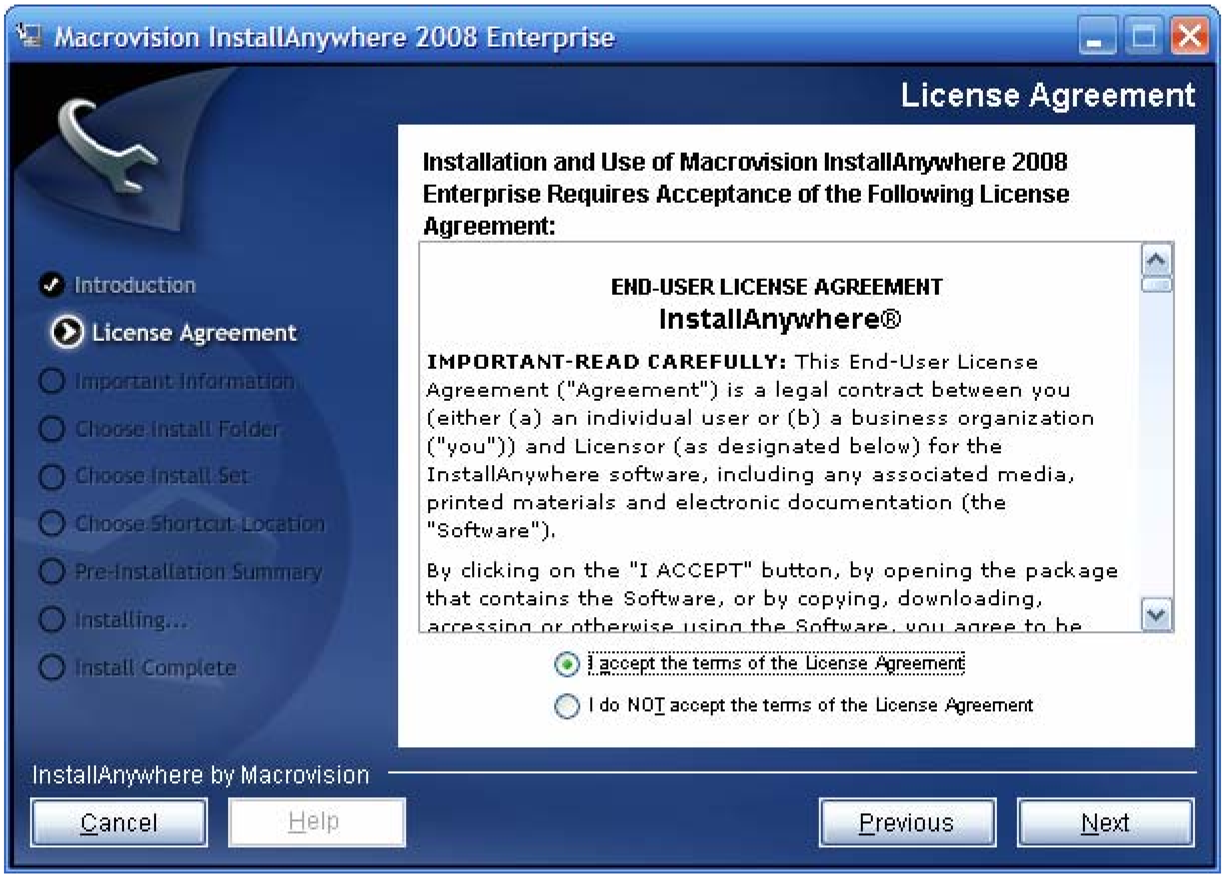
InstallAnywhere will guide you through the installation of Macrovision InstallAnywhere 2008 Enterprise.

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

Click the 'Next' button to proceed to the next screen. If you want to change something on a previous screen, click the 'Previous' button.

You may cancel this installation at any time by clicking the 'Cancel' button.

**Figure 2-3: Informational Pages**
The left pane of the Installer can present a list of steps that the installer will perform. These steps will be updated as the installation progresses. In the right pane the Installer can present information and accept information from the end user.



Installation and Use of Macrovision InstallAnywhere 2008 Enterprise Requires Acceptance of the Following License Agreement:

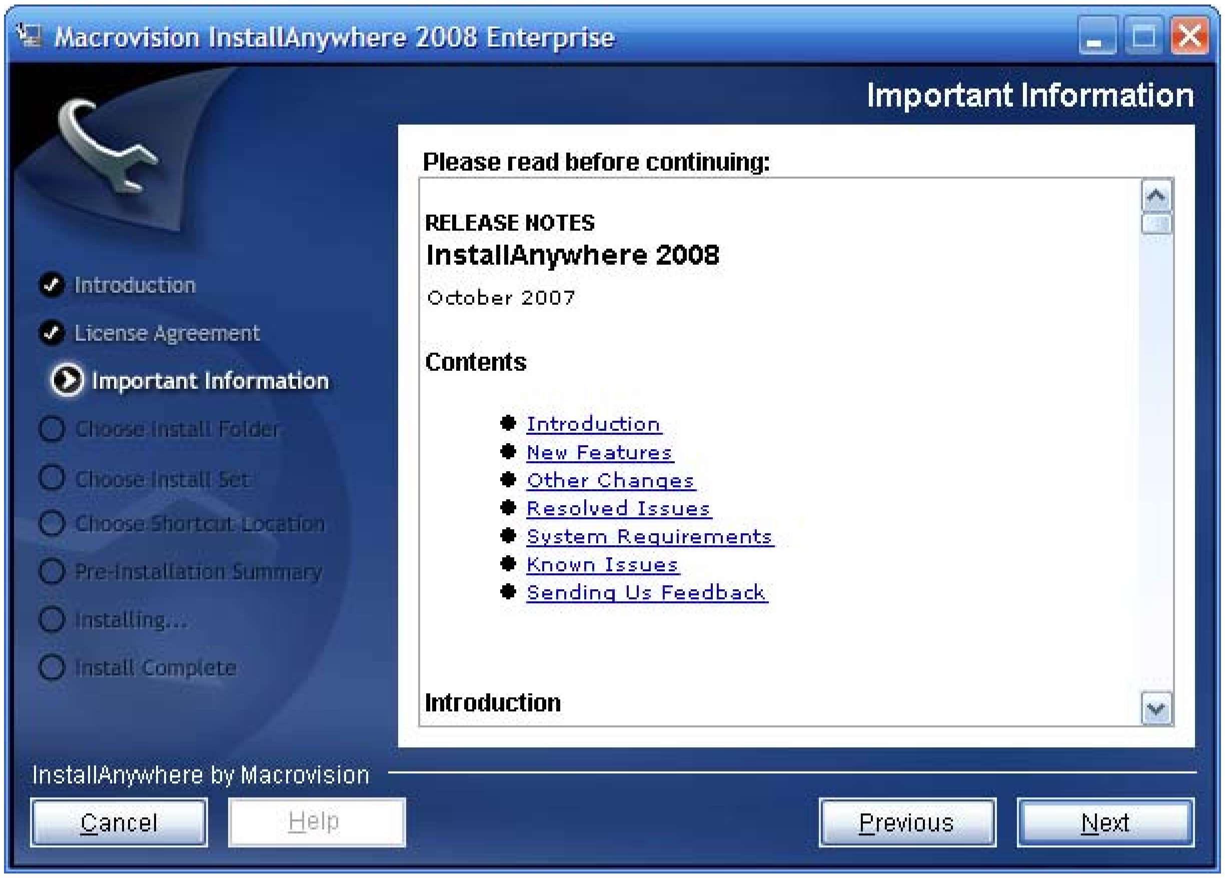
END-USER LICENSE AGREEMENT
InstallAnywhere®

**IMPORTANT-READ CAREFULLY:** This End-User License Agreement ("Agreement") is a legal contract between you (either (a) an individual user or (b) a business organization ("you")) and Licensor (as designated below) for the InstallAnywhere software, including any associated media, printed materials and electronic documentation (the "Software").

By clicking on the "I ACCEPT" button, by opening the package that contains the Software, or by copying, downloading, accessing or otherwise using the Software, you agree to be

[✓ I accept the terms of the License Agreement]
I do NOT accept the terms of the License Agreement

**Figure 2-4: Present Options to the End User**
With the InstallAnywhere installer, a license agreement is presented. The user must accept before the installation can continue.



Please read before continuing:

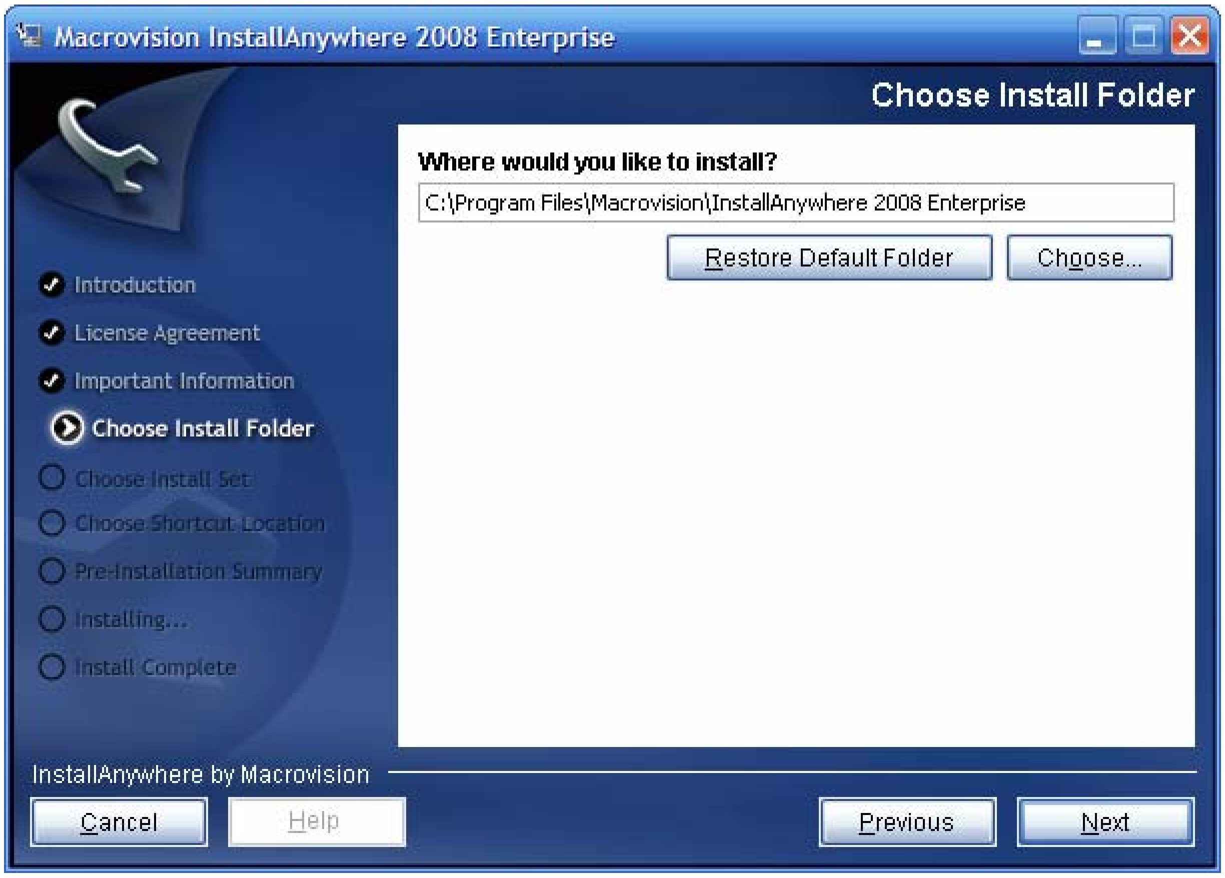
RELEASE NOTES
InstallAnywhere 2008
October 2007

Contents

- Introduction
- New Features
- Other Changes
- Resolved Issues
- System Requirements
- Known Issues
- Sending Us Feedback

Introduction

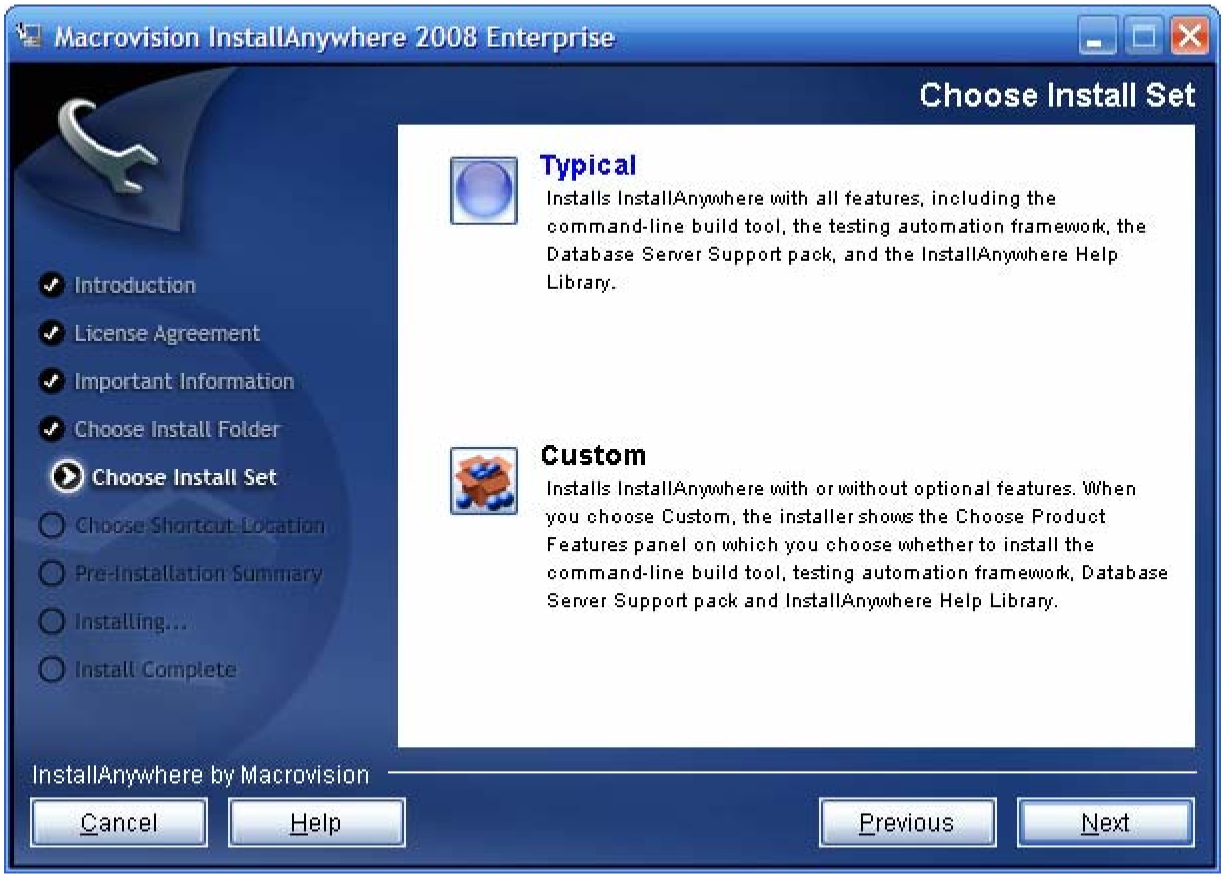
**Figure 2-5: Present HTML from within the installer**
A great deal of information can be packaged in the installer, even HTML, including hyperlinks. The InstallAnywhere installer uses this panel to display the product's release notes.



Where would you like to install?

C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise

**Figure 2-6: Request Installation Configuration Information**
The installer can request installation configuration information from the end user; in this case, the installation location.

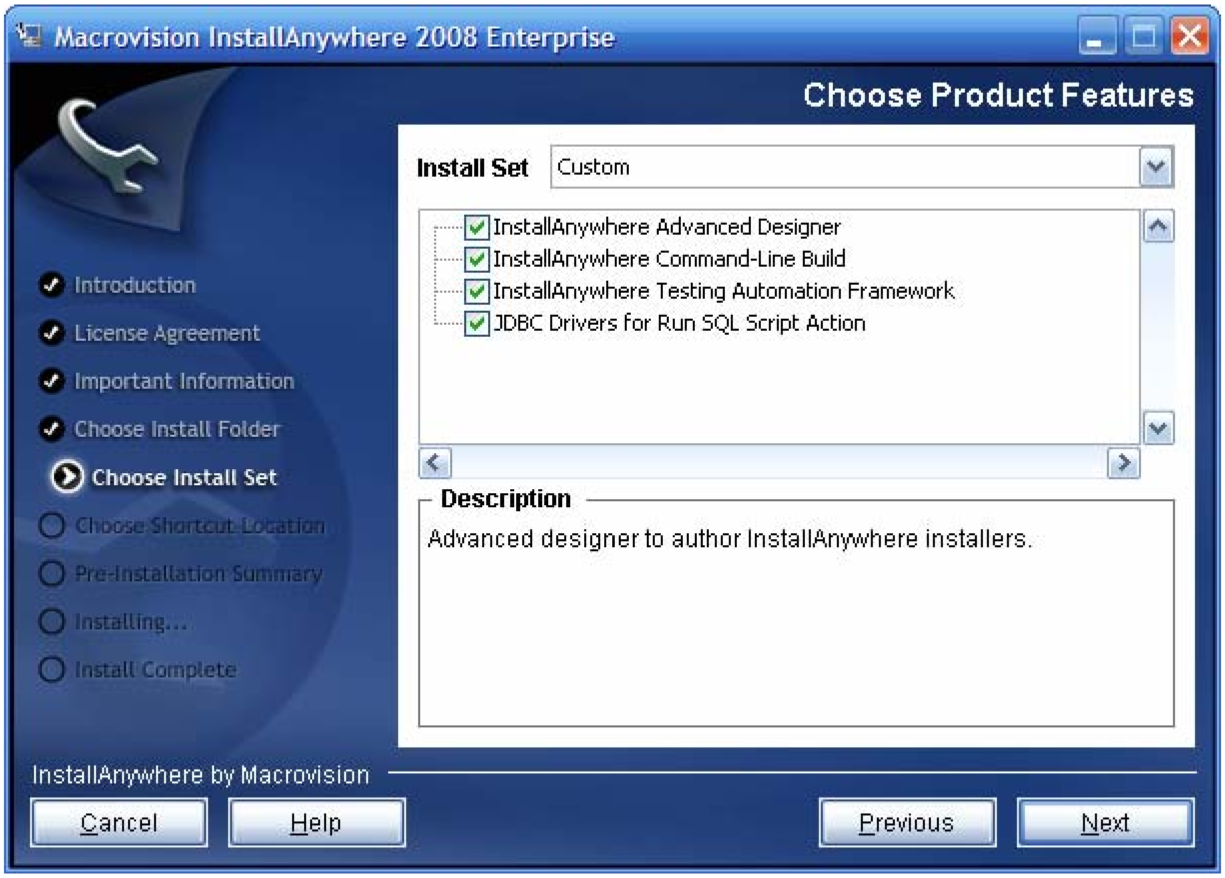


Choose Install Set

Typical
Installs InstallAnywhere with all features, including the command-line build tool, the testing automation framework, the Database Server Support pack, and the InstallAnywhere Help Library.

Custom
Installs InstallAnywhere with or without optional features. When you choose Custom, the installer shows the Choose Product Features panel on which you choose whether to install the command-line build tool, testing automation framework, Database Server Support pack and InstallAnywhere Help Library.

**Figure 2-7: Choose Install Set**
The installer lets the user select an install set, which is a prepackaged group of product features. Selecting the Typical install set installs all the InstallAnywhere features, while selecting the Custom install set prompts the user for the specific features to install.



Choose Product Features

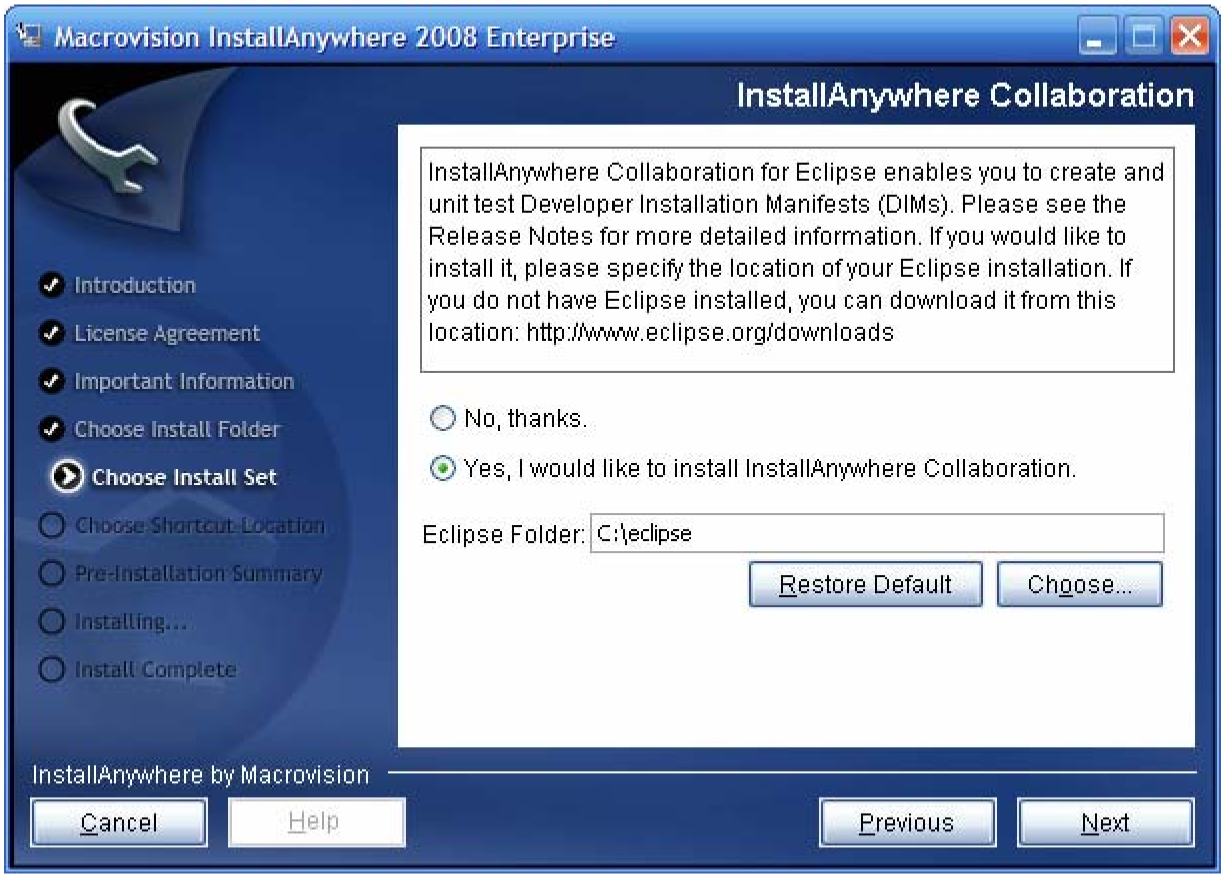
Install Set: Custom

- [✓] InstallAnywhere Advanced Designer
- [✓] InstallAnywhere Command-Line Build
- [✓] InstallAnywhere Testing Automation Framework
- [✓] JDBC Drivers for Run SQL Script Action

Description

Advanced designer to author InstallAnywhere installers.

**Figure 2-8: Choose Product Features**
If the user selects the Custom install set, the installer prompts the user with a list of individual product features that can be installed separately.



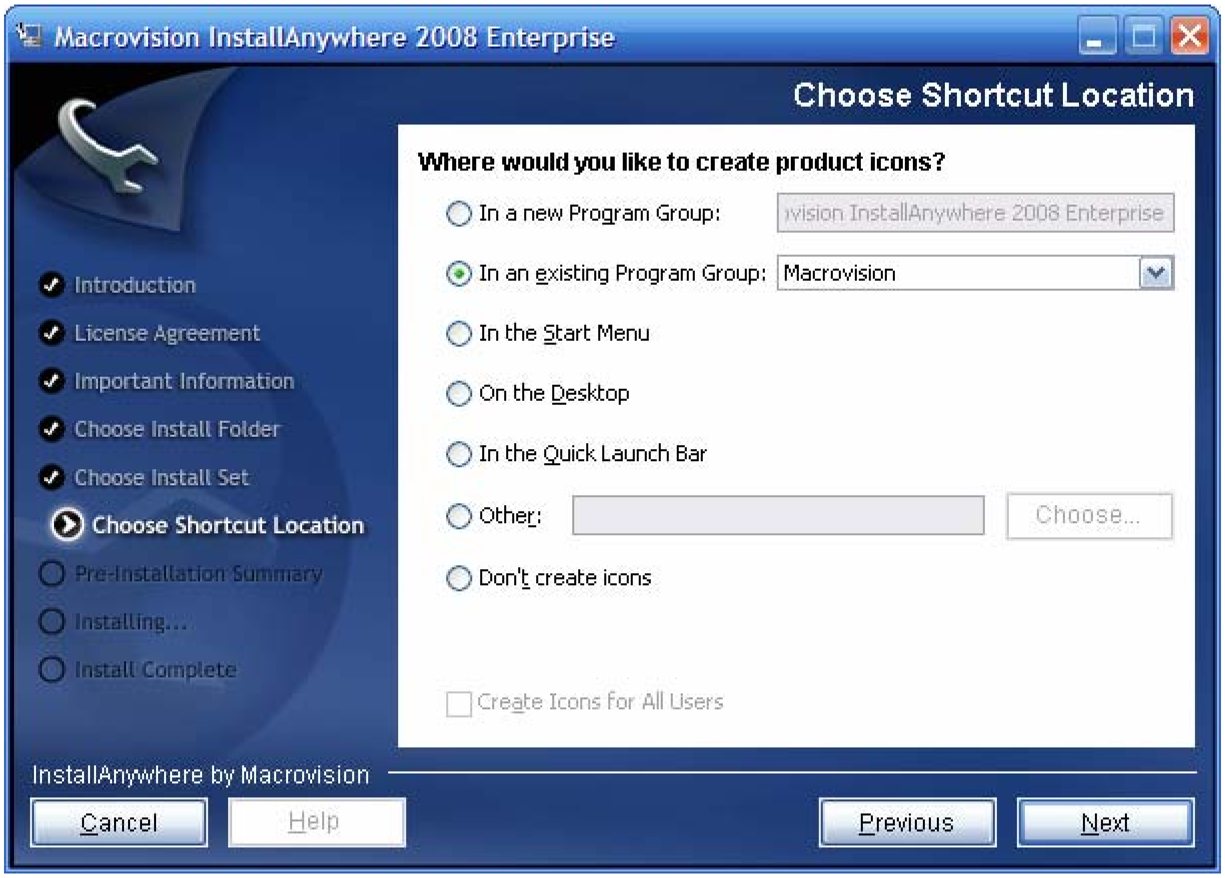
InstallAnywhere Collaboration for Eclipse enables you to create and unit test Developer Installation Manifests (DIMs). Please see the Release Notes for more detailed information. If you would like to install it, please specify the location of your Eclipse installation. If you do not have Eclipse installed, you can download it from this location: http://www.eclipse.org/downloads

No, thanks. [✓ Yes, I would like to install InstallAnywhere Collaboration.]

Eclipse Folder: C:\eclipse

**Figure 2-9: InstallAnywhere Collaboration**

InstallAnywhere supports the use of DIMs as a way for product developers to collaborate with installation developers. The InstallAnywhere installer provides the option to install InstallAnywhere Collaboration for Eclipse, which is a tool for creating DIMs that can be consumed by InstallAnywhere. Use of DIMs in InstallAnywhere is described in Chapter 4.


Macrovision InstallAnywhere 2008 Enterprise

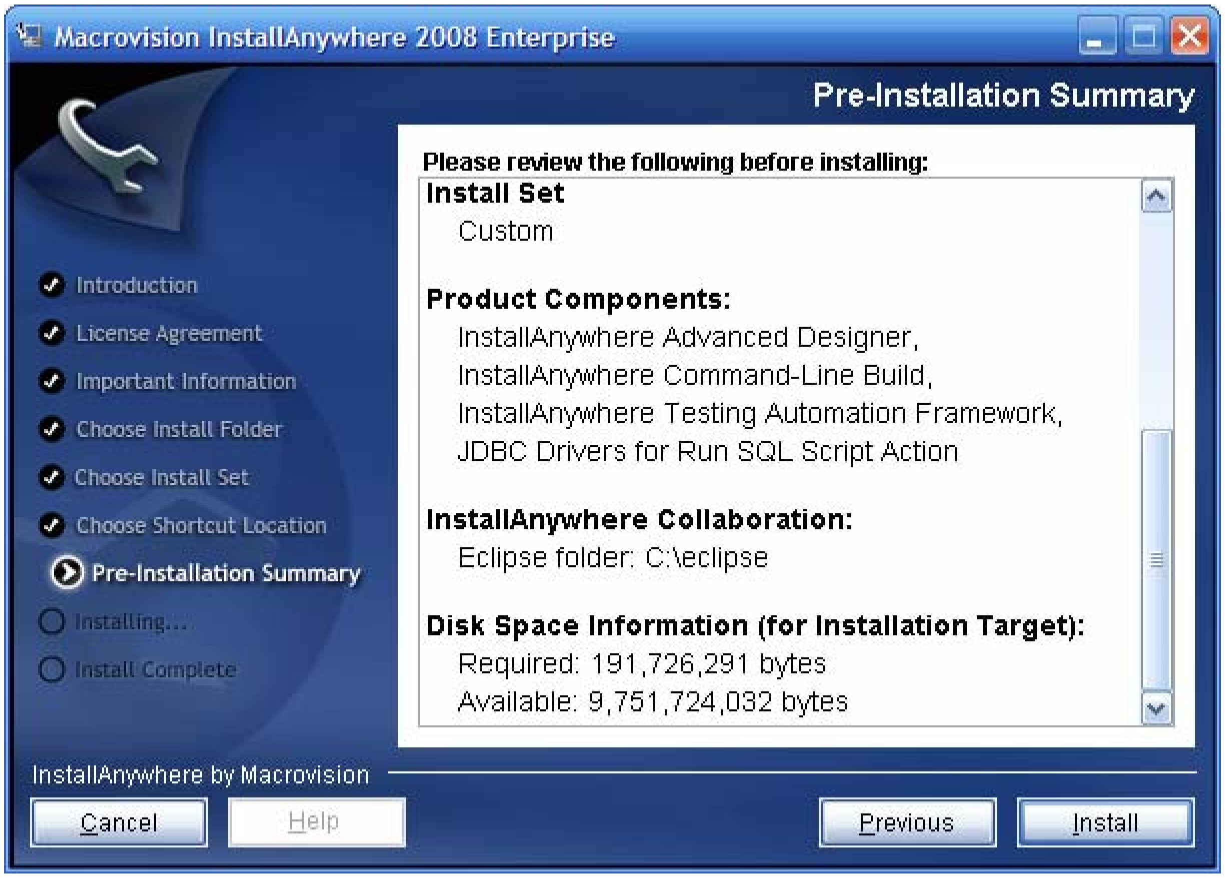
Choose Shortcut Location

Where would you like to create product icons?

- [ ] In a new Program Group: ›vision InstallAnywhere 2008 Enterprise
- [✓] In an existing Program Group: Macrovision
- [ ] In the Start Menu
- [ ] On the Desktop
- [ ] In the Quick Launch Bar
- [ ] Other:
- [ ] Don't create icons

- [ ] Create Icons for All Users

**Figure 2-10: End User Shortcut Definition**
The InstallAnywhere installer requests that the user choose a location for shortcuts to be installed. When the installation is done on a Unix system, the same panel would reflect links rather than shortcuts, and on a Macintosh system the user would choose where to install aliases.



Please review the following before installing:

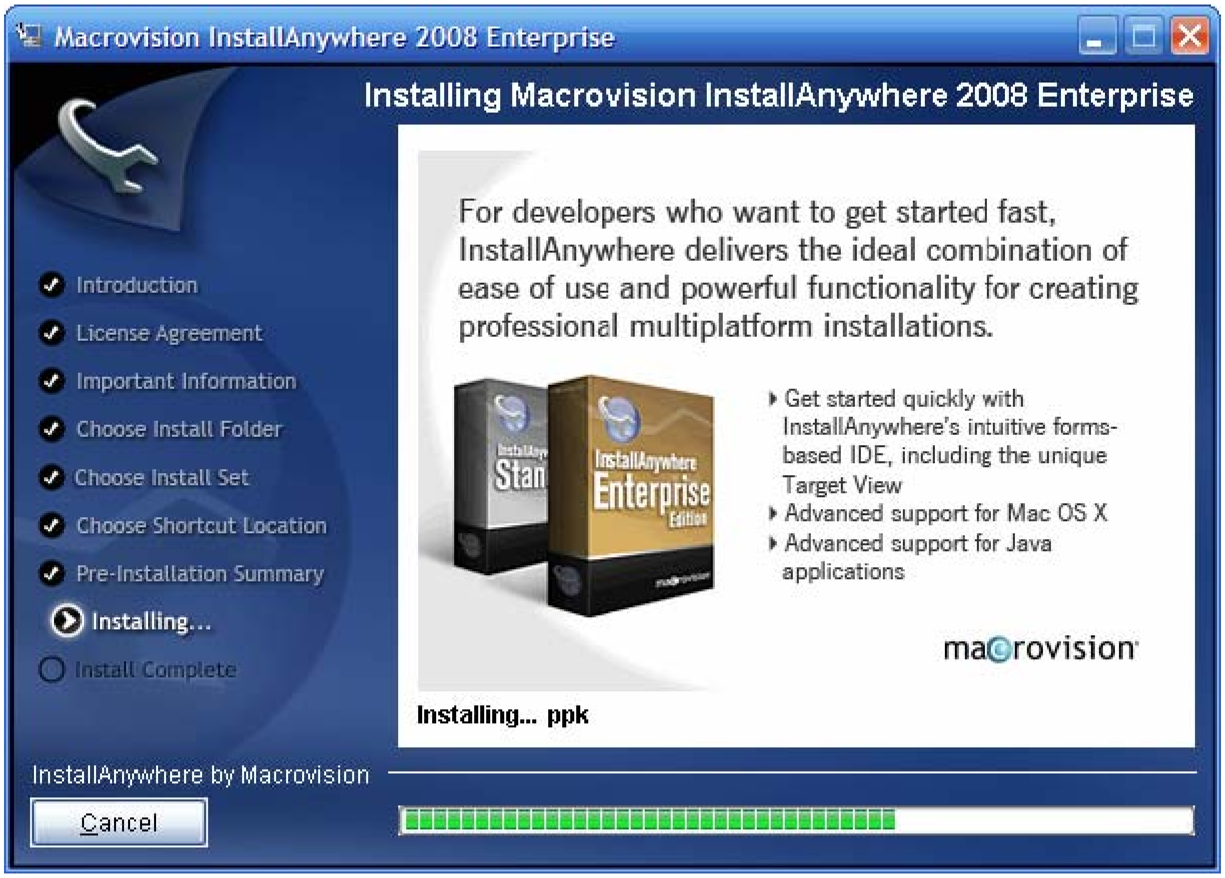
Install Set
Custom

Product Components:
InstallAnywhere Advanced Designer,
InstallAnywhere Command-Line Build,
InstallAnywhere Testing Automation Framework,
JDBC Drivers for Run SQL Script Action

InstallAnywhere Collaboration:
Eclipse folder: C:\eclipse

Disk Space Information (for Installation Target):
Required: 191,726,291 bytes
Available: 9,751,724,032 bytes

**Figure 2-11: Pre-install Summary**
The InstallAnywhere installer displays a summary of information gathered in the installation so far. This summary includes disk space calculations and user choices.



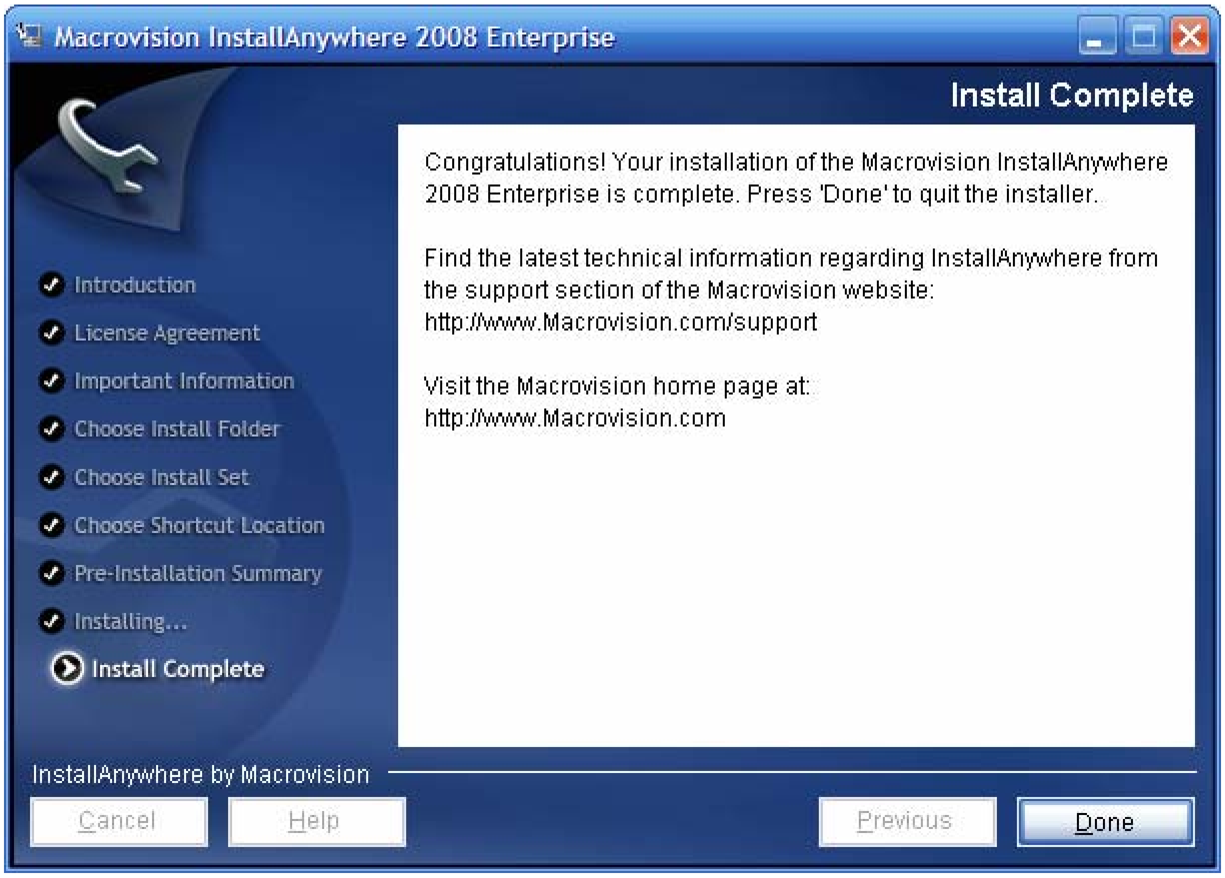
Installing Macrovision InstallAnywhere 2008 Enterprise

For developers who want to get started fast, InstallAnywhere delivers the ideal combination of ease of use and powerful functionality for creating professional multiplatform installations.

- Get started quickly with InstallAnywhere's intuitive forms-based IDE, including the unique Target View
- Advanced support for Mac OS X
- Advanced support for Java applications

Installing... ppk

**Figure 2-12: Present Billboards, Animated Graphics, and Progress Bar**
Finally, the installer proceeds to lay down files. During the actual file install, the user is presented with a progress bar and textual feedback. The installer can display animated graphics about the product or about other available products offered (in this case by Macrovision).



Congratulations! Your installation of the Macrovision InstallAnywhere 2008 Enterprise is complete. Press 'Done' to quit the installer.

Find the latest technical information regarding InstallAnywhere from the support section of the Macrovision website:
http://www.Macrovision.com/support

Visit the Macrovision home page at:
http://www.Macrovision.com

**Figure 2-13: Install Complete**
When the file installation is complete, the installer presents a panel indicating that the install has completed successfully (or, should there be any errors, which errors have occurred).

Conveniently, at the completion of this demonstration, you will have installed the InstallAnywhere product you will be using for the remainder of the training session.

## *The Server-Side Installer Experience*

InstallAnywhere also includes features that can be used in the installation of server-side products. The needs of a server product installation are often different than that of a client application. Server-side installations normally require more information from the user, and in many cases the installation will not be run in a graphical environment. One way InstallAnywhere accommodates this type of installation is through support for console installations.

Using the same intuitive development interface as client-side installations, and in graphical environments the same graphical installer, the user can be presented configuration choices to assist with the configuration of the installation. In addition, you can use InstallAnywhere's Get User Input panel to request arbitrary input from the user, such as requiring the user to choose an application server. The Get User Input panel does not require you as a developer to write any code, and can be configured in a matter of seconds.

# Chapter 3: The InstallAnywhere Developer Experience

- The InstallAnywhere Wizard
- Building Your First Installer

InstallAnywhere has two authoring modes. The Project Wizard makes the process easier by making choices for developers, while the Advanced Designer gives developers greater control of an installer project.

InstallAnywhere opens displaying the first frame of the Project Wizard, unless the default preference has been changed using the **Preferences** command in the **Edit** menu. To access an existing InstallAnywhere project, click **Open Existing Project**, click the project name in the Open Recent Project list, and then click **Advanced Designer**.

The general process for developing an InstallAnywhere project is:

2. *Create a new project*
3. *Set project information*
4. *Add pre-install actions*
5. *Install tasks*
6. *Add post-install actions*
7. *Set installer UI options*
8. *Configure the project uninstaller*
9. *Build the project*
10. *Test the project*

The introductory tutorial builds an installer using the Project Wizard, which does not allow configuring pre-install or post-install actions.

## The InstallAnywhere Wizard

InstallAnywhere provides two distinct interfaces for creating projects and building InstallAnywhere installers. In this section, you will see how to use the InstallAnywhere Project Wizard. This intuitive wizard guides you through the creation of a basic InstallAnywhere project customized for your product.

You can build your first installer in less than five minutes with the six-step Project Wizard. This intuitive design tool also sets the Classpath and finds the Main Class for a Java application automatically.

## Building Your First Installer

The following tutorial teaches how to build an installer for a sample Java application, called “OfficeSuite for Java,” which is included in the InstallAnywhere folder.

1. *Create New Project*
   a. *Launch InstallAnywhere.*

      On the first screen, the Create New Project option should already be selected.

   b. *Click* **Save As** *to save and name the project.*

      The **Save New Project As** dialog box appears. By default the project is named My_Product, but you can use any name.

   c. *Click* **Save** *to confirm the name and close this dialog box.*
   d. *Click* **Next** *to move to the next step of the Project Wizard.*

2. *Set Info*

   Set Info defines basic information about the installer, such as the product name (as displayed on the installer), the name of the installer to be produced, the name of the destination folder, and the application name.

   a. *Type the information in the appropriate text boxes. For this tutorial, use the following:*

      | | |
      |---|---|
      | Product Name: | `OfficeSuite` |
      | Install Folder Name: | `OfficeSuite` |
      | Application Shortcut Name: | `OfficeSuite` |

      The default Install Folder Name value $PRODUCT_NAME$ is an InstallAnywhere variable, which expands to the string product name at run time. Variables are described later in this course.

   b. *Click Next to move to the next step in the Project Wizard.*

3. *Install tasks.*

   a. *Add Files*

      i. *Click* Add Files*. The* Add Files to Project *dialog box appears. Browse through the list to find the* `OfficeSuiteSourceFiles` *folder. The* `OfficeSuiteSourceFiles` *folder is within the InstallAnywhere installation folder.*

      ii. *Click* Add All *to add the ImagesAndDocs and OfficeSuite2000 folders, which are inside the OfficeSuiteSourceFiles folder.*

         These files should appear in the Files to Add list.

         This type of file linking adds a static list of files to your project: any files you later add to this source directory will not automatically be added to your project. To specify a directory from which InstallAnywhere should regenerate a dynamic list of source files during each build, you can use the SpeedFolder functionality, described later in this course.

      iii. *Click* Done*.*

         The selected files should appear in the File/Folder Hierarchy. (The User Install Folder location $USER_INSTALL_DIR$ is another example of an InstallAnywhere variable. Its value initially represents the default installation location for your product's files, and the value changes if the user selects a non-default install location.)

      iv. *Click* Next *to move to the next step in the Project Wizard.*

   b. *Choose Main Class*

      Choose Main Class selects the starting class for the application. A Java application contains one or more classes that implement a method called *main*; this wizard panel is where you specify the class whose *main* method you want to execute when a user launches your LaunchAnywhere executable. (If you are not installing a Java application, click Next without specifying a main class.)

      This frame also allows developers to specify custom icons (in GIF format) for the LaunchAnywhere executable file.

      i. *Click* Automatically Find Main Classes *at the bottom of the screen.*

      ii. *Select the Main Class.*

      iii. *Specify a custom icon for the LaunchAnywhere executable by clicking* Change *and choosing a 32x32 or a 16x16 pixel GIF for the application icon. (Note that Windows-only .ico files are not supported.) Navigate to the* `Image and Docs` *folder and choose* *`OfficeIcon.gif`.*

      iv. *Click* OK *to confirm and close the dialog box.*

         The icon will appear on the main screen.

      v. *Click* Next *to move to the next step in the Project Wizard.*

   c. *Set Classpath*

i. *Set Classpath automatically configures a Java application's classpath, which is a list of directories and JAR files containing classes used by the application. For example, to deploy a Java application packaged as a JAR file, the JAR file is required on the application's classpath. This is reflected in the command used to manually launch a Java application, such as:*

*java -cp TrainApp.jar TrainingAppMainClass*

*In this case, TrainApp.jar is on the classpath.*

Click *Automatically Set Classpath*. InstallAnywhere will calculate which files need to be added to the classpath. A small "CP" icon will appear at the bottom of those folders.

ii. *Click Next to move to the next step in the Project Wizard.*

4. *Build Installer.*

   The first several items on the Build Installer screen, from Mac OS X through Unix (All), represent installers that are double-clickable on their respective platforms. The final option, Other Java-Enabled Platforms, is a "pure" Java installer that can be invoked from the command line on any Java-enabled platform. Developers may also choose to build installers with an embedded Virtual Machine, where the embedded VM will be used to run the installation. Installers that are built without VMs are smaller, and download faster than installers bundled with one. The InstallAnywhere Web Install process will allow end users to choose the appropriate installer for their system.

   Pick the appropriate destination platforms and click **Build**.

   The installer folder will be placed in a sub-directory in the same location as the project file. This location cannot be changed.

5. *Test.*

   a. *Now that an installer has been built, it can be tested by clicking Try It.*

   b. *After deploying the sample installer:*

      - On Windows, go to the OfficeSuite program group and choose OfficeSuite.
      - On Unix, "cd" to the directory where the program was installed and type OfficeSuite.
      - On Mac OS X, double-click the OfficeSuite icon on the desktop.

   c. *After launching OfficeSuite for Java, quit by selecting Exit from its File menu.*

      It is possible to post the installer folder to a Web server and install the software onto another platform as well.

TIP: Hold down the Control (Ctrl) key while the installer launches to see the debug output (Windows only).

   d. *Run the completed installer.*

      When building for a platform other than that on which the installer is being developed, transfer that installer, and run it manually. By default, installers will be located in the `Build_Output` directories found in the same folder as the `.iap_xml` project file. Within the `Build_Output` folder, will be `Web_Installers`, and or `CDROM_installers`. From within each of these directories, choose the platform to test. For the CDROM installer, transfer the entire contents of the CDROM_Installers folder.

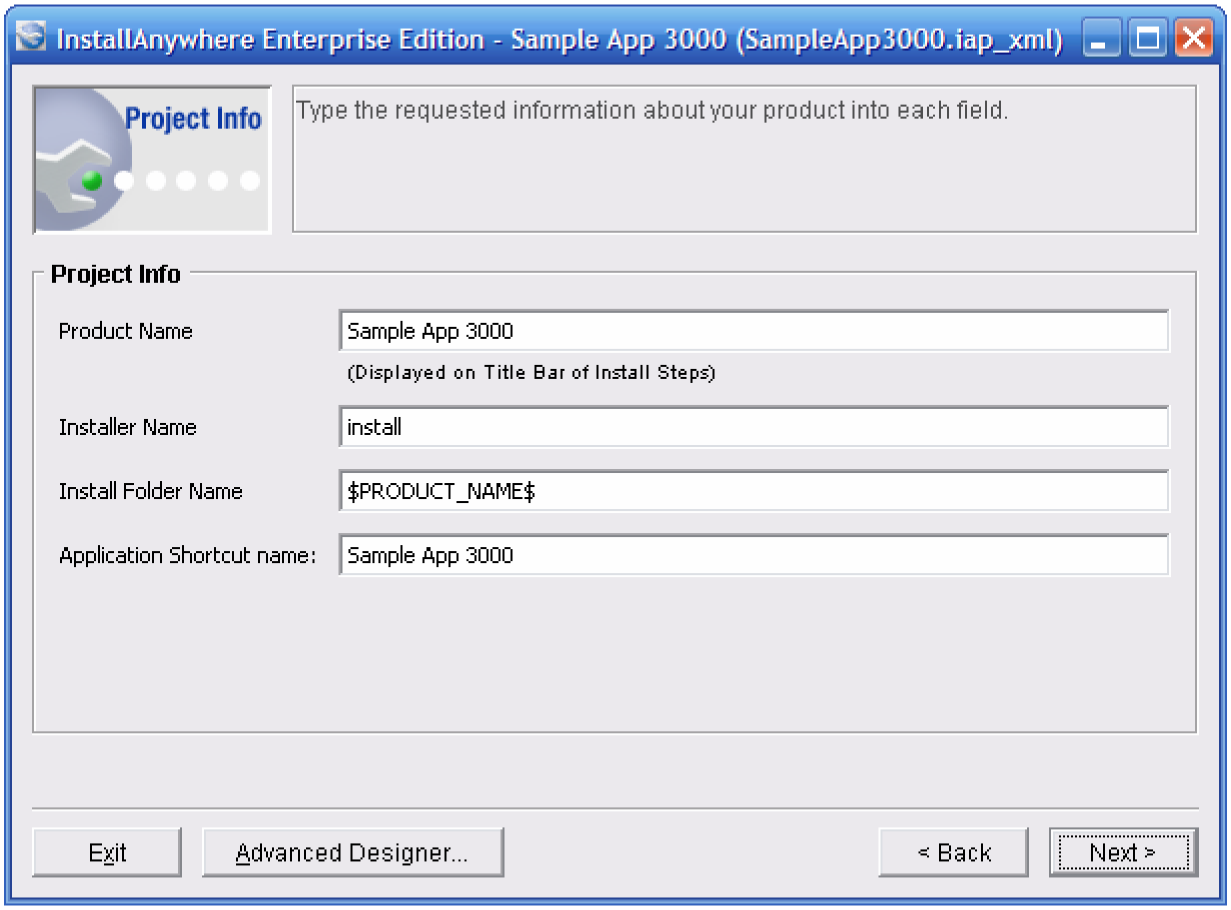
# Chapter 4: Key Concepts in InstallAnywhere

- **Authoring Environments**
- **Installer Types**
- **Installer Modes**
- **Install Sets, Features, and Components**
- **Installer Interface GUI**
- **Actions**
- **Rules**
- **Uninstaller**
- **LaunchAnywhere**
- **InstallAnywhere Variables**
- **Magic Folders**
- **SpeedFolders**
- **Strict VM Selection**
- **Project File**
- **Manifest Files**
- **InstallAnywhere Collaboration and DIMs**
- **FLEXnet Connect**

This chapter gives a conceptual framework to InstallAnywhere installer development. InstallAnywhere provides many complex options for creating installer functionality.

## Authoring Environments

InstallAnywhere offers two authoring environments. One environment is the Project Wizard, which guides you through creating a new installer quickly, using a simple step-by-step wizard interface for describing your project, linking to files, defining shortcuts and icons, and building releases.


InstallAnywhere Enterprise Edition - Sample App 3000 (SampleApp3000.iap_xml)
Project Info
Exit
Advanced Designer...
< Back
Next >

Type the requested information about your product into each field.

Project Info

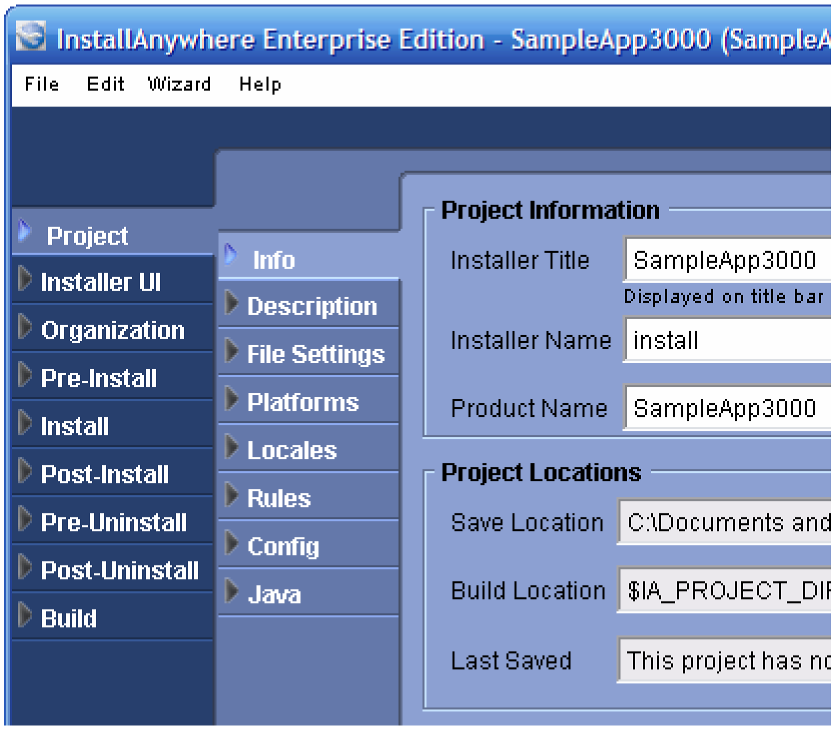
Product Name: Sample App 3000
(Displayed on Title Bar of Install Steps)

Installer Name: install

Install Folder Name: $PRODUCT_NAME$

Application Shortcut name: Sample App 3000

The other environment is the Advanced Designer, which provides much finer control over installer functionality. The Advanced Designer enables you to define multiple Install Sets, add customized splash screen graphics, and execute commands from within the installation process, along with many other advanced features.



Project Information

Installer Title: SampleApp3000

Displayed on title bar

Installer Name: install

Product Name: SampleApp3000

Project Locations

Save Location: C:\Documents and

Build Location: $IA_PROJECT_DIF

Last Saved: This project has nc

You can begin an installer project in the Project Wizard and then switch to the Advanced Designer (by clicking the Advanced Designer button in the Project Wizard) to define advanced functionality.

## Installer Types

InstallAnywhere offers two types of installers: Web Installers and CD-ROM Installers. Merge Modules and Templates, which are essentially InstallAnywhere sub-projects, are also installer types.

### Web Installers

Web Installers are packaged into a single self-extracting executable file by platform, and are appropriate for distribution over the Web or through e-mail. They use a Self-Extractor to prepare the source files at the start of the installation, and therefore require more temporary disk space.

### CD-ROM Installers

In a CD-ROM Installer, all of the installation resources are located in an external archive. There is still a Self-Extractor, but it extracts only the installer engine. Therefore, CD-ROM installers use less temporary space and start up faster, but they are not appropriate for Web or electronic distribution. CD-ROM installers are also good for distribution on DVD, and the installer can span multiple CDs or DVDs if the installation data are larger than the capacity of the selected media type.

### Merge Modules

Merge Modules are essentially installer sub-projects that can be created independently of one another and later merged together. A Merge Module is a reusable collection of installation functionality. A Merge Module is just like an installer, complete with features, components, panels, actions, and files. However, a Merge

Module cannot be installed without being part of an InstallAnywhere project; instead, you use Merge Modules to include the functionality of one installer within another installer.

### *Templates*

A template is the starting point for every new installer project. A template can be a simple empty project, or it can contain everything a regular project would contain, such as license agreements, custom graphics and billboards, and even files. Templates provide a convenient way for maintaining a consistent set of project options across multiple installers.

## *Installer Modes*

InstallAnywhere installers can run in three different modes:

- GUI—graphical user interface with wizard panels and dialog boxes
- Console—also known as command-line interface, this mode is meant for remote installations over telnet, or on systems without a graphical windowing environment
- Silent—these installers do not interact with the user at all, and are suitable for distribution when all of the settings are already known or provided in a response file

## *Install Sets, Features, and Components*

The hierarchy of Install Sets, Features, and Components is one of the most important concepts to understand when using the InstallAnywhere installer development environment.

InstallAnywhere provides levels of granularity for end-user installation options. Install Sets and Features may be selected by the end user. Install sets are groupings of features, such as Typical Install or Minimal Install. Features are meant to identify specific units of functionality of your product. While both Install Sets and Features are made up of files, there is not necessarily a direct correlation between these larger organizational groupings and the files.

Components are groupings of the specific files and actions of your product, and are invisible to the end user. A component may also include a group of registry changes or other elements needed to make a feature work properly. Components are used for the organized sharing of resources, for versioning, are uniquely identified, and are the organization tool of the installer developer, not the installer user. A developer could create a component for each feature in the application, a component for shared libraries used by all the features, and a component for the help system.

Though developers may assign files, folders and actions directly to Product Features, it is best to think of features as groupings of components. Install sets such as Typical Install or Minimal Install are groupings of Features. The interaction between the three levels should be addressed when planning the options to present to the end user.

## *Installer Interface GUI*

Most aspects of an InstallAnywhere installer's user interface can be modified. As a developer of an InstallAnywhere installer, you can alter text strings as well as graphics, such as the splash screen, installer screens, panel additions, background images, and billboards. Developers may even add their own animated GIF files to provide a multimedia experience.

## *Localization*

Nearly every text string in an InstallAnywhere project can be localized. Translations of the text of built-in InstallAnywhere screens and dialog boxes are already provided. The Enterprise Edition supports 31 different locales, and the Standard Edition supports 9.

If developers want to further modify text strings by locale, the string files are output each time an installer is built, in a folder called “Project locales” which will be next to the build output folder. The files are named by locale code. For example, the default English (with locale code `en`) locale file has the name `custom_en`.

These locale files contain the text strings grouped by the name of the action to which it belongs. Developers may alter the text strings, and upon the next build of the installer the new localized text will be displayed with the action.

## *Look and Feel*

InstallAnywhere provides many options for altering the look and feel of the installer. For illustrations of these types of customizations, see Chapter 8.

### Splash Screen

InstallAnywhere installers present a splash screen at the initial launch of the installer. This screen is displayed for a few seconds while the installer prepares the wizard. The splash screen is an ideal introduction to your product, and is an opportunity to set the mood and image for your product. The splash screen will also appear on the HTML page generated for the InstallAnywhere Web Install Applet. It can be either a GIF, PNG, or JPEG image of any size, although the preferred size is 470 by 265 pixels.

### GUI Panel Additions

The Additions to GUI Installer Panels option allows developers to display a list of steps or an image along the left side of the installer’s panel.

### Background Images

This Background image feature enables you to create a truly unique installer. The Background image is the graphical background for every panel in your installer. Naturally, background images are supported only in GUI-mode installers.

### Billboards

Billboards are graphics that the installer will display during the installation of files. Billboards generally convey a marketing message, a description of the product, or simply something entertaining for the end user to see as data transfer is taking place. Each billboard added will be displayed for an equal amount of time, based on actions within the installation. If an installation has very few files and many billboards, each billboard will only be displayed for a short time.

Billboards can be GIF, PNG, or JPEG files, and should be 587 by 312 pixels in size. Billboards can even use animated GIF files, providing the end user with a richer media experience during their installation.

## Actions

InstallAnywhere supports an extensible action architecture that gives developers the ability to perform operations during installation. Some of these actions are as simple as installing files and folders and as complex as creating modifying text files, executing custom code during the installation process, or extracting contents from a compressed file.

Actions may occur in the background, not requiring any user input, or may require user input. General/Install Actions do not require any user input. Panel Actions and Console Actions request user input. Panel Actions display a graphic element that requests user input. Console Actions display a command-line request.

### General Actions

Most General Actions make system changes, search the target system for data, or read data from the target system. Examples include creating or deleting folders, modifying a text file, or reading data from the Windows registry. These actions are generally transparent to the end user, and do not require any user input.

### Panel Actions

Panel Actions are requests for user input that appear inside the graphical installer wizard. Examples are the standard panels for welcoming the user, prompting for a password or a product destination folder, and displaying summary information at the end of installation.

### Console Actions

Console Actions are displays of information and requests for user input used during a command-line installation.

### Plug-in Actions

Custom code can be integrated with the InstallAnywhere Designer and will appear as a plug-in.

### Action Groups

Action groups make InstallAnywhere projects more manageable and easier to understand. They enable the developer to logically group of set of actions or panels in Pre-Install, Post-Install, and Post-Uninstall. Rules applied to the Action Group affect all of the actions or panels contained inside it. Action groups are useful with complex installations that contain numerous actions and panels.

## Rules

InstallAnywhere Rules can be applied to any action within the InstallAnywhere installer, as well as to organizational units such as Install Sets, Features, and Components.

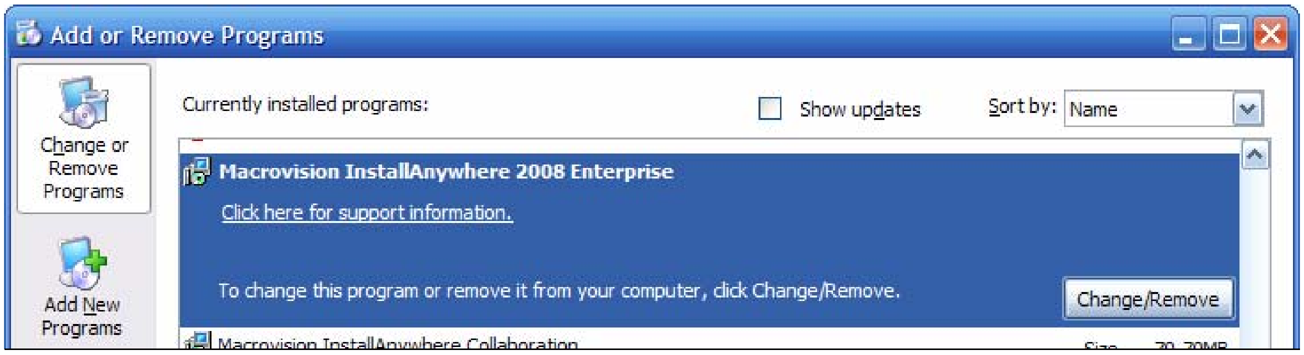
InstallAnywhere uses variable-based Boolean rules to control most aspects of installer behavior. The Rules logic allows developers to create simple and complex logic systems that determine which actions will occur. The rules can be structured based on end-user input, or on conditions determined by the installer.

## Uninstaller

InstallAnywhere automatically creates an uninstaller for each project. The uninstaller can be removed manually. The InstallAnywhere uninstaller typically removes all files and actions that were installed during the installation, although you can control this behavior as desired.

The uninstaller is much like the installer. It is a collection of panels, consoles, and actions. It keeps track of what the installer has done, and contains a record of every action run during install time. All Pre-Uninstall panels, actions, and consoles run first. Then the uninstaller removes files and performs the uninstall operation of any action found in the files tree. Finally, Post-Uninstall actions are performed.

In addition, an InstallAnywhere installer can create an Add or Remove Programs entry on Windows systems, similar to the following.


Add or Remove Programs
Change or Remove Programs
Add New Programs
Currently installed programs:
Show updates
Sort by:
Name
Macrovision InstallAnywhere 2008 Enterprise
Click here for support information.
To change this program or remove it from your computer, click Change/Remove.
Change/Remove
Macrovision InstallAnywhere Collaboration

## LaunchAnywhere

A native executable used to launch a Java application, LaunchAnywhere is Macrovision’s Java application launcher technology. LaunchAnywhere technology creates double-clickable icons on Windows and Mac OS X. On Unix platforms, a command-line application is created.

A LaunchAnywhere Java application launcher automatically locates an appropriate Java Virtual Machine (JVM), either bundled with the application or already installed on the system, and sets the Java options and settings (such as heap size) depending on the developer’s specifications. LaunchAnywhere sets the classpath, redirects standard out and standard error, passes in system properties, environment variables, and command-line parameters, and launches the Java application. LaunchAnywhere hides the console window by default for GUI applications, or can be set to display the console for text-based applications. All LaunchAnywhere settings are configured within InstallAnywhere, and are automatically set when the installer installs the application.

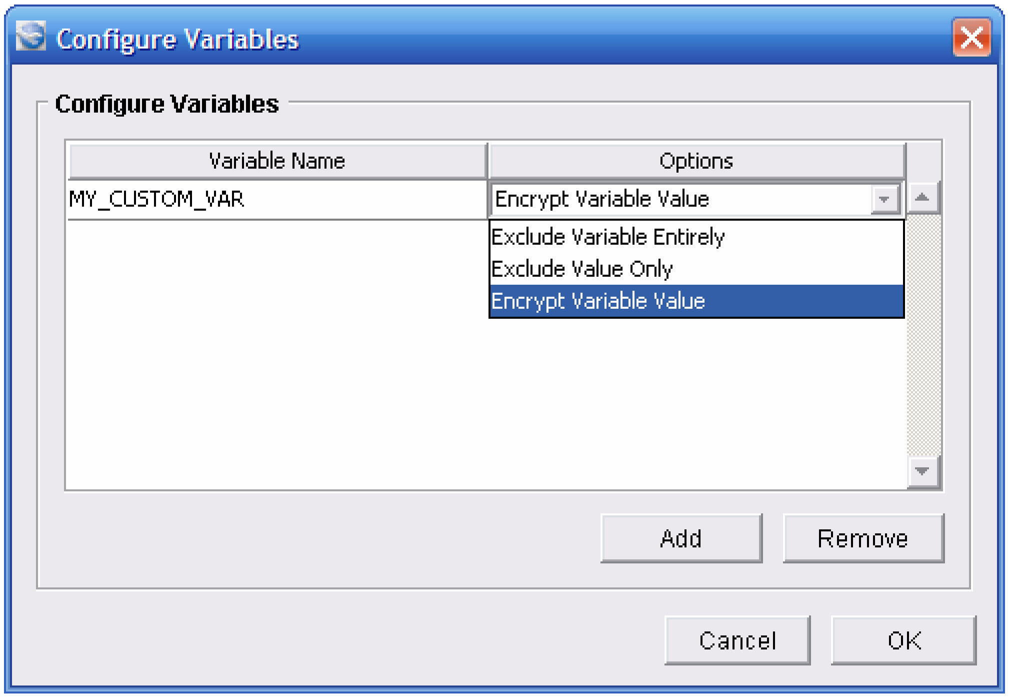
## InstallAnywhere Variables

During installation, InstallAnywhere keeps track of dynamic values through the use of *variables*. Almost every dynamic value in InstallAnywhere, such as the path that a Magic Folder (the following section defines Magic Folders) refers to, is represented by an InstallAnywhere variable. Some variables can be modified or accessed in order to affect the functionality or behavior of an installer.

Examples of standard InstallAnywhere variables are `$PRODUCT_NAME$`, which contains the product name as defined in the project; and `$/$` or `$\$`, which resolves (at run time) to the appropriate platform-specific file separator. InstallAnywhere provides a Set InstallAnywhere Variable action to create and modify variables and their values at run time.

InstallAnywhere variables are the key to any InstallAnywhere-based installation. They enable developers to control the flow of information, as well as the flow of the installation. They enable developers to store information obtained from the target system or from end users, and then to create rules to determine operations based on that information. Developers can even output that information to configuration files or other resources to be used by the application.

If variables contain sensitive information, you can specify to encrypt or exclude variables from installer logs and response files. In the Project > Info task, the Configure Variables section contains a Configure button; clicking Configure opens the following panel, in which you click Add, enter a variable name, and specify whether to encrypt the value or exclude it.



| Variable Name | Options |
|---|---|
| MY_CUSTOM_VAR | Encrypt Variable Value |

Exclude Variable Entirely
Exclude Value Only
Encrypt Variable Value

(For variable encryption, the Project > Java subtask enables you to specify which encryption algorithm to use.)

Note that variables in merge modules are not encrypted. For more information about how to use sensitive information in merge modules, see Macrovision Knowledge Base article **Q113554**.

A list of standard InstallAnywhere variables is presented in Appendix A.

## *Magic Folders*

InstallAnywhere uses Magic Folders to define installation locations. These Magic Folders are a way of keeping track of installation locations. InstallAnywhere can install to any Magic Folder or subfolder of a Magic Folder. Magic Folders represent a specific location, such as the user-selected installation folder, the desktop, or the location for library files. At install time, the installer determines which operating system it is running on, and sets the Magic Folders to the correct absolute paths. Some Magic Folders are platform-specific and some are predefined by InstallAnywhere. You can install to nearly any standard location on any supported platform. InstallAnywhere also provides user controlled Magic Folders which can be set as the developer needs them.

NOTE: Not all Magic Folders are available in every InstallAnywhere edition.

## *Magic Folder Examples*

InstallAnywhere uses Magic Folders to define most installation locations. They can represent either a fixed or variable path, and can be used to place a single file in the appropriate location on different target platforms. Many Magic Folders are predefined by InstallAnywhere, and in Enterprise Edition, allow you to install to nearly all-standard locations across our supported platforms.

| **Folder Name** | **InstallAnywhere Variable** | **Destination** |
|---|---|---|
| User Installation Directory | `$USER_INSTALL_DIR$` | The installation folder, as specified by the end user. You can specify a default value for this variable in the Project Info screen in the Advanced Designer and choosing a location in the Default Install Folder area of the screen. |
| Programs Folder (Platform Default) | `$PROGRAMS_DIR$` | The default application directory on the destination system. (Program Files on Windows, in the Applications folder on Mac OS X, and the logged-in-end user's home account on Unix.) |

"User Installation Directory" (`$USER_INSTALL_DIR$`) resolves to the folder the end user selects as their desired installation directory when the installer is running.

"Programs Folder" (`$PROGRAMS_DIR$`) resolves to the default application folder for the target platform's operating system. If a particular Magic Folder does not make sense on a target platform, actions that create files ("Install File" or "Create Alias, Link, Shortcut") will not install them. Actions that do not install files, such as "Execute Command," are not affected, and will run normally.

In the Properties section of each Install File and Create Folder action is a list where to select that item's destination, such as its Magic Folder. Some user environment Magic Folders now resolve on Linux. They will detect which GUI environment is being run (either KDE or Gnome) and resolve them appropriately.

### Variables and Magic Folders

Every Magic Folder has an associated InstallAnywhere variable. These variables are first initialized when the installer starts up. Changing the value of a Magic Folder Variable will change the installation destination for the Magic Folder. This technique can be used for any of the folders. For example, changing the value of the `$USER_INSTALL_DIR$` through InstallAnywhere will change where all the files inside the Install Folder Magic Folder will install. With three exceptions, these variables are initialized at install time and will not change except through using custom code or the Set InstallAnywhere Variable action. The exceptions are:

| | |
|---|---|
| `$USER_INSTALL_DIR$` | This variable is initialized to the default value determined in the Project task in the Advanced Designer. If the end user selects a different folder, then its value can change at the Choose Install Folder step. |

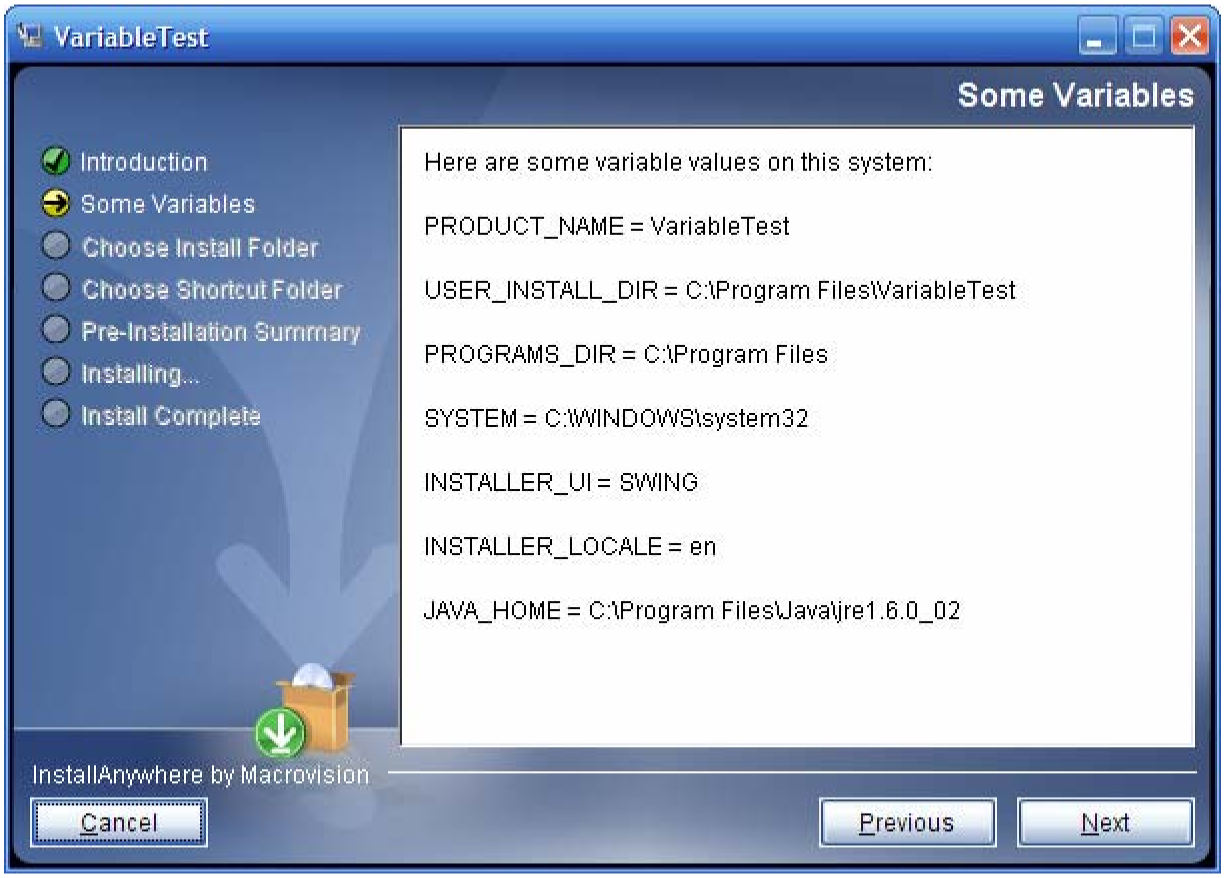
| | |
|---|---|
| `$USER_SHORTCUTS$` | This variable is initialized to the default value determined by the Platforms task of Advanced Mode. If the end user selects a different location, then its value can change at the Choose Alias, Link, Shortcut Folder installation step. |
| `$JAVA_HOME$` | This variable is initialized to the default value determined by the Platforms task of Advanced Mode. If the end user selects a different location, then its value can change at the Choose Alias, Link, Shortcut Folder install step.<br>Installer without VM: Defaults to the value of the Java property `java.home`. If the end user selects a VM, then its value can change at the Choose Java Virtual Machine step.<br>Installer with VM: Defaults to the value `$USER_INSTALL_DIR$/jre`. If the end user selects a VM already on their machine, then it can change when the `$USER_INSTALL_DIR$` changes, or at the Choose Java Virtual Machine step. |

NOTE: Variables cannot be set to themselves unless they are defined with the Evaluate at Assignment option. For variables defined without Evaluate at Assignment checked, you cannot set USER_MAGIC_FOLDER_1 = USER_MAGIC_FOLDER_1$/$test to append /test to USER_MAGIC_FOLDER_1. InstallAnywhere allows direct and indirect recursion only with InstallAnywhere variables that use Evaluate at Assignment. Otherwise, this condition causes an error.

For a simple example, an installation project might contain a Display Message panel whose message contains many InstallAnywhere variable expressions of the form $*VARIABLE_NAME*$:

```
Here are some variable values on this system:
PRODUCT_NAME = $PRODUCT_NAME$
USER_INSTALL_DIR = $USER_INSTALL_DIR$
PROGRAMS_DIR = $PROGRAMS_DIR$
SYSTEM = $SYSTEM$
INSTALLER_UI = $INSTALLER_UI$
INSTALLER_LOCALE = $INSTALLER_LOCALE$
JAVA_HOME = $JAVA_HOME$
```

At run time, the panel might appear similar to the following:



Here are some variable values on this system:

```
PRODUCT_NAME = VariableTest

USER_INSTALL_DIR = C:\Program Files\VariableTest

PROGRAMS_DIR = C:\Program Files

SYSTEM = C:\WINDOWS\system32

INSTALLER_UI = SWING

INSTALLER_LOCALE = en

JAVA_HOME = C:\Program Files\Java\jre1.6.0_02
```

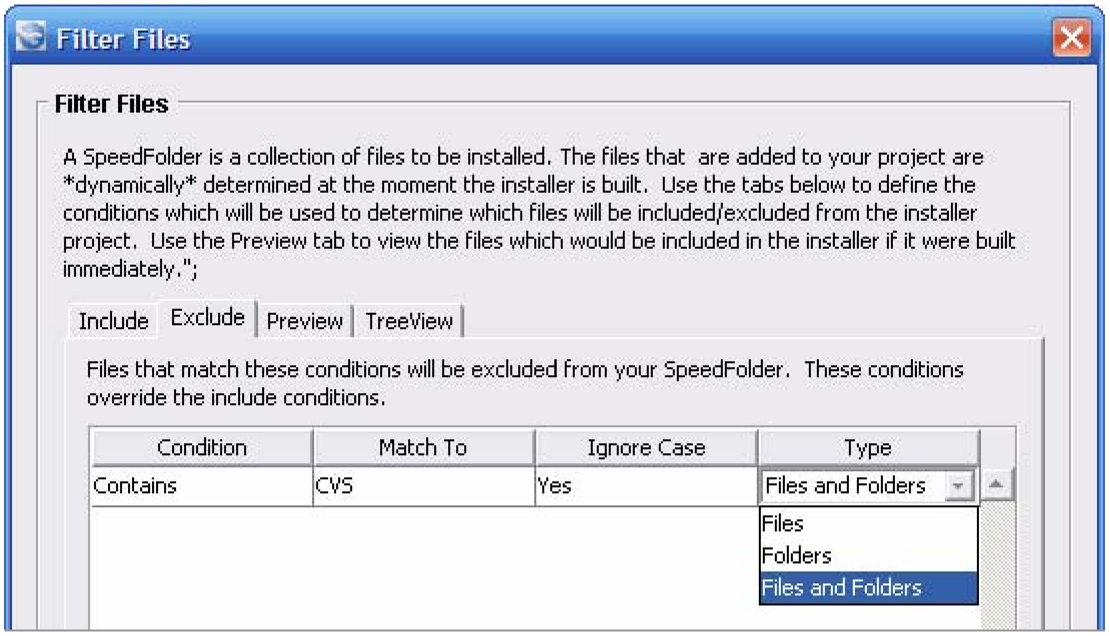
## *SpeedFolders*

Using SpeedFolders will dramatically increase installation speed and memory efficiency. Similar to folders that are added to a project using the Add Files method, SpeedFolders represent a container of other folders and files that are to be installed on the destination computer. SpeedFolders are a pointer to a particular folder, as opposed to a traditional folder, in which every item inside of it is a separate action. However, unlike normal folders, SpeedFolders and the contents they represent are treated as a single action, rather than each item representing an individual item. This combining of items lowers memory requirements and speeds up the installation.

SpeedFolders are ideal for use in an automated build environment. The contents of a SpeedFolder are determined at build time. At the time the installer is built, all of the contents of the folder on the build system (excepting items that have been marked to filter out) are added to the installer recursively. SpeedFolders are used to specify that a folder and all of its contents and sub-folders on the development system are to be automatically updated and included in the installer at the time the project is built. Standard folders (non-SpeedFolders) require developers to add or remove any files that are present or absent since the last installer build, or an error will occur.

SpeedFolders have filters that allow inclusion or exclusion of files that meet particular naming criteria. Individual files or folders in a SpeedFolder cannot be assigned to different components, nor can SpeedFolders be converted into traditional directories, or traditional directories into SpeedFolders. To convert one type to the other, you must delete the one folder and replace with the other type of folder.

InstallAnywhere allows you to add directory or folder filters. This can, for example, allow you to exclude the CVS directories left by the Concurrent Versions System, or exclude .java directories, which may contain un-compiled source code.



A SpeedFolder is a collection of files to be installed. The files that are added to your project are *dynamically* determined at the moment the installer is built. Use the tabs below to define the conditions which will be used to determine which files will be included/excluded from the installer project. Use the Preview tab to view the files which would be included in the installer if it were built immediately.";

Files that match these conditions will be excluded from your SpeedFolder. These conditions override the include conditions.

| Condition | Match To | Ignore Case | Type |
|---|---|---|---|
| Contains | CVS | Yes | Files and Folders |

## *Strict VM Selection*

InstallAnywhere includes strict virtual machine selection.

This feature allows you to specify not only the family—but the specific version of the Java virtual machine which you would like your installer, your installed product, or both to use. This ability, combined with the VM Shared Component feature, means that you may not need to bundle a virtual machine in cases where it was previously required. In some cases, this can significantly reduce your installer size.

You can specify strict VM selection parameters in your launcher—for your installed application—by setting the LaunchAnywhere property `lax.nl.valid.vm.list`, or you can set the property for your installer in the **Project > Java** tab of the InstallAnywhere Advanced Designer.

The values for these properties can be any space-delimited combination of the following general operators:

- ALL (any VM)
- JDK (any JDK)
- JRE (any JRE)

Alternately, you can use a strict VM expression such as "JRE_1.5.1_03" or "JDK 1.4.2_02", joining JDK or JRE, with an underscore character, to a version number.

"JRE_1.4.2_02" allows the installer or application to run only against the JRE 1.4.2_02. A value of "JDK_1.5.0_06" allows the installer or application to run only against a JDK 1.5.0_06.

And finally, you can specify minimum or wildcard versions of a specific VM, using the + or * operators:

- JRE_1.4+ selects any JRE-type JVM of version 1.4.0_0 or greater.
- JDK_1.4.2* selects any JDK-type JVM of the 1.4.2 series.

If more than one of these expressions is present, consider them combined with an OR operator. In other words, a VM is valid if it matches any of the given expressions.

The optional JDK or JRE specifies which type of VM is valid. If specified, it must be followed by an underscore character. You can only specify one or the other.

The version number can have varying degrees of precision, however we recommend having at least the major and minor version numbers specified.

The + or * operator at the end of a version are used to specify a version range. When using these operators, it is assumed that any unspecified version part is zero (specifying “1.6” is interpreted as 1.6.0_0). The + operator means “at least this version”, and the * operator means “of this version”. If you do not specify an operator, only versions that exactly match the specified version are valid (“1.4” does not validate for 1.4.2_02 JVMs).

## Project File

InstallAnywhere stores every project in its own XML file. These XML-based project files can be checked in and out of source control systems, and can be modified with text and XML editors. For added flexibility, project files may also be modified using XSL transformations, providing the ability to modify referenced file paths, or other attributes. Several XML and XSL tools to work on the XML project file can be found in the InstallAnywhere application folder, inside the `XML Project File Tools` directory.

## Manifest Files

InstallAnywhere uses a manifest file to identify the files that will be put in the installer.

Manifest files allow you to build installers directly from a list of files without needing to open and modify entire project files within InstallAnywhere. Useful when groups of developers are working on a project, manifest files can be used to combine files. Development groups only need to supply a list of files, rather than provide partially built options within InstallAnywhere (though that capability is also available through the use of Merge Modules and Templates).

**NOTE:** Another feature that supports modular, team-based development is InstallAnywhere's support for DIMs (Developer Installation Manifests). For more information, see the following section.

Developers can allow builds to succeed even if files referenced in the manifest file cannot be found. This flexibility allows for a single manifest file to identify all portions of a development project, promoting parallel development of software projects.

Manifest files’ ability to set Unix file permissions give the developer fine-grained control over the files that will be written to Unix target platforms.

## InstallAnywhere Collaboration and DIMs

InstallAnywhere Collaboration supports software development teams by enabling software application developers to collaborate with release engineers on the design, development, and deployment of their software subsystems. InstallAnywhere Collaboration is a standalone product that plugs in to Eclipse.

Application developers can use InstallAnywhere Collaboration to capture installation requirements and store them in a Developer Installation Manifest (DIM) file.

DIMs are named with the .dim extension and are structured in XML. Each .dim file describes the installation requirements for one subsystem of the entire software product. Because the DIMs are authored by the software developers who design the subsystems and know all of their installation requirements, they ensure that those requirements are included in the .dim file. Any DIM file—whether authored with InstallAnywhere Collaboration or InstallShield Collaboration, in Eclipse or Microsoft Visual Studio—can be referenced by any InstallShield 11.5 (or later) or InstallAnywhere 8 (or later) project.

## *FLEXnet Connect*

InstallAnywhere integrates with FLEXnet Connect (formerly “Update Service”). FLEXnet Connect enables you to initiate communication with your customers—such as notifying them of changes to your product. Using the Enable Update Notifications panel action, you can easily customize an action that automatically configures your products to integrate with FLEXnet Connect and send update notifications to your customers.

# Chapter 5: Basic Installer Development Strategies

- **Installation Planning**
- **Installation Goals**

## Installation Planning

Occasionally, you will find that planning an installation is simple. Put the files to disk in their specified location, and the application will just work. However, this situation is usually not the case. In today’s world of systems integration, installation stacks, suite installers, and client–server application development, you are far more likely to run into a very complex installation scenario—one requiring multiple steps, multiple products, and intricate configuration steps.

The idea behind using a fully featured installer such as InstallAnywhere is to minimize the impact that this sort of complexity will have on your customers and your end users.

As such, it is important to carefully plan your installation process and installation needs prior to beginning development.

## Installation Goals

When planning your installation process, consider the goals and targets of your installation.

- Is it required to allow a non-technical end user to install a complex product or as a highly flexible installer that can be used in a number of environments by expert users?
- What platforms and architectures will your deployment project target?

To help you plan your installation process, an Installation Planning Worksheet can be used to structure and manage your installation development project. A worksheet template can be found in Appendix J, “Installation Planning Worksheet”.

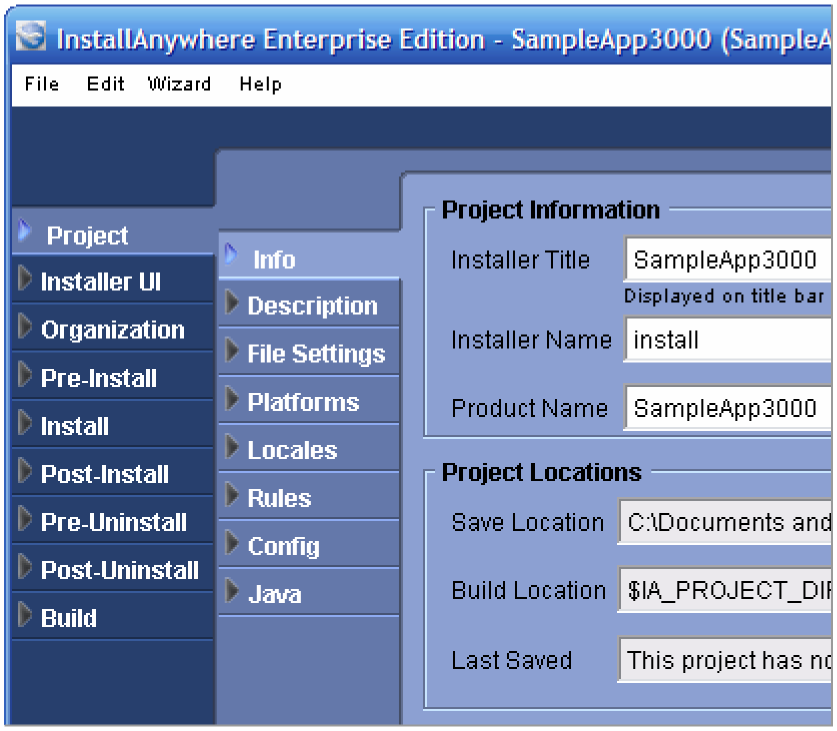
# Chapter 6: An Introduction to the Advanced Designer

- **Building an Installer with the Advanced Designer**
- **Defining Installer Projects and the Product Registry**
- **File Settings—Timestamps and Overwrite Behavior**
- **Platforms**
- **Locales**
- **Rules Before the Pre-Install Task**
- **Creating Debug Output**
- **Virtual Machines**
- **Quick Quiz**

The InstallAnywhere Advanced Designer has an intuitive, graphical interface which allows developers to manage all aspects of their installer project. All the features of InstallAnywhere are available in this easy to use integrated development environment.

To access the InstallAnywhere Advanced Designer, click the Advanced Designer button after selecting a project file or creating a new project.

The Advanced Designer is divided into “Tasks”, which are represented by tabs found along the left side of the window. Each tab represents tasks and settings specific to each installation project.



Project Information

Installer Title: SampleApp3000
Displayed on title bar
Installer Name: install
Product Name: SampleApp3000

Project Locations

Save Location: C:\Documents and
Build Location: $IA_PROJECT_DIF
Last Saved: This project has nc

| | |
|---|---|
| Project | Settings related to your specific project. These include general settings, file settings, and localization settings. |
| Installer UI | Set the look and feel for the installer by adding background images, billboards, and other graphical elements. |
| Organization | Manage Install Sets, Features, Components, and Merge Modules. |
| Pre-Install | An ordered sequence of panels and actions that occur before file installation. |
| Install | Manage the file installation tree and installation-time actions. |
| Post-Install | An ordered sequence of panels and actions that occur after file installation. |
| Pre-Uninstall | An ordered sequence of panels and actions that occur before file uninstallation. |
| Post-Uninstall | An ordered sequence of panels and actions that occur after file uninstallation. |
| Build | Manage build settings, including bundling of a Java Virtual Machine. |

Each Advanced Designer task contains sub-tabs that offer greater fine-tuning of InstallAnywhere’s features. For an example, go through the Advanced Designer tutorial.

## *Exercise 6.1 Building an Installer with the Advanced Designer*

In this exercise we will rebuild the previous OfficeSuite Installer using the Advanced Designer. The Advanced Designer offers a much wider range of configuration over InstallAnywhere's many options than the Project Wizard allows.

1. *Create New Project*

   a. *Launch InstallAnywhere.*

      On the first screen, the Create New Project option should already be selected.

   b. *Select the Basic Project Template*

      This template should already be selected

   c. *Click Save As to save and name the project.*

      The Save New Project As dialog box appears. InstallAnywhere will use this name as the name of the product in the installer project.

   d. *Select the Advanced Designer button.*

      This selection will open the newly created project file in the InstallAnywhere Advanced Designer. Advanced Designer will open to the Project > Info task. This task sets the basic installer options such as the name of the product, the installer title, and the installer name. The installer name will be the name of the executable file InstallAnywhere creates. This tab also sets the location to build the installer and the settings for the generation of installation logs.

   e. *Complete the Installer Title and Product Name fields.*

      For now, we will skip the Installer UI, Organization, and Install tasks.

2. *Pre-Install Actions*

   a. *Select the Pre-Install task.*

      The Pre-Install task sets the panels and action that occur prior to the installation of files. By default, a new InstallAnywhere project contains the following panels:

      Introduction: This panel allows developers to introduce the product or installation process.

      Choose Install Folder: This panel enables end users to choose the installation location for the product.

      Choose Alias, Link, Shortcut Folder: This panel enables end users to specify the location for any Mac OS Aliases, Windows Shortcuts, and Unix Symlinks (used as shortcuts) that will be installed.

      Pre-Install Summary: This panel provides end user with a summary of various installation settings prior to the installation of files.

      Actions in the Pre-Install task will occur in the order set in the task list. In a default project an Introduction panel will be followed by a Choose Install Folder panel, followed by a Choose Alias, Link, Shortcut Folder panel, and so on. The order of panels and actions can be manipulated using the arrow buttons in the middle right of the Advanced Designer screen.

      The behavior and content of panels can be modified by highlighting each panel. The dialog along the bottom half of the Advanced Designer will change to reflect the panel selected. In InstallAnywhere's vocabulary, this is known as a customizer, and is available for each action and panel in the installer.

3. *Define the installation tasks*

   a. *Select the Install task from the far left side of the Advanced Designer.*

      The Install task defines the files to install, the folder location to install those files and the order of the tasks that need to happen as the files are being installed.

By default, the InstallAnywhere Install task has a folder called Uninstall `$PRODUCT_NAME$`, which contains any InstallAnywhere uninstaller actions, and a comment action with instructions pertaining to the uninstaller.

Actions (including, but not limited to, the installation of files) in the **Install** task list occur in order with actions at the top of the installation occurring first.

HINT: The Advanced Designer implements a Drag-and-Drop interface in many tabs and tasks. In the Install task, actions and files can be moved by selecting and dragging them. A dark underline appears in the location where the file or action will be placed.

TIP: Leave the Uninstaller creation in its default place in the installation (although the folder structure can be changed). For organizational purposes, it's generally best to have the uninstaller creation action first.

Since the advanced tutorial mainly replicates the tasks for the OfficeSuite installer from the Project Wizard, those same files will be added.

b. *Add Files*

- Use the File Chooser to browse to the OfficeSuite Source Files folder found in the InstallAnywhere Installation Directory. The folder can also be drag and dropped into the Install task.
- Add the OfficeSuite2000 folder and its contents.

After adding the files, the files will be displayed in the file installation tree in the Advanced Designer window.

File trees may be expanded or contracted within the InstallAnywhere Advanced Designer Install task by clicking on the "+" or "-" boxes at the apex of the tree branches. Objects may be moved up and down or into and out of subfolders in the file tree by highlighting the object, and using the right, left, up, and down arrows (or dragging and dropping the files into the correct locations) found in the middle right of the Install task screen.

4. *Add a LaunchAnywhere Executable to the Install task.*

a. *Select the Add Launcher button.*

A LaunchAnywhere Executable (LAX) is a unique native executable, created by InstallAnywhere, that is used to launch a Java application. While the InstallAnywhere Wizard specifically asks to select a main class and automatically creates a single launcher, the Advanced Designer allows developers to add as many launchers as they would like.

There are two ways to add a LaunchAnywhere Launcher to an InstallAnywhere Project file. The Create LaunchAnywhere for Java Application option may be selected from the Add Action palette, or can be added by clicking the Add Launcher button on the middle control bar in the Advanced Designer.

i. *Highlight the* ***User Install Folder*** *in the Advanced Designer, and click the* ***Add Launcher*** *button.*

ii. *Click OK.*

When adding a launcher, InstallAnywhere will automatically introspect into the added files (including introspecting into JAR and or ZIP files) to find class files with Main Methods specified.

iii. *Choose the* ***com.acme.OfficeSuite*** *as main class for the application.*

Since OfficeSuite is a simple project, we're presented with only the `com.acme.OfficeSuite` class.

b. *Click OK to continue.*

NOTE: The Add Launcher button has not only added the launcher to the file structure, but also created a Shortcut, Link, or Alias action in the Shortcuts' Destination Folder Magic Folder. This location is variable and will be specified by the Choose Alias, Link, and Shortcut panel in the pre-install section.

c. *Customize the Launcher*

The appearance the launcher will have as a shortcut can now be customized. Highlight the launcher. The customizer along the lower portion of the Advanced Designer screen will change to reflect the options for the Create LaunchAnywhere for Java Application action.

In the lower middle right of the customizer (below the arguments field) are a set of buttons that control the icon associated with the launcher. The default icon is a teal tile with a coffee cup, and a rocket ship icon.

i. *Click* Change *to alter the icon.*

ii. *In* the Choose Icon *dialog, click* Choose GIF File*.*

iii. *Select a GIF or JPG file to use as an icon. For this tutorial, use the* `OfficeSuiteIcon` *in the* `Images and Docs` *folder within the* `OfficeSuiteSourceFiles` *folder.*

NOTE: Interlaced GIF files cannot be used with InstallAnywhere. The conversion process does not support these files and their use can result in blank icons.

NOTE: For Mac OS X, provide an ICNS file (created with iconbuilder—part of the Mac OS X Developer Tools).

d. *Set the InstallAnywhere Classpath*

InstallAnywhere maintains a general classpath that is used to create launchers for the Java Application.

i. *In the InstallAnywhere Advanced Designer, click the* **Set Classpath** *button.*

A blue CP icon will appear on folders and archives that the process has added to the classpath.

ii. *Select the* ***Project > Java*** *tab along the left side of the Advanced Designer window to view the Classpath as determined by the Set Classpath action.*

Since OfficeSuite is a simple product, we'll have only the main "OfficeSuite2000" folder (which contains loose class files). If our example project contained JAR or ZIP files containing classes, they would also have been added. If a file is added mistakenly on the Classpath, it can be removed at this point or by highlighting that file in the installation tree and un-checking the In Classpath option box in the Customizer for that file.

5. *Post-Install Actions*

The Post-Install Task list specifies actions and panels to occur after the installation of files. Like Pre-Install, the Post-Install step is ordered with the top actions occurring first.

By default, InstallAnywhere has added two actions to the InstallAnywhere project. These actions are:

Panel: Install Complete

This panel appears when the installation has completed successfully. This action is determined by the status of the `$INSTALL_SUCCESS$` variable. This panel will display only if the `$INSTALL_SUCCESS$` does not contain any error condition.

Restart Windows

This action restarts a Windows system if the installer determines that it is necessary.

InstallAnywhere installations are controlled primarily by InstallAnywhere Rules. As an example of an InstallAnywhere Rule, highlight the Restart Windows action in the OfficeSuite Project. In the customizer in the lower portion of the screen, select the Rules tab.

The InstallAnywhere Rules customizer will appear in the lower portion of the Advanced Designer. The rules set on the Restart Windows action are simple rules set to compare InstallAnywhere Variables. InstallAnywhere Rules are Boolean and allow the file, panel, or action to be installed, displayed, or run only if the rule resolves to `True`.

a. *Click the* **Add Action** *button to open the Action Palette.*

The Action Palette is divided by tabs that vary based on the task that is active at the time the palette is called.

b. *Select* **Execute Target File** *found under the* **General** *tab.*

The Execute Target File action is used to execute files that are included as part of the installation, and consequently it is available only in the Install, and Post-install portion of the installation. (The Execute Target File is not available in "Pre-Install" because files cannot be executed that are not installed yet!)

c. *To add the action, click* **Add**.

The Palette remains open so additional actions may be added.

d. *To select the target, click the* **Choose Target** *button.*

The **Choose an Action** dialog represents the file installation tree specified in the **Install** task. Files can be executed in this stage. To execute the just installed OfficeSuite application, choose the launcher for that application.

NOTE: Choose the actual OfficeSuite Launcher, and not the shortcut (which should share the same Icon). Shortcuts, especially on Windows and Mac OS systems, are pointers and are not inherently executable. InstallAnywhere will not execute a shortcut.

Customizer Options

By using the Command Line field modifications can be made to the command line used to execute the file, such as adding a handler, or an argument to the execution.

NOTE: Do not remove or modify the `$EXECUTE FILE TARGET$` entry, as this represents the file to execute. To specify a handler, prepend an executable path; to specify an argument, append a file path. These paths MUST be absolute; however, the paths can include InstallAnywhere variables.

The user experience for this action can be tailored by using the Options fields.
The option to suspend the installation until the process is complete. This is particularly useful in cases where a later step in the installation is dependant on the execution. There is also a subtask that allows developers to specify an indeterminate progress bar with a message. This task can be used if the execution may take some time (for example, an execute action that installs another product, or configures a database or other application).

The **Show Please Wait** panel option will display a message panel to the user while the execution is occurring.

The **Suppress First Window** option allows developers to suppress the first window on Microsoft Windows platforms. This option is particularly useful in suppressing the appearance of the `cmd.exe` window when executing batch files, or command line executables.

NOTE: If the execute action panel was added at a location other than the bottom of the Post-Install task, move it now. Either utilize the up and down arrows, or drag the action to the bottom of the task list.

6. *Build Installer*

The InstallAnywhere **Build** task allows the options that will be used to build the installer(s) to be set. In this task platforms for the build can be set, configuration options for bundled virtual machines, and platform optimization and installer type.

HINT: For early testing, build only for the development platform. Each additional platform adds to the time required to build, cycling through run-rebuild-run-rebuild stages. A faster build will make the development process easier.

On the **Build Targets** tab in the **Build** task, select the platform(s). Selecting **With VM**, will “bundle” the installer with a VM. **With VM** is only selectable for platforms which have a VM pack. VM packs should be placed in the `<InstallAnywhere>/resource/installer_vms` folder, and InstallAnywhere should be restarted to refresh the available VM packs. (Depending on your build settings and the VMs available on your system, it may be necessary to obtain additional VMs by clicking **Download Additional VM Packs**.)

The **Build** task also includes the **Distribution** and **Build Log** tabs.

The **Distribution** tab allows developers to set options for the type of installers to build, and the optimization options for each installer. As the installer being built in this tutorial doesn’t contain any platform specific files, it will not need to be optimized at this point. However, if the installer did include platform specific files, these files would be optimized based on the application of **the Check Platform** rules.

The Build Log tab displays the XML log of previous builds.

a. *Click **Build Project** to build the OfficeSuite installer.*
   *The Build information dialog will appear.*

b. *Click the blue arrow on the lower left of that dialog to see the build details console.*

   When the build is complete, there will be a notification (in this case, the Build should take a minute or less).

7. *Testing.*

   After the Build process is complete, try the installer by selecting either the Try Web Install or Try Installer button. In this case, use the Try Web Install button to launch our browser and the InstallAnywhere Web Install Page generated by the build process.

   a. *Click **Try Web Install**. The **Web Install Page** will load, and should request a security access.*

   b. *Grant this access to allow the Web Install Applet to run the InstallAnywhere installer.*

      The web installer can now be launched with just one click.

   c. *Click the **Start Installer for Windows** button below the image.*
      *The applet will check for sufficient disk space, “download” the installer, and execute the installer.*

   d. *Run the installer.*

      After the Install Complete Panel, the installer should launch OfficeSuite. The OfficeSuite Icon can now be selected from the Windows Start Menu to run the installed product.

## ***Defining Installer Projects and the Product Registry***

### *Product Registry*

The product registry is essentially a product configuration database which keeps track of features and components of products for the operating system. It is the product registry which accomplishes tasks such as associating file name extensions with applications. InstallAnywhere makes entering vendor and product information to uniquely identify their product in the product registry information easy.

NOTE: Correctly setting the Product ID and Version are critical to using the Find Component in Registry action. It is by checking the Product ID that InstallAnywhere finds the locations of components in the Registry.

Product ID and vendor information is entered in the Project > Description subtask.

### *Installer Identification and Version*

Installers—just like the software products they are installing—need to be given names and versions. Just as names and versions help track changes in a software product, InstallAnywhere helps uniquely identify versions of installers. InstallAnywhere also provides an installation log which details the files installed and the actions execute by the installer. The developer defines whether to create an installation log, the format of the log, whether it should be created in plain text or XML format, and whether the installation log should be removed if the application is uninstalled.

NOTE: To set the Installation log install location, set the InstallAnywhere Variable `$INSTALL_LOG_DESTINATION$`.

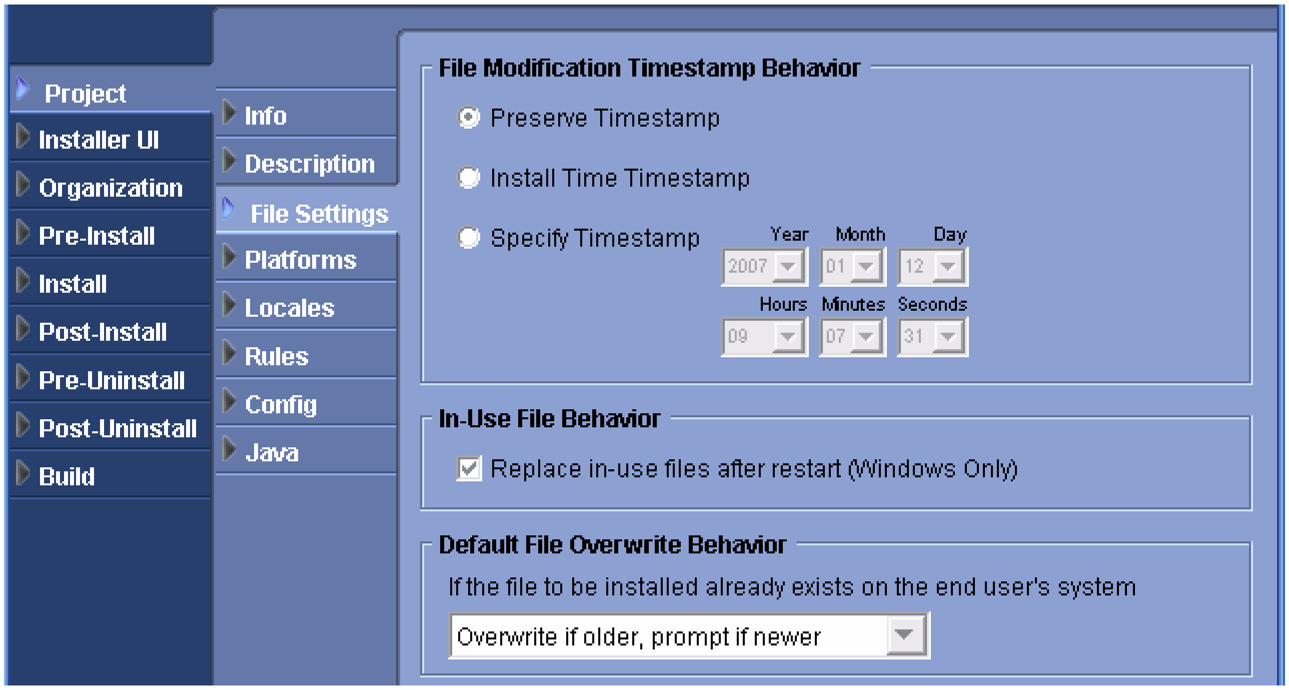
The Project > Info task defines basic information about the installer that is to be created, displays information about the InstallAnywhere installer project, and enables the developer to make decisions about the installer installation log.

## *File Settings—Timestamps and Overwrite Behavior*

When installing software, whether a new product or a newer version of a product, there is the possibility of overwriting files that already exist on the target system. InstallAnywhere uses timestamps to uniquely identify files with the same name. InstallAnywhere also allows the developer to set the type of overwrite behavior—whether to prompt the end user, whether to overwrite the older file.

NOTE: When installing to Windows operating systems, there may be files that are in use. When the Replace in-use files after restart option is selected, the installer action will detect if files that are being installed are overwriting files that are in use. If there are files in use, InstallAnywhere will register these files with the Windows Product Registry, so they can be correctly installed when the system is restarted.

Timestamps, in-use file behavior, and overwrite behavior are defined in the File Settings subtask.



File Modification Timestamp Behavior

[✓ Preserve Timestamp]

Install Time Timestamp

Specify Timestamp Year 2007 Month 01 Day 12 Hours 09 Minutes 07 Seconds 31

In-Use File Behavior

[✓] Replace in-use files after restart (Windows Only)

Default File Overwrite Behavior

If the file to be installed already exists on the end user's system

Overwrite if older, prompt if newer

### Installed File Timestamps

The File Modification Timestamp Behavior section enables developers to timestamp installed files in three different ways.

- Preserve Timestamp: This selection maintains the default timestamp on the file. That is, the time that file was last modified as shown by the operating system where the file was created and/or saved. For example, the file would show the time it was last modified and not when it was installed.
- Install Time Timestamp: This selection sets the creation property and the file modification property to the time that the files are installed on the target system. With this option all the installed files would have the same creation and file modification properties.
- Specify Timestamp: This selection enables developers to place a specific timestamp on installed files. Specific date and time stamps may be selected from the scroll lists.

NOTES about Timestamps:

The files' timestamp property may be set to both before and after the current date.

InstallAnywhere displays all timestamps in the system's local time zone. Internally, InstallAnywhere automatically maintains those timestamps in Greenwich Mean Time (GMT), but the timestamps display in the local time zone.

### Default Overwrite Behavior

When the installation contains files that also exist in the installation locations on the target system, the installer must know how to determine whether to overwrite files. Files are considered the same when they are named the same and have the same path. Whether to overwrite or not install the files is dependent on the timestamps of the files on the target computer and the timestamp of the installation files.

**NOTE**: For Windows systems, InstallAnywhere also includes an overwrite-after-restart option for files that are in use during installation. To enable this option, click Replace in-use files after restart.

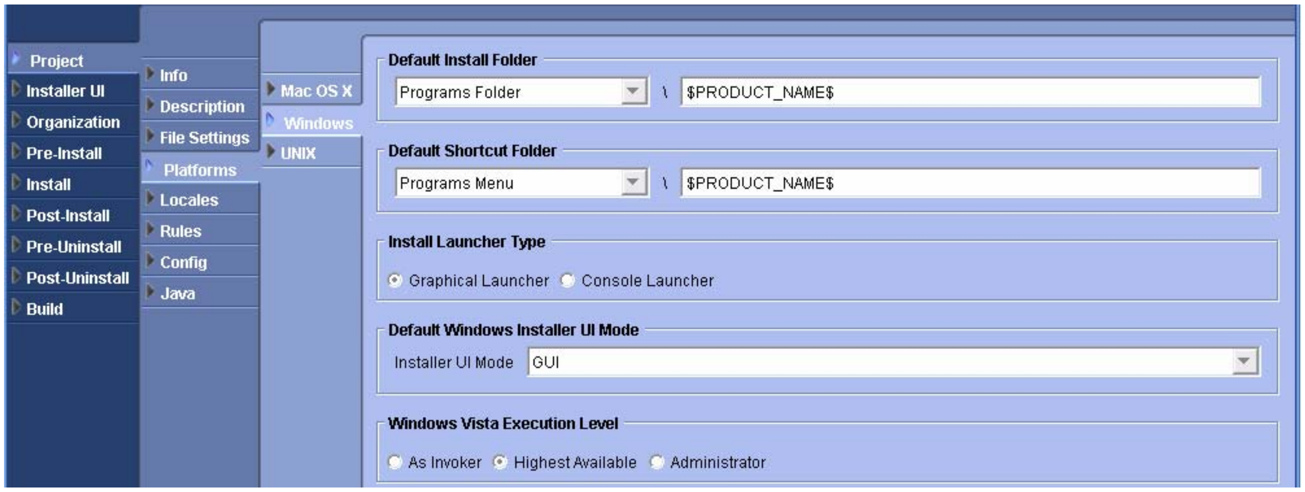
The default behavior is to overwrite older files, to prompt if newer. The options for determining overwrite behavior may be prompted or set as a default.

| **Overwrite Option** | **Select this option to…** |
|---|---|
| Always Overwrite | …install files without giving the user the choice whether to overwrite files which currently exist on the computer. |
| Never Overwrite | …leave existing files untouched on the user's computer rather than overwrite them with files that are being installed. The user is given no option. |
| Overwrite if older, do not install if newer | …overwrite existing files on the user's computer that are the same as files that are being installed if the installed files are newer (have a later timestamp) than the existing files. The user is given no option. |
| Overwrite if older, prompt if newer | …overwrite existing files on the user's computer that are the same as files that are being installed if the installed files are newer (have a later timestamp) than the existing files without giving the user the option, but prompting if the installed files are older than the existing files on the user's computer. |
| Prompt if older, do not install if newer | …prompt the user if the existing files on the user's computer are older than the installation files. If the existing files are newer, there will not be a prompt and the installation files will not be installed. |
| Always Prompt User | …prompt the user whenever an installation file exists on the target computer. |

## *Platforms*

While InstallAnywhere runs on any Java-enabled platform, there are features such as default install folders and default link folders (Unix), default shortcut folders (Windows), and default alias folders (Mac OS X) that should be defined separately for each target operating system.

The Platforms task is separated into different platforms. For Windows, you specify the default install and shortcut folders, and choose between using a graphical launcher or a console launcher.



Default Install Folder: Programs Folder \ $PRODUCT_NAME$

Default Shortcut Folder: Programs Menu \ $PRODUCT_NAME$

Install Launcher Type: [✓ Graphical Launcher] Console Launcher

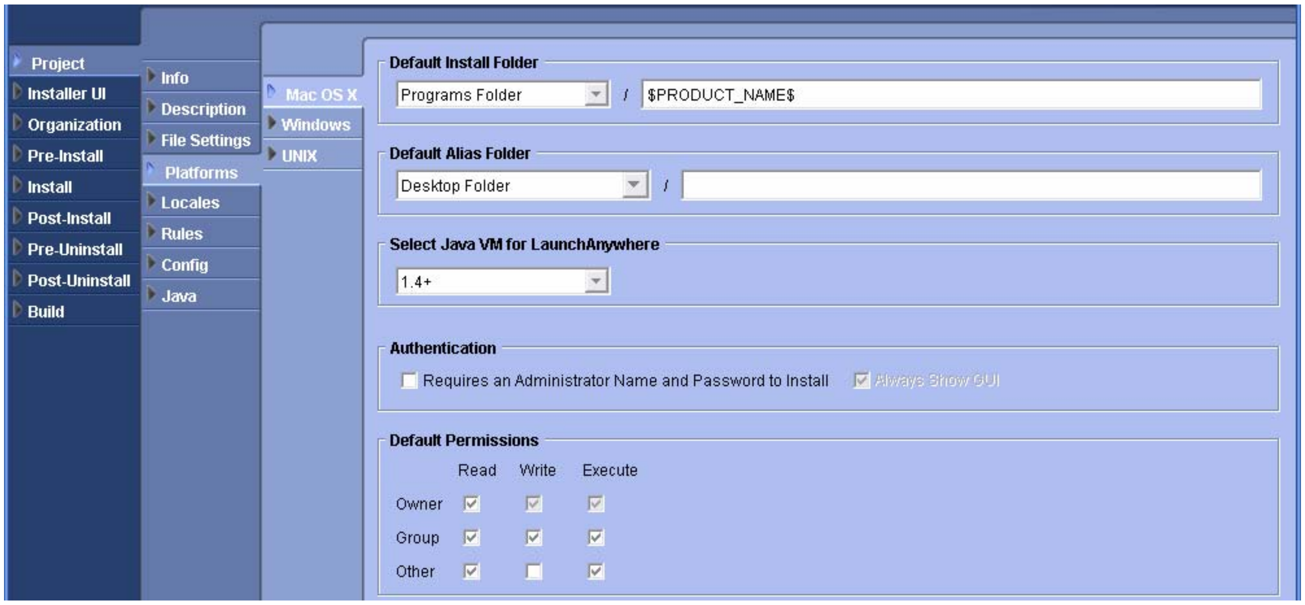
Default Windows Installer UI Mode: Installer UI Mode GUI

Windows Vista Execution Level: As Invoker [✓ Highest Available] Administrator

To support User Account Control (UAC) on Windows Vista target systems, you can also specify the execution level:

- **As Invoker**: The launcher acquires the same execution level as its parent process.
- **Highest Available**: The launcher requests the highest execution level (Windows privileges and user rights) available to the current user.
- **Administrator**: The launcher requires local admin privileges to run. Depending on the privileges of the current user account and the configuration of the target system, this setting may result in a launcher that will not start.

Mac OS X adds defining default permissions for files and folders that will be created on the target system. The developer can also enable installer authentication (providing the end user correct permissions to install if they are not running as a privileged user) and the ability to set which VM versions.



Default Install Folder: Programs Folder / $PRODUCT_NAME$

Default Alias Folder: Desktop Folder /

Select Java VM for LaunchAnywhere: 1.4+

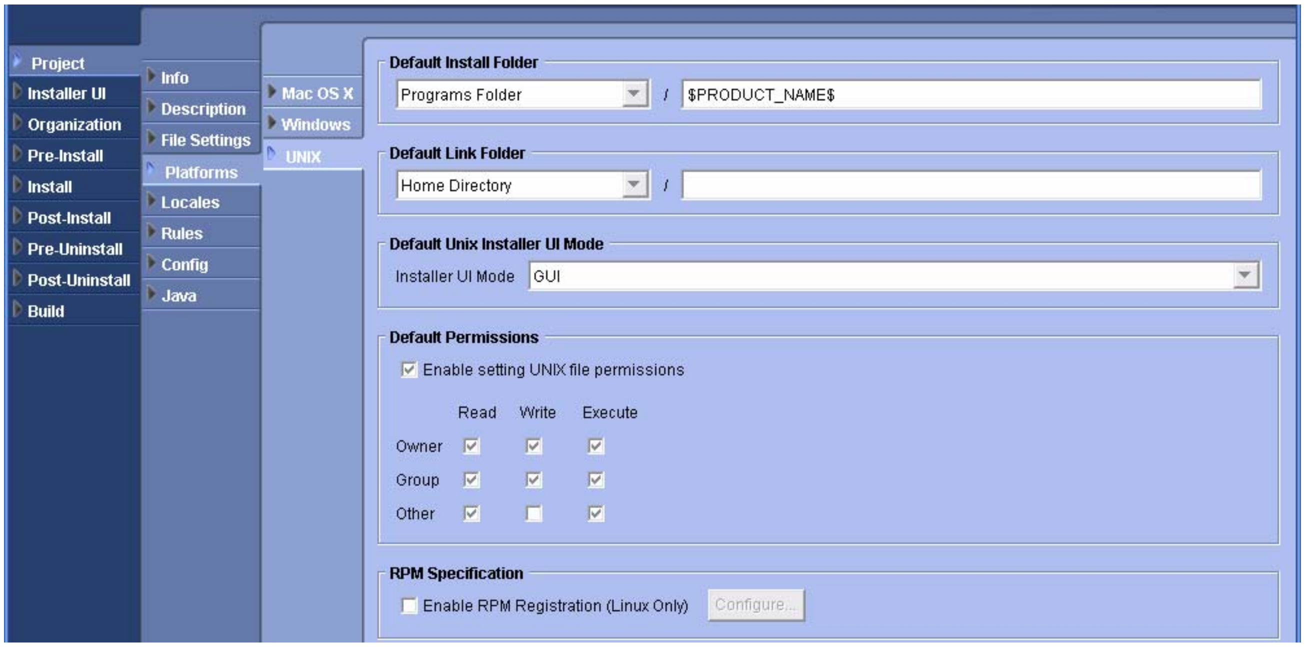
Authentication: ☐ Requires an Administrator Name and Password to Install ✓ Always Show GUI

Default Permissions

| | Read | Write | Execute |
|---|---|---|---|
| Owner | ✓ | ✓ | ✓ |
| Group | ✓ | ✓ | ✓ |
| Other | ✓ | ☐ | ✓ |

The Unix task shows similar settings. For Linux target systems, you can enable RPM (Red Hat Package Management) support. RPM is a package of installation tools that InstallAnywhere installers will use in the

Linux environment and other Unix environments. The RPM feature enables the installer to interact with and make entries into the RPM database.



Default Install Folder: Programs Folder / $PRODUCT_NAME$

Default Link Folder: Home Directory /

Default Unix Installer UI Mode — Installer UI Mode: GUI

Default Permissions — ☑ Enable setting UNIX file permissions

| | Read | Write | Execute |
|---|---|---|---|
| Owner | ✓ | ✓ | ✓ |
| Group | ✓ | ✓ | ✓ |
| Other | ✓ | | ✓ |

RPM Specification — ☐ Enable RPM Registration (Linux Only)

## Locales

The Locales subtask defines the languages for which the installer will be created. A locale is enabled when it is checked.

Each enabled locale will generate a locale file that will be placed in a folder that is in the same directory as the InstallAnywhere project file. To customize a locale, customize this file. For more information about locales and localization, see Chapter 21, “Localizing and Internationalizing InstallAnywhere Installers”.

## Rules Before the Pre-Install Task

Some rules should be evaluated before any installation tasks, even Pre-Install tasks, occur. These rules, such as checking if the target system is a proper platform for this installation, if the user is logged into the root, or has the necessary permissions to perform the installation, can be added in the Project > Rules subtask.

## Creating Debug Output

Installer debug output information can be useful for tracking down issues in an installer. InstallAnywhere developers can enable debug output as well as select if it should be sent to a file or to a live console.

### Installer Debug Output

If the “Send stderr to” or the “Send stdout to” field is left blank, the output of the installer will be discarded. To send the output to a live console to monitor the output, enter `console` in the text field. To send the information to a file, enter the file name.

Note that InstallAnywhere 2008 introduces classes that enable automated testing of installers using JUnit tests. For information and examples, see the `gui-test-auto` subdirectory of the InstallAnywhere distribution.

## Virtual Machines

With InstallAnywhere developers can define a valid list of Java VMs their installer can use. This option can be used to select VMs that have been fully tested. LaunchAnywhere searches for VMs sequentially based on VM type. Valid VM types are listed in the LaunchAnywhere Executable property, `lax.nl.valid.vm.list`. LaunchAnywhere uses the following approaches on each platform:

- Windows: first search on the system path, then the system Registry.
- Unix: search the system path.
- Mac OS X: LaunchAnywhere will use the VM specified in the Project > Platforms > Mac OS X task.

InstallAnywhere developers can also set the heap size for the VMs.

NOTE: Change the heap size when experiencing out-of-memory conditions. With large installations that have many files to install, the heap size may need to be increased.

### Optional Installer Arguments

To support Java VM configuration options which are not available through the InstallAnywhere Advanced Designer, specify additional command line parameters to pass to the Java VM used by the installer through the use of the Optional Installer Arguments > Additional Arguments field.

### Java

The Project > Java subtask enables developers to fine tune the classpath settings and decide whether to install the bundled Java VM. Developers may choose not to install a VM, Install the VM only while performing the installation, or to leave the VM on the target system. If developers choose to install the VM, the VM Install folder pull down list provides a variety of locations.

Available in the Java subtask, starting with InstallAnywhere 8.0, is the Add Service Support for Custom Code checkbox. InstallAnywhere provides a service layer that adds a rich suite of APIs for use with custom code actions. This checkbox must be enabled if you use the `FileService`, `SecurityService`, `SystemUtilService`, `Win32RegistryService`, or `Win32Service` API calls. If you use one of those calls with this checkbox unchecked you will get a `NoClassDefFound` exception and the custom code will not execute properly.

## *Quick Quiz*

1. Which rule is used to determine which “Install Complete” message to display?
   A. Check Platform
   B. Compare InstallAnywhere Variables
   C. Compare Time Stamps
2. Which two indicators show the Classpath to be used for your LaunchAnywhere launched application?
   A. A list in the Classpath Task
   B. An indicator on the file/folder icon
   C. A beeping tone when mousing over the file
3. When would you use the “Suppress First Window” option on an Execute Target File action?
   A. Executing a Windows Batch file
   B. Running a sub installer
   C. Running a Unix shell script

Answers: 1.B | 2.A, B | 3. A

# Chapter 7: Build Options

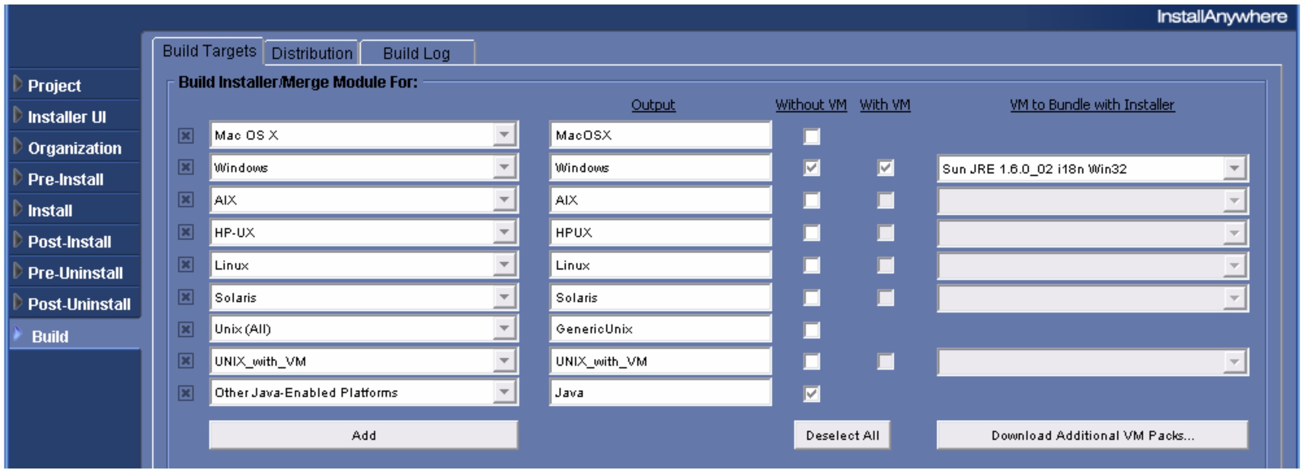
- **Dynamic Build Targets**
- **VM Packs**
- **Distribution**

With three different build outputs, InstallAnywhere Users can build installers for deployment over the Web, deployment on a CD-ROM, or through a Merge Module. Templates—a specialized Merge Module intended for maintaining look and feel, as well as retaining standardized installation panels, such as the license agreement or choose install folder panels, or even files—are also created in the Build task.

In the Build task, developers define the target operating system for the installer as well as defining the form for distributing the installer. In Build > Build Targets, operating systems are defined. Developers also define whether to provide a VM (Java Virtual Machine) with the installer to provide greater ease of use for the end user.

## Dynamic Build Targets

Starting with InstallAnywhere 2008 (Enterprise Edition), you can define *dynamic build targets*: using the Build Targets tab, you can create and delete build targets, and create more than one build target per platform. For example, a single project can contain a build target for Windows with no VM, Windows with an IBM VM, and a Sun VM.



Build Installer/Merge Module For:

| | | Output | Without VM | With VM | VM to Bundle with Installer |
|---|---|---|---|---|---|
| ☒ | Mac OS X | MacOSX | ☐ | | |
| ☒ | Windows | Windows | ✓ | ✓ | Sun JRE 1.6.0_02 i18n Win32 |
| ☒ | AIX | AIX | ☐ | ☐ | |
| ☒ | HP-UX | HPUX | ☐ | ☐ | |
| ☒ | Linux | Linux | ☐ | ☐ | |
| ☒ | Solaris | Solaris | ☐ | ☐ | |
| ☒ | Unix (All) | GenericUnix | ☐ | | |
| ☒ | UNIX_with_VM | UNIX_with_VM | ☐ | ☐ | |
| ☒ | Other Java-Enabled Platforms | Java | ✓ | | |

When you click Build Project, all of the targets with a selected check box in the Without VM or With VM column will be built.

While InstallAnywhere provides options for many flavors of Unix, it also allows the creation a generic Unix (Unix (All)) and of other, custom flavors. To create an installer for a flavor of Unix that is not in the list of platforms. Select an existing target (or create a new target) of type UNIX_with_VM, and if desired enter the custom target's name in the Output field.

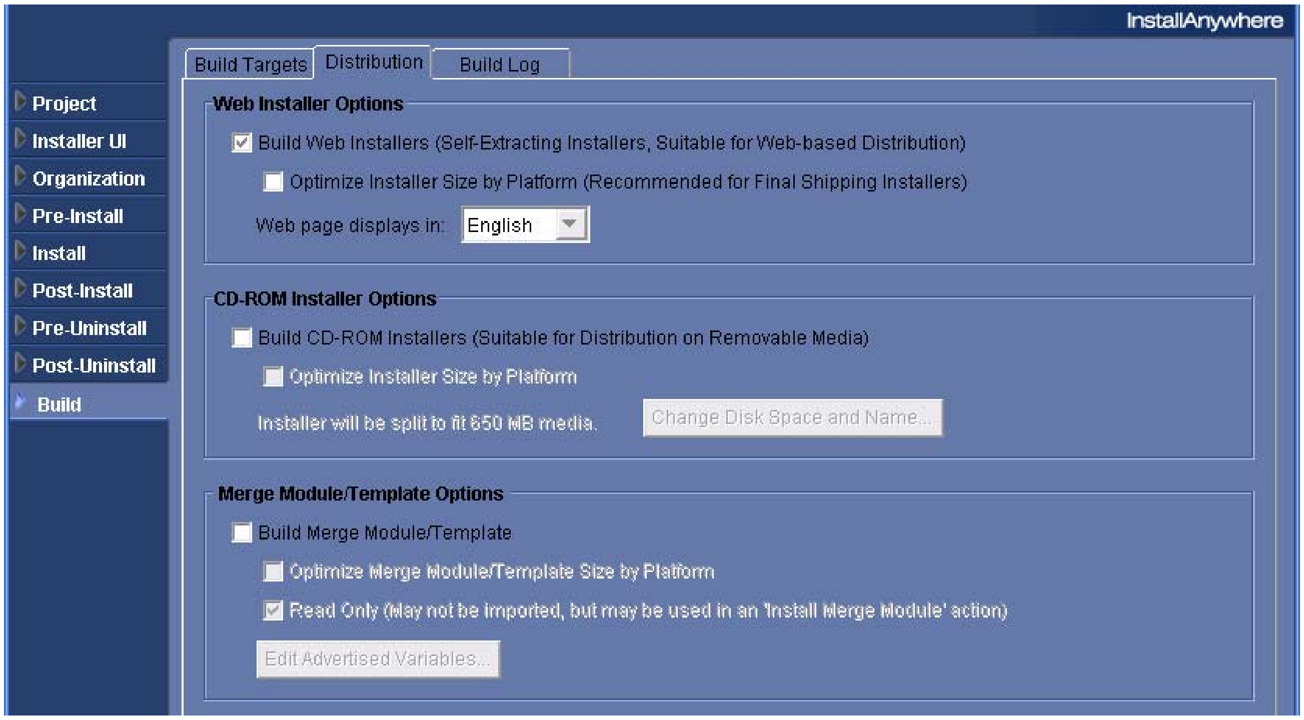
## VM Packs

The Macrovision web site provides VM packs for you to download. Clicking the Download Additional VM Packs button in the Build Targets tab brings up the VM Packs section of the Macrovision web site.

VM packs are stored as `.vm` (zip/jar) files, which contain the archive of a Java VM and a `vm.properties` file. These VM packs must be stored as resources in the directory `<InstallAnywhere>/resources/-installer_vms`. The selected bundled VM will be saved on a project-by-project basis. You can also add new VM-pack locations using Edit > Preferences > Resources > VM Pack Resource Paths.

## Distribution

In the Distribution subtask, you define the form for the installer to be distributed. Developers can build and optimize an installer on a single or multiple CD-ROM discs, an installer for use over the Web, or an installer to be launched from an HTML file.



Web Installer Options

[✓] Build Web Installers (Self-Extracting Installers, Suitable for Web-based Distribution)

[ ] Optimize Installer Size by Platform (Recommended for Final Shipping Installers)

Web page displays in: English

CD-ROM Installer Options

[ ] Build CD-ROM Installers (Suitable for Distribution on Removable Media)

[ ] Optimize Installer Size by Platform

Installer will be split to fit 650 MB media.

Change Disk Space and Name...

Merge Module/Template Options

[ ] Build Merge Module/Template

[ ] Optimize Merge Module/Template Size by Platform

[✓] Read Only (May not be imported, but may be used in an 'Install Merge Module' action)

Edit Advertised Variables...

The Distribution subtask also enables you to build and optimize Merge Modules and Templates.

### Web Installers

The web installer is a single executable file that contains all of the necessary installation logic. Building the web installer also generates an HTML page and embedded java applet to make downloading the installer over the web easy. Select Optimize Installer Size by Platform to minimize the size of the final installers by excluding platform-specific resources (this is determined by evaluating Check Platform Rules). The developer also has an option to select in which language to build the target web page.

### CD-ROM/DVD Installers

CD-ROM/DVD installers consist of multiple files meant to be burnt onto one or more CD-ROM disks, DVD-ROM disks, or other removable media. They can also be placed on network volumes to provide easier to access to large installers. The output of this build process can be directly burned onto disk.

InstallAnywhere CD/DVDs Installers have the ability to span multiple CDs/DVDs. By default, the installer will automatically segment the installer into a new disk if the size of the installer exceeds the media size (default: 650 MB). To control when InstallAnywhere will span to new disks, configure your disk names and size by clicking the “Change Disk Space and Name” button. This allows the developer to set the size for each disk, as well as set its name. The name will be displayed during the install process when the installer asks for the next disk in a set.

## Burning CD-ROM Installers

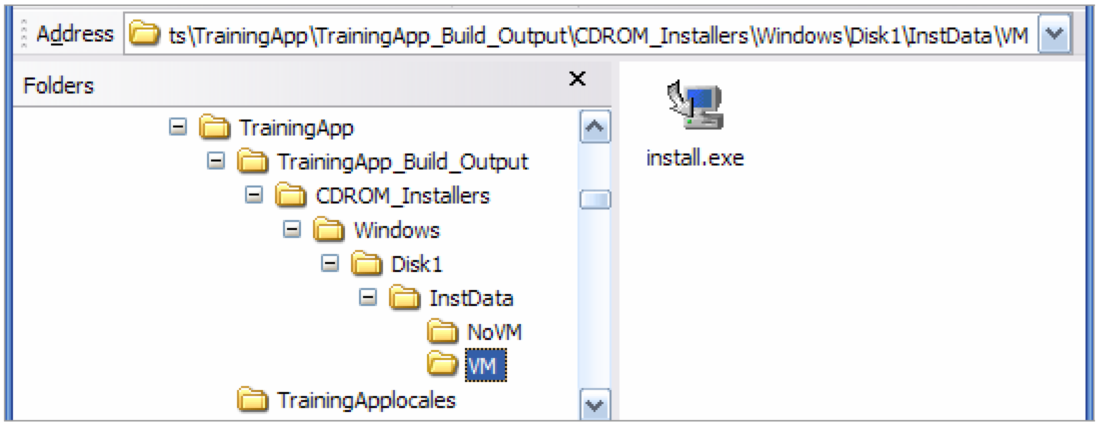
The directory structure for CD-ROM installers is:

```
Platform1/Disk1/InstData/
  |-...
  |-MediaId.properties
  |-Resource1.zip


Platform1/Disk2/InstData/
  |-MediaId.properties
  |-Resource2.zip


Platform1/Diskn/InstData/
  |-MediaId.properties
  |-Resourcen.zip


...
```

Disk 1 typically contains an installer binary, often inside a VM or No VM directory. For example, when the Without VM and With VM options for Windows are both checked on the Build Targets task, InstallAnywhere will place both VM and No VM subdirectories, each with an install.exe, inside Disk1\InstData. So the directory structure for Disk 1, in this case, is the following:

```
Windows/Disk1/InstData/
  |-No VM
    |-install.exe
  |-VM
    |-install.exe
  |-MediaId.properties
  |-Resource1.zip
```

This build might appear in Windows Explorer as follows:



However, on other platforms, such as Mac OS X, Unix (All), and Other Java-Enabled Platforms, the install binary is contained in the Disk1/InstData directory. On Mac OS X, for example, the directory structure for Disk 1 is:

```
MacOSX/Disk1/InstData/
  |-install.app
```

```
|-MediaId.properties
|-Resource1.zip
```

When burning CDs or DVDs, ensure that the folders Disk1, Disk2, etc., are burned as-is to the disk. Burning only the contents of these folders will cause installers to work incorrectly. The directory structure for the disk burning application should look like:

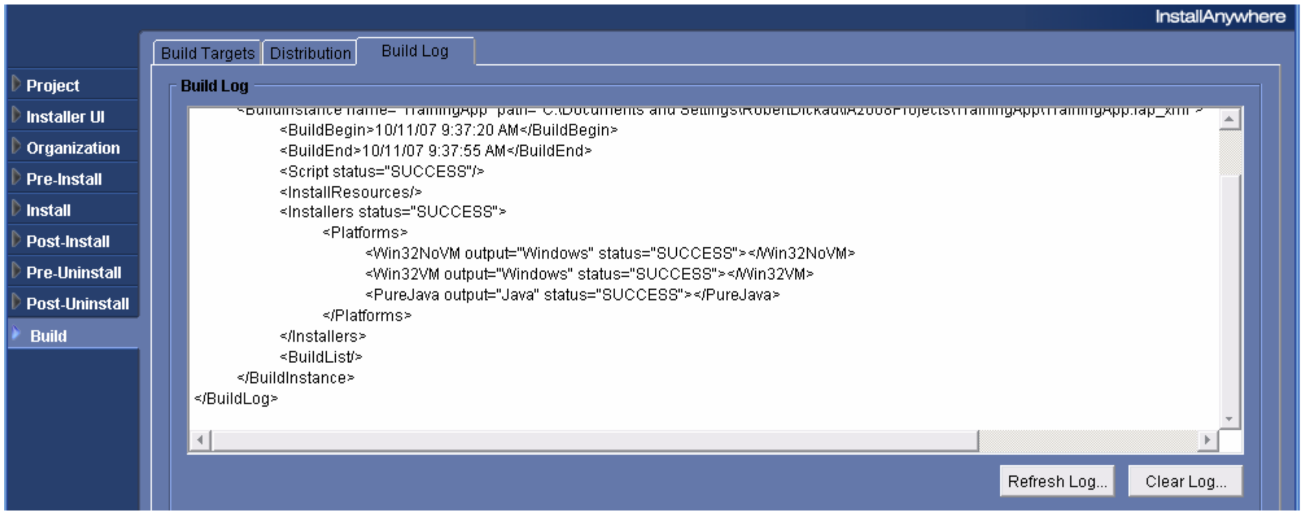
```
<ISO CD NAME>
  |-Disk1
  |-Disk2
```

## Merge Modules and Templates

Developers can also build Merge Modules. Merge Modules enable you to create installers that can easily integrated into other InstallAnywhere installers. More information on Merge Modules and Templates is available in Merge Modules and Templates in Chapter 17.

## Build Log

The **Build Log** window displays an XML log of the build once an installer project is successfully built. Click **Refresh Log** to display the current log. Click **Delete Log** to remove the log.



```
    <BuildInstance name="TrainingApp" path="C:\Documents and Settings\RobertDickau\IA2008Projects\TrainingApp\TrainingApp.iap_xml">
        <BuildBegin>10/11/07 9:37:20 AM</BuildBegin>
        <BuildEnd>10/11/07 9:37:55 AM</BuildEnd>
        <Script status="SUCCESS"/>
        <InstallResources/>
        <Installers status="SUCCESS">
            <Platforms>
                <Win32NoVM output="Windows" status="SUCCESS"></Win32NoVM>
                <Win32VM output="Windows" status="SUCCESS"></Win32VM>
                <PureJava output="Java" status="SUCCESS"></PureJava>
            </Platforms>
        </Installers>
        <BuildList/>
    </BuildInstance>
</BuildLog>
```

# Chapter 8: Basic Installer Customization

- Customizing Your InstallAnywhere Installer Look and Feel
- Using Installer Rules
- Using Rules to Control Visual Elements
- Managing Installer Flow Based On End-User Input
- Quick Quiz

InstallAnywhere installers are almost infinitely customizable. You control how the install looks as well as the tasks the installer will accomplish. You have complete control over what (if anything) appears on the end user's screen, what order the actions will occur, where files are to be installed, how each panel looks in a graphical installer, what messages appear to the end user, and many more.

In this chapter, we'll cover some of the myriad customization options for your installer project, beginning with customizing the appearance of the installer, and progressing to customizing installer flow, commencing with our first complex installer project.

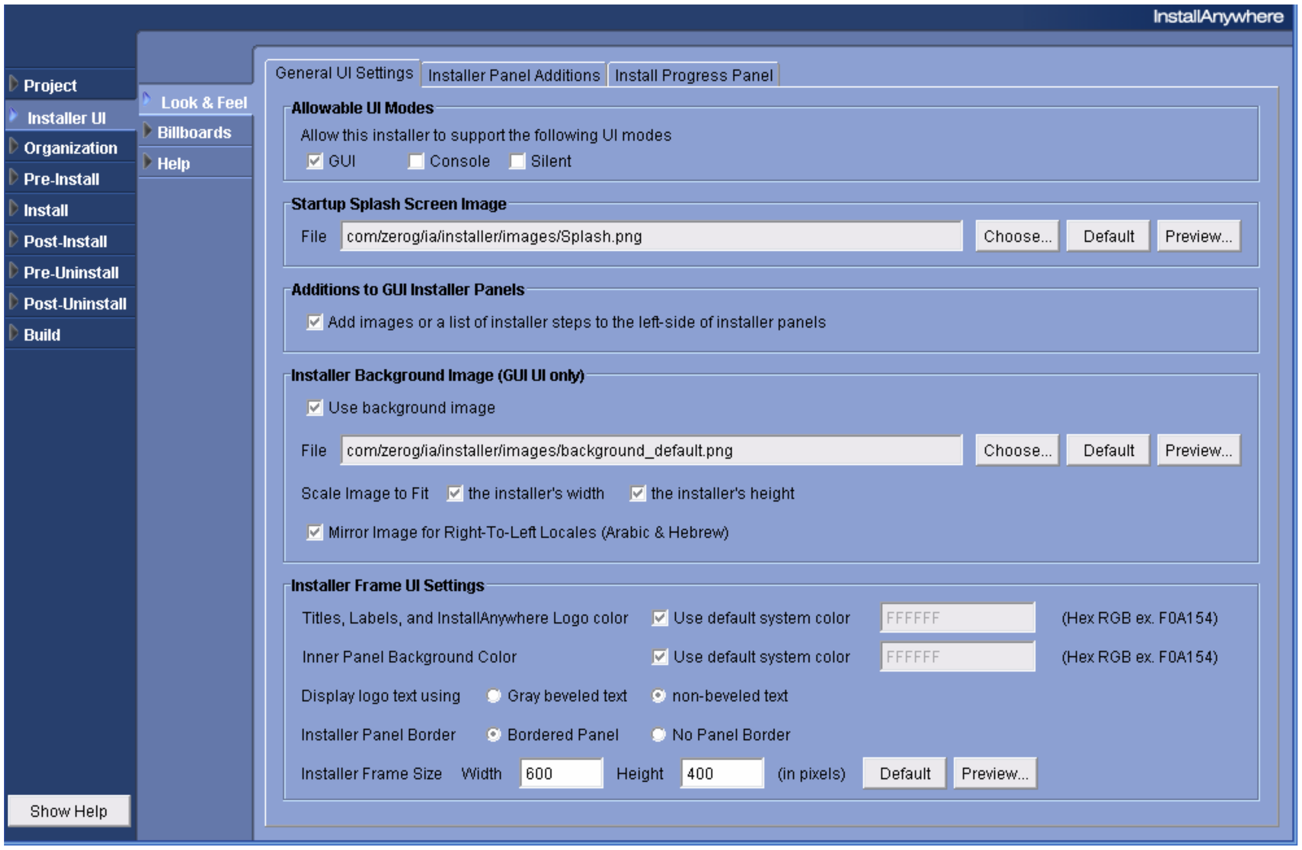
## Customizing Your InstallAnywhere Installer Look and Feel

One of the keys to a professional looking installer is the appearance of the installer itself. Most developers want their installer to reflect the image of their product or company. InstallAnywhere allows you the ability to customize your installer to provide your end users with an installation experience that matches your product's graphics, your target audience, and or organizational image and branding.

## Exercise 8.1 Exploring Look and Feel

InstallAnywhere provides many options for altering the look and feel of the installer. You may add splash screens, display a list of steps or an image along the left side of the installer panel, add a background image that will display behind the steps, or add billboards, graphics (even animated graphics) that display in the large right hand display of the installer.

The following settings are controlled with the Look & Feel subtask of the Installer UI task.



**Allowable UI Modes**

Allow this installer to support the following UI modes

☑ GUI ☐ Console ☐ Silent

**Startup Splash Screen Image**

File: com/zerog/ia/installer/images/Splash.png — Choose... Default Preview...

**Additions to GUI Installer Panels**

☑ Add images or a list of installer steps to the left-side of installer panels

**Installer Background Image (GUI UI only)**

☑ Use background image

File: com/zerog/ia/installer/images/background_default.png — Choose... Default Preview...

Scale Image to Fit ☑ the installer's width ☑ the installer's height

☑ Mirror Image for Right-To-Left Locales (Arabic & Hebrew)

**Installer Frame UI Settings**

Titles, Labels, and InstallAnywhere Logo color ☑ Use default system color FFFFFF (Hex RGB ex. F0A154)

Inner Panel Background Color ☑ Use default system color FFFFFF (Hex RGB ex. F0A154)

Display logo text using: Gray beveled text [✓ non-beveled text]

Installer Panel Border: [✓ Bordered Panel] No Panel Border

Installer Frame Size Width 600 Height 400 (in pixels) Default Preview...

## Installer UI Modes

Defined by the Allowable UI Modes setting, InstallAnywhere installers can run in several different modes defined by the end-user interface. They are:

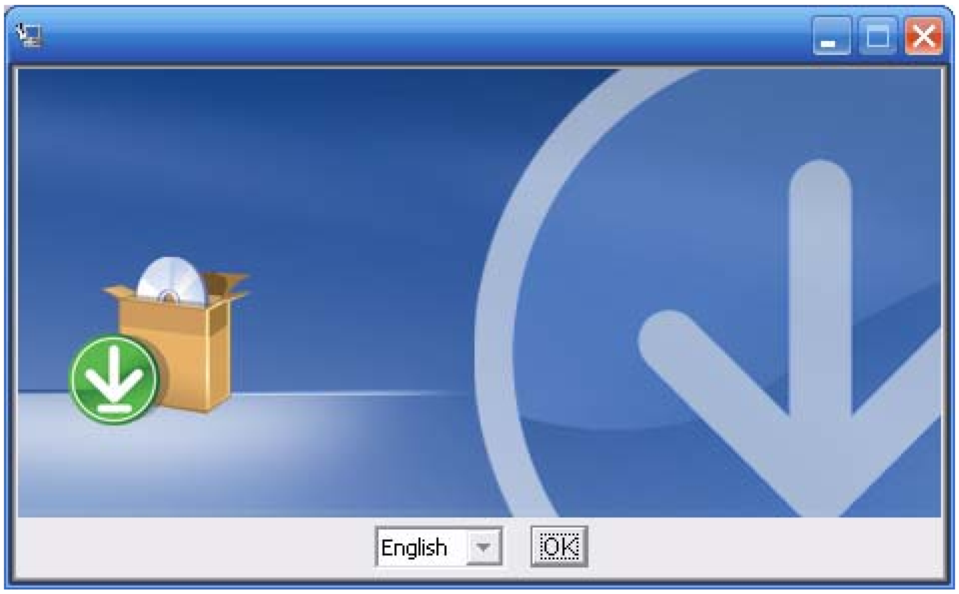
| | |
|---|---|
| GUI | The GUI mode uses Swing, an end-user interface toolkit provided by the Java Foundation Classes. InstallAnywhere's GUI installer interface provides a rich end-user experience including background graphics, rendered HTML, and alpha transparency for graphics used in the installer. The InstallAnywhere installer itself is an example of a GUI installer. |
| Console | Console mode provides a TTY or terminal-style interface that can allow an interactive installation on a system lacking a graphical end-user interface. (Note that you may still need to set a display and/or have X Windows running.) |
| Silent | Silent installers are, as the name implies, a silent installer that requires and provides no end-user interaction. Silent installers can either run without any input, or can accept data from a properties file containing the values for specific InstallAnywhere variables used to control the installation. |

Starting with InstallAnywhere 2008, the legacy AWT GUI mode is no longer supported.

## Splash Screens

InstallAnywhere installers present a Splash Screen at the initial launch of the installer. This screen is displayed for a few seconds while installer resources are extracted and the installer environment is set up. The Splash Screen is an ideal introduction to your product, and an opportunity to set the mood and image for your product. The Splash Screen will also appear on the HTML page generated for the InstallAnywhere Web Install Applet. You specify your splash screen in the Startup Splash Screen Image section of the General UI Settings tab, pictured above.

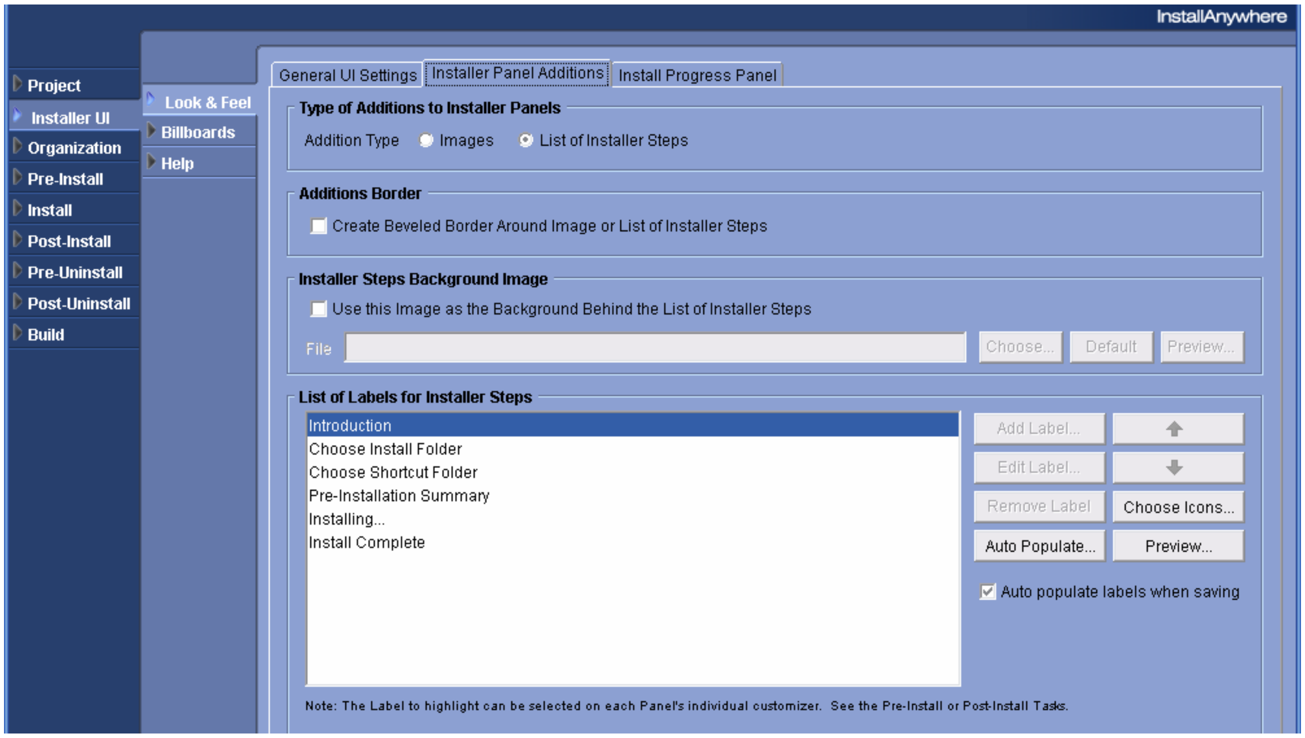
The default splash screen appears similar to the following:



## *GUI Panel Additions*

The Additions to GUI Installer Panels option allows developers to display a list of steps or an image along the left side of the installer's panel.

You specify the details of the additions in the Installer Panel Additions tab of the Look & Feel task:



Type of Additions to Installer Panels

Addition Type: Images [✓ List of Installer Steps]

Additions Border

☐ Create Beveled Border Around Image or List of Installer Steps

Installer Steps Background Image

☐ Use this Image as the Background Behind the List of Installer Steps

File:

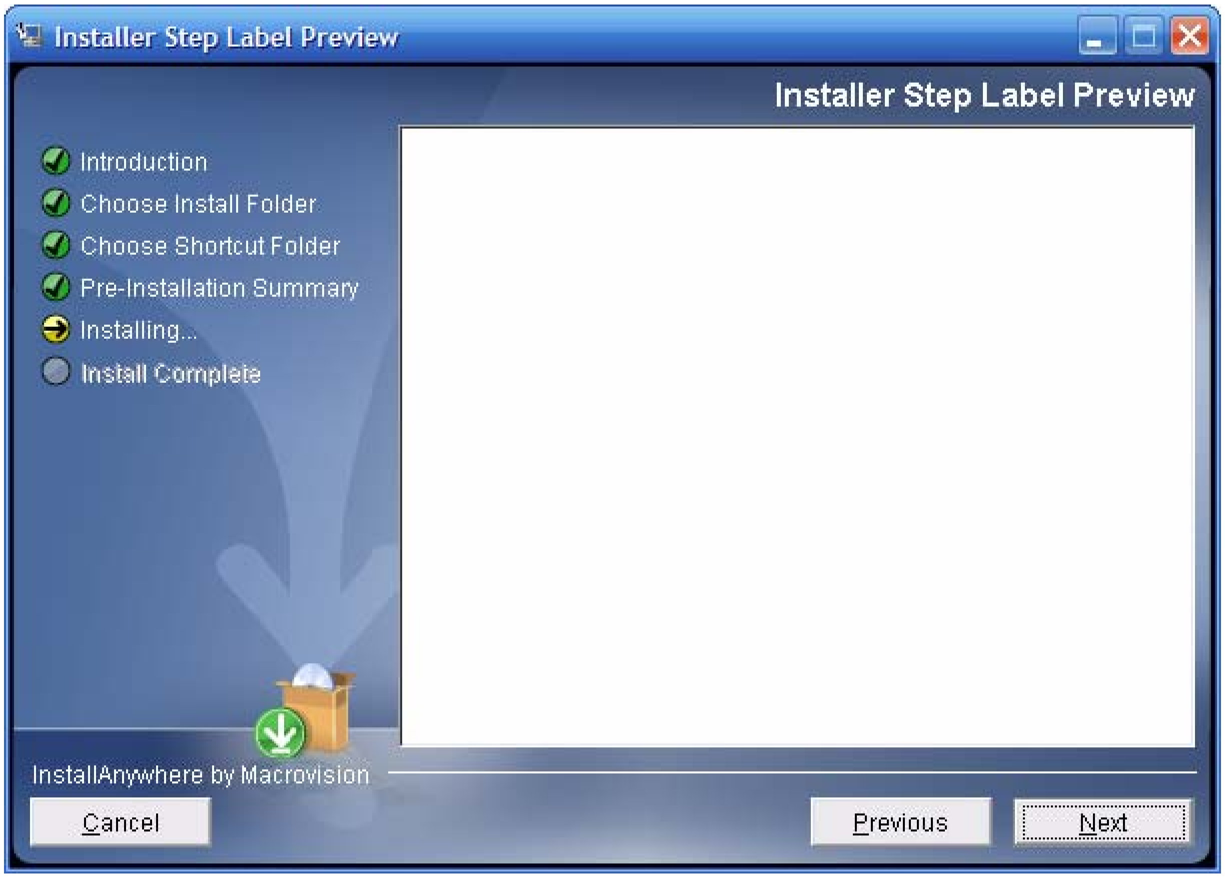
List of Labels for Installer Steps

- Introduction
- Choose Install Folder
- Choose Shortcut Folder
- Pre-Installation Summary
- Installing...
- Install Complete

☑ Auto populate labels when saving

Note: The Label to highlight can be selected on each Panel's individual customizer. See the Pre-Install or Post-Install Tasks.

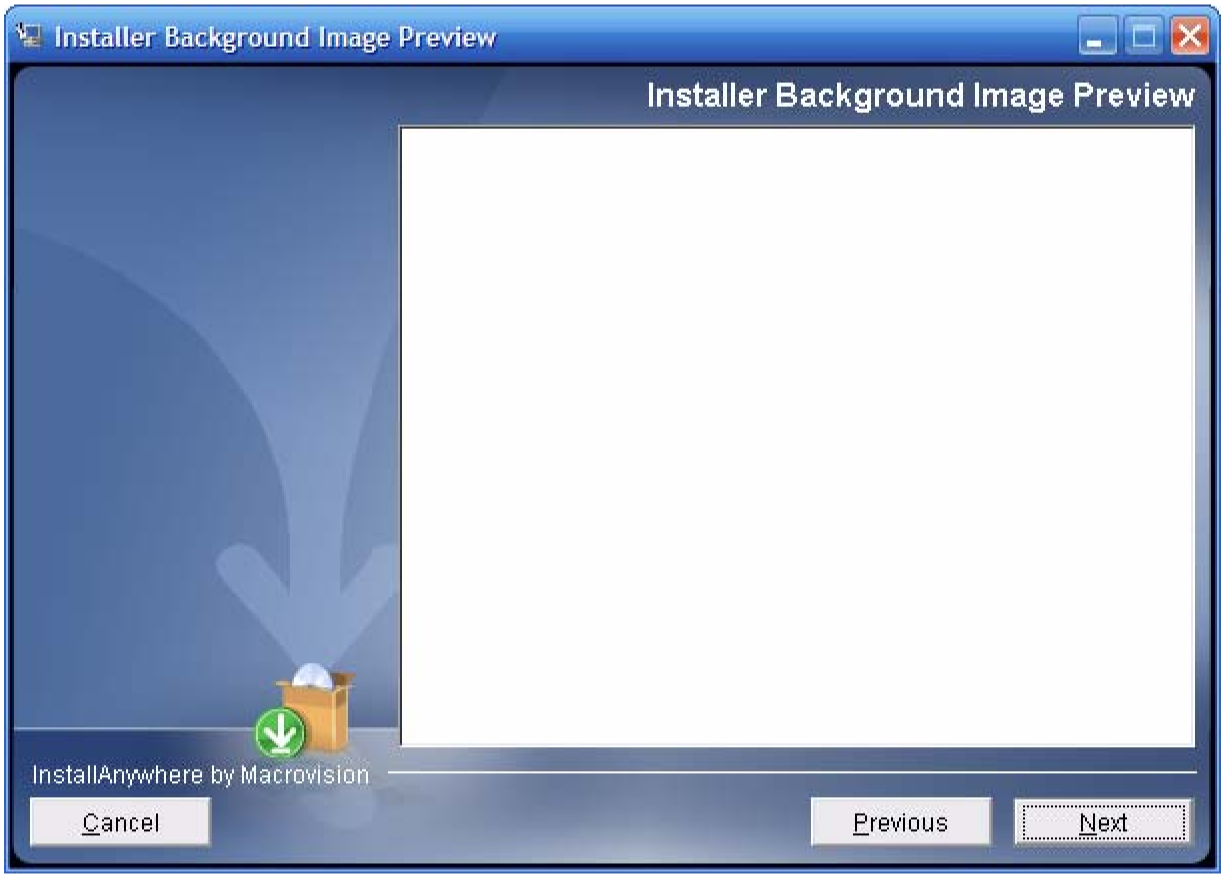
At run time, the additions might appear similar to the following:



## *Background Images*

InstallAnywhere has the ability to add customized background images to your installer. This feature enables you to create a unique installer using the ability to superimpose the left-hand installer steps or image screen and the right-hand informational or interactive rectangle upon a background image.

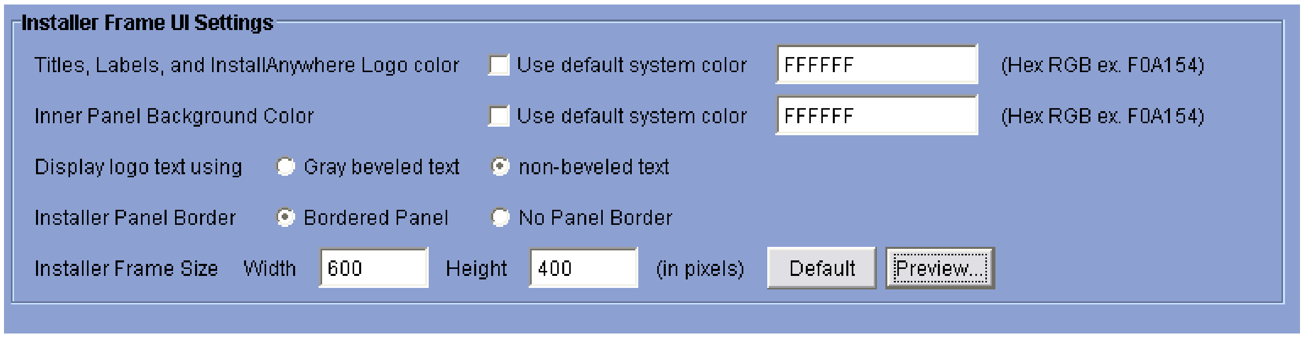
The default image appears similar to the following:


Installer Background Image Preview
Installer Background Image Preview
InstallAnywhere by Macrovision
Cancel
Previous
Next

In addition, you can specify the reverse the image when the installer runs using a right-to-left locale (Arabic or Hebrew). Localization issues are discussed in Chapter 21.

## Frame UI Settings

InstallAnywhere includes the ability to modify the color scheme of the installer text, and the installer panel's background color. The color codes are the standard hex representations of RGB values. This enables you to match the text and background color schemes of your installer to your organization's branding.



Installer Frame UI Settings

Titles, Labels, and InstallAnywhere Logo color ☐ Use default system color FFFFFF (Hex RGB ex. F0A154)

Inner Panel Background Color ☐ Use default system color FFFFFF (Hex RGB ex. F0A154)

Display logo text using ○ Gray beveled text [✓ non-beveled text]

Installer Panel Border [✓ Bordered Panel] ○ No Panel Border

Installer Frame Size Width 600 Height 400 (in pixels)

## Billboards

Billboards are graphics that the installer will display during the installation of files. You can use files to convey a marketing message, a description of your product, or simply something entertaining for your end user to see as the file installation is occurring. Each billboard you add will be displayed for an equal amount of time, based on actions within the installation. If you have very few files and many billboards, each will be displayed for only a short time.

The billboard action includes support for animated GIF files, which enables you to add animations to your billboard, providing your end user with a rich media experience during installation.

## Help

Help is available to the users of the Install Anywhere Installer. Help may either be HTML or plain text. With HTML you use HTML tags to define formatting. With plain text there is no formatting. Help may be associated with a panel, or the same help text may be displayed regardless of the panel being displayed to the end user.

## Exploring the Installer UI Tasks

The Look & Feel task includes several subtasks defined in the tabs across the top of the task. Each of these tasks controls a specific portion of the installer.

1. *Open the OfficeSuite Project you created earlier, and browse to the* ***Project > Look & Feel > General UI Settings*** *task.*

   The **Installer UI > Look & Feel** subtask contains three tabs which enable developers to configure the look and feel of the installer. These tabs allow developers to customize many graphic elements and progress panels within the installer. Within these three tabs are different Preview buttons which will display the look and feel of the installer with the current settings.

   The **Installer UI > Look & Feel > General UI Settings** tab sets general look and feel settings for the installer. Developers select whether the installer will have a GUI, console, or silent mode. (An installer may support all the modes at the same time.) The GUI (Swing) mode offers the ability to have a background image, such as a company logo, to be displayed in the installer. The specific background image can also be defined in this tab of the Installer UI task.

2. *Select the UI Mode.*

   In the **Allowable UI Modes** section, select **GUI**.

   If we wanted a silent or console installer, we could enable these modes here.

3. *Explore Splash Screens.*

   a. *In the* ***Startup Splash Screen*** *Image section, click the* ***Preview*** *button to see an example of the splash screen.*

   InstallAnywhere installers present a splash screen at the initial launch of the installer. This screen is displayed for a few seconds while the installer prepares the wizard.

   b. *Click Choose to select an image for the splash screen.*

   Several splash images can be found in the `ImagesAndDocs` folder.

   The splash screen will also appear on the HTML page generated for the InstallAnywhere Web Install Applet. It can be a GIF, a PNG, or a JPEG file of any size, although the preferred size is 470 by 265 pixels.

4. *In* ***the Installer Background Image*** *section*

   a. *In the Installer* ***Background Image*** *section, select* ***Preview****.*

   b. *In the* ***Installer Background Image*** *section, click* ***Choose****.*

   c. *Select a background graphic.*

   d. *In the* ***Installer Background Image*** *section, select* ***Preview****.*

   The InstallAnywhere installation includes a number of background images, which are free for your use (and notably without royalties). These images are located in the graphics/background folder within the InstallAnywhere installation.

The background image is behind the entire GUI image, behind both the left area (which can have install steps) and the right informational rectangle. The Background image you choose will appear in every panel in your installer. Naturally, background images are supported only in GUI Installers.

5. *Make Additions to the GUI Installer Panels.*

   a. *In* **the Additions to GUI Installer Panels** *section, select* **Add images or a list of installer steps** *to the left side of installer panels.*

      While selecting **Add images or a list of installer steps** watch the remaining two tabs of **the Installer UI > Look & Feel** task. As the option is selected the **Installer Panel Additions** and **Install Progress Panel** tabs will become active.

      If **Additions to GUI Installer Panels** is not selected the **Installer Panel Additions** or **Install Progress Panel** tabs will not be accessible because there will be no additional progress panels to modify. These additional panels are either images or labels.

   b. *Select* **the Installer Panel Additions** *tab.*

      The **Look & Feel > Installer Panel Additions** tab allows developers to customize installer panels for the left hand install progress steps rectangle.

   c. *In the Types of Additions to Installer Panels, select Images.*

      If **Images** is selected, no install labels will be displayed. Selecting **Images** provides the ability to select a default image in the **Default Installer/Uninstaller Panel Image** section of the **Installer Panel Additions** tab. When **Images** is selected the default image may be overridden by specifying an image in the **Install Progress Panel** tab. The developer may also choose not to display an image, or select to use the same image as the previous panel.

      Later you will see how to put a background image behind steps in the left-hand install progress steps rectangle.

   d. *Click* **Preview***.*

   e. *In* **Types of Additions to Installer Panels***, select* **List of Installer Steps***.*

      Selecting **List of Installer Steps** provides labels that can be customized.

      If **List of Installer Steps** is selected in the **Look & Feel > Installer Panel Additions** tab, the **Installer Steps Background Image** option will be available if **Use this Image as the Background Behind the List of Installer Steps** is selected. The **Installer Steps Background Image** option enables developers to select a specific image to display in the rectangle on the left hand side where installation step labels will be displayed.

      When using the GUI UI mode, you can specify a transparent image to be displayed on top of the background image selected in the **Look & Feel > General UI Settings** tab.

NOTE: The size of the install progress pane is 380 by 270 pixels. Installer dimensions may change slightly by platform to better display text and different fonts.

      The bottom pane of the **Installer Panel Additions** allows developers to view and alter the installation steps labels. The buttons to the right enable developers to add or remove labels, or change the order of the labels. Developers can edit the text string that is displayed. The **Auto Populate…** button adds an installer panel for every panel action added to the Pre-Install and Post-Install tasks.

   f. *Alter Order of Steps*

      Using the arrows allows you to change the order of the steps.

   g. *Edit Label*

      Change a label, `Pre-Installation Summary` can be changed to `Summary`.

   h. *Change Icons*

      **Choose Icons…** enables developers to alter the small square graphics that are to the left of the text labels. The default icons are double arrows for the current step or steps to be completed and a check mark for installation steps that have been completed.

6. *Select **Installer UI > Billboards**.*

   Billboards are images that appear in the large right hand pane of the installer while files are being installed. Billboards generally convey a marketing message, a description of the product, or simply something fun for the end user to see as the file installation is occurring.

   a. *In the **Billboard** section near the bottom of the window, click **Preview***

   b. *Near the center of the screen, click the **Add Billboard** button.*

   c. *In the InstallAnywhere 2008 Enterprise\OfficeSuiteSourceFiles\ImagesAndDocs folder, select billboard1.gif.*

   There are several billboard graphics available in the ImagesAndDocs directory within the `OfficeSuiteSourceFiles` folder. In this case, we advise adding two billboards to the installation (more than two and the appearance of the panels would be too short due to the small number of files in this installation).

   Billboards can be GIF, PNG, or JPEG files, and should be 587 by 312 pixels in size.

NOTE: The size of the billboard pane is 587 by 312 pixels. Installer dimensions may change slightly by platform to better display text and different fonts.

   d. *Click **Choose**.*

   e. *Select `billboard2.gif`.*

   f. *Click **Preview**.*

   Each billboard added will be displayed for an even amount of time, based on actions within the installation. If an installation has very few files, and many billboards, each billboard will only be displayed for a short time. Several billboard graphics may be added for larger (and longer) installations. For small installations, like the tutorial OfficeSuite example, only one billboard will show. Billboards may also be assigned to features, and will only be displayed if the feature they are associated with installs.

   When adding multiple billboards, the billboards will display in the order they are shown in the **Installer UI > Billboards > Billboard List**.

7. *Explore Help Options.*

   a. *Browse to **Installer UI > Help**.*

   b. *In the top section, select **Enable installer help**.*

   c. *In **Help Text Format**, select **HTML**.*

   d. *In **Help Context**, select **Use the same help text for all panels**.*

   e. *In the **Title** text field, enter **Sample Help Text**.*

   f. *In the **Help Text** text field, enter*
   `<b>OfficeSuite</b> Help <br> This is an <i>example</i> of HTML help.`

   g. *Click **Preview**.*

   h. *Click **Close**.*

8. *Rebuild the project*

   a. *Click **Build**.*

   b. *Click **Build Project** to build the OfficeSuite installer.*

   c. *Click the blue arrow on the lower left of that dialog to see the Build Details console.*

9. *Test web applet page customization.*

10. *Test installer customizations.*

## *Introducing Conditional Logic*

InstallAnywhere uses variable-based Boolean rules to control most aspects of installer behavior.

In the following segment, we'll cover basic implementation of these rules, and of some of the methods by which the end user interacts with the rules based architecture.

InstallAnywhere Rules can be applied to any action within the InstallAnywhere installer, as well as to organizational units such as Install Sets, Features, and Components (all of which we will discuss a little later in this chapter).

The Rules logic allows you to create simple and complex logic systems that determine what actions will occur. The rules can be structured based on end-user input, or on conditions determined by the installer.

There are a handful of preset rules included in InstallAnywhere:

| | | |
|---|---|---|
| **Check File/Folder Attributes** | E | This rule allows you to check the attributes of a file or directory that already exists on the target system. The rule allows you to check if the object exists, whether it is a file or a folder/directory and whether it readable and/or writable. |
| **Check If File/Folder Exists** | E S | This rule is designed to be applied to individual file/folder install actions. It will check to see if the file or folder to which it is attached already exists in the specified install location. You can choose to install either if it does, or does not exist at that location. |
| **Check Platform** | E S | This rule allows you to specify action or files to be run or installed only on specific platforms. The platform is determined by the Java virtual machine, and reported to the installer. |
| **Check System Architecture** | E S | This rule allows you to specify actions or files to be run/installed only on specific system architecture (32-bit or 64-bit). The system architecture is normally determined by the Java virtual machine, and reported to the installer, but in some cases may be specified in a registry action. |
| **Check User-Chosen Language** | E S | You can use Check User-Chosen Language to make installation decision based on the locale chosen by the end user at installation time. |
| **Compare File Modification Timestamp** | E | This rule allows you to compare the timestamp of an existing target file in order to make an overwrite decision. |
| **Compare InstallAnywhere Variables** | E S | This rule allows you to make a simple string comparison of any InstallAnywhere variable. You can check if a variable equals, does not equal, contains, or does not contain a value. |
| **Evaluate Custom Rule** | E | Custom Rules are rules built using the specifications outlined in the InstallAnywhere API and can be tailored to fit the needs of your installation. More concerning custom and API development will be covered later in the curriculum. |
| **Match Regular Expression** | E | The Match Regular Expression rule allows you to compare a string, or InstallAnywhere Variable to a regular expression of your choosing. Regular Expressions (`regexp`) are an industry standard method of expressing a variable string. You can find considerable information on regular expressions, including |

archives of use full expressions, and web applications that can be used to verify the validity of your expression on the World Wide Web.

## Exercise 8.2 Using Installer Rules

InstallAnywhere Rules can be implemented in a number of actions within the installer. However, the first set of rules evaluated in the installer is the Installer Rules. These rules, set in the Project > Rules task, allow you to control the complete installer based on rules.

For example, let's consider that our OfficeSuite product is designed to run only on the following systems: Windows 2000, Windows XP, and Windows Vista, but not Windows NT; on Mac OS X, and on Linux.

Add this condition to the OfficeSuite installer.

1. *Add the rule.*
    a. *Open the* ***Project > Rules*** *task*
    b. *Click* ***Add Rule****.*
    c. *Select* ***Check Platform****.*
    d. *Click* ***Add****.*
2. *Set the condition for the rule*
    a. *In the customizer for the Check Platform rule, select Windows 2000, Windows XP, Windows Vista, Mac OS X, and Linux from the left-hand (Do Not Perform On) column.*

       You can hold down the Ctrl or Apple keys to select multiple items.
    b. *Move the selected items to the right (Perform On) column by clicking the arrow.*
3. *In the customizer below the Check Platform section, modify the message that will appear if an end user attempts to run the installer on a platform other than those you've specified.*
4. *Build and run the installer.*

   If you are running on a platform other than those that we've specified, the installer should run normally.
5. *Return to the project, and remove the platform you are working on at the moment. Rebuild, and re-run the installer. You should see the message you entered indicating that the platform was disallowed.*

NOTE: Before continuing, be sure to add the rule back in. This will enable your installer to run properly in the next exercise.

## Exercise 8.3 Using Rules to Control Visual Elements

Often, you'll want certain panels and or other visual elements to appear only under certain conditions. For example, notes explaining errata on Windows shouldn't appear on Mac OS X. Like the installer rules you can use rules to control visual elements in the Pre, and Post-Install tasks. In this next set of exercises, we'll introduce some elements of visual control, and introduce a few of the available panels, and other actions.

1. *Click* ***Pre-Install Task*** *from the task list along the left hand side of the InstallAnywhere Advanced Designer.*
2. *Click* ***Add Action*** *near the middle of the screen.*

   This step will open the Action Palette. The Action Palette has several tabs for differing types of actions.
3. *Select the Panels tab on the palette and add the following panels to your project:*

| | |
|---|---|
| **License Agreement** | This panel allows you to display a license agreement to your end user. The end user must choose to accept the agreement in order to continue. You can set the default state of the radio buttons (Accept or Decline) and choose a file to use for a license agreement. You'll find a License.txt file in the ImagesAndDocs folder within the OfficeSuite Source Files that can be used for this installer. The License Agreement Panel can also utilize HTML files, which give you a degree of control over the text formatting and allow you to link to external documents. |
| **Display Message** | The Display Message Panel allows you to simply display a text message to the end user during the installation. This can be useful for conveying information about installation choices that the end user has made. This panel is also particularly useful in debugging installer issue having to do with InstallAnywhere Variables. You can add Display Message panels with variables resolved to test variable values you are using in rules.<br>Place any text of your choosing on the Display Message Panel, however try including several InstallAnywhere Variables, using the $VARIABLE$ notation so that they are resolved.<br>Example: This installer is running on a `$prop.os.name$` `$prop.os.version$` system named `$prop.computername$` and is running against a `$prop.java.vendor$` `$prop.java.version$` VM. |
| **Important Note** | The important note panel allows you to display a text or HTML file without the radio buttons found on the license agreement panel. It is particularly useful for displaying Readme or errata type documents. |

You can choose to place these panels in any order or location within the pre-install, although for authenticity's sake it is recommended that you place them after the **Introduction** panel and before the **Choose Install Folder** panel.

4. *Add rules to actions and/or panel actions.*

We've now created a set of install panels that will appear in the pre-install section of the installer. Now, let's add a rule to one of the panels.

a. *Select* **Display Message***.*

b. *In the* **Title** *text bar, name the Display Message "Running as root" so it will be easily recognized if other Display Messages are used.*

c. *In the text box* **Enter message to be displayed during installation***, enter Display only if running as root.*

d. *In the* **Customizer***, click* **Rules***.*

e. *Click* **Add Rule***.*

f. *Select Compare InstallAnywhere Variables, then click* **Add***.*

g. *Create the following rule:*

Install Only If:

| Operand1 | | Operand2 |
|---|---|---|
| `$prop.user.name$` | `equals` | `root` |

Can you tell the purpose of this rule? This rule should restrict the first Display Message Panel so that it appears only if the end-user name for the end user running the installer is root.

h. *Create another similar rule:*

Install Only If:

```
Operand1                                   Operand2
$prop.user.name$     does not equal       root
```

The second panel will only display if the end user is not root. Generally, this type of rule would only be used to display panels or execute actions for administrative end users on Unix systems.

Notice the "R" that appears in the upper right corner of the Panel Icon in the Advanced Designer. This visual identifier serves to indicate that a Rule has been applied to the action.

5. *Build and run the new installer project.*

   Notice the addition of the new panels. If you added the rule to your display message panel, you'll not see that panel. Try removing that rule, and re-running. The panel should now appear.

The example we've just considered is, of course, over simplified. However, the concepts are important to understanding the basic behavior of InstallAnywhere functionality. By now, you should have a basic understanding of InstallAnywhere Variables, and a basic understanding of InstallAnywhere Rules. In the next segment, we'll be building a more complex installer, integrating end-user input with the concepts covered here.

## Exercise 8.4 Managing Installer Flow Based on End-User Input

In many cases the path that an end user will take through an installer depends on the choices made in different steps within the installation procedure. InstallAnywhere provides methods to gather input from end users, which you can leverage to control your installation.

In this first example, we'll return to an action we added to our OfficeSuite Installer—the execute action added in the Post-Install task.

Generally, it's nice to ask the end user if they would like to launch the application when the installation is complete. In order to add this functionality, we'll need both a method to ask the end user if they would like to launch the application, and a method by which we can control that action.

In previous sections, we've seen the Rules methods that can be used to prevent the installer from displaying certain panels, and we've learned a little about the InstallAnywhere Variable architecture that is used to store information within the installer.

In this next exercise, we'll put the two together in a useful manner.

### Step One: Retrieving End-User Input

1. *Open the* Post-Install *task of the project.*

   Later we'll actually create some new projects, but for now we'll continue to use and abuse our OfficeSuite installer.

2. *Open the* Action Palette *and from the* Panels *tab, add* Panel: Get User Input - Simple*.*

   The Get User Input - Simple panel allows you to retrieve a single type of information from the end user and store it in a single InstallAnywhere Variable for later use. The panel allows input in text fields, choice menus, pop up menus, radio buttons, or check boxes.

3. *Define and configure the panel.*

   a. *In the Title field of the customizer for the Get User Input - Simple panel, enter Launch.*

   b. *In the Prompt text field enter, Would you like to Launch OfficeSuite?*

      As we only want to ask a Yes or No question, we'll utilize the radio buttons option as the input method.

c. *Choose* ***Radio Buttons*** *from the pull down menu on the middle right of the customizer.*

Now, we'll configure the panel to present our options.

d. *Click the Configure button, then Add.*

Clicking the **Configure** button will open a dialog where you can add labels for the buttons and set their default states.

e. *A field will appear in the dialog. In the left hand portion of the field, type your message. For this example, we'll use: Yes, Launch OfficeSuite now.*

Make sure the word `Yes` is capitalized. The capitalization of the message is important because the string from the label will be stored exactly as you enter it in the Results Variable—the variable which stores the results of the end user's input.

f. *Now click twice in the Default Value field to the right, then choose Selected.*

4. *Create the No message for the panel.*

   a. *Click* ***Add****.*

   b. *In the* ***Label*** *column enter* `No, I'll open OfficeSuite later, thanks`*.*

   Be sure that `No` is capitalized in your message.

   c. *Click* ***Set Variable****.*

5. *Set the Results Variable to $LAUNCH_APPLICATION$.*

   The default Results Variable is `$USER_INPUT_RESULTS$`, however you can change the variable to fit your needs or your naming scheme.

6. *Click the* ***Preview*** *button. You should see two radio buttons.*

7. *Before continuing, make sure that you've placed the get end-user input panel in the Post-Install tree prior to the Execute Target File action.*

## *Step Two: Applying the End User's Choice*

Now, to control the **Execute Target File** action we added earlier, we'll add a rule to that action. We want the action to occur only if the end user has selected the Yes option. Since that information is stored in the variable we selected in the **Get User Input - Simple** panel, we'll use a Compare InstallAnywhere Variables rule.

1. *Add the Compare InstallAnywhere Variables rule by selecting the Execute Target File action, and choosing the Rules tab from the customizer in the lower portion of the window.*

   Use **Add Rule** to add the **Compare InstallAnywhere Variables** rule. We'll add the following rule:
   **Install Only If:**

| Operand 1: | | Operand 2 |
|---|---|---|
| *`$LAUNCH_APPLICATION$`* | *`contains`* | *`Yes`* |

   If you used a variable other than `$LAUNCH_APPLICATION$` in the results variable in the Get **User Input - Simple** panel, use that variable here.

HINT: Although it is strictly necessary only when retrieving variables, it's a good idea to use the `$` notation when setting variables as well. This makes it easier to keep straight what is a variable and what is a literal value.

2. *Rebuild and re-launch the installer.*

   You should be able to choose whether or not to launch the application. If the application launches, even when you've chosen "No," check your Rule to ensure that you are correctly comparing the variable and that the case of the value is correct.

## Quick Quiz

1. Which InstallAnywhere Rule would you use to verify an entry that a user had made in a text field (for example, checking to see if they have entered a valid telephone number)?
   A. Check Platform
   B. Compare InstallAnywhere Variable
   C. Match Regular Expression
2. What notation is used in InstallAnywhere to indicate that a variable's value, as opposed to its literal name, be returned?
   A. #Variable#
   B. $VARIABLE$
   C. $Variable
3. If the logged in username is Jim, what effect will the following rule have on a panel?

   Install Only If:

   | Operand 1: | | Operand 2: |
   |---|---|---|
   | `$prop.user.name$` | `equals` | `jim` |

   A. The panel will display
   B. The panel will not display

Answers: 1.C | 2.B | 3. B

# Chapter 9: Installer Organization

- **Install Sets, Features, and Components**
- **Organizing Features and Components**
- **Using InstallAnywhere’s Basic Installer Organization**
- **Adding Components**
- **Assigning Files to Components**
- **Removing Empty Components**
- **Integrating Components Already Installed on the Target System**
- **Adding Features**
- **Quick Quiz**

So far, we've worked exclusively with a very simple file set, a single product with a single install option. While it would be nice if all installations were this easy, it would also generally negate the need for installer software. Most installations are complex, and offer the user the choice of multiple options, multiple products, and even more options within those multiple product installations. InstallAnywhere provides three logical groupings for managing product installation options—Install Sets, Features, and Components.

## Install Sets, Features, and Components

Install Sets, Features, and Components are one of the most important concepts to understand when using the InstallAnywhere installer development environment. InstallAnywhere provides levels of granularity for end-user installation options. Install Sets and Features may be selected by the end user.

Install sets are groupings of features such as Typical Install or Minimal Install. Features are meant to identify specific units of functionality of your product. While both Install Sets and Features are made up of files, there is not necessarily a direct correlation between these larger organizational groupings and the files.

Components are groupings of the specific files and actions of your product, and are invisible to the end user. A component may also include a group of registry changes or other elements needed to make a feature work properly. Components are used for the organized sharing of resources, for versioning, are uniquely identified, and are the organization tool of the installer developer, not the installer user. A developer could create a component for each feature in the application, a component for shared libraries used by all the features, and a component for the help system.

Though developers may assign files, folders and actions directly to Product Features, it is best to think of features as groupings of components. Install sets such as Typical Install or Minimal Install are groupings of Features. The interaction between the three levels should be addressed when planning the options to present to the end user.

InstallAnywhere's organization features allow you to both manage the installation from a developer standpoint, and offer your users the maximum number of installation options.

### Install Sets

Install Sets are the simplest and broadest organizational concept within InstallAnywhere. Install Sets are a set of product features which represent high-level, easily selectable, installation options. End users can choose only one install set. These sets are generally options such as: Typical and Minimal, or Client only and Client and Server. End users select their desired Install Set using the Choose Install Set panel or console. Install Sets are made up of Features.

### Features

Features are a logical grouping of capabilities in a product. Features are effective if developers want to give their end users fine-grained choices. Features are meant to identify distinct parts of the product so the end user may choose whether or not to install them. It is up to the developer to define the logical grouping of components into features by assigning components to the features.

Features may be hierarchical, and there can be as many as desired, but there needs to be at least one. One example of Features in a hierarchy would be a Documentation sub-feature and a Samples sub-feature added beneath the main Help feature.

Several Features make up each Install Set, and each Feature can belong to one or more Install Sets. End users select Features as an option from the Choose Install Set panel or console. Features are visible to the end user only when the user chooses to customize an Install Set.

Files can be assigned directly to Features, or you can assign Components to Features. InstallAnywhere actually creates Components for you if you assign Files directly to Features.

End users can uninstall specific Features if desired.

## *Components*

Components are the smallest piece of an installation handled by the installer. From a developers perspective they are the building blocks of applications or features. End users never see or interact with Components.

There are many advantages to having fine-grained control over components. Common components may be shared between multiple installers, multiple versions of a product, or multiple products. For example, two products in a suite could have several shared components.

Components are uniquely identified so developers may update a specific component or use the Find Component in Registry action to locate a particular component. Components are versioned as well as having a unique ID, so that a particular version of a component on a system can be searched for to see if the latest version has been installed at a particular location.

Several Components make up each Feature, and each Component can belong to one or more Feature. Components are made up of files and actions. Although you can assign files and actions to Components, you can also assign Files and actions directly to features and let InstallAnywhere automatically create the Components.

A file or action may belong to one component only. All installers must have at least one component, and can have as many as needed.

The InstallAnywhere uninstaller is component-based, and can provide feature-level uninstallation functionality.

HINT: For most installations, you will not need to manipulate the components in any way. Components are automatically generated based on the way that you've assigned files to Features and Install Sets.

### Types of Components: Shared Components and Component Dependencies

Shared Components and Component Dependencies are part of InstallAnywhere's installer organization toolbox. Using Shared Components and Component Dependencies, you can create installers that leverage external components, either developed in house or by a third party, as part of your software distribution strategy. These components can be included as part of a suite installer, be defined as pre-requisite dependencies for your package, and can even allow multiple applications to share common components across a system.

### Shared Components

InstallAnywhere's shared component feature defines a method by which you can share components in your installation with other, later installations—or by which your installation can make use of other previously installed components.

## Component Dependencies

Component Dependencies allow you to specify requirements for your installation that may, or may not be included in your package. For example, if your installation required a database component be installed as a separate package—your installer could specify that database component as a dependency, and would let your end user know if they had met that dependency at install time.

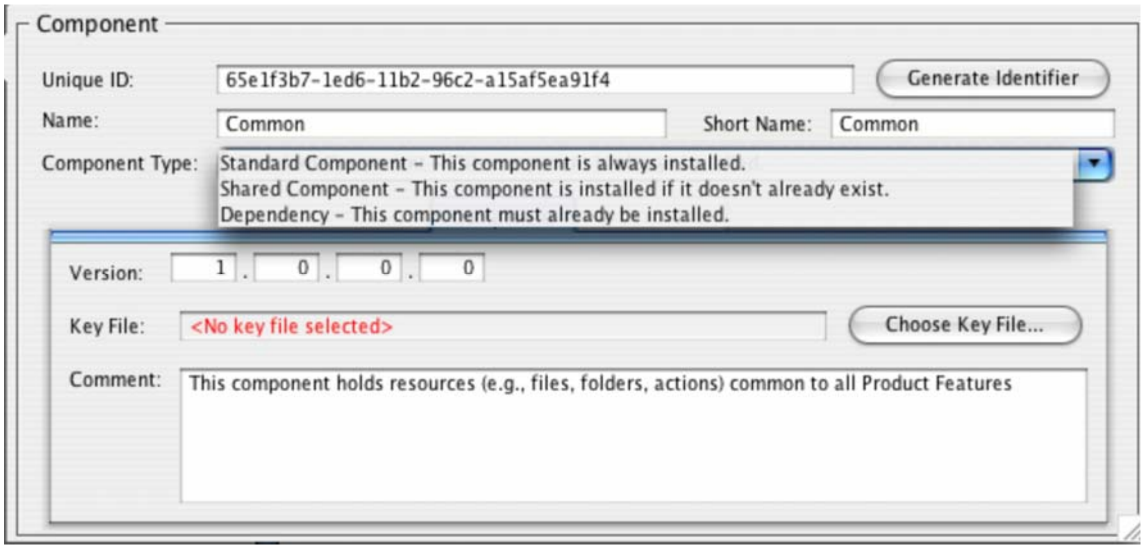
## Defining Shared Components and Dependencies

Component behavior and properties are defined in the InstallAnywhere Advanced Designer's Organization > Components subtask. As you define components, you have the option of defining its dependency and sharing options, as well as all the standard component options. Each new component will give you the option of specifying one of three organization types:

**A Standard Component**—Meaning that your component will always be installed, regardless of previous installations or dependencies. The uninstaller for the product with which they were installed uninstalls standard components.

**A Shared Component**—Meaning that this component will be installed if it has not already been installed on the system, and can be made available to other applications or installations. At uninstall, the Shared Component will only be removed if no other applications are referencing it. The last installation referencing the Shared Component will uninstall it.

**A Dependency**—Meaning that rather than actually installing the component, the installer will require that it has already been installed on the system. Dependencies are not elements of your install per se, but rather are pre-requisite requirements that the installer will enforce.



Unique ID: 65e1f3b7-1ed6-11b2-96c2-a15af5ea91f4
Name: Common
Short Name: Common
Component Type:
Standard Component - This component is always installed.
Shared Component - This component is installed if it doesn't already exist.
Dependency - This component must already be installed.
Version: 1 . 0 . 0 . 0
Key File: <No key file selected>
Comment: This component holds resources (e.g., files, folders, actions) common to all Product Features

Shared Components enable you to spread common components across your development enterprise. If a team has developed a database distribution used by other groups, and that component has been installed as a shared component, you can leverage that component in your own installation.

### The Virtual Machine Component

An example of a shared component is the “InstallAnywhere VM Component”—a shared component automatically created by the InstallAnywhere advanced designer. The VM Component allows you to share a single VM between multiple installations—meaning that if several projects in your organization use the same Java Virtual Machine, you don’t need to install that VM more than once. The different installations can reference the same virtual machine, and the last installation making use of the VM will remove it on uninstall.

## *The Organization Task*

The Organization task enables developers to arrange Install Sets, Product Features, Components, and Merge Modules (Merge Modules will be discussed in depth in another chapter). Install Sets and Features allow for levels of installation options for the end user of the installer. Components are the smallest widget that can be selected by a Feature set. Install Sets are groupings of Features, and are an organizational tool for the developer of the installer. Components may be much more than files, they can be sophisticated actions that are required to install and run applications or features properly.

There is an interaction between the Install Sets, Features, and Components subtasks, as well as the Install task. If an install set is added in the Organization > Install Sets task, features can be assigned to that install set in Organization > Features. If a feature is added in Organization > Features components can be assigned to that feature in Organization > Components. If a component is added in Organization > Components files and/or actions can be assigned to that component after the files and/or actions are added in the Install task.

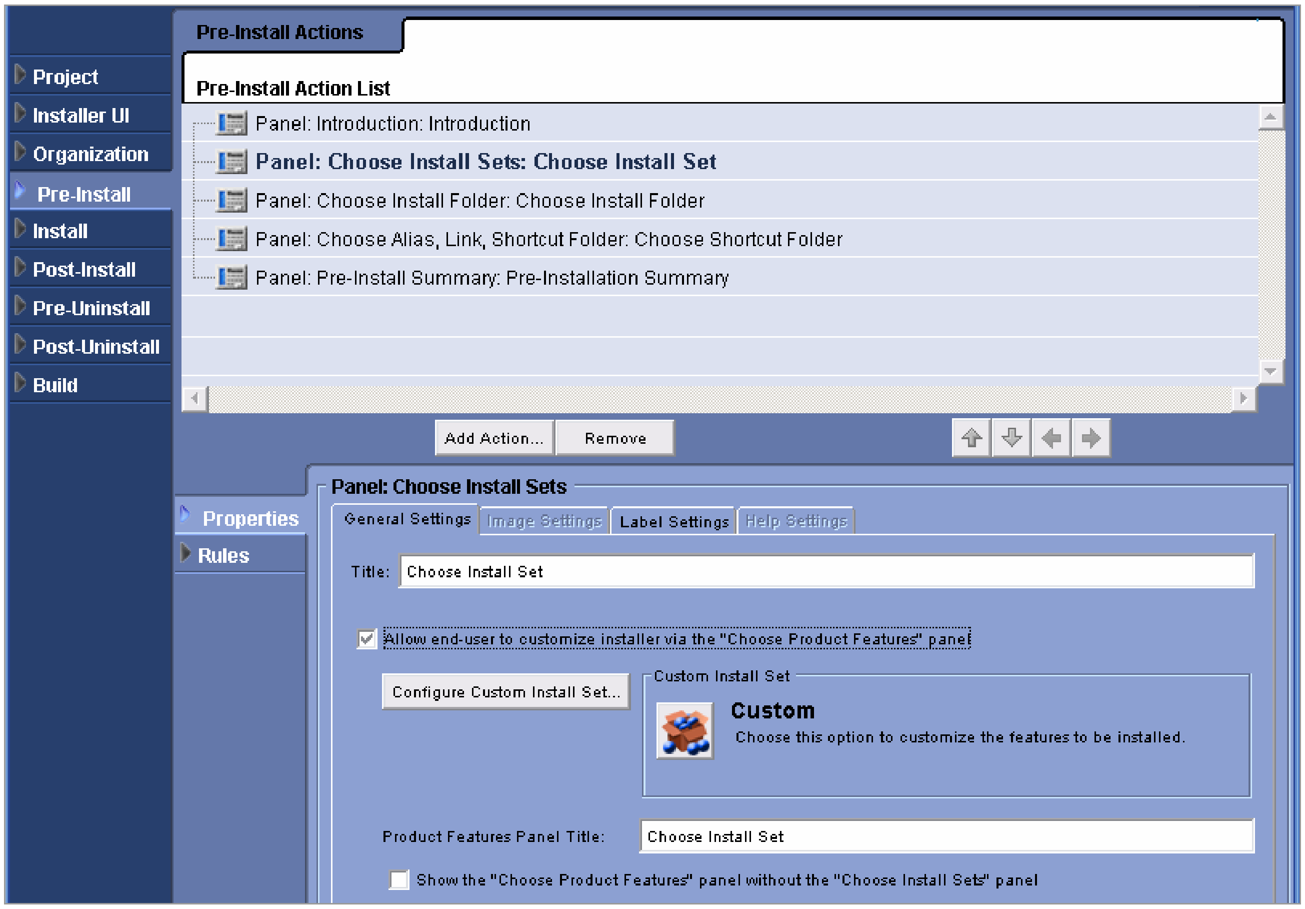
### Install Sets

The Organization > Install Sets subtask allows developers to add, name, remove, or order Install sets in the installer. In the Install Set List developers define which install set (or sets) to use as the default option to provide to the end user. Features are assigned to install sets in the Organization > Features subtask.

When the installer requests install set information, each install set is represented by a graphic element. The Choose Image... button enables developers to select the graphic element.

Rules may be associated with an install set, and that association is created by selecting Rules in the customizer and adding rules. The rules for install sets are evaluated before the install set is installed. If the rules on the Install Set evaluate to false, the Install Set will not be displayed.

In addition to defining your own install sets, you can enable the Custom install set, where the user can manually select which features to install, using the Customizer of the Choose Install Sets panel. After adding the Choose Install Sets panel to your Pre-Install task, select the check box labeled “Allow end-user to customize installer…”



Pre-Install Action List

- Panel: Introduction: Introduction
- **Panel: Choose Install Sets: Choose Install Set**
- Panel: Choose Install Folder: Choose Install Folder
- Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
- Panel: Pre-Install Summary: Pre-Installation Summary

Panel: Choose Install Sets

Title: Choose Install Set

[✓] Allow end-user to customize installer via the "Choose Product Features" panel

Custom Install Set

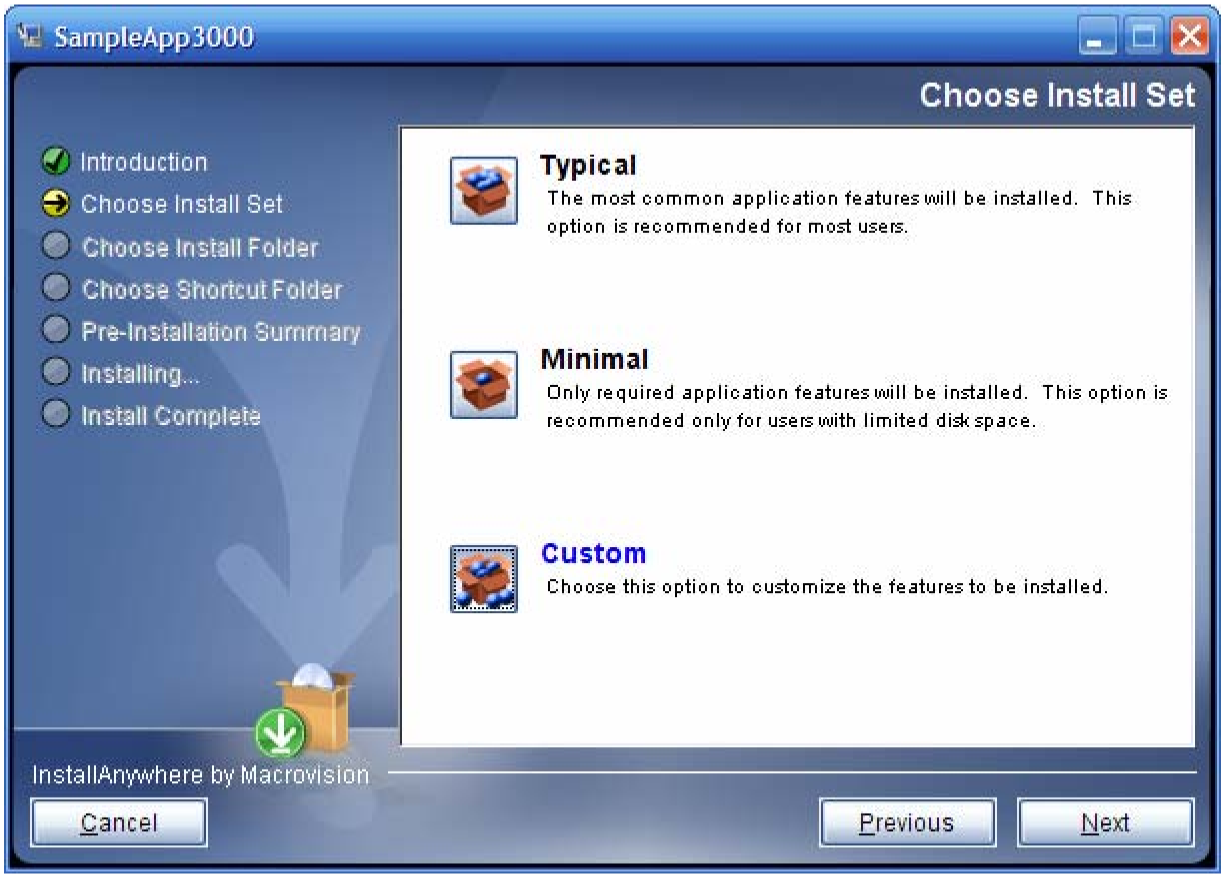
**Custom**
Choose this option to customize the features to be installed.

Product Features Panel Title: Choose Install Set

[ ] Show the "Choose Product Features" panel without the "Choose Install Sets" panel

Using the Customizer, you can also modify the custom install set’s name, description, and icon.

At run time, the Choose Install Sets panel containing the Custom install set appears as follows.


SampleApp3000

Choose Install Set

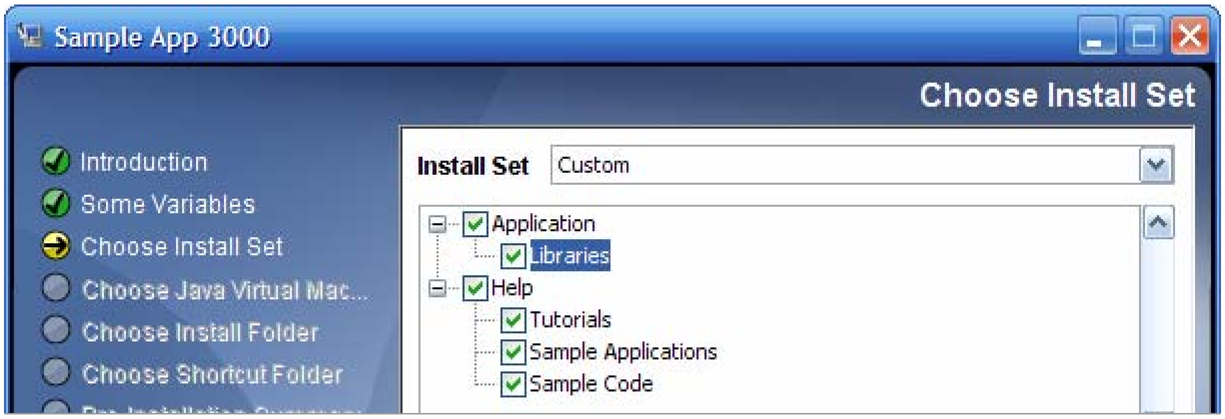
**Typical**
The most common application features will be installed. This option is recommended for most users.

**Minimal**
Only required application features will be installed. This option is recommended only for users with limited disk space.

**Custom**
Choose this option to customize the features to be installed.

## Features

The Organization > Features subtask enables developers to add, name, remove, or order features. In addition to creating a single layer of top-level features, you can use the right-arrow and left-arrow buttons to “promote” and “demote” features in a multi-level feature tree. The following figure illustrates a feature tree with two levels of features: top-level features and their subfeatures.


Sample App 3000

Choose Install Set

Install Set: Custom

- ✓ Application
  - ✓ Libraries
- ✓ Help
  - ✓ Tutorials
  - ✓ Sample Applications
  - ✓ Sample Code

Rules may be associated with a feature set, and that association is created by selecting Rules in the customizer and adding rules. The rules for feature sets are evaluated before the feature set is installed. If the rules on the Feature evaluate to false, the Feature will not be displayed.

### Components

The Key File is a file that must be present in all subsequent versions of the component. The Key File is used to define the component's location when the Find Component in Registry action is used.

The Organization > Components subtask enables developers to add, name, remove, order, identify, and version components.

Rules may be associated with component sets, and that association is created by selecting Rules in the customizer and adding rules. The rules for component sets are evaluated before the component set is installed.

## *Organizing Features and Components*

Components are the lowest level of organization in an installer. Each product must have at least one component but most installers will by default contain at least two components, as the uninstaller is considered a component of its own.

InstallAnywhere's component architecture is designed to allow developers to plan for future releases, suite installers, and other uses of their software elements in their deployment plan.

InstallAnywhere will automatically create components as you add files to your project and assign them to Features. This approach, while working well for most projects, does not give you the most flexibility. To realize the ultimate benefits of componentized software, you should manually manage the creation of components.

### *Best Practices for Components*

When using components, first determine and organize which components to add.

A few things to keep in mind:

- Make unique components for files that will need to be updated separately. For example, a "Help" feature may have both a User Guide and Javadocs. However, the User Guide may be updated more frequently than the Javadocs. Make the two items separate components so a unique "User Guide" component may be added which can be versioned and updated individually.
- Components should make logical sense. When building a Suite Installer, keep in mind the pieces of applications that are shared between different products. When componentizing a product for versioning purposes, designate the version of the component in the Organization > Component > Properties task when the component is added.

NOTE: If you are using Components, we recommend that you do not modify files and features using the Install task. If you modify which files are assigned to a particular feature using this option, components will be modified automatically.

### *Best Practices for Features*

Features are effective if you want to provide end users fine-grained choice in terms of what they install. For example, you might have a main application feature, a shared libraries feature, and a help feature. To make sure end users can choose which feature gets installed; enable the Choose Install Sets panel action via the Pre-Install task in the Advanced Designer. To ensure that your end users can choose which features get uninstalled, enable the Feature Level Uninstall option in the Create Uninstaller action.

A few things you should know about Features:

- Features are logical groupings of components.
- Feature designation arranges your components by function.
- Features may be hierarchical.
- You can create as many features as you wish, but every project needs at least one.
- Features are visible to the end user.
- You'll also want to ensure that your features are as independent as possible, each being as close to an individual package as possible. However, keep in mind that features do not include dependencies, so if multiple features share dependant files sets, those files must be added to each feature (this is merely organizational, and does not, in any way, affect the size of your installation).

### *Best Practices for Install Sets*

The InstallAnywhere **Choose Install Set** panel will only display approximately four features with complete descriptions. You should design your installation with a minimal number of options—or so that rules eliminate invalid Install Sets prior to the display of the option panel.

## *Exercise 9.1 Using InstallAnywhere's Basic Installer Organization*

While we'll cover component architecture later in the in the course, we've advanced enough to delve into the basics of installer organization.

In this next exercise, we'll extend our OfficeSuite installer to employ InstallAnywhere Features, and Install Sets.

1. *If it's not already open, open your OfficeSuite Project in the InstallAnywhere Advanced Designer.*
2. *Click on Install.*

   Depending on what files you added when initially creating the project, you probably have either a directory called OfficeSuiteSourceFiles, or a directory called OfficeSuite2000.
3. *If you added the entire OfficeSuiteSourceFiles directory, you can skip onto the next step. If not, we're now going to add some auxiliary files so that we have a little more to work with in the installer.*
   a. *Click the* **Add Files** *button.*
   b. *Use the file chooser to select the ImagesAndDocs directory from within the OfficeSuite Source Files directory. Add this directory and its contents to your installer.*
4. *If you originally added the entire OfficeSuiteSourceFiles directory, we'll make a little change that will improve the "cleanliness of the project, and will make it easier to organize. Using either the left and right arrows, or the drag and drop functionality, move the OfficeSuite2000 and ImagesAndDocs folders into the root of your installation—the $USER_INSTALL_DIR$.*
5. *Remove the OfficeSuiteSourceFiles directory by highlighting it and clicking Remove.*

   The `OfficeSuiteSource Files` directory should now be empty.

   You should now have three separate directories under the root of your installation: `OficeSuite2000`, `ImagesandDocs`, and `UninstallerData`.

6. *Rename the OfficeSuite2000 and ImagesAndDocs folder. For this example, use the names Program and Data for OfficeSuite and ImagesAndDocs.*

**NOTE:** InstallAnywhere allows you to rename resources independently of the source directories and files. Simply highlight the file you need to rename, and alter the entry in the name field. This allows you to use multiple instances of the same file, but with different names. It also allows you to dynamically name files when necessary, using InstallAnywhere variables in the name field in the InstallAnywhere advanced designer.

7. *Browse to the* ***Organization > Install Sets*** *task.*

   A default InstallAnywhere project contains two Install Sets—Typical, and Minimal. We'll add a third called Documentation Only.

**NOTE:** The Customizer available for the Install task, allows you to set an image and description to appear on the "Choose Install Sets" panel. The Choose Install Sets panel presents your end user with large radio style buttons with a description of the installation option. At this stage, you should add descriptions of the Typical (All features) Minimal (application only) and Documentation Only (Only the documentation) Install Sets.

8. *Select Typical as the default by clicking in the check boxes to the right of the Install Set names in the panel above the customizer.*

   Now, we're ready to assign features to the Install Sets. In this simple example, we only have two features: Application and Documentation.

9. *Navigate to the* ***Organization > Features*** *task.*

   Notice that two default features—application and help—have been created as part of the default project. As these meet our needs, we'll not have to add any additional features; however the process by which they are added is identical to that used for Install Sets.

10. *Add a description for the features we'll use in this example.*

    Now, we'll assign the features to the Install Sets. We've defined three distinct sets; two have only one feature, and one has both of the features.

**NOTE:** The check boxes along the right hand side of the Install Tree within the Install task are used to assign files, folders, and actions to features within the task.

    a. *Using the checkboxes to the right of the panel, select* ***Typical*** *and* ***Minimal*** *for the* ***Application*** *feature.*
    b. *Select* ***Typical*** *and* ***Documentation*** *for the "Help" feature, making sure to leave out "Minimal."*

11. *Assign files to features in the* ***Install*** *task.*

    a. *Browse to the Install task.*
    b. *Assign the entire contents of the OfficeSuite2000 folder to the Application feature.*
    c. *Assign the entire contents of the ImagesAndDocs folder to the Help feature.*

    Now, we have all the files and actions assigned properly, and our organization is complete. However, we've still not presented a method to our end user by which they can alter the choice of Install Sets (in its current configuration, our installer will simply use the default install set).

12. *In order to offer our user a choice, we'll need to add a panel to the Pre-Install that will allow them to select.*

    a. *Browse back to the Pre-Install task, where we'll add the panel.*

       This panel is, of course, a built-in option in InstallAnywhere.

    b. *Open the Add Action palette, and from the Panels tab, choose Panel: Choose Install Sets.*

c. *Use the arrows or the drag and drop functionality to move the panel to a location after the introduction, but before the user selects an installation location.*

NOTE: The **Choose Install Sets** panel has several options that directly effect the options the user is presented with. The checkbox **Allow end-user to customize installer via the 'Choose Product Features' panel** will allow the user to choose individual features to install. While not strictly necessary in an installation as simple as our current example, we suggest engaging this option so that you may see the results.

13. *We're now ready to build our first "Organized Installer".*

   a. *Browse to the* **Build** *task.*

   b. *Build the installer.*

   The option to "Show "Product Features" without the "Choose Install Sets" panel" allows you to present only the features to your user, creating a highly flexible installer. For this example, do not engage this option.

   When running the installer, note the **Choose Product features** panel. Select the **Custom Install set** so that you can see the results of selecting the checkbox "Allow end-user to customize installer via the "Choose Product Features" panel."

## Adding Components

Components are managed in the **Organization > Components** task. Each component is created with a long **Name**, a **Short Name**, and a **Unique ID**. The **Version** and **Key File** may be specified. Key Files are single files that are a core part of the component which identify the component. A Key File should be a file that will always be included in a component.

## Assigning Files to Components

To assign files to a component, use the **Assign Files to Components** option in the **Install** task to assign files to the components.

## Removing Empty Components

Sometimes you may have components that are no longer needed or do not have any files assigned to them. To remove empty components from the project click **Clean Components** in the **Organization > Components** task.

## Integrating Components Already Installed on Target System

If the installer needs a component that should already be installed on the target system, use the Find Component in Registry action to locate that component. This action searches for the component by using the UUID, the unique identifier specified for the component. The component's location can then be used as an installation location by the installer.

The installer can compare versions and locate the highest version found. The key file may also be searched for.

## Adding Features

Features are managed in the **Organization > Features** task. You can add and remove Features, as well as assign Features to Install Sets.

## *Quick Quiz*

1. To which basic InstallAnywhere organization element are files assigned?
   A. Install Sets
   B. Features

2. In which InstallAnywhere Advanced Designer Task are Files assigned to features?
   A. The Features Task
   B. The Components Task
   C. The Install task

Answers: 1.B | 2.C

# Chapter 10: Introduction to Advanced Actions and Panel Actions

- **Actions**
- **Using Panels in Pre-Install**
- **Using Install Task Actions**
- **Quick Quiz**
- **Application Servers and Database Servers**

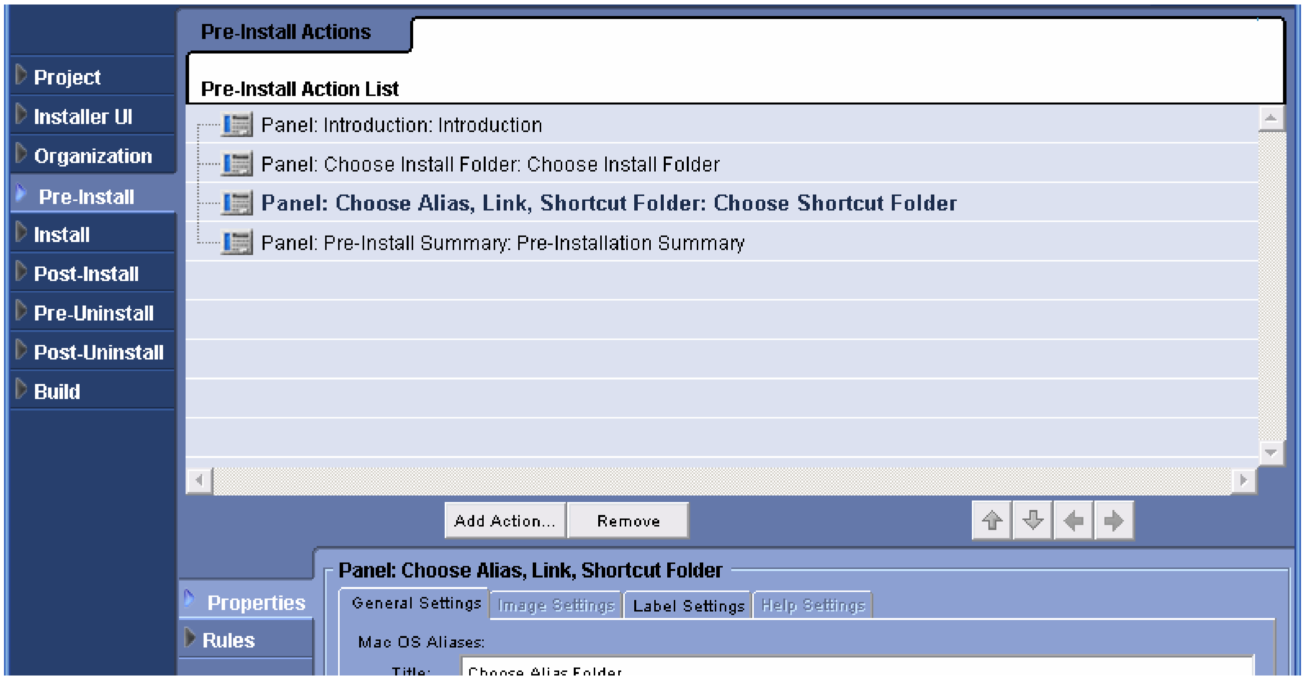
In this chapter, we'll cover the use of some of InstallAnywhere's more advanced and more useful installer *actions*.

InstallAnywhere actions represent operations that will be performed by the installer or uninstaller. Actions may be as simple as installing files or displaying a panel, or as sophisticated as executing custom code during the installation process.

## Actions

Actions can be added only in the Advanced Designer. Actions can be added to the tasks that represent real-time operations being accomplished by the installer: Pre-Install, Install, Post-Install, Pre-Uninstall, and Post-Uninstall.

Actions are executed in the order that they appear in the task, from top to bottom. Actions may be reordered either by use of the up and down arrows. The left and right arrows move actions out of or into folders. For example, the following figure shows the actions contained in the Pre-Install task.



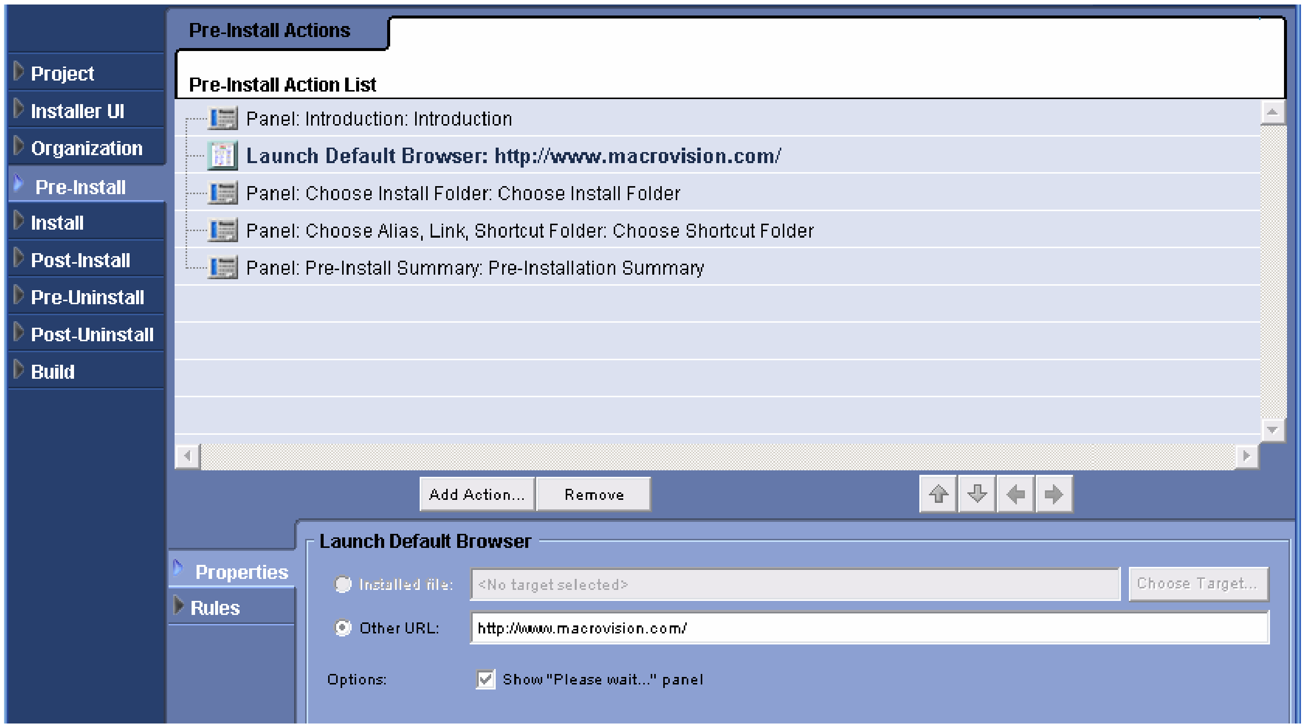
Panel: Introduction: Introduction
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

Panel: Choose Alias, Link, Shortcut Folder
Mac OS Aliases:
Title: Choose Alias Folder

To add an action, click Add Action. Depending on the current task, the list of actions you can add will be different.

Once an action has been added, its Customizer will appear in the bottom half of the Advanced Designer. Developers modify the action in the customizer. The Properties area enables developers to add file locations and variables to the action, as well as detail how developers would like the action to run.

For example, the Launch Default Browser action has a simple customizer: you can specify whether to launch an HTML file or external URL, and whether to display a “Please Wait” panel. The customizer appears as in the following panel.



Pre-Install Action List

- Panel: Introduction: Introduction
- Launch Default Browser: http://www.macrovision.com/
- Panel: Choose Install Folder: Choose Install Folder
- Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
- Panel: Pre-Install Summary: Pre-Installation Summary

Launch Default Browser

Installed file: <No target selected>

[✓ Other URL:] http://www.macrovision.com/

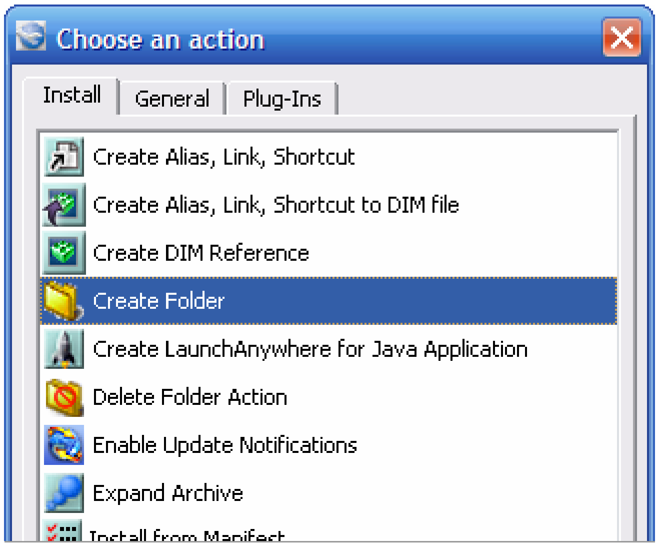
Options: [✓] Show "Please wait..." panel

The Rules area enables developers to add specific rules to an action. Once an action has a rule added to it, its icon will be modified to display a small R on it, so developers will be able to see that there is a rule associated with it. This icon badge may help test or modify an installer if there is a long list of actions. For more information about rules, see Chapter 8.

## *Action Availability by Task*

Some actions are available only to some tasks. The Choose an Action dialog box will present only actions which are appropriate for the task within which the developer is working. For example, Add Action from within the Install task will not provide any actions that require user input.

For the Install task, the Choose an Action dialog box will have a tab for install tasks and one for general tasks. Actions under the General tab are available from all tasks. The list of available Install actions appears similar to the following figure.


Choose an action
Install
General
Plug-Ins
Create Alias, Link, Shortcut
Create Alias, Link, Shortcut to DIM file
Create DIM Reference
Create Folder
Create LaunchAnywhere for Java Application
Delete Folder Action
Enable Update Notifications
Expand Archive
Install from Manifest

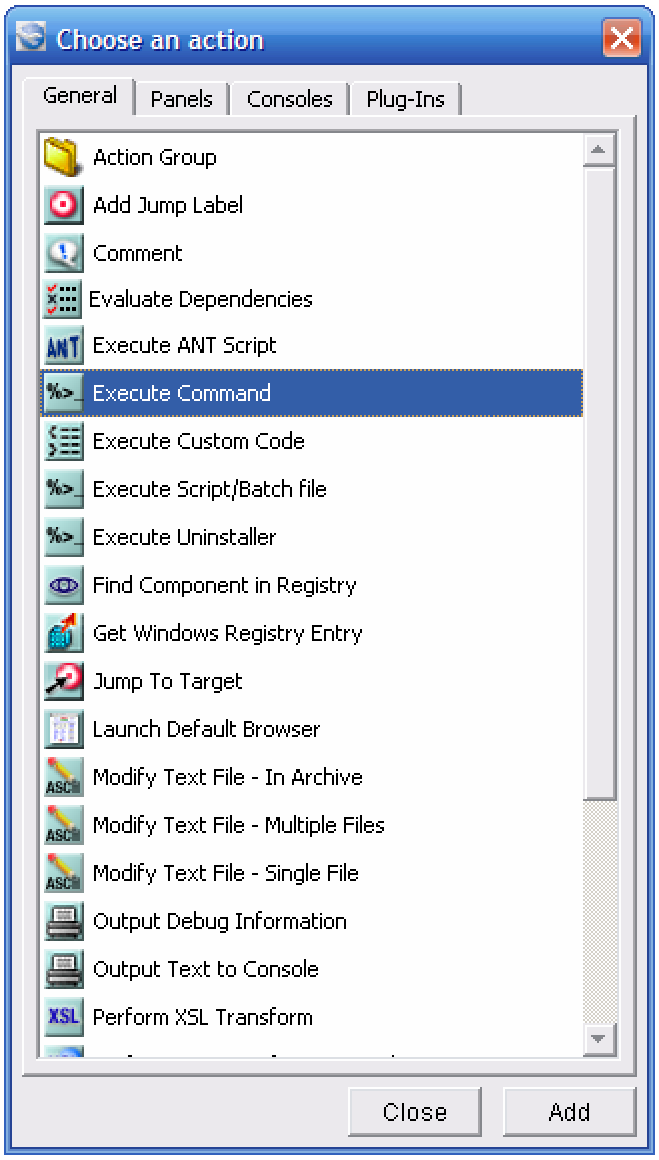
The **Pre and Post-Install** and **Uninstall** tasks have other actions listed under Panels, Consoles, and the Plug-Ins tabs.

NOTE: The Plug-ins tab will appear when a custom code plug-in has been added.

Not all InstallAnywhere actions are available in each stage of the installation. For example, very few actions pertaining to installed files are available in the pre-install section. We will examine some common actions in this section.

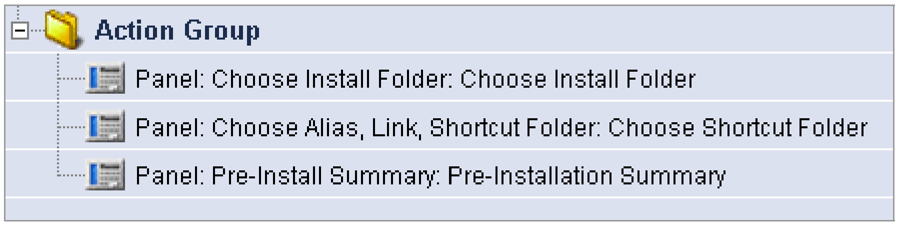
## *General Actions*

General Actions are actions that perform tasks related to installer performance, or act on your target system. The following figure shows some of the general actions: Execute Command, Modify Text File, and Perform XSL Transform.



## *Action Groups*

You can also add *action groups* to a task. Action groups enable you to group installer actions into logical groups—which can then be controlled by rules. This enables you to easily apply the same conditional rules to a large number of actions. For example, on a large multiplatform installer, it may be necessary to perform a particular set of actions only on specific platforms. These tasks can be combined into action groups, which can then be controlled by a single rule, or a single set of rules. In previous versions you would have had to apply those rules to each individual action.


Action Group
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

Action groups also enable you to organize your own work into logical sets. For example, you might combine tasks that represent a particular state of your installation into a single action group. Even if that group doesn't require any rules, you may still find it advantageous to have those actions grouped into a logical set.

Action groups are available only in the Pre-Install and Post-Install sections of the installer.

### Pre-Install/Uninstall and Post-Install/Uninstall Actions

Pre-Install/Uninstall actions are executed before Files actions which are executed before Post-Install/Uninstall actions. Pre-Install actions are generally actions that determine what to install on the target system, or even if the installation should occur. Actions added from the Install task define what will occur on the target system, like creating folders, expanding archives or moving files. Post-Install actions are generally actions that require files to have been installed; examples are showing the Readme file or launching the application that was just installed. Actions that occur within the Pre-Install or Post-Install tasks will not be uninstalled.

NOTE: When an Action is executed in Pre-Install or Post-Install, it may be executed more than once if an end user clicks Previous and then Next repeatedly. This could cause several errors for actions that modify files, such as Modify Text File - Single File or Register Windows Service. For these types of actions, it is recommended you execute them during the actual installation (within the Install task) to prevent this type of error.

The Pre-Install Task sets up those actions that will occur prior to the installation of files. In general, this is the portion of the installation in which most configuration options are offered. Most developers choose to require information be input at this stage, and automate later configurations based on that input. This enables your user to make one sets of entries near the beginning of a long install process, and then leave the installation relatively unattended during the actual install.

## Examples of Common Actions

This section describes the behavior and settings of some of the common actions you can add to the various project tasks. For a list of predefined actions, see Appendix C and the InstallAnywhere help library.

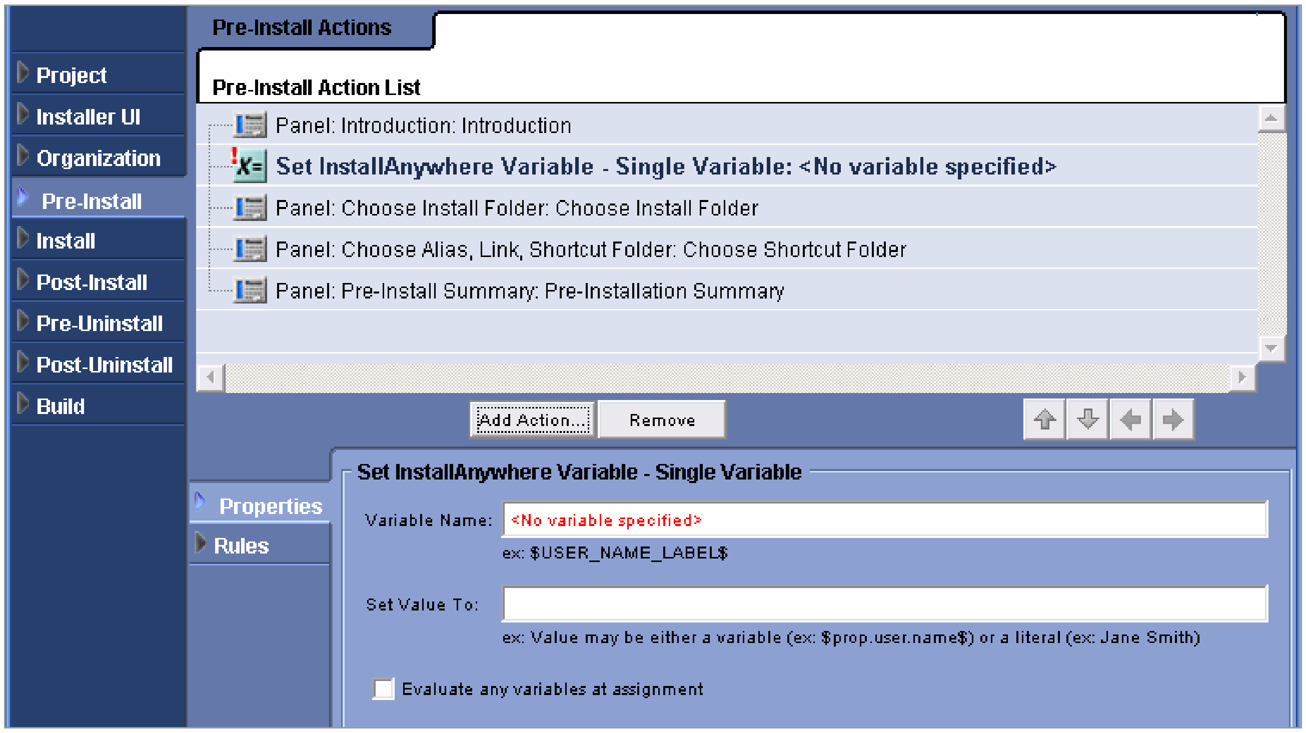
Some commonly used actions are the following:

- Set InstallAnywhere Variable
- Display Message
- Set System Environment Variable

### Set InstallAnywhere Variable Action

Most installation programs must set the values of InstallAnywhere variables. As described in Chapter 4, InstallAnywhere variables are used to communicate data between the various panels and actions that make up an installation. To create or define one InstallAnywhere variable, you can use the action called "Set InstallAnywhere Variable—Single Variable" (there is also a "Multiple Variables" variant).

For example, suppose you want to create a custom variable called MY_CUSTOM_VAR, and set its value to the hard-coded string "Custom data". (After having done this, you can use the expression $MY_CUSTOM_-VAR$ in an action to expand to its value at run time.) To begin, select the Pre-Install task in your installation, select the action you want to precede your action (such as the first action, typically the Introduction panel), and click Add Action. In the Choose an Action panel, select “Set InstallAnywhere Variable—Single Variable”, click Add, and click Close. The new action should appear as follows.



Panel: Introduction: Introduction
Set InstallAnywhere Variable - Single Variable: <No variable specified>
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

Set InstallAnywhere Variable - Single Variable

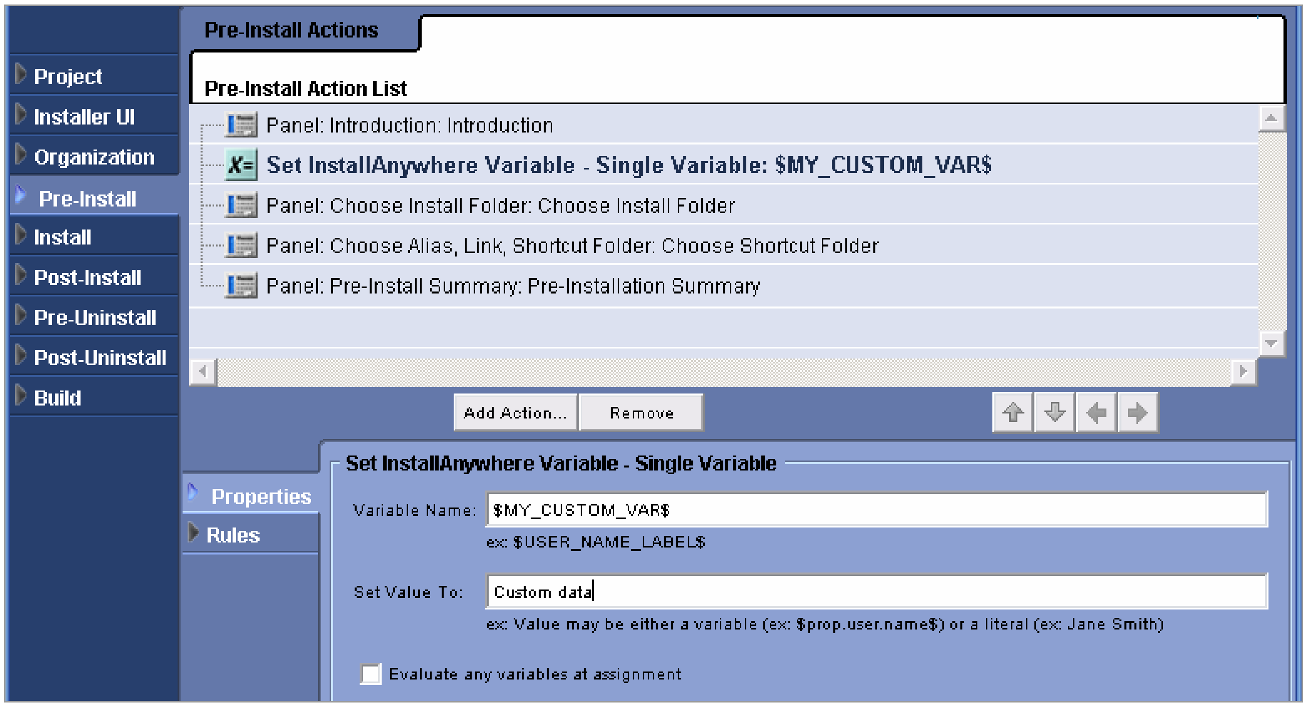
Variable Name: <No variable specified>
ex: $USER_NAME_LABEL$

Set Value To:
ex: Value may be either a variable (ex: $prop.user.name$) or a literal (ex: Jane Smith)

☐ Evaluate any variables at assignment

If necessary, you can move the action up and down in the list by clicking the arrow buttons. Note that the action’s icon displays a red exclamation point as an overlay, to indicate that the action is missing a required setting, in this case the variable name.

You specify the property name and initial value using the action’s Customizer at the bottom of the screen. For this example, enter **$MY_CUSTOM_VAR$** for the name, and enter **Custom data** for the value. (As described in the InstallAnywhere help library, you can base the value of a variable on another variable; you can also specify to evaluate the variable’s value immediately to avoid troubles with recursive property definitions.) The action should now appear as follows.



Pre-Install Action List

Panel: Introduction: Introduction
Set InstallAnywhere Variable - Single Variable: $MY_CUSTOM_VAR$
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

Set InstallAnywhere Variable - Single Variable

Variable Name: $MY_CUSTOM_VAR$
ex: $USER_NAME_LABEL$

Set Value To: Custom data
ex: Value may be either a variable (ex: $prop.user.name$) or a literal (ex: Jane Smith)

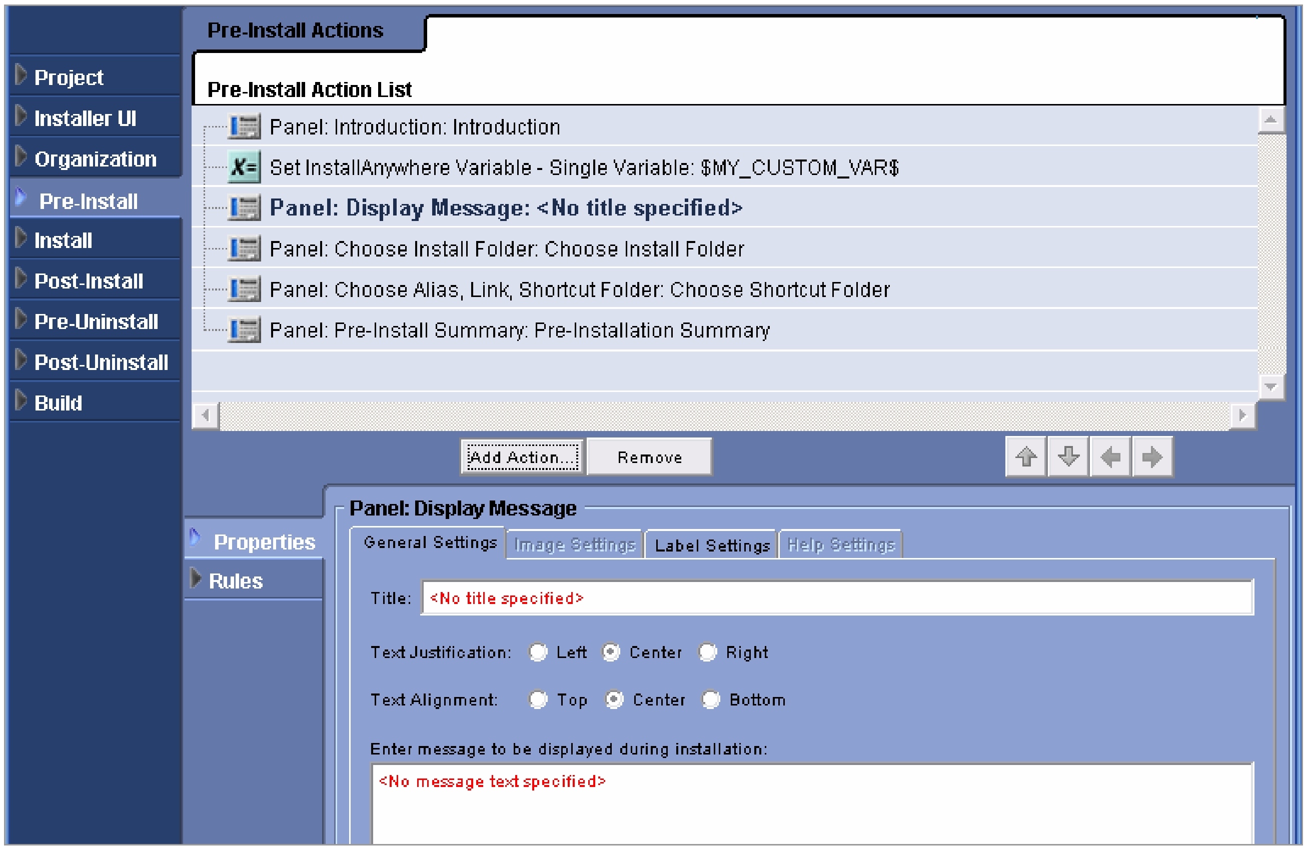
☐ Evaluate any variables at assignment

You can then use the expression $MY_CUSTOM_VAR$ in a later action, and the expression will be replaced with the variable's value.

## *Display Message Panel*

Another common requirement is to display an informative message to the end user at run time. To handle this requirement, InstallAnywhere provides a Display Message panel that you can add to your tasks.

For this example, you can create a panel that will display the value of the InstallAnywhere variable created in the previous example. To begin, select the Set InstallAnywhere Variable action (the action that is to precede the new panel), and click Add Action. In the Choose an Action panel, activate the Panels tab select "Panel: Display Message", and then click Add and Close. The new action should appear as follows.



Pre-Install Action List

- Panel: Introduction: Introduction
- Set InstallAnywhere Variable - Single Variable: $MY_CUSTOM_VAR$
- **Panel: Display Message: <No title specified>**
- Panel: Choose Install Folder: Choose Install Folder
- Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
- Panel: Pre-Install Summary: Pre-Installation Summary

Panel: Display Message

Title: <No title specified>

Text Justification: Left [✓ Center] Right

Text Alignment: Top [✓ Center] Bottom

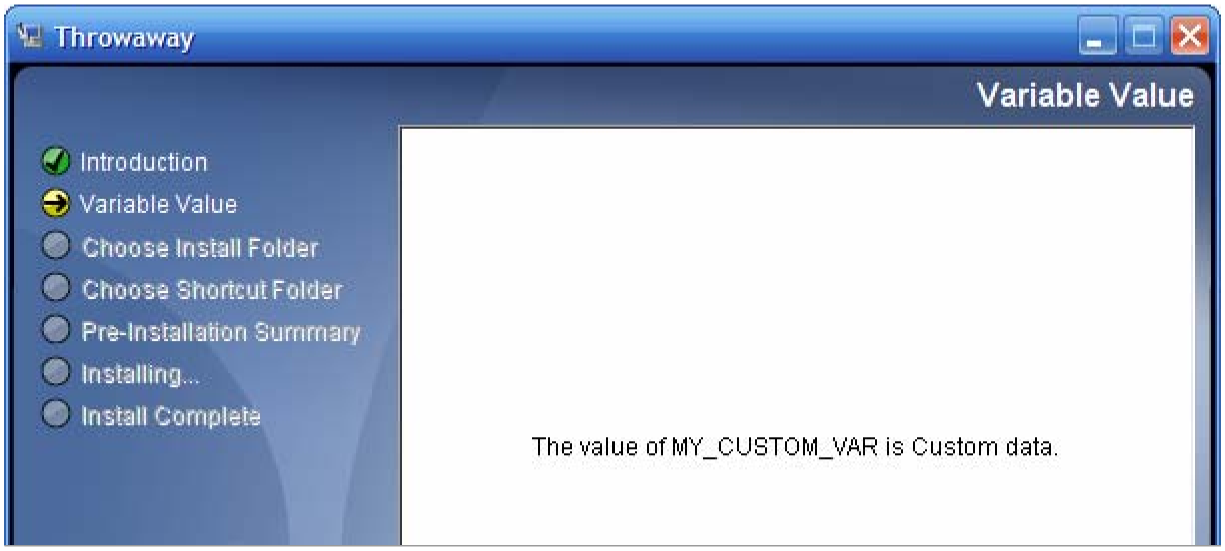
Enter message to be displayed during installation:

<No message text specified>

The Customizer for the new panel action accepts the panel title and message, along with text justification and alignment. For this example, enter the title **Variable Value**, and enter the message “The value of MY_CUSTOM_VAR is $MY_CUSTOM_VAR$.”

The special format $VAR$ expands to the variable’s value. To display a dollar sign without triggering variable expansion, you can use the predefined variable `$DOLLAR$`. See Appendix A for a complete list of predefined InstallAnywhere variables.

At run time, the panel would appear similar to the following.



Variable Value

The value of MY_CUSTOM_VAR is Custom data.

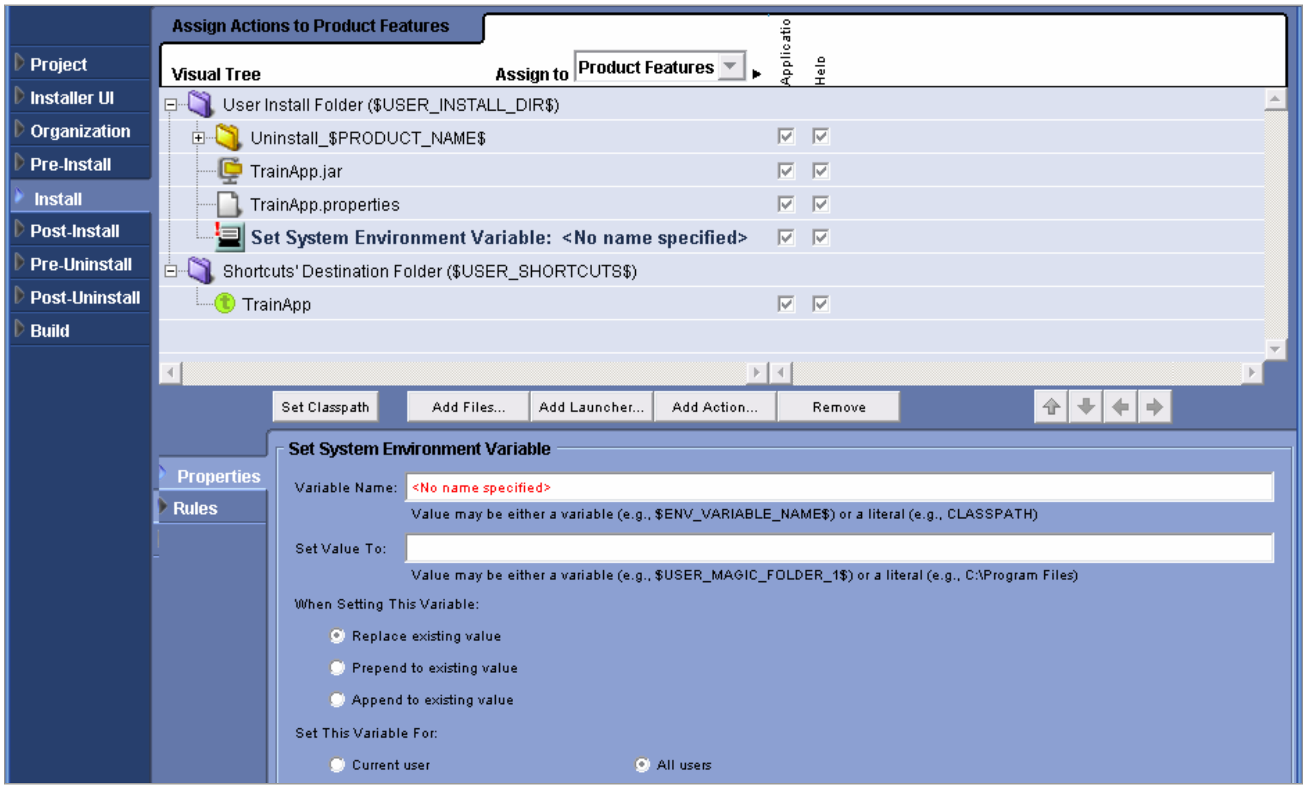
You can use the Label Settings tab of the panel's Customizer to specify the text that appears in the left-hand list of steps on the panel at run time.

This process of displaying the value of one or more variables in a Display Message panel is commonly used as a simple debugging technique.

## *Set System Environment Variable*

Setting the value of an environment variable on the target system is an example of an action that should take place during the Install task. To modify an environment variable on the target system, you can use the Set System Environment Variable action.

In this example, select the Install task, click Add Action, and select Set System Environment Variable, followed by clicking Add and Close. The empty action might appear similar to the following.



Set System Environment Variable

Variable Name: <No name specified>

Value may be either a variable (e.g., $ENV_VARIABLE_NAME$) or a literal (e.g., CLASSPATH)

Set Value To:

Value may be either a variable (e.g., $USER_MAGIC_FOLDER_1$) or a literal (e.g., C:\Program Files)

When Setting This Variable:

[✓ Replace existing value]

Prepend to existing value

Append to existing value

Set This Variable For:

Current user [✓ All users]

As with other actions, you use the action's Customizer to specify the action's settings. In this case, you specify the environment variable name (such as **TEST_PATH**), its value (such as **$MY_CUSTOM_VAR$**), and whether to replace any existing value or to add the new value to the beginning or end of an existing value. You can also specify whether the variable should be set for the current user or for all users.

The following exercises will give you additional practice with creating and defining actions.

## *Exercise 10.1 Using Panels in Pre-Install*

Add the following panels to your installation and rebuild to investigate their appearance and configuration options.

1. *Add a Choose File panel.*

This panel requests that the user select a file, either by typing in the full path name or by browsing. The panel may require the user to browse to select the file. You name the InstallAnywhere variable the panel returns by entering it into the Selected File field. The directory where the file is located will also be returned to the variable named by the Parent Folder field.

2. *Add a **Choose Folder** panel.*

   This panel requests that the user select a folder, much in the same way the **Choose File** panel works. The panel returns a variable you name in the **Selected Folder** field. You may also check to see if you have write permissions for the chosen folder.

3. *Add a **Choose Java VM** panel.*

   This panel searches for a Java VM on the target system. The panel may also prompt the user to install a VM.

4. *Add a **Find File/Folder** panel.*

   The **Find File/Folder** panel conducts searches on the target system, depending on a number of criteria which the developer may define. The InstallAnywhere variable the panel returns is named in the **Results Variable** field.

5. *Add a **Get Password** panel.*

   The **Get Password** panel requests a password from the user. The password may then be validated, compared against an index which enables different passwords to unlock different features, or saved in a variable.

NOTE: The **Get Password** panel can be configured either to simply store a masked entry or to confirm against a file. For this exercise, use the `password.txt` file in the `ImagesAndDocs` directory included with OfficeSuite.

6. *Add a Display Message panel to exhibit the results variables from each of your previous panels.*

7. *Build and run your installer.*

Console installs, and their associated actions will be covered later.

## Exercise 10.2 Using Install Task Actions

1. *Browse to the **Install** task.*

2. *Add an Install SpeedFolder action*

   a. *Click **Add Action**.*

   b. *Click the **Install** tab.*

   c. *Select **Install SpeedFolder**.*

   d. *Click **Add**.*

3. *Add a **Set System Environment Variable** action.*

| | |
|---|---|
| Variable Name: | `$PRODUCT_NAME$_DIR` |
| Set Value to: | `$USER_INSTALL_DIR$` |
| When setting this Variable: | `Append to existing value` |
| Set this Variable for: | `Current user` |

4. *Add a **Set Windows Registry - Single Entry**.*

   a. *Click the **General** tab.*

   b. *Select **Set Windows Registry - Single Entry**.*

   c. *In the customizer, define the registry entry:*

| | |
|---|---|
| Comment: | `Set Windows Registry Test` |
| Registry Key: | `-HKEY_LOCAL_MACHINE\SOFTWARE\$PRODUCT_NAME$` |
| Value Name: | `InstallDirectory` |
| Data Type: | `String` |

| | |
|---|---|
| Data: | `$USER_INSTALL_DIR$` |
| Uninstall Options: | `Remove if value has not changed` |

5. *Add **Show Message Dialog** action. Display the following text.*

| | |
|---|---|
| Variable Name is | `$PRODUCT_NAME$_DIR` |
| Value is | `$USER_INSTALL_DIR$` |

6. *Add an **Execute Script/Batch File** action. Enter the following commands.*

```
@echo off
echo enter script
mkdir $DESKTOP$$/$TestDir
mkdir $DESKTOP$$/$TestDir
```

7. *Add another **Show Message Dialog** action. This time display the following text.*

```
STDOUT: $EXECUTE_STDOUT$
STDERR: $EXECUTE_STDERR$
EXITCODE: $EXECUTE_EXITCODE$
```

8. *Rebuild and run your installer.*

   Note the files installed by the SpeedFolder, the changes made to the system registry, and environment variable created. The changes on the target system should reflect all of the changes you have made to the project.

9. *To add an action to an installer, click Add Action.*

   The Add Action button is present wherever Actions are available. Clicking Add Action will display the Choose an Action dialog box.

## *Quick Quiz*

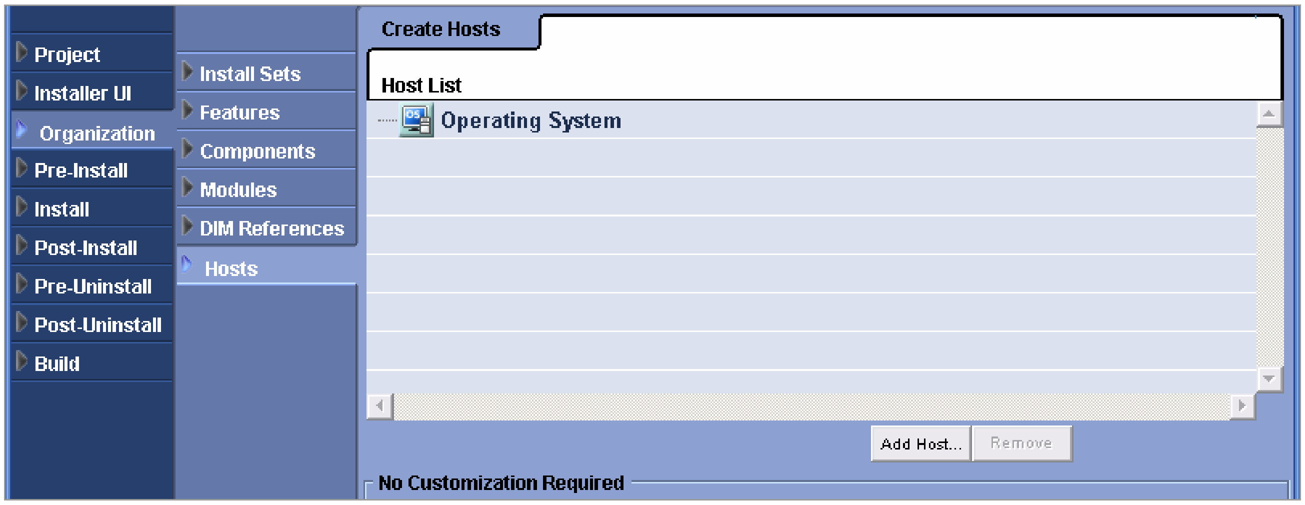
1. Expand Archive Action is available in:
   A. Pre-Install
   B. Post-Install
   C. Install
   D. All of the above
2. Where would you place an action that requires an uninstall equivalent?
   A. Pre-Install
   B. Post-Install
   C. Install
   D. All of the above

Answers: 1.C | 2.C

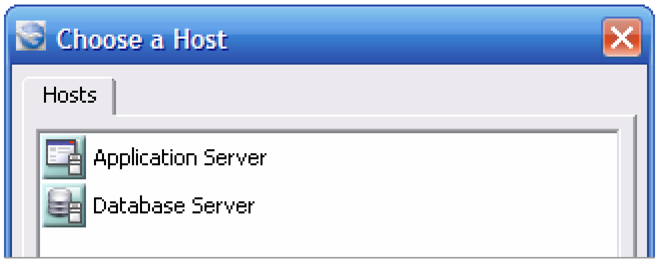
## *Application Servers and Database Servers*

InstallAnywhere 2008 Enterprise Edition introduces support for Application Server and Database Server hosts, to which you can respectively assign WAR/EAR deployment actions and SQL script actions.

In order to use the application server and database server functionality, you must define an associated *host*. The hosts defined by your project are located in the Organization > Hosts task. By default, every project contains an Operating System host, representing the target to which files, links, and other system changes are made.

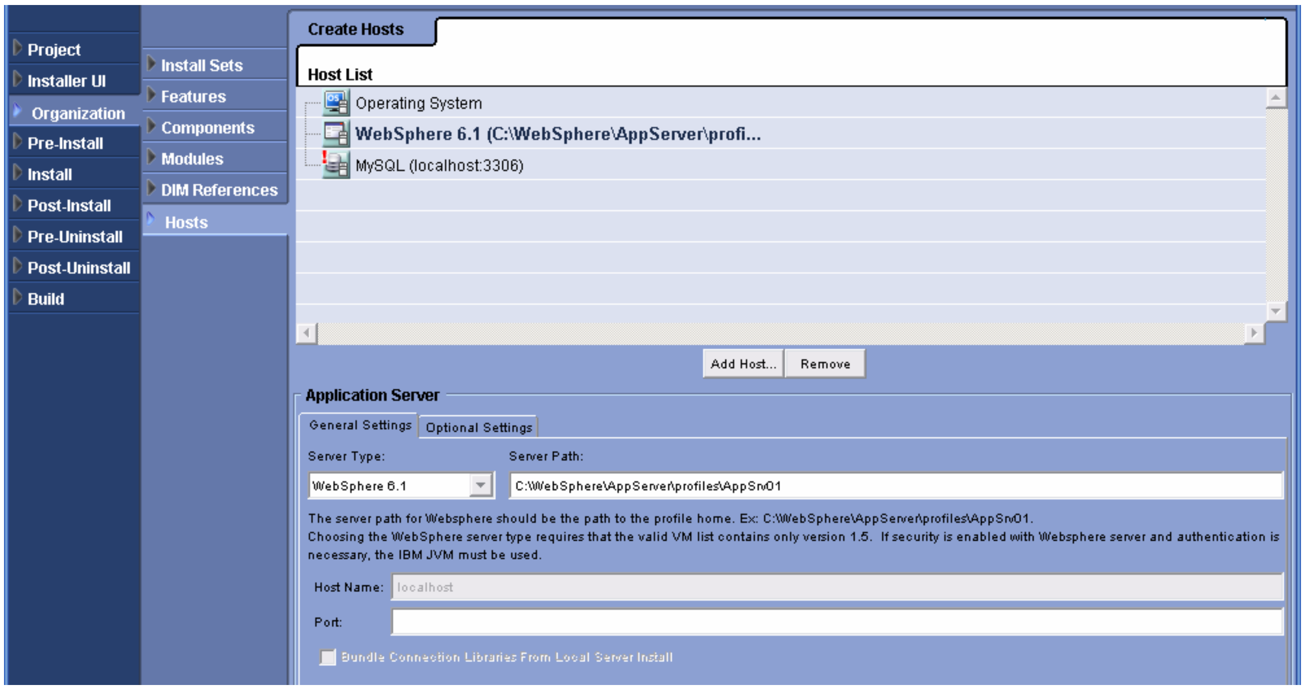


To add a host to your project, click Add Host and select the desired host type in the Choose a Host panel.



After you have added a host, you set its properties in the customizer view while the host is selected. For example, the following figure shows the customizer for an Application Server host. Note that one of the general settings for an Application Server host is the server type (WebSphere, Tomcat, and so forth), and different server types can have different general and optional settings.

Moreover, if any host is missing a required setting or is improperly configured, its icon in the Host List column displays a red exclamation point as an overlay. See the InstallAnywhere help library for detailed information about the various host properties.



Host List

- Operating System
- WebSphere 6.1 (C:\WebSphere\AppServer\profi...
- MySQL (localhost:3306)

Application Server

Server Type: WebSphere 6.1

Server Path: C:\WebSphere\AppServer\profiles\AppSrv01

The server path for Websphere should be the path to the profile home. Ex: C:\WebSphere\AppServer\profiles\AppSrv01.
Choosing the WebSphere server type requires that the valid VM list contains only version 1.5. If security is enabled with Websphere server and authentication is necessary, the IBM JVM must be used.

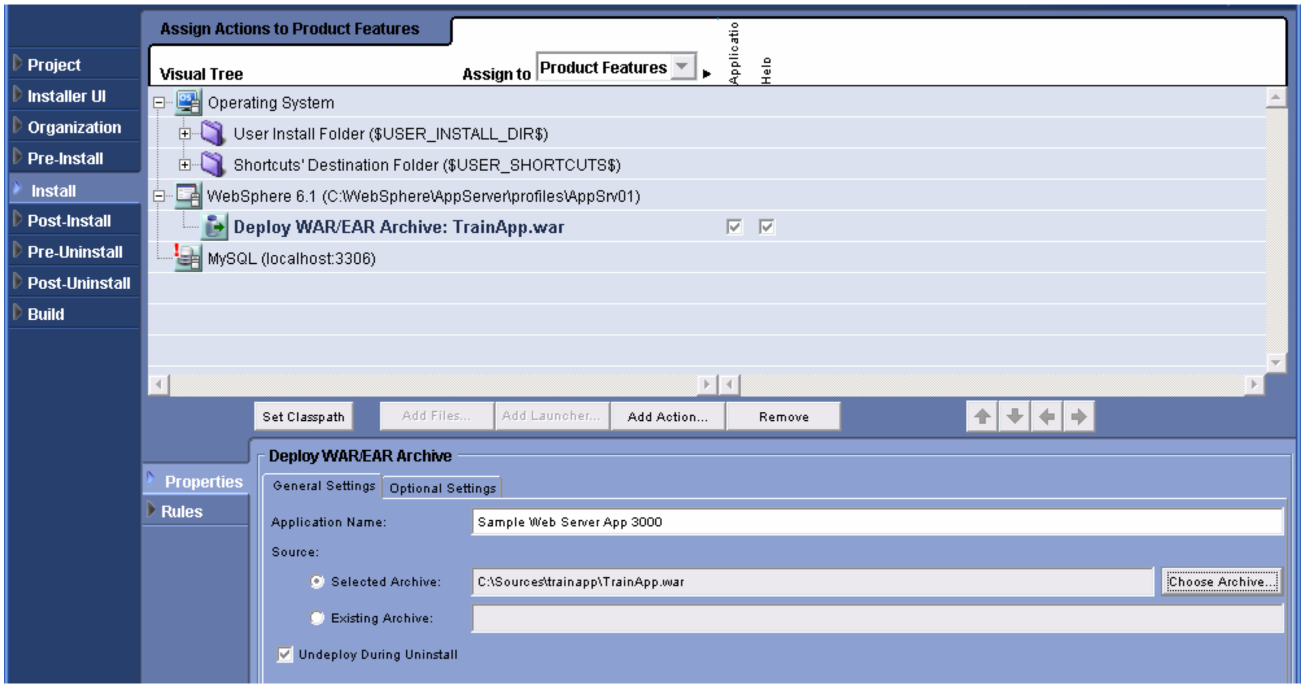
Host Name: localhost

Port:

☐ Bundle Connection Libraries From Local Server Install

When working with Database Server hosts, note that not every JDBC driver ships with InstallAnywhere. For information about obtaining any of the JDBC drivers that do not ship with InstallAnywhere, see Macrovision Knowledge Base article **Q113500**.

After you have added the desired host types, you can add a Deploy WAR/EAR Archive action to an Application Service host, or add a Run SQL Script action to a Database Server host. To add these actions, begin by opening the Install task of your project; the new hosts should appear in the Visual Tree, along with the standard Operating System host.



Visual Tree

- Operating System
  - User Install Folder ($USER_INSTALL_DIR$)
  - Shortcuts' Destination Folder ($USER_SHORTCUTS$)
- WebSphere 6.1 (C:\WebSphere\AppServer\profiles\AppSrv01)
  - Deploy WAR/EAR Archive: TrainApp.war ✓ ✓
- MySQL (localhost:3306)

Deploy WAR/EAR Archive

Application Name: Sample Web Server App 3000

Source:

● Selected Archive: C:\Sources\trainapp\TrainApp.war

○ Existing Archive:

☑ Undeploy During Uninstall

To add the action, select the desired host and click Add Action. For an Application Server host, the Deploy WAR/EAR Archive action appears in the Choose an Action panel, and for a Database Server host, the Run SQL Script action appears in the Choose an Action panel.

As with any type of action, you specify the action's general and optional settings in the action's customizer. For example, the previous figure shows the customizer for a particular Application Server target, where you specify the application name, the application source archive, and whether to undeploy the application when the user uninstalls your product. For details of any action's settings, see the InstallAnywhere help library.

## Common Properties

The following list contains common properties found in action customizers in InstallAnywhere.

| Property | Description |
|---|---|
| Comment | Sets the name of the action in the visual tree. |
| Do not uninstall | Tells an action to not attempt to undo the results of the action at uninstall time. |
| If file already exists on end user's system | Overrides the default behavior for how to resolve conflicts between installed files and pre-existing files. |
| In Classpath | Puts the item on the classpath for all LaunchAnywhere executables installed. |
| Installed File/Existing File | Determines whether the file is being installed, or already exists on the end user's system. |
| Override default Unix/Mac OS X permissions | Sets the file permissions to a specific value for this action. |
| Path | Shows the path where the action will be installed. |
| Show Indeterminate Dialog | Brings up an indeterminate progress bar to show progress to the end user while a external process is executing. |
| Source | Shows the path where the item currently exists on the developer's system (displays the source path if source paths are being used). |
| Store process exit code in | Sets the value of the InstallAnywhere Variable to the process exit code. |
| Store process `stderr` in | Sets the value of the InstallAnywhere Variable to the process standard error. |
| Store process `stdout` in | Sets the value of the InstallAnywhere Variable to the process standard out. |
| Suspend installation until process completes | Pauses the installer until the launched process completes. |

## Panel Action Settings

Panel Actions (commonly called Panels) are the means for requesting user input through a graphical interface.

Graphic Installers may show the installation steps through a set of labels—words which represent the step. Installers may also display specific images for the steps. When Images is selected in the Installer UI > Look & Feel > Installer Panel Additions > Type of Additions to Installer Panels, the customizer for the panel in the Pre-Install and Post-Install task will enable the use of the Image Settings tab. If List of Installer Steps is selected the Label Settings tab will be enabled.

NOTE: These settings are unavailable to panel actions in the Uninstaller. Panel actions in the Uninstaller use the default values set in the Installer UI > Look & Feel task.

### Image Settings

Use panel image settings to choose a specific image to display on the chosen panel. Developers may choose to use the default panel image, display an image specific to that panel, or display no image at all.

### Label Settings

The Label Settings tab in the customizer enables developers to preview the labels and the icon images. The labels are highlighted, and marked as the installation progresses. The installer build process will auto populate the list based on the panel titles.

NOTE: Using the Installer UI > Look & Feel task's Installer Panel Additions tab and the Labels settings tab found on each individual panel's customizer, developers can assign multiple panels to the same label. Thus, if there are numerous steps, or if the installer has several panels for the same step the interface can be adjusted as needed.

To control label order, or to edit the content of the label, in the Installer UI > Look & Feel task's Installer Panel Additions tab use the Arrows and other control buttons found to the left of the list of panels.

### Help

Selecting Enable installer help in the Installer UI > Help subtask provides a Help feature for the installer program.

Selecting HTML enables greater formatting control of the help text. (You cannot apply HTML tags to the title.) To format the help text, use the HTML formatting tags. For example,

```
<B>MyHelp</B> <I>Information</I>
```

causes InstallAnywhere to display **MyHelp** *Information*.

Developers may either set a single help message, which they can define in this window, or customize help for each installer screen. To customize help for each installer screen, select Use different help text for each panel. Add the customized help in the Help tab of the action customizer at the bottom of the Pre-Install, Post-Install tasks.

## *Additional Action Information*

### LaunchAnywhere

A LaunchAnywhere Executable is an executable file that is used to launch a Java application on any LaunchAnywhere-compatible platform (All Windows, Unix platforms, and Mac OS X). LaunchAnywhere enables end users to double-click an icon (Windows or Mac OS X) or type a single command (Unix) to start a Java application. The LAX is also in charge of configuring the Java application environment by setting the classpath, redirecting standard out and standard error, passing in system properties, environment variables, and command-line parameters, and many other options.

The launcher looks at a configuration file `<MyLauncherName>.lax` to determine how the launcher runs. This `lax` file is created during the installation, and is placed in the same location as the launcher.

A list of lax properties is located in Appendix I, “LAX Properties”.

## Manifest Files

Manifest files are text files which specify a list of files and directories. The manifest file has a certain format (listed below). The format specifies the file’s source, its destination (which is relative to the location of the action in the Visual Tree of the Install task), and optionally, which Unix file permissions it should have and if it should be placed on the classpath. At build time, this file is analyzed, and its contents are placed into the installer.

### Manifest File Format

For files:

```
F,[SOURCEPATH]relative_path_to_source_file,./relative_path_to_destination_file
F,absolute_path_to_source_file,./relative_path_to_destination_file
```

To put files on the classpath:

```
F,absolute_path_to_source_file,./relative_path_to_destination_file,cp
```

To set a file’s permissions on Unix:

```
F,[SOURCEPATH]relative_path_to_source_file,./relative_path_to_destination_file,755
```

For directories:

```
D,[SOURCEPATH]relative_path_to_source_dir[/],./relative_path_to_destination_dir[/]
D,absolute_path_to_source_dir[/],./relative_path_to_destination_dir[/]
```

Examples:

```
F,$IA_HOME$/path/to/source/file.txt,./destination/path/thisfile.txt
F,/absolute/path/to/source/file.txt,./destination/path/thisfile.txt,cp,655
D,$IA_HOME$/path/to/dir,./destination/path/dir
D,/absolute/path/to/dir,./destination/path/dir
```

# Chapter 11: Managing Installation Locations with Magic Folders

- **Magic Folders and InstallAnywhere Variables**
- **Magic Folders**

## Magic Folders and InstallAnywhere Variables

Magic Folders represent a specific location, such as the user selected installation directory, the desktop, or the location for library files. At install time, the installer determines which operating system it is running on, and sets the magic folders to the correct absolute paths. Many Magic Folders are platform-specific and many are predefined by InstallAnywhere to standard locations across InstallAnywhere-supported platforms.

Every Magic Folder has an associated InstallAnywhere variable. These variables are initialized when the installer begins. Changing the value of a Magic Folder variable will change the destination to which the Magic Folders installs. Changing the value of the `$USER_INSTALL_DIR$` through InstallAnywhere will change where the files will install.

With three exceptions, these variables are initialized at install time and will not change except through using custom code or the Set InstallAnywhere Variable action. The exceptions are:

- `$USER_INSTALL_DIR$`: This is initialized to the default value determined in the Platforms task in the Advanced Designer. Its value can change at the **Choose Install Folder** step, if the end user selects a different folder.
- `$USER_SHORTCUTS$`: This is initialized to the default value determined by the Platforms task in the Advanced Designer. Its value can change at the **Create Alias**, **Link**, **Shortcut Folder** install step if the end user selects a different location.
- `$JAVA_HOME$`: Installer without VM: Defaults to the value of the java property `java.home`. Its value can change at the Choose Java Virtual Machine step if the end user selects a VM Installer with VM: Defaults to the value specified in the **Project > Java** task. It can change when `$USER_INSTALL_DIR$` changes, or at the **Choose Java Virtual Machine** step if the end user selects a VM already on their machine.

NOTE: Variables cannot be set to themselves unless they are defined with the Evaluate at Assignment option. For variables defined without Evaluate at Assignment checked, you cannot set USER_MAGIC_FOLDER_1 = USER_MAGIC_FOLDER_1$/$test to append /test to USER_MAGIC_FOLDER_1. InstallAnywhere enables direct and indirect recursion only with InstallAnywhere variables that use Evaluate at Assignment. Otherwise, this condition causes an error.

InstallAnywhere defines a number of magic folders as defaults, mostly those representing common installation location. One of the keys to understanding magic folders is to understand that they resolve to different locations based on the configuration of the target system.

For example:

The Magic Folder System Drive Root (`$SYSTEM_DRIVE_ROOT$`) resolves to `/` on Unix, Linux, and Mac OS X, and to `C:\` (or the root of the drive containing the Windows directory).

Other Magic Folders are reserved, but are defined by actions in the installer, or actions taken by the end user. These magic folders are important to the operation of the installer.

One such Magic Folder is the “User Install Dir” (`$USER_INSTALL_DIR$`) the directory that represents the root of the installation as specified by the developer, or as chosen by the user. This directory structure is not defined by the installer, but is defined on a project-by-project basis.

Another such Magic Folder type is the “User Magic Folder.” While InstallAnywhere defines many installation paths via pre-defined Magic Folder, Macrovision cannot have accounted for all possible installation needs when implementing the technology. As such, the User Magic Folder is introduced. These variable based installation paths are designed to enable you to define a path manually, or using any of InstallAnywhere’s dynamic installation tools. You can create your own User Magic Folders by setting the associated variables, and then simply adding your resources to that magic folder in the install step.

## Exercise 11.1 Magic Folders

In this exercise, we’ll implement a new installer project that serves to demonstrate the use, and flexibility of the InstallAnywhere Magic Folders architecture.

1. *Open InstallAnywhere and create a new project in the Advanced Designer.*

   Yes, at least for the moment, OfficeSuite will be retired.

2. *On your desktop (or in a location where you can easily find them), create the following files (they need not have any content):*
   *`Desktop.txt`*
   *`SystemDriveRoot.txt`*
   *`Home.txt`*
   *`Temp.txt`*
   *`Programs.txt`*
   *`System.txt`*
   *`MagicOne.txt`*
   *`MagicTwo.txt`*

3. *Add all files you’ve just created to your installer project.*

4. *Browse to the* ***Install*** *task.*

   For each file, we’ll be choosing a magic folder that will result in the file being installed to a location that matches the file name. This will enable us to see the installation process in action, and to see how the magic folders resolve.

5. *Set Magic Folder destinations for each file.*

   a. *For the Desktop.txt file, highlight the file, and in the customizer, use the pull down menu, Path, to select Desktop Folder.*

   b. *Highlight SystemDriveRoot.txt and select System Drive Root.*

   c. *Highlight the Home.txt file and select Home Directory.*

   d. *Repeat, selecting the matching Destination Path name, until reaching the MagicOne and MagicTwo files. For these files, select USER_MAGIC_FOLDER_1 and USER_MAGIC_FOLDER_2 respectively.*

6. *Define the User Magic Folders that we’ve used.*

   As the folders must be defined before they're installed, we'll define them in the pre-install section of the installer.

   a. *Browse to the* ***Pre-Install*** *task.*

      As we're only exploring the magic folders in this installer, we can remove all the panels from the pre-install. This decision will enable our installer to run more quickly, and without interaction.

      Now, to define the User Magic Folders, we need to create the InstallAnywhere variables that define them.

   b. *Add a Set InstallAnywhere Variable action by clicking Add Action, then in the General tab, choose Set InstallAnywhere Variable - Single Variable.*

      This action will define the `$USER_MAGIC_FOLDER_1$` variable, and as such, define the location for the `MagicOne.tx`t file.

   c. *Select a location where you have write permissions, and enter that path as the value for the file.*

7. *Add another Set InstallAnywhere variable action which we'll use to define $USER_MAGIC_FOLDER_2$.*

   For this variable, try using one of the Magic folder variables to help define the path. The variables used in Magic Folders are described in the appendix. We suggest using something like `$SYSTEM_DRIVE_ROOT$$/$MagicFolder`. The `$/$` will resolve to the proper separator `/` or `\` depending on the system used. It is one of the standard InstallAnywhere variables, and we suggest using it in place of the system specific file separators.

8. *Build and run your installer.*

   After execution, you'll want to check each file location that you've specified with a magic folders to see were, and in fact if, the file was correctly installed. The `Desktop.txt` file should appear on your desktop. The other files will appear in the locations that have been specified. The only tricky one here is the `Home.txt` file. This should appear in the user's home directory, which is highly variable. On most Windows systems this will be in `C:\Documents and Settings\USERNAME` and on Unix and derivative systems `~$USER`.

For a complete list of predefined Magic Folders, see Appendix B, "InstallAnywhere-Provided Magic Folders", or the InstallAnywhere help library.

# Chapter 12: Applying Basic and Intermediate Development Concepts

- Concept Review
- Debugging InstallAnywhere Installers

This chapter will consist of a single unstructured exercise where we will utilize much of what we've discussed in sections up until this point. After completion of an installer project, we will test, and debug the installer utilizing InstallAnywhere built in debugging features.

## *Concept Review*

You will build a single installer utilizing each of the concepts, actions, and panels listed. The idea is to create an installer which you plan and create using a number of InstallAnywhere development concepts.

1. *Build an Installer with the following guidelines:*

| | |
|---|---|
| **Magic Folders** | Your installer should include the use of at least one magic folder other than the core `USER_INSTALL_DIR` and `SHORTCUTS`. |
| **LaunchAnywhere** | Your installer must contain at least one LaunchAnywhere Launcher. |
| **InstallAnywhere Variables** | Your installer should display an understanding of, and reasonable management of InstallAnywhere variables. |
| **InstallAnywhere Rules** | You should implement rules that control installer behavior, and install path options. |
| **Your installer should implement the following actions and panels** | • Panel: Choose Alias, Link, or Shortcut<br>• Panel: Choose File or Choose Folder<br>• Panel: Choose Install Folder<br>• Panel: Display Message<br>• Panel: Get Password<br>• Panel: Get User Input - Simple<br>• Panel: Install Failed<br>• Panel: Install Success<br>• Panel: Install Summary<br>• Panel: Introduction<br>• Panel: Show License Agreement |
| **Your installer should implement the following InstallAnywhere actions** | • Add Comment<br>• Create Alias, Link, or Shortcut<br>• Create LaunchAnywhere for Java Application<br>• Execute Command<br>• Execute Target File<br>• Get Windows Registry<br>• Install File<br>• Install Folder<br>• Install Uninstaller<br>• Output Text to Console<br>• Set InstallAnywhere Variable<br>• Set Windows Registry Entry - Single<br>• Show Message Dialog |

2. *When you have completed your installer project, test your installer. Does it meet your expectations?*

## *Debugging InstallAnywhere Installers*

Using the installer we've just constructed, we'll explore some of InstallAnywhere's built in debugging features. There are several methods available to debug InstallAnywhere installers. Deciding upon which method to use depends in part on the installer development cycle—during installer development or later if an end user has a problem with the installer.

Most InstallAnywhere installers utilize Macrovision's LaunchAnywhere technology. Along with many convenient features for end users (double clickable launchers, native-like user experience) LaunchAnywhere launchers provide a host of built in debugging features.

## During Installer Development

InstallAnywhere provides a project specific debugging feature that will enable you to create a debug file for each installer. To activate the project specific debugging feature:

1. *In the InstallAnywhere Advanced Designer, select **Project > Config**.*

   From here you'll see two fields within the Installer Debug Output section. In these fields, you can enter a path to the text file where the installer will place output when run. Both entries should point to the same file. These paths should be absolute, and can be managed using Java paths, rather than the system specific paths. This will enable you use one entry for multiple platforms.

2. *Set the output files:*

   | | |
   |---|---|
   | *Send stderr to:* | */tmp/outputfile.text* |
   | *Send stdout to:* | */tmp/outputfile.text* |

   These settings will direct the output to `/tmp/outputfile.txt` on a Unix or derivative system, and to `C:\tmp\outputfile.txt` on a Windows system. You'll need to make sure that the directory /tmp or c:\tmp exists on the target system in order for the output file to be created. The file itself (in this example `outputfile.text`) will contain most, if not all information needed to debug InstallAnywhere installations.

NOTE: Because these files will not be uninstalled, we recommend that this feature be deactivated prior to the final build of your product installer. However, some developers have chosen to leave the output intact to make debugging any issue that arises post-development easier.

## Post Development

Post Development suggestions should help when debugging a customer problem, or other post development issues.

### Debugging a Win32 installer

To view or capture the debug output from a Win32 installer, hold down the Ctrl key immediately after launching the installer and until a console window appears. Before exiting the installer, copy the console output to a text file for later review.

On some Windows ystems, run the installer once with the Ctrl key down, resetting the scroll back buffer for the console window, and then quit and run the installation again.

If there are problems capturing the console output, try a slightly more convoluted method. First, launch the installer and enable it to extract the necessary files. Once it reaches the "Preparing to Install…" window, when given the opportunity to choose a language or to go to the Windows "temp" directory, look for a temp folder which starts with an "I" followed by many digits (for example, "I1063988642"). Ensure it is the most recent directory, by sorting the directories by their "modified" date. Open the directory, and there should be a file called "sea_loc". Delete this file. Now go back to the installer, click OK, and at the first opportunity, cancel the installation.

Now go back to the directory inside the temp directory, where the file "sea_loc" was deleted. There should be another directory called "Windows"; open it. There should be an `.exe` file (most likely `install.exe`). There should also be another file with the same name except it will have a `.lax` extension. Open it with a plain text editor and edit the lines:

```
lax.stderr.redirect= AND lax.stdout.redirect= to be:
lax.stderr.redirect=output.txt AND lax.stdout.redirect=output.txt
```

After these changes have been made, save the file and launch the `.exe`. When the installation is complete there should be an `output.txt` file in the same directory as the `.lax` file. The `output.txt` file will contain the same information as that generated to the console.

## Debugging a Unix/Linux installer

To capture the debug output from the Unix command line developers need to perform the following: enter one of the following (based on which shell) at the command line prior to executing the installer:

`export LAX_DEBUG=true` or `setenv LAX_DEBUG true` or `LAX_DEBUG=true` or set `LAX_DEBUG` or whatever would be appropriate for the Unix shell.

Then run the installer. This will redirect the debug output to the console window you are currently in, and this output will help debug the installer.

If you would like to redirect the output to a file, you'll need to set the `LAX_DEBUG=file`. Once you launch the installer, a file called `jx.log`, containing debug output will be generated in the directory containing the installer.

## Debugging a Mac OS X Installer:

InstallAnywhere utilizes the standard output layers in Mac OS X to display output. To gather debugging output from an OS X installer, launch `console.app`. This output is found in `/Applications/Utilities`. To retain this information, cut and paste information from the console window to a file.

## Debugging a Pure Java Installer

There are two methods to debug the Pure Java, or other platforms installers.

- Place a file named `ia_debug` (lowercase) in the same directory as the JAR, which contains the installer. Placing this file will not direct the output into this file, but its existence will redirect the output to the console.
- Set the General Settings to create output prior to building the installer.
- Set the output in **Project > Config** as described above.

## Debugging LaunchAnywhere Launched Executables:

Since InstallAnywhere installers use LaunchAnywhere executables, the above procedures are also useful for debugging installed applications that make use of the LaunchAnywhere Java launcher technology. Generally, however, it's quite simple to alter the LAX file to enable the launcher to always generate output. This behavior can then be changed upon qualification and final release.

To generate debug output:

1. *In the InstallAnywhere Advanced Designer, highlight the launcher.*

2. *Click the **Edit Properties** button.*
3. *Alter the values for the following variables:*

   ```
   lax.stderr.redirect=   AND   lax.stdout.redirect= to be:
   lax.stderr.redirect=output.txt  AND  lax.stdout.redirect=output.txt
   ```
4. *After a normal installation, edit the .lax file as described in the preceding instructions.*
5. *This procedure will have to be repeated for each installation.*

**NOTE:** For Unix, set `LAX_DEBUG=true`. For Mac OS X, open the `Console.app`

## Reviewing Debug Information

InstallAnywhere debug output will generally appear as in the following sample (truncated):

```
InstallAnywhere 2008 Enterprise Build
Wed Dec 10 17:13:04 PST 2008

Current Total Java heap = 24575 kB
Current Free Java heap = 22581 kB

No arguments.

java.class.path = C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\resource C:\Program
Files\Macrovision\InstallAnywhere 2008 Enterprise\resource\swingall.jar C:\Program Files\Macrovision\InstallAnywhere
2008 Enterprise\resource\compiler.zip C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\IAClasses.zip
C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\lax.jar C:\Program Files\Macrovision\InstallAnywhere
2008 enterprise\jre\lib\rt.jar

ZGUtil.CLASS_PATH = C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\resource C:\Program
Files\Macrovision\InstallAnywhere 2008 Enterprise\resource\swingall.jar
C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\resource\compiler.zip
C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\IAClasses.zip C:\Program
Files\Macrovision\InstallAnywhere 2008 Enterprise\lax.jar


java.version  = 1.5.0
java.vendor  = Sun Microsystems Inc.
java.home  = C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\jre
java.class.version = 49.0
```

The debug information will show vital information such as the VM in use, the VM version, the locale, the system architecture, OS, and other features.

## Using Output Debug Information Actions

InstallAnywhere offers an Output Debug Information action. This action outputs information stored by, or available to the installer. This information can be either directed to the standard output, or can be directed to a file for later review. While all the options available in Output Debug Information have useful purposes, the most useful for troubleshooting are the Print InstallAnywhere Variables and Print Java Properties options. Both of these options enable easy access to those variables most often used in rules formulation. If developers are experiencing errors related to rules (or, for example, an action that should occur is not occurring), use the Output Debug Information to verify the values that InstallAnywhere has perceived for each of the variables and Java properties used by the InstallAnywhere rules.

## Debugging Using Display Message Panel

Often, it’s desirable to debug some portions of an installation during installer development. One simple feature of InstallAnywhere Enterprise Edition enables developers to add a display message panel that can display specific InstallAnywhere Variable values.

For example:

Add a rule that states: `Install Only If $prop.os.name$=Solaris`

The install continues, and the action assigned this rule does not execute.

So, add a Display message panel. The message is: `The prop.os.name is: $prop.os.name$`

When the install is run, the value of `prop.os.name` is SunOS, not Solaris, the rule can be reformulated to match the proper name.

# Chapter 13: Advanced Installer Concepts

- **Console Installers**
- **Building a Console-Enabled Installer**
- **Silent Installers**
- **Building a Silent-Mode installer**

In a contemporary information systems environment, an end user is likely to find a number of heterogeneous systems. They'll be working with different platforms, different operating environments, and different interaction environments. InstallAnywhere does its best to help you meet the needs of these end users by presenting installer options that can run in any number of environments.

In enterprise level environments it's not uncommon for end users to install applications to servers and other remote systems. In these cases, a rich Graphical User Interface such as that provided by InstallAnywhere's standard installer modes is not always desirable. You may find that your end users will a need command line interface mode installer, or even silent installations that require no end-user interaction.

In the InstallAnywhere Enterprise Edition, you'll find support for both Console and Silent mode installations. Console mode provides your end user with a text only interface similar to that found in ANSI terminal applications. Silent mode provides an automated non-interactive installation mode which can either run entirely from the defaults set by you, use intelligent logic to determine installation parameters, or read configuration information from a simple response file.

These modes provide enormous flexibility in your installation, enabling you to give end users a choice of the mode that best meets their needs.

## Console Installers

Console mode installers enable your end users a non-graphical user interface structured to enable interaction through text only. InstallAnywhere does not automatically provide console alternatives for panels you have added to your installer. You must provide consoles for each panel that you have included.

Console mode is intended to add support for non-graphical environments such as those common on so called "headless" Unix systems.

In general, InstallAnywhere console actions provide parity with panels provided in the graphical modes. In the next exercise, we'll discuss building console installers, and use some of the consoles to reproduce one of our earlier installers.

Console mode mimics the default GUI steps provided by InstallAnywhere, and uses standard input and output. The biggest advantage to console mode is that Unix developers no longer need X Windows (X11) to run their installers.

Console mode enables text to be output to the console line-by-line. It does not support any formatting, clearing of the screen, or positioning of the cursor.

```
+---------------------------------------------------------+
| CHOOSE ALIAS, LINK, SHORTCUT FOLDER                    |
|                                                        |
| Where would you like to create application shortcuts?  |
|                                                        |
| 1) In the Start Menu                                   |
| 2) On the Desktop                                      |
| 3) Don't create shortcuts                              |
|--------------------------------------------------------|
| Please make a selection [1, 2, or 3], and then         |
| press ENTER.                                           |
|--------------------------------------------------------|
```

To trigger a console installer from the command line, type the following command:

```
installername -i console
```

**NOTE:** To enable console mode for your Windows installer, choose the Console Launcher option for Install Launcher Type in Project > Platforms > Windows.

## *Exercise 13.1 Building a Console-Enabled Installer (the return of OfficeSuite)*

In this exercise we'll build a console-enabled version of our OfficeSuite installer.

1. *Open InstallAnywhere and create a new project, OfficeSuiteConsole.*
2. *Open the InstallAnywhere Advanced Designer.*
3. *Setup the project similar to the previous OfficeSuite installer.*
   a. *Add the OfficeSuite2000 and ImagesAndDocs directories from the OfficeSuiteSourceFiles directory within your InstallAnywhere installation.*
   b. *Set up your features, and Install Sets so that the user is presented with at least two, and preferably three, installation options.*
   c. *Add the necessary panels in pre-install to enable the users choice of installation options (Choose Install Sets)*
   d. *Add launchers and setup the classpath for office suite.*
4. *Setup the installer to enable the user to use Console Mode. In* Installer UI > Look & Feel > General UI Settings*, enable Console in the Allowable UI Modes section.*
5. *Switch to the Pre-Install task. Click Add Action, and select the Consoles tab. Add the following console actions:*

   Console: Introduction
   Console: Choose Install Sets
   Console: Choose Install Folder
   Console: Choose Link Folder
   Console: Pre-Install Summary
   Console: Ready to Install

   NOTE: While you can insert the consoles anywhere in the install, as long as their order represents what you would like, it's generally best to insert them paired with their graphical equivalent. This helps to keep your flow and organization even.

6. *Browse to the Post-Install Section and insert the following consoles.*
   Console: Install Complete
   Console: Install Failed
7. *Add rules to the Install Complete and Install Failed actions so that the appropriate action will display based on the value of the InstallAnywhere variable $INSTALL_SUCCESS$.*
8. *Rebuild your installer, choosing Linux as one of your target platforms. The instructor will provide you with address logon information to a Linux system where you will test the installer.*
9. *Ftp the installer to your test system. Run the installer using the following command:*

   ```
   sh ./[installername].bin -i console
   ```

   The `–i` option tells the installer to default to the mode specified. In most cases, this is necessary, as InstallAnywhere will make an attempt to attach to a graphical environment unless otherwise specified. You can however set default modes by removing the option for an installer to run in graphical mode.

## Silent Installers

Silent mode, which enables an installer to run without any user interaction, is fully supported on all Unix platforms. A near-silent mode is possible on Windows, and Mac OS X. InstallAnywhere and end-user-defined variables may be set through command-line parameters and/or a properties file.

To trigger a silent installer from the command line, type the following command:

```
installername -i silent
```

You may also call a properties file from the command line:

```
installername -f <properties file>
```

You may use the direct or the relative path to the properties file.

NOTE: InstallAnywhere variables may be incorporated in these values, and they will be resolved at install time.

### Using Response Files and Silent Installers

Silent mode is an InstallAnywhere UI mode that is useful for enterprise class systems. In silent mode, InstallAnywhere has no end-user interaction, and runs either on the defaults provided by the developer, or by providing a response file from which the installer retrieves the values for various InstallAnywhere variables used to control the install.

InstallAnywhere can automatically record end-user installer choices in a response file. To record a response file, run the installer from the command line, specifying `-r`, or click the **Always Generate Response File** checkbox in the **Project > Info** subtask. When the installer runs, it records end-user choices in a file named `installer.properties` in the same directory as the installer.

InstallAnywhere installers automatically check the directory in which they reside for a file called `installer.properties` or `[installername].properties`. You can also indicate a properties file for the installer to use by specifying the following command-line switch.

```
-f /path/to/properties file
```

This file utilizes a simple `key=value` format.

For example, the console installer we've previously built has effectively one real option—the installation directory.

The properties file might look like:

```
INSTALLER_UI=silent
USER_INSTALL_DIR=[select directory]
```

The `INSTALLER_UI` variable enables you to specify the installer mode in the properties file, negating the need to use the `-i` silent command line switch.

## Exercise 13.2 Building a Silent-Mode installer

1. *Open InstallAnywhere, and create a new project.*
2. *Set up the project*
   a. *Create your install files, and launcher.*
   b. *Set the classpath.*

3. *In **Installer UI > Look & Feel > General UI Settings**, enable Silent in the Allowable UI Modes section.*
4. *Build your installer.*
5. *Create a Properties File installer.properties with the following contents:*
   *INSTALLER_UI=silent*
   *USER_INSTALL_DIR=[select directory]*
6. *Place your properties file in the same directory as your executable.*
7. *Run the installer.*
8. *Verify the output results.*

   Did the installer install where you expected?

**NOTE:** For more information on silent installers, see Silent and Console Installers in the InstallAnywhere Help Library.

# Chapter 14: Uninstaller Issues

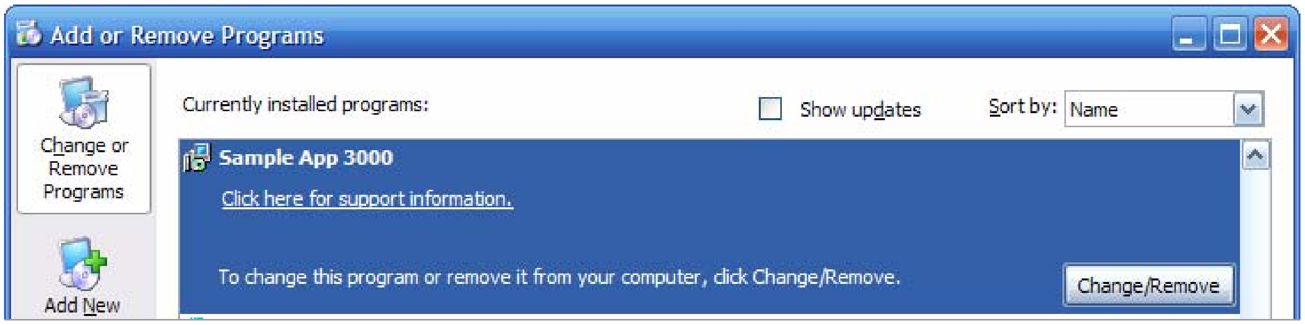
- About the Uninstaller
- Feature Uninstall
- Uninstaller for Multiple Products

As important as properly designing your installer is designing your uninstaller. Most simple projects will not require much uninstaller customization. However, if you are installing multiple projects, using merge modules, or installing server applications, you may wish to add additional functionality to your uninstaller. In InstallAnywhere 2008, you can now customize the uninstaller in the same way you can customize the installer.

## About the Uninstaller

InstallAnywhere automatically creates an uninstaller for the project which can be removed manually. The InstallAnywhere uninstaller removes all files and actions that occur during the Install task of the installation. Actions added in other phases of the installation cannot be removed using the uninstaller, and should be accounted for in the install phase.

On Windows platforms, InstallAnywhere automatically creates an Add or Remove Programs entry.



Sample App 3000

Click here for support information.

To change this program or remove it from your computer, click Change/Remove.

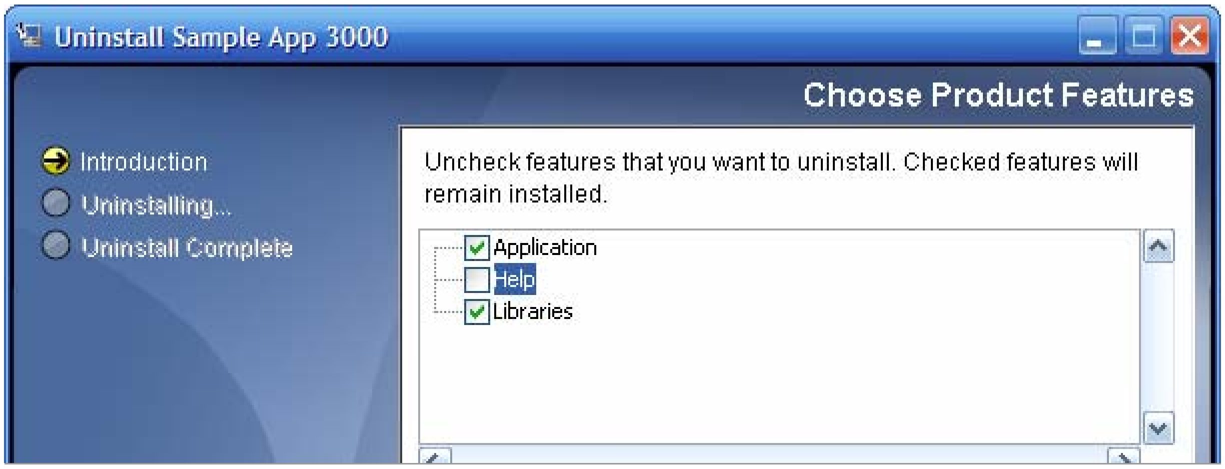
To disable this entry, you can set the predefined `$REGISTER_UNINSTALLER_WINDOWS$` variable to “false”.

The uninstaller is much like the installer. It is a collection of panels, consoles, and actions. It keeps track of what the installer has done, and contains a record of every action run during install time. All Pre-Uninstall panels, actions, and consoles run first. Then the uninstall functionality of actions in the Install task are called. Lastly, Post-Uninstall actions are run.

## Feature Uninstall

Each installer project has one uninstaller. All features are registered with the uninstaller through a local registry. If the Choose Feature panel is included in the uninstaller, the user will be offered the option to uninstall only certain features.

There are two options for controlling the behavior of a feature-level uninstall. The default behavior, illustrated in the following figure, is that installed features appear checked at uninstall time, and that clearing a feature’s check box causes it to be uninstalled.


Uninstall Sample App 3000

Choose Product Features

Uncheck features that you want to uninstall. Checked features will remain installed.

- [✓] Application
- [ ] Help
- [✓] Libraries

In the settings for the Choose Features to Uninstall panel, you can specify to use the opposite behavior, that installed features are displayed unselected, and selecting a feature causes it to be uninstalled.

A feature-level uninstall enables end users to choose specific features to uninstall. If an end user opts to uninstall one feature that has a shared component with a feature they were not planning to uninstall, the uninstaller recognizes this conflict and does not uninstall the shared component.

## *Uninstaller for Multiple Products*

Each uninstaller is tied to a certain product. It does this through its Product ID (found in the Project > Description task). In order to have one uninstaller function for a group of separate products, it is necessary that every project have the same Product ID. Each separate product should then be "demoted" to a feature. The uninstallers for these separate projects must also share the same uninstaller name and installer location. An example:

| | |
|---|---|
| **Product:** | Acme Office Suite (ID: 97338341-1ec9-11b2-90e2-a43171489d33) |
| **Installer 1:** | Acme Word Processor and Acme Spreadsheet |
| **Product ID:** | 97338341-1ec9-11b2-90e2-a43171489d33 |
| **Features:** | Acme Word Processor and Acme Spreadsheet |
| **Installer 2:** | Acme Slide Show |
| **Product ID:** | 97338341-1ec9-11b2-90e2-a43171489d33 |
| **Features:** | Acme Slide Show |
| **End Result:** | One uninstaller for all 3 "Features" |

# Chapter 15: Source and Resource Management in InstallAnywhere

- How Source Paths Work
- Managing Source Files
- The Resource Manager
- Creating Source Paths
- Quick Quiz

In today's software development environment, it is very rare that a developer is working in a vacuum, alone in a dark room, coding away at a project that he or she will be responsible for from the planning stages to the customer desktop. Development occurs in tightly integrated teams, often in parallel. A core team tweaks the product while those responsible for release and deployment work at creating the deployment packages.

Source Paths enable developers to reference file resources using variable paths, instead of absolute paths. This enables you to share a project file with other team members, even when the file resources are located at different paths on their development systems. With Source Paths, you can even use the same project file on different types of operating systems. For example, you can share a project between Unix and Windows.

## How Source Paths Work

Source Paths will automatically be substituted for the most complete path possible. For example, if you have two Source Paths defined

`$SOURCEPATH1$ = D:\temp\dir\foo`

`$SOURCEPATH2$ = D:\temp`

Then when you add a file such as `D:\temp\dir\foo\hello.txt` the file will be referenced by `$SOURCEPATH1$/hello.txt`, since `$SOURCEPATH1$` has the most complete path match available.

If a team member opens this project and the Source Path is not defined, a dialog will appear asking to locate and redefine the Source Path.

There are several predefined Source Paths that exist in any project. They cannot be changed or edited. They are:

`$IA_HOME$` is the location on the system where InstallAnywhere is running. A common location might be `C:\Program Files\InstallAnywhere`.

`$IA_PROJECT$` is the location on the system where the InstallAnywhere project is located.

`$USER_HOME$` is the User Home folder.

### Adding Source Paths

Source paths can be added through the use of Source Path variables, in one of two different ways.

1. *InstallAnywhere Preferences.*
    a. *Choose* **Edit > Preferences***.*
    b. *Select the* **Source Paths** *tab.*
    c. *Click* **Add***.*
    d. *Type the Source Path Variable name, such as RESOURCE, in the table (Do not type the dollar sign ($) around the path name. It will be added automatically when it is used.*
    e. *Click under Folder in the table. A button will appear labeled Choose Folder. Click this button to select the target location for the Source Path Variable (for example, c:\resources\test.txt).*
2. *Set the System Environment Variables.*
    a. *Access the environment variables. On Windows, right-click on My Computer on the Desktop, choose properties, choose Advanced, and click Environment Variables. On Unix or Mac OS X, modify the proper shell configuration file or set the variable directly using the shell.*

b. *Add an environment variable for the source path, pre-pended with IA_PATH_ tag.*

For example, to set the source path `SOURCE_PATH`, set the environment variable `IA_PATH_SOURCE_PATH`.

### *Updating the Location of Files and Resources*

When the location of a file or folder has changed, simply change the folder location listed for the source path. By changing the source path to the new location, InstallAnywhere will update the references to the resources automatically.

If you open a project and InstallAnywhere cannot find the resources, either because they have moved or they no longer exist, you will be asked to locate the resource or remove the resource from the project. If you want to open a project without updating the location of the resource, change the Project Loading preference found on the General Settings tab after setting Edit > Preferences to Never. Projects will be opened without checking the location of each resource. Instead, resources will be checked only when you build.

With this in mind, InstallAnywhere introduced features designed to help you manage resources shared between developers—from file resources included in the installer, to project files themselves, to individual component packages that can be merged into a single larger, or suite installer.

## *Managing Source Files*

Normally, InstallAnywhere uses absolute paths to reference your products included files and other resources added to your installer. This means that by default, your installer must be built with all files in the same location as when they were added to the project. This is enforced by a component of the InstallAnywhere Designer called the InstallAnywhere Resource Manager.

## *The Resource Manager*

The Resource Manager helps you keep track of files that are needed for your installation, and will prompt you to find those files, or remove them from the project if they are missing. In its default mode, the Resource Manager checks to see if necessary files are present when projects are loaded, saved, or built; however, this behavior can be altered in the Preferences. Open Edit > Preferences or click the About InstallAnywhere Button on the initial screens, then select the Preferences button from that screen. This will take you to the InstallAnywhere Preferences control panel. From this panel you can manage many of the features of InstallAnywhere, including the behavior of the Resource Manager.

Resource Manger settings can be found on the General Settings tab within the Preferences Panel. The two settings here that affect Resource Manager behavior are:

| | |
|---|---|
| **Project Loading** | Check “always” or “never” to determine if the Resource Manager should check for specified resources when the project is loaded. If never is selected, the InstallAnywhere advanced designer will enable you to work with a project file regardless of whether resources specified in the project are present at their specified locations. |
| **Command Line Builds** | This option affects the way that the Resource Manager will treat resources missing when a command-line build is executed. By default, the build will fail, requiring you to add or return the resources to their specified locations. Selecting Continue without the missing files will enable the installer to build without the files. |

## *Adding Source Path Management Capability to Your Installer Project*

InstallAnywhere enables team development while working on installers. Use this feature to share an installer project across your entire development team, working with common source control management (SCM) tools. Instead of having to map the entire project to one machine, InstallAnywhere leverages Source Path Management variables so that developers can work on the same project file in a disparate computing environment.

Source Paths resolve to the most complete value:

If `$TEMP$` = D:\temp\dir\fred

And `$TMP$` = D:\temp,

The file `D:\temp\dir\fred\hello.txt` becomes `$TEMP$\hello.txt and not $TMP$\dir\fred\hello.txt`

### Enabling/Disabling Source Paths

To enable or disable any of these Source Paths, click **Edit > Preferences > Source Paths** and select or clear the option you would like.

### Default Source Paths

There are three default source paths that exist in any project. They cannot be changed or edited. They are:

- `$IA_HOME$` is the location on your system where you are running InstallAnywhere. A common location might be `C:\Program Files\InstallAnywhere`.
- `$IA_PROJECT$` is the location on your system where your InstallAnywhere project is located.
- `$USER_HOME$` is the User Home directory on all platforms.

### Adding Source Paths

Source paths can be added through the use of variables.

1. *The Edit menu.*
   a. *Choose* ***Edit > Preferences*** *and then select the Source Paths tab.*
   b. *Click* ***Add****.*
   c. *Type the Access Path Name, such as* `RESOURCE`*, in the textbox.*
   d. *Type the Folder in the space allotted (for example,* `c:\resources\test.txt`*).*

   **NOTE:** Do not type dollar signs (`$`) around source paths when you add them into Preferences.

2. *Modify the PathManager.properties file.*
   a. *Go to* `InstallAnywhere`*\resource*`\preferences` *and open the* `PathManager.properties` *file.*
   b. *Add any number of source paths and precede each one with* `IA_PATH`*; the resource example would look like* `$IA_PATH_RESOURCE$`*.*
3. *Set System Environment Variables.*
   a. *Access your environment variables. On Windows, right-click on* ***My Computer*** *on the Desktop, choose* ***Properties****, choose* ***Advanced****, and click* ***Environment Variables****.*

b. *Add source paths. These are stored in the same format as Source Paths created in the Preferences area of InstallAnywhere.*

### Using Source Paths in Your Project

Go to the Install task and add the file `test.txt` that is in your resources directory.

### Switching Access Path Locations

When the location of a file or folder is changed, simply change the location where you have listed your source paths. By pointing your source path to the new location, your installer will update its resources automatically.

## *Exercise 15.1 Creating Source Paths*

Add the following user-defined Source Path, and enable the following default Source Paths.

1. *OFFICE_SOURCE (user-defined): point this Source Path to the following directory.*
   *<InstallAnywhere root>/OfficeSuiteSourceFiles*
2. *IA_PROJECT_DIR (predefined): this will point to the directory in which your project file resides.*

### *Managing Resources in the InstallAnywhere Project File*

While not recommended as a Best Practice, it is possible to manage resources directly from the project file itself.

InstallAnywhere versions newer than InstallAnywhere 5.0 utilize a new XML project file format (previous versions of the software utilized a binary format based on a Java class file).

As the `.xml_iap` file is essentially a plain text file, it is possible to manipulate it directly. The file can be managed using simple search and replace techniques, or using more advanced functions such as sed or awk scripts, or even an Extensible Style Language Transform.

## Quick Quiz

1. Which of the following are valid ways to set Source Paths?
   A. The Edit Menu (in the Advanced Designer)
   B. Modifying the `PathManager.properties` file
   C. Set System Environment Variables
   D. All of the above

2. What must be prepended to Environment Variables to make it a valid Source Path?
   A. `IA_SOURCE_`
   B. `IA_PATH_`
   C. `SOURCE_PATH_`

Answers: 1.D | 2.B

# Chapter 16: Advanced Interface Options

- **Installer Panel Additions**
- **Creating Installer Logic Using Jump Labels and Actions**
- **Quick Quiz**

In earlier segments of the course, we discussed some basic interface customizations. In this chapter, we'll discuss advanced modifications to both the flow of the installer interface, and changes to the graphical interface itself.

## *Installer Panel Additions*

InstallAnywhere enables you to modify the appearance of the installer panels to convey information to your user, or to present a branded image for your product (or both, should you so desire).

We've already discussed customizing the background and splash screen images for an installer. You can also customize the appearance of the area on the left hand side of an installer panel. This area can contain: a consistent image, an image that differs for each panel, a list of installer steps that serves as a progress indicator, or a combination of images and labels.

### *Panel Images*

In Project > Look & Feel > Installer Panel Additions, you can customize the appearance of the left side panel. You can choose the type of addition whether images or a list of installer steps. You can also opt to include borders and background images.

### *Panel Labels*

This task also enables you to specify the order of labels that will appear, and the icon image (such as an arrow) that will appear beside them. These labels are highlighted, and marked as the installation progresses. You can enable the installer build process to auto-populate the list based on the panel titles you've entered, or you can manually assign panels to a label in the settings found in each panel's customizer. To control label order, or to edit the content of the label, use the arrows and other control buttons found to the left of the list of panels.

NOTE: Using the Project > Look & Feel > Installer Panel Additions tab and the Labels settings tab found on each individual panel's customizer, you can assign multiple panels to the same label. Thus, if you have numerous steps, or if your installer may have several panels for the same step you can tweak the interface to meet your needs.

### *Advanced Installer Interface Actions*

In previous sections we covered the use and implementations of the Get User Input Panel, and mentioned the Advanced Get User Input Panel. Here, we'll discuss the features of the Advanced Get User Input Panel in more depth.

The Get User Input-Advanced panel, unlike its little brother, Get User Input-Simple, enables you to create panels that include more than one type of data, and that store that information in more than one variable.

This enables you, for example, to create a panel that collects registration information:

- Text Fields for: First Name, Last Name, Company, Street, Address, and Zip/Postal Code
- Pop Up Menu for: State/Province and Country
- Check boxes for: Preferred Contact Method

The Advanced Get User Input Panel interface is complex. Each element is added based on the type of element, then configured individually. If the elements added require more space than the graphic panel allows, the panel will add a scrollbar. You can use the Preview button to see the layout of your fields.

### *Using Jump Actions and Logic*

Within a specific task, InstallAnywhere enables you to “jump” back and forth based on a combination of Jump Actions, Jump Labels, and InstallAnywhere rules.

You can create conditions where end users must repeat a task, or meet certain criteria before they can progress with the installation. You can also enable end users to skip over a large portion of the configuration options if they merely want to accept all your defaults.

Jump actions are created by inserting Jump Labels at places in the install that you want to be able to jump to. These labels function much in the same was as HTML target or anchors. In the location where you would like the jump to occur, place a Jump Action. This action (you’ll need to add conditional rules to it or your end user will constantly be jumping) enables you to jump to the target specified.

## *Exercise 16.1 Creating Installer Logic Using Jump Labels and Actions*

In this exercise, we’ll use the advanced features that we’ve implemented so far in this segment.

Create an installer that does the following:

1. *Retrieves information from an end user. The input panel should use at least three input types.*
2. *Presents the information to the end user for verification. If they do not accept the information, returns them to the input panel.*
3. *Writes the collected information to a file.*
4. *Offers the end user the option to enter more information.*
5. *If they choose to enter more info, returns to the initial input panel.*
6. *Modify the labels so that your configuration process shares one label.*

## *Quick Quiz*

1. What happens if your Get User Input Panel fields exceed the allotted space?
   A. The Panel Scrolls
   B. The information splits into two panels
   C. The installer deletes everything it cannot display
2. Multiple labels can be assigned to the same action or panel. True or False?
   A. True
   B. False

Answers: 1.A | 2.B

# Chapter 17: Advanced Organizational Concepts

- Integrating Find Component in Registry Action
- Merge Modules and Templates
- Importing a Design Time Merge Module
- Creating Merge Modules
- Quick Quiz

## Integrating Find Component in Registry Action

If your installer uses a component already installed on your target system, or one that should already be installed on the target system, you can use InstallAnywhere's Find Component in Registry (this is referring to the InstallAnywhere Registry not the Windows Registry) action to locate that component. You can then utilize that component in your installation, or use that installation location as a path within your installation.

This action enables you to specify the component to discover using the UUID, the unique identifier specified for the component. You can then request that only the highest version be found, that the installer compares versions, or search for the key file. The action sorts the count of components found, the versions found, and the locations found in variables specified by you.

## Merge Modules and Templates

Merge Modules support the easy creation of suite installers, subinstallers, and templates, delivering reusability from project to project, within development teams, across the enterprise, or from third-party providers.

A suite installer is an installer for a suite of applications. Each application in a suite may be collected in a merge module, so that they may be easily added to a different mix of applications.

Templates are generally used as starting points for installer projects. Installer items that remain unchanged such as the license agreement panel would be saved in an InstallAnywhere template. You may also want to create a template to maintain the look and feel for your installer projects.

### Merge Modules

Merge Modules are essentially Installer sub-projects that can be created independently of one another and later merged together. Like an installer, a Merge Module is a reusable collection of installation functionality, complete with features, components, panels, actions, and files. However, a Merge Module cannot be installed on its own; instead, developers use Merge Modules when they want to include the functionality of one installer within another installer.

Merge Modules provide many benefits and provide solutions to complex installation requirements. For instance:

- Combine several Merge Modules from different products to create a "Suite Installer."
- Independent development teams in different locations can create Merge Modules for different software components. A release engineer can combine those Merge Modules into a single product installer.
- Create self-contained units of installer functionality for reuse in future installer projects. For instance, if the same software component needs to be in several different installers, build it into a Merge Module and make it available for all of the installer developers.
- Save common installer functionality, such as License Agreement panels and Custom User Input panels, into Merge Modules to simplify future installer project creation.
- Combine Merge Modules from third-party software packages to build complex software "Solutions," without having to figure out how to install each individual package.

Macrovision provides many third-party Merge Modules on its Web site. See http://www.macrovision.com/downloads.

- Use a Merge Module as the starting point for a new installer project. These Merge Modules are referred to as Templates, and are covered in another section.
- Any installer project can be built into a Merge Module. And any Merge Module can be used within any other installer project.
- Merge Modules are created as an option through the installer build process. Since a Merge Module contains all of the resources for a project, it is just like building an installer. They can be built automatically when the installer is built, or they can be explicitly built from the Advanced Designer (check the **Build Merge Module** option on the **Build** task, under the Distribution tab) or from the command line (use the +merge option).

Merge Modules can be merged into an existing installer in one of two ways:

- In the **Organization > Modules** task, click **Import Merge Module** to merge a merge module into the current installer. All of the merge module's features, components, files, actions, and panels (optionally) will be combined into the current project, enabling developers to further customize any settings.
- Merge Modules can also be installed as self-contained sub-installer units, without merging them into the current project. This is useful if developers do not know what will be in a Merge Module, or they will not be modifying any settings. Merge Modules added in this manner are run as silent sub-installers.

Merge modules can be integrated with a project in one of two ways:

- Use the **Install Merge Module** action and select **Bundle Merge Module at Build Time**, if the merge module is available when ready to build the installer. These Merge Modules will be included in the actual generated installer.
- Use **the Install Merge Module** action and select **Locate Merge Module at Install Time** to have the installer install a Merge Module that is available at install time, but external to the installer. The Merge Module can be either on the end user’s system or stored on a CD. If the location is a folder that contains several Merge Modules, they will all be installed.

Other important things to know about Merge Modules:

- **Read Only option**: Merge Modules can be locked, preventing them from being opened, used as templates, or being merged into an installer. Read Only Merge Modules can only be installed as a self-contained installer unit.
- **Optimize Merge Module Size by Platform option**: Separate merge modules will be created for each platform. Each will only contain the resources needed for that specific platform. Do not use this option if merge modules will be imported into another installer. Importing a merge module requires a non-optimized merge module.
- **Advertised Variables**: these are InstallAnywhere variables that will be necessary to set before a Merge Module can be installed using the **Install Merge Module** action. On the **Build** task, under the **Distribution** tab, click **Edit Advertised Variables** to add variables, set default values, and add comments. Use Advertised Variables to inform master installers of settings required for a Merge Modules configuration.

- InstallAnywhere Variables can be passed to the merge module when using the Install Merge Module action. Only selected variables will be passed to the merge module. By default, any Advertised Variable set by the Merge Module (Advertised Variables are set when the Module is built) will be automatically passed in. Specific variables can also be passed in through the customizer of the Merge Module. For example, if the Magic Folder variable `$IA_PROJECT_DIR$` was advertised by the Merge Module, it will be passed in. If the variable `$OTHER_VARIABLE$` was not advertised, but was set in the customizer of the Install Merge Module action, it, too would be passed in.

## *Templates*

A template is the starting point for every new installer project. A template can be a simple empty project, or it can contain everything a regular project would contain, such as license agreements, custom graphics and billboards, and even files.

A template is simply a Merge Module that has been placed within the `iatemplates` folder inside the InstallAnywhere installation folder. When you create a new project, you have the option of starting from a Template. When you start from a Template, a copy of the template is created and saved.

Templates are great for large installer teams, where you want everyone to have a consistent starting point, or for starting a new project based upon an older one.

## *Merge Module Types*

### Design Time Merge Modules

Use Design Time Merge Modules to integrate a Merge Module into your main installer project. Once a Merge Module has been imported, it is fully integrated into your master project file; all files, actions, and panels will appear as if they were a part of your main installer project. Merged projects may be added and removed; however, any changes that are made once they've been imported will be lost if you remove a merged project from the suite.

For example, if you were to import a Merge Module into your master installer and then modify a few panels, when you save the project, those changes will be saved. If you remove the imported Merge Module from the suite, all of these changes will be lost, regardless of how many times your main installer project is saved.

Design time Merge Modules display all panels you add to the project. They are the only Merge Module type that is not run silently.

NOTE: Only non-optimized Merge Modules may be imported as Design Time Merge Modules.

### Build Time Merge Modules

A Build Time Merge Module is a Merge Module that is included in your installer at build time, and installed by the master installer. Unlike Install Time Merge Modules, Build Time Merge Modules are included with the master installer when you build the installer project. At install time, the Master Installer will install the Build Time Merge Module. To specify that the Merge Module be Build Time, select the option labeled, Bundle Merge Module at Build Time, from the Install Merge Module action customizer and pick a Merge Module with the Choose Merge Module button.

Build Time Merge Modules are packaged along with the master installer project in one `.iap_xml` file. Build Time Suite Installers can build a number of separate installers into a single executable. In this scenario, a single master installer runs a number of Merge Modules silently during the installation process. The master installer is responsible for the user interface and for passing properties files to the Merge Modules so that they run with the correct configuration information. The Merge Modules may also advertise specific properties they require to operate properly.

### Install Time Merge Modules

An Install Time Merge Module is a Merge Module that is executed by the main installer at install time. Install Time Merge Module(s) are external to the main installer project. At install time, the master installer looks for the Merge Module at the specified path and launches whatever Merge Module it finds there. If the Install Time Merge Module path points to a directory then all Merge Modules contained in that directory will be installed. This enables Suite installers to be updated without having to update the master installer package

To specify that the Merge Module will be an install time Merge Module, select the Locate Merge Module at Install Time option from the Install Merge Module action customizer and then put a path in the text field next to it.

### Dynamic Merge Modules

In previous versions of InstallAnywhere, importing a merge module was a one time event—after the merge module was imported—the installer wouldn't look to the module for any changes, and any modification would have to take place within the parent project.

Merge Modules can be configured to be dynamic—meaning that the InstallAnywhere advanced designer will check for updates to the imported module at load and build time.

To use a dynamic Merge Module, you must first have built your sub component as a Merge Module, without the "Read Only" flag. Then, In the Organization > Modules task, click Import Dynamic Merge Module to merge the component into the current installer. All of the merge module's files, actions, and panels (optionally) will be combined into current project. The actions will appear as in an action group in Pre-Install and Post-install. Dynamic Merge Modules are recommended if you have merge modules whose contents constantly change. A parent project will automatically refresh dynamic merge modules when the parent project is loaded and built. This enables another group to continue parallel development on a Merge Module and its components, with those changes coming into the master installer automatically at build time.

NOTE: Merge modules cannot be authenticated. If you need authentication for a Merge Module, add it to your main installer project. Then, the Merge Module will inherit this setting during the installation.

## *Creating Merge Modules and Templates*

You create merge modules and templates much in the same way as regular installers are created. Add any panels, actions, and rules just as you would for a typical installer project, then before building in Build > Distribution > Merge Module/Template Option select Build Merge Module/Template. Once Merge Modules are built, they have almost the same contents as a regular installer project, except they don't contain an `IAClasses.zip` file or a launcher.

### Build Options

When you have an InstallAnywhere project ready to be made into a merge module, go to the Build > Distribution > Merge Module/Template Option. Select Build Merge Module/Template. Build options are used to optimize the size of the merge module, define whether it is to be read only, and edit the advertised variables for the merge module.

### Merge Module Size

Merge modules will contain an approximation of their required size (based on the largest amount of space any given module could need), but you may override this size with your own calculation by setting an advertised variable. The variable `$DISK_SPACE_REQUIRED$` may be set to override the automatic approximation.

### Creating Merge Modules as Read Only

You have the option of designating Merge Modules as “read only”. This option protects the integrity of the Merge Modules; the only way it can be added to an installer project is through the Install Merge Module action. This type of module cannot be integrated with the main installer project using the Project > Modules task.

### Advertised Variables

Advertised Variables are a list of all variables in a Merge Module installer project. The main installer project can pass InstallAnywhere variables to a Merge Module at install time. The Merge Module will then use these variables as regular InstallAnywhere variables.

To add your own variables to be passed to a Merge Module, go to the customizer for the Install Merge Module action in the designer, and click Edit Variables to add some variables.

There may be cases however when the Merge Module cannot run without already having had some variables set. Those variables should be “advertised” so the Merge Module can be configured easily. To do this, click Edit Advertised Variables on the build settings tab and setup your variables.

Now, if you go to the main installer project, add a Merge Module, and look at the settings for variables that will be passed to the Merge Module, you will see that the names of advertised variables have been added to your list and some may have been set to default values. Change the values as you’d like, and these will be passed to the Merge Module at install time.

## *Adding Advertised Variables*

To add advertised variables, go to the Build task and then select Build Settings. Click Edit Advertised Variables. The Edit Advertised Variables dialog box appears.

NOTE: In the Edit Advertised Variables dialog box, list all variables to be set in the installer project.

The Variable Name is the same as it was defined for the Merge Module. The Value is the corresponding value in the main installer project. For example, if you included the common Magic Folder `$IA_PROJECT_DIR$` in your Merge Module, and you wanted that to correspond with the value of `$IA_PROJECT_DIR$` in the main installer project, the Variable name and the Value would both be `$IA_PROJECT_DIR$`.

NOTE: Variables in InstallAnywhere must be expressed with dollar signs ($) on either side.

## *Adding Merge Modules*

To add a Merge Module, go to the Install task and click Add Action. Select Install Merge Module and click Add. For information on the Merge Module action customizer, see the Actions section below.

## Importing a Design Time Merge Module

To import a Merge Module:

1. *Choose **Organization** > **Modules** and then click **Import Merge Module**.*
2. *Navigate to a <productname>.iam_zip file, select it, and click **Open**.*

   The Import Settings dialog box appears.
3. *Select the tasks you'd like to be imported. You can import just the Install task panels and actions (the default choice) or any combination of Pre-Install, Install, and Post-Install.*

   NOTE: Merging Pre-Install actions will result in duplicate actions. We recommend that you do not merge the **Pre-Install** task for this reason, or if you do that you take care to clean up any duplicate actions.
4. *Choose how to import the Product Features of this Merge Module.*

   If you choose to import each Feature as a top-level Feature, each feature will be given equal weight hierarchically. If you choose to import each Feature as the child of a new Feature, a new feature will be created, titled <MergeModuleName>, with all of the Merge Module features appearing below it hierarchically.
5. *Click OK.*

   The merged project appears on the Suite tab. The panels and actions you have merged should appear in their respective tasks, with an added "M" badge to their icons to identify their Merge Module status.

### Merge Module Customizer

The Merge Module customizer will be populated automatically with the information about the Merge Module. You may add comments and/or notes in the textbox provided.

## Exercise 17.1 Creating Merge Modules

1. *Create a Merge Module that can be used to install OfficeSuite as a part of a larger install.*
2. *Create an OfficeSuite installer without graphical elements.*
3. *Set up the "Advertised Variables" so that the user can access the OfficeSuite install options from the master installer.*
4. *Build your OfficeSuite Merge Module.*
5. *Build a "fake" master installer to pass information to your Office Suite installer as a sub installer.*
6. *Import your Office Suite Merge Module into a master project.*

## Quick Quiz

1. Which Merge Module Type will enable you to modify your files in the advanced designer?
   A. Design Time
   B. Build Time
   C. Install Time

3. Which Merge Module Type will be included in the Master uninstaller?
   A. Design Time
   B. Build Time
   C. Install Time

4. Which Types of Merge Modules can be used in a silent installer?
   A. Design Time
   B. Build Time
   C. Install Time
   D. All of the above

5. Which answers are reasons to use an installer template?
   A. To have an installer project which contains standard panels, graphics, and files
   B. To maintain the look and feel of installers
   C. To create an installer for a suite of applications
   D. All of the above

Answers: 1.A | 2.A | 3. D

# Chapter 18: Integrating InstallAnywhere with Automated Build Environments

## *Integrating InstallAnywhere with Automated Build Environments*

InstallAnywhere's current distributions include a number of features that can be utilized to integrate your deployment project into an automated build environment or process. This includes the ability to build your project from a "headless" system, make some modifications to the project without utilizing the InstallAnywhere Advanced designer, and add a task that will enable you to integrate InstallAnywhere with the Java-based Apache Ant build tool.

### *InstallAnywhere Command-Line Build Facility*

InstallAnywhere Enterprise and Standard Editions include functionality that enables you to perform command-line builds. This enables you to build your completed project without instantiating InstallAnywhere's graphical Advanced Designer Interface.

The Command Line Build facility comes in the form of a second executable called build (`build.exe` on Windows systems). This executable takes, as an argument the full path to an InstallAnywhere `.iap_xml` file, and by default will simply build the installer as specified in the project file.

For example, to build the Office Suite installer that we've been developing throughout this course, you might use a command line similar to the following:

```
C:>"C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise\build.exe"
"C:\OfficeSuite\OfficeSuite.iap_xml"
```

In this case, the installer build would occur with the settings stored in the project file from the last build or save of the project.

(Note that for Windows Vista systems, an additional command-line builder `buildasinvoker.exe` is provided for users who do not have administrative privileges on the build system.)

If a command-line build is successful, the builder returns the value 0 (zero) to the environment; a nonzero exit code indicates a specific error. (On most systems, you can check the exit code using the command `echo $?` or `echo $status`. On Windows system, you can use the command `echo %ERRORLEVEL%`.) For a list of specific exit codes, see Appendix E or the InstallAnywhere help library.

There are times at which you may need or want to build an installer with different settings than those stored in the project file. Obviously, you could simply open the project file, make the changes, and build the project. However, the build executable accepts additional parameters that enable you to make a number of changes to the build settings of the installer.

### *Examples*

To build installers for `MyProduct` project with the previously saved build settings:

```
build MyProduct.iap_xml
```

To build installers for `MyProduct` project overriding the build platforms and building for Mac OS X and Linux only (turning all others off):

```
build MyProduct.iap_xml +x +l -s -j -w -u
```

To build installers for MyProduct project, overriding the saved build settings for that project with a build properties file:

```
build MyProduct.iap_xml -p BuildProperties.xml
```

A build properties file template named BuildProperties.xml can be found in:

```
<InstallAnywhere>/resource/build/BuildProperties.xml
```

It provides a sample of all possible build settings and can be customized to suit your build requirements.

For a list of command-line switches supported by the builder and a list of abbreviations for platforms to add or remove from a project, see Appendix D, “Build Arguments”.

## Digitally Signed Installers

Newer version of Microsoft Windows include security features which can warn users that they are about to run “unsigned code”—meaning that the operating system is not sure of the origins of the executable. At times, this can be confusing to users, and can add to support costs. InstallAnywhere 8 introduced the ability to sign InstallAnywhere installers, and thus enable your installers to meet the requirements of the Windows security sub-system.

In order to sign installers for Windows you must first have a valid digital certificate. You will need to obtain this certificate from a Certifying Authority prior to signing your installer.

To digitally sign installers you will need two key files: a .pvk file (a private key), and a .spc file (the previously mentioned code signing certificate). You’ll also need signcode.exe—the Microsoft tool which enables you to integrate your key and certificate into the signature, and sign your installer.

If digital signatures aren’t already in use at your organization, you’ll need to download signcode.exe Microsoft’s download center. It is included in a package called codesigningx86.exe. The application has both a GUI mode, and command line mode, and is often run from the command line as part of an automated build. The command line can be as simple as the following:

```
signcode /spc myCert.spc /v mypkey.pvk "install.exe"
```

The signcode application has numerous command line parameters, each of which enables you to customize the way that your application is signed. You can obtain more information on signcode.exe’s options at `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfFileSigningToolSigncodeexe.asp`.

## Ant Build Integration

Ant is a powerful, Java-based build tool developed by the Apache Foundation’s Jakarta Project. It can be used to control complex build tasks in Java and other development environments. Ant manages specific actions though “tasks”, which can either be part of the core Ant distribution or available as extensions.

InstallAnywhere provides an Ant task to build installers from Ant. The InstallAnywhere Ant task is located in your InstallAnywhere application folder:

```
<InstallAnywhere>/resource/build/iaant.jar
```

To integrate the InstallAnywhere Ant task in an Ant project, you must set the classpath of the InstallAnywhere Ant task to the location of `iaant.jar`. Note that use of `iaant.jar` requires Java 1.4 or later.

Ant uses an XML file to specify the order of tasks for your build process. More information on Ant can be found on the Apache Foundation's Ant Project Web site (`http://ant.apache.org`).

## Task Definition

You add a task definition to your Ant project for the InstallAnywhere Ant task as follows:

```
<taskdef name="buildinstaller"
         classname="com.zerog.ia.integration.ant.InstallAnywhereAntTask"/>
    <classpath>
        <pathelement path="<InstallAnywhere>/resource/build/iaant.jar" />
    </classpath>
</taskdef>
```

## Task Settings

After defining the task, specify any parameters necessary for the build settings:

```
<buildinstaller
  IALocation="C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise"
  IAProjectFile="C:\Projects\myproject.iap_xml"
  additionalparameter=value>
  <configuration>
    <target platform="windows">
      <outputDir>win</outputDir>
      <buildWithNoVM>false</buildWithNoVM>
      <buildWithVM>true</buildWithVM>
      <bundledVM>SunJRE160_00i18nWin32.vm</bundledVM>
    </target>
    ...
  </configuration>
</buildinstaller>
```

Replace the `IAlocation` value with the absolute path to your own InstallAnywhere application folder.

Specify the path and file name of the project to build in the `IAProjectFile` parameter.

All other properties are optional. The parameters closely match the properties found in the BuildProperties.xml file described above. For a complete list of build parameters and other options, see Appendix F, "Build Properties".

# Chapter 19: Custom Code

- **Writing Custom Code**
- **Quick Quiz**

The majority of tasks needed to deploy the application can be handled using InstallAnywhere’s built-in actions and panels. If there is functionality not covered by InstallAnywhere’s built-in actions and panels, developers can create their own custom components using the Custom Code API.

InstallAnywhere offers an open Application Programming Interface (API) which enables developers to write Java code that can run within InstallAnywhere's architecture. Using the API also gives access to additional functionality in InstallAnywhere, such as its unique Variables and resource loading features. Developers can use the API to create custom actions and GUI elements that seamlessly interact with and extend the InstallAnywhere framework.

All custom code that is going to run within the InstallAnywhere framework must be written in Java. The major forms of custom code are actions, panels, consoles, and rules:

| | |
|---|---|
| **Custom Code Actions** | These actions run within InstallAnywhere’s action framework, alongside the default InstallAnywhere actions. |
| **Custom Code Panels** | These panels run within InstallAnywhere’s graphical interface during the installation process. The developer can use this mechanism to add panels to the Installer not provided in InstallAnywhere's default panels. |
| **Custom Code Consoles** | These actions run within InstallAnywhere’s console interface during the installation process. Developers can use this mechanism to add custom console elements to the Installer. |
| **Custom Code Rules** | These rules are evaluated when the action they are associated with is about to be executed. Custom Code rules need to return a Boolean value defining whether the action is to be run. |

Macrovision provides sample custom code actions, panels, console actions, and rules. These can be found in the `CustomCode` folder in the root installation directory of InstallAnywhere. These samples can be used as examples of how to implement InstallAnywhere custom code using the API. Additionally, there is a wide variety of custom code actions and panels located in the Downloads > Add-Ons > Custom Code area of the Macrovision website.

## ***Writing Custom Code***

### *Custom Code Actions*

Custom Code Actions enable you to create non-graphical actions which can manipulate data on your target system, affect InstallAnywhere Variables, pre-populate various install options, or even execute native code (using JNI or native executions). In fact, Custom Code Actions enable you to perform nearly any action possible with Java.

#### A Custom Code Action Example

This section describes how to write your own CustomCodeAction, and leads you through an example (called `com.zerog.ia.customcode.samples.SampleAction`). This section does not explain how to make your code work with InstallAnywhere’s Advanced Designer or the Execute Custom Code action.

**NOTE:** Assume that you know the goals of your action, but have not yet written any code to implement it. After each relevant step, we will show an example of how this can be done.

1. *Create your main class file and package it. This is the file that you will specify in the Class: field in the InstallAnywhere Advanced Designer (see Execute Custom Code documentation)*

```
package com.zerog.ia.customcode.samples;
```

2. *To create a custom action, your class must first extend the abstract class com.zerog.ia.api.pub.CustomCodeAction. This class provides the interface through which you will interact with the InstallAnywhere runtime.*

```
package com.zerog.ia.customcode.samples;

import com.zerog.ia.api.pub.*;

public class SampleAction extends CustomCodeAction
{
   public void install(InstallerProxy ip) {}
   public void uninstall(UninstallerProxy up) {}
   public String getInstallStatusMessage( ) {}
   public String getUninstallStatusMessage( ) {}
}
```

3. *Description of public void install (InstallerProxy ip) {}*

   The installer calls this method at the point during installation where the CustomCodeAction was added. The InstallerProxy instance provides methods to access installer information, set installation status, and control installation flow. SampleAction, will display a message to the end user in a modal dialog box.

```
package com.zerog.ia.customcode.samples;

import com.zerog.ia.api.pub.*;
import java.awt.*;
import java.awt.event.*;
public class SampleAction extends CustomCodeAction implements ActionListener
{
    public void install(InstallerProxy ip)
    {
        // Get Some Variables from the Variable Manager
        String title = ip.substitute("$SAMPLE_ACTION_TITLE$");
        String msg = ip.substitute("$SAMPLE_ACTION_MESSAGE$");

        Label l = new Label(msg);

        Button b = new Button("Close");
        // For internationalization, the button label could be
        // externalized and referenced as follows:
        //
        // Button b = new Button(ip.getValue("SampleAction.Close"));
        b.addActionListener (this);

        // Here we create a dummy Frame to use as the parent for the
        // Dialog - it is never shown.
        Frame myFrame = new Frame( );

        Dialog myDialog = new Dialog(myFrame, title, true);
        Dimension bounds = Toolkit.getDefaultToolkit( ).getScreenSize( );

        myDialog.setLayout(new BorderLayout( ));

        myDialog.add(BorderLayout.NORTH, l);
        myDialog.add(BorderLayout.CENTER, b);
        myDialog.pack( );

        // Center the Dialog
        Rectangle abounds = myDialog.getBounds( );
        myDialog.setLocation((bounds.width - abounds.width)/2,
            (bounds.height - abounds.height)/2);

        myDialog.show( );
    }

    public void uninstall(UninstallerProxy up) { }
    public String getInstallStatusMessage( ) { }
    public String getUninstallStatusMessage( ) { }
    public void actionPerformed(ActionEvent ae)
    {
        // Dispose the dialog and its hidden parent frame.
        Button source = (Button)ae.getSource( );
        Dialog parent = (Dialog)source.getParent( );
        Frame fparent = (Frame)parent.getParent( );
        parent.dispose( );
        fparent.dispose( );
    }
}
```

4. *Description of public void uninstall (UninstallerProxy up) {}*

   This method is called by the uninstaller at uninstall-time, before the deletion of any files. The UninstallerProxy instance provides access to any information written at install-time. SampleAction will display a small window with a message.

```
public void uninstall(UninstallerProxy up)
{
    String title = "Uninstall";

    Button b = new Button("Close");
    b.addActionListener(this);

    Frame myFrame = new Frame( );

    Dialog myDialog = new Dialog(myFrame, title, true);
    Dimension bounds = Toolkit.getDefaultToolkit( ).getScreenSize( );

    myDialog.setLayout(new BorderLayout( ));

    myDialog.add(BorderLayout.NORTH, l);
    myDialog.add(BorderLayout.CENTER, b);
    myDialog.pack( );

    // Here we center the Dialog
    Rectangle abounds = myDialog.getBounds( );
    myDialog.setLocation((bounds.width - abounds.width)/2,
        (bounds.height - abounds.height)/2);

    myDialog.show( );
}
```

5. *Description of public String getInstallStatusMessage() {}*

   The installer calls this message to display a status message in the progress bar during installation. The message is displayed while the install (InstallerProxy ip) method is called. The method has no effect if the action is a Pre-Install, or Post-Install Action.

```
public String getInstallStatusMessage( )
{
    return "Displaying a small installation message...";
}
```

6. *Description of public String getUninstallStatusMessage() {}*

   This method is called by the uninstaller to display a status message in the progress bar during uninstall. The message is displayed while the uninstall (UninstallerProxy up) method is called.

```
public String getUninstallStatusMessage( )
{
        return "Displaying a long uninstallation message...";
    }
```

7. *Description of CustomCodeAction member variables*

   The CustomCodeAction class provides "InstallerProxy installerProxy" and "UninstallerProxy uninstallerProxy" as member variables. Use this to access the proxies at any point during execution of the custom code actions, rather than only during installation/uninstallation. Uninstall and install still receive proxies as parameters while called, this is for ease of use as well as for backwards compatibility. You can use either the passed proxy, or the member variable proxy, and receive the exact same functionality inside these functions See the Javadoc for CustomCodeAction for more information.

Sample code is located on the accompanying CD.

Starting with InstallAnywhere 2008, you can also implement the `build` method in your custom code action to perform tasks at build time, rather than at run time. Using this method, you can add resources to the archive and get build target and distribution settings. For more information, see the Javadoc help for the `CustomCodeAction` and `CustomBuildServices` classes.

## *Custom Code Panels*

There may be times when you find that InstallAnywhere's included panels do not meet your needs. InstallAnywhere's API addresses this with the introduction of Custom Code Panels. Custom Code Panels are the graphical equivalent of Custom Code Actions. They enable you to present a UI to your end user in combination with any task that you may need in a graphical installer.

The Custom Code Panel provides you with a framework to which you can add components necessary for your particular task. Each Custom Code Panel extends the core InstallAnywhere install panel, and as such provides your custom developed panel with the same look and feel, and same available elements as the standard InstallAnywhere panels. The default custom code panel provides the framework for Swing GUI elements.

```
public boolean setupUI(CustomCodePanelProxy ip)
{
    // clear panel if setupUI is called multiple times
    if(true)
        removeAll( );
    // set up layout manager
    setLayout(new FlowLayout( ));
    Label label1 = new Label("Custom Code Panel Test");
    add(label1);
    return true;
}
```

### A Custom Code Panel Example

This section describes how to write your own CustomCodePanel, and leads you through an example (called `com.zerog.ia.customcode.samples.SamplePanel`). This section does not explain how to make your code work with InstallAnywhere's Advanced Designer or the Custom Code Panel action.

**NOTE:** We assume that you know the goals of your panel, but have not yet written any code that implements it. After each relevant step, you will see an example of how it can be done.

1. *Create your main class file and package it. This is the file that you will specify in the Class: field in the InstallAnywhere Advanced Designer (see Custom Code Panel documentation).*

   ```
   package com.zerog.ia.customcode.samples;
   ```

2. *To create a custom action, your class needs to extend the abstract class com.zerog.ia.api.pub.CustomCodePanel. This class provides the interface for interacting with the InstallAnywhere installer.*

```
package com.zerog.ia.customcode.samples;

import com.zerog.ia.api.pub.*;

public class SamplePanel extends CustomCodePanel
{
public boolean setupUI(CustomCodePanelProxy customCodePanelProxy) {}
public void panelIsDisplayed( ) {}
public boolean okToContinue( ) {}
public boolean okToGoPrevious( ) {}
public String getTitle( ) {}
}
```

3. *Description of public boolean setupUI(CustomCodePanelProxy customCodePanelProxy) {}*

This method gets called prior to the Panel being displayed. This method is useful for initializing the contained Components and variables. SamplePanel will use another example class, BrowserLauncher, to demonstrate how to launch a URL through a custom code panel. To do this, import the class `edu.stanford.ejaldbert.BrowserLauncher`, and make sure to include this class file in the custom code panel Jar archive. Use the setupUI method to setup the UI and add components to our panel. This step includes adding labels, textfields buttons, and an ActionListener. You'll want to ensure that you specify a LayoutManager for your panel. Though Custom Code Panel does specify a default FlowLayout, by specifying the LayoutManager your components appear as expected within the InstallAnywhere Custom Code Panel framework. This example also creates a few member variables of the SampleConsole class.

```
package com.zerog.ia.customcode.samples;
import com.zerog.ia.api.pub.*;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import edu.stanford.cs.ejalbert.*;
public class SamplePanel extends CustomCodePanel implements ActionListener
{
private boolean inited = false;
private TextField tf;
private Button b;

public boolean setupUI(CustomCodePanelProxy customCodePanelProxy)
{
// Use a boolean flag here to prevent duplicate GUI elements.
if (inited == true)
return true;
inited = true;

Label label = new Label("URL");
label.setFont(new Font("Dialog", Font.BOLD, 12));

b = new Button("Go");
b.addActionListener(this);

tf = new TextField (40);

// Get a default URL string from InstallAnywhere and use it if it
// is not empty.
String def = customCodePanelProxy.substitute("$DEFAULT_URL$");
System.out.println("Def: " + def);
if (!def.equals(""))
{
    tf.setText(def);
}
else
{
    tf.setText("www.macrovision.com");
}

add(label);
add(tf);
add(b);

return true;
}
public void panelIsDisplayed( ) {}
public boolean okToContinue( ) {}
public boolean okToGoPrevious( ){}
public String getTitle( ) {}
}
```

4. *Description of public void panelIsDisplayed(UninstallerProxy up) {}*

   This method is called immediately after the Panel is displayed. This is useful for doing some processing while the Panel is displayed, without having to wait for the okToContinue method to be called. Of course, this method will never be called if setupUI returns false. SamplePanel does nothing during this method and is left empty.

```
public void panelIsDisplayed( )
{
}
```

5. *Description of public boolean okToContinue() {}*

   This method gets called prior to continuing on with the installer. If this method returns true, then the installer continues to the next action, otherwise the installer prevents the user from continuing. This is useful for verifying end-user input or setting InstallAnywhere variables. In the SamplePanel an InstallAnywhere Variable is set, before continuing.

```
public boolean okToContinue( )
{
    // Set an IA variable based upon the textfield's value, then
    // continue.
    customCodePanelProxy.setVariable("$CHOSEN_URL$", tf.getText( ));
    return true;
}
```

6. *Description of public boolean okToGoPrevious() {}*

   Similar to `okToContinue`, this method gets called prior to returning to a previous step in the installer if the end user clicks the previous button. In the SamplePanel this method simply returns true.

```
public boolean okToGoPrevious( )
{
    return true;
}
```

7. *Description of public String getTitle() {}*

   This method returns the String to be displayed as the title of this panel. In SamplePanel “Launch URL” is returned as the title.

```
public String getTitle( )
{
    return "Launch URL";
}
```

8. *Description of CustomCodePanel member variables*

   The CustomCodePanel class provides “CustomCodePanelProxy customCodePanelProxy” as a member variable. This panel enables developers to access the proxy at any point during execution of the custom code panel, rather than only during setupUI. While setupUI is called, it still receives the CustomCodePanelProxy as a parameter (for ease of use as well as backward compatibility). Inside this method, it is important to note that you can use either the passed proxy, or the member variable proxy, and receive the exact same functionality. See the Javadoc for CustomCodePanel for more information.

Sample code is located on the accompanying CD.

## *Custom Code Consoles*

InstallAnywhere's Custom Code Console provides a similar, customizable framework to that provided by Custom Code panel that enables you to add components of your choosing to a generic InstallAnywhere interface. This console enables the installer to display text or extract information from the end user when running in console mode.

The framework is designed to provide a text only interface, and as such, the available components are somewhat different.

A simple Custom Code Console might look like this:

```
=================================================================
Test Console
------
Please select one of the following options.
->1- first choice
  2- second choice
  3- third choice
ENTER THE NUMBER FOR YOUR CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:
=================================================================
```

### A Custom Code Console Action Example

This section describes how to write your own `CustomCodeConsoleAction`, by leading you through an example (`com.zerog.ia.customcode.samples.CustomCodeConsoleAction`). This section doesn't explain how to make your code work with InstallAnywhere's Advanced Designer.

NOTE: We assume that you know the goals of your console action, but have not yet written any code that implements them. After each relevant step, we will show an example of how it can be done.

1. *Create your main class file and package it. This is the file that you will specify in the Class: field in the InstallAnywhere Advanced Designer (see Execute Custom Code documentation).*

   ```
   package com.zerog.ia.customcode.samples;
   ```

2. *To create a custom console action, your class must first extend the abstract class com.zerog.ia.api.pub.CustomCodeConsoleAction. This class provides the interface through which you will interact with the InstallAnywhere runtime.*

   ```
   package com.zerog.ia.customcode.samples;
   import com.zerog.ia.api.pub.*;
   public class SampleAction extends CustomCodeConsoleAction
   {
       public boolean setup( ) {}
       public void executeConsoleAction( ) throws PreviousRequestException{}
       public String getTitle( ) {}
   }
   ```

3. *Description of public boolean setup() {}*

   The installer calls this method prior to the ConsoleAction being displayed. This method is useful for initialization needed by the action and returns true if the console should be displayed. If this method returns false, the console is not displayed and the installer continues with the next action. This CustomCodeConsoleAction example uses the setup method to populate a vector from an InstallAnywhere variable.

   ```
   package com.zerog.ia.customcode.samples;
   ```

```
import com.zerog.ia.api.pub.*;
import java.util.vector;
import java.util.StringTokenizer;

public class SampleConsole extends CustomCodeConsoleAction {

public boolean setup( )
{
// Use a StringTokenizer to populate the vector ‘choices’ from
// the InstallAnywhere variable $SAMPLE_CONSOLE_LIST$

String list = cccp.substitute("$SAMPLE_CONSOLE_LIST$");
StringTokenizer st = new StringTokenizer(list, ",");

while (st.hasMoreTokens( ))
{
    choices.addElement(st.nextToken( ));
}
return true;
}

public void executeConsoleAction( ) throws PreviousRequestException {}
public String getTitle( ){}
}
```

4. *Description of public void executeConsoleAction() {}*

   This method is called when the installer is ready to display the console action. Most if not all of the console input and output should originate from the call into this action via this method. This example uses the executeConsoleAction method to prompt the end-user to select a choice from a list.

```
public void executeConsoleAction( ) throws PreviousRequestException
{
// Get an instance of ConsoleUtils. We will use ConsoleUtils to help
// construct the console prompts.
ConsoleUtils cu = (ConsoleUtils)cccp.getService(ConsoleUtils.class);

// Use the substitute method to get the variable:
// $SAMPLE_CONSOLE_PROMPT$
String prompt = cccp.substitute("$SAMPLE_CONSOLE_PROMPT$");

// Use the built-in features of the ConsoleUtils class to ask
// the user to select from a list
int userChoice = cu.createChoiceListAndGetValue(prompt, choices);

// Set the result as an InstallAnywhere variable
String result = choices.elementAt(userChoice).toString( );
cccp.setVariable("$SAMPLE_CONSOLE_CHOICE$", result);
}
```

5. *Description of public String getTitle() {}*

   This method returns the String to be displayed as the title of this console. This example just returns “Sample Console” as the title.

```
public String getTitle( )
{
    String title = "Sample Console";
    return title;
}
```

6. *Description of CustomCodeAction member variables*

The `CustomCodeConsoleAction` class provides "`CustomCodePanelProxy cccp`," but instances of the class `ConsoleUtils` must be instantiated specifically. See the Javadoc for `CustomCodeConsoleAction` for more information.

Sample code is located on the accompanying CD.

## *Custom Code Rules*

InstallAnywhere's rules architecture enables you to extend the existing rule set by adding Custom Code Rules. These rules, based on the InstallAnywhere API, provide you with a method of making any action in the InstallAnywhere installer conditional. However, unlike the built-in rules, your code provides the condition. This feature is especially useful in situations where your action is dependant on a variable outside of the InstallAnywhere architecture.

An InstallAnywhere Custom Code Rule is effectively an action which, when executed, returns a simple Boolean value. If the rule returns "true" the action to which it is attached will install or execute. If the rule returns "false" the action will be prevented from occurring.

### A Custom Code Rule Example

This section describes how to write your own CustomCodeRule, and leads you through an example (called `com.zerog.ia.customcode.samples.SampleRule`). This section doesn't explain how to make your code work with InstallAnywhere's Advanced Designer.

NOTE: We assume that you know the goals of your rule, but have not yet written any code to implement it. After each relevant step, we will show an example of how this can be done.

Create your main class file and package it. You will specify this file in the `Class: field` in the InstallAnywhere Advanced Designer (see Execute Custom Code documentation).

```
package com.zerog.ia.customcode.samples;
```

1. *To create a custom rule, your class must first extend the abstract class com.zerog.ia.api.pub.CustomCodeRule. This class provides the interface through which you will interact with the InstallAnywhere runtime.*

   ```
   package com.zerog.ia.customcode.samples;
   import com.zerog.ia.api.pub.*;

   public class SampleRule extends CustomCodeRule {
         public abstract boolean evaluateRule( )
   }
   ```

2. *Description of public abstract boolean evaluateRule()*

   This method is called at install-time when evaluating the rules set on a given installer action. It is very important that this method return quickly so that unnecessary lag is not experienced in the installer.

   ```
   package com.acme; // you should change this to your own package
   import com.zerog.ia.api.pub.*;
   public class CustomCodeRuleTemplate extends CustomCodeRule
   {
   public boolean evaluateRule( )
   {
   //This method resolves all of the InstallAnywhere variables in a
   //string. If the string contains a variable, and resolving that
   //variable then contains another variable, that too will be
   //resolved until all variables have been resolved. If a variable
   ```

```
//cannot be resolved, it evaluates to an empty string ("").
String myString = ruleProxy.substitute("$myVariable$");

//This method returns the value of the named variable.
//If no variable is
//defined for that name, returns null.
//String myString = (String)ruleProxy.getVariable("myVariable");
//This method sets the named variable to refer to the value. If
//the variable was already set, its previous value is returned.
//Otherwise, returns null.
Object previousValue = ruleProxy.setVariable("myVariable", "theValue");

//For Internationalization support. Gives access to locale-
//specific static GUI strings.
String myString = ruleProxy.getValue("myKey", Locale.ENGLISH);

//For Internationalization support. Gives access to locale-
//specific static GUI strings. Returns the resource for the
//user’s chosen installation locale.
      String myString = ruleProxy.getValue("myKey");
      return true;
}
}
```

3. *Description of CustomCodeRule member variables*

   The CustomCodeRule class provides “CustomCodeRuleProxy customCodeRuleProxy” as a member variable. This enables developers to access the proxy at any point during execution of the custom code rule, rather than only during evaluateRule. See the Javadoc for CustomCodeRule for more information.

Sample code is located on the accompanying CD.

**NOTE:** InstallAnywhere contains a services layer that adds a rich suite of APIs for use with custom code actions. Many of these APIs mirror the functionality present in InstallShield MultiPlatform (ISMP) services. The File service, Security service, SystemUtil service, Win32 service, Win32 Registry service, and the Windows Account Privileges service from ISMP are available starting with InstallAnywhere 8.

## *Quick Quiz*

1. If you added code to the uninstall method in a Custom Code Action, where must you add the Action in order for the uninstall method to be called.
   A. Pre-Install
   B. Install
   C. Post-Install

2. Can Custom Code be localized?
   A. Yes
   B. No

3. Which type of Custom Code will run in all supported install modes (GUI, Console, and Silent)?
   A. CustomCodeAction
   B. CustomCodePanel
   C. CustomCodeConsoleAction

Answers: 1.B | 2 Yes | 3. A

# Chapter 20: Developing and Using Custom Code Actions

- **Custom Code and InstallAnywhere Variables**
- **Accessing InstallAnywhere Variables via Custom Code**
- **Executing External Scripts and Executables via Custom Code Action**
- **Best Practices for Custom Code Development**
- **How to Write Custom Errors in the Installation Log**
- **Custom Code Development Exercises**
- **How to Package and Execute Custom Code with an Installer**
- **Plug-Ins**

## *Custom Code and InstallAnywhere Variables*

For information on variables, see the section on InstallAnywhere Variables. Custom Code can both get and set InstallAnywhere Variables. This ability can be useful in two cases:

A project can set InstallAnywhere Variables, which can be used as parameters for Custom Code. For example, if a Custom Code Action is designed to know the directory to which the end user was installing, use the Set InstallAnywhere Variable action to set

`PARAM_1` to `$USER_INSTALL_DIR$`

Then, in the Custom Code Action, call `InstallerProxy.substitute("$PARAM_1$")` to determine its value.

InstallAnywhere Variables may also be used to store return values from Custom Code Actions. These Variables can be queried by other actions or rules. To do this, set the InstallAnywhere variable's value by calling the method `InstallerProxy.setVariable("PARAM_1", <OBJECT>).`

Remember that these Variables are treated as Strings, and will have `java.lang.Object.toString` called upon them to determine their value in InstallAnywhere. Once `PARAM_1` has been set it can be accessed in the normal way, using `$PARAM_1$` in a later InstallAnywhere rule or action.

A common problem when accessing the values of InstallAnywhere variables from within Custom code is the improper use of the `getVariable` and `substitute` methods. The most common problem is the use of `getVariable` on a Magic Folder object (for instance, `$USER_INSTALL_DIR$`) and then trying to cast this object to a String. Instead, use `substitute`, which resolves the contents of the variable to a string, instead of casting the object.

## *Accessing InstallAnywhere Variables via Custom Code*

There is a Proxy class for each of the four types of Custom Code which provide methods for classes extending the Custom Code classes to access information in an InstallAnywhere installer, and locate and access resources. Each of these Proxy classes provides a method called `getVariable` and `substitute`. Both of these methods can be used to access InstallAnywhere variables from within your custom code.

### substitute

```
public String substitute(String var)
```

This method fully resolves a String that may contain embedded InstallAnywhere variables

The String returned is guaranteed to resolve all InstallAnywhere variables embedded in the parameter passed to this method. Variables contained (embedded) within the String are fully and recursively resolved (i.e., they are resolved recursively). InstallAnywhere variables that are to be resolved are identified by their surrounding dollar signs ($). If no value has been set for a given variable name, that variable is resolved to the empty String.

For example, calling this method on the String:

"The files have been `$PLACED$` in `$USER_INSTALL_DIR$`"

would return a String with `$PLACED$` and `$USER_INSTALL_DIR$` resolved to the `String` values of the objects represented by the InstallAnywhere variables.

This method is particularly useful for resolving file system paths represented by InstallAnywhere variables for Magic Folders and is the preferred mechanism for retrieving this type of data.

### getVariable

```
public java.lang.Object getVariable(String var)
```

This method returns the literal Object represented by an InstallAnywhere variable

InstallAnywhere variables are identified by their surrounding dollar signs ($). If no value has been set for a given variable name, null is returned.

The `getVariable` method does not recursively resolve variables. If the intention is to get the String representation of a particular InstallAnywhere variable, the `substitute` method is generally preferred.

For example, `getVariable("$USER_INSTALL_DIR$")` would return the MagicFolder object for the current install location, while `substitute("$USER_INSTALL_DIR$")` would return a String representing the absolute path to the current install location.

## *Executing External Scripts and Executables via Custom Code Action*

Here is a simple Custom Code Action that enables you to extract the exit code and output produced when executing a command or target file during install time.

```
package com.zerog.ia.customcode.actions;

import com.zerog.ia.api.pub.*;
import java.net.*;
import java.io.*;

public class EnhancedExecuteTargetFile extends CustomCodeAction
{
      public void install( InstallerProxy ip ) throws InstallException
{

try
{
      String fileToExecute = ip.substitute("$FILE_TO_EXECUTE$");
      String executeString = "";

      if(fileToExecute.endsWith(".sh") || fileToExecute.endsWith(".bin"))
      {
            System.err.println("Shell script found - using /bin/sh");
            executeString = "/bin/sh " + fileToExecute;
      }
      else
      {
            executeString = fileToExecute;
      }

      System.out.println("About to execute: " + executeString);
      final Process p = Runtime.getRuntime( ).exec(executeString);

      final StringWriter errWriter = new StringWriter( );
      final StringWriter outWriter = new StringWriter( );

      Thread thread = new Thread( )
      {
            public void run( ) {
             InputStream err = p.getErrorStream( );
             int c;
             try
             {
              while((c = err.read( )) >= 0)
              {
                  errWriter.write(c);
                 }
                }
                catch (IOException ioe){ }
         }
    };
          Thread thread2 = new Thread( )
          {
                public void run( )
                {
                 InputStream err = p.getInputStream( );
                 int c;
                 try
    {
                  while((c = err.read( )) >= 0)
                  {
      outWriter.write(c);
            }
            }
      catch (IOException ioe){}
                }
    };

    thread.start( );
    thread2.start( );
    p.waitFor( );

    String stdErr = errWriter.toString( );
    String stdOut = outWriter.toString( );
    String exitCode = new Integer(p.exitValue( )).toString( );


    System.out.println("StdErr Output: " + stdErr);
    System.out.println("StdOut Output: " + stdOut);
    System.out.println("Exit Code: " + exitCode);


     ip.setVariable("CC_STD_ERR", stdErr);
     ip.setVariable("CC_STD_OUT", stdOut);
     ip.setVariable("CC_EXIT_CODE", exitCode);
  }
    catch(Exception e)
    {
  throw new NonfatalInstallException("Execute Command Custom Code Failed while
  reading Error Stream");
    }
  }
  public void uninstall(UninstallerProxy up) throws InstallException
  {
  }

  public String getInstallStatusMessage( )
  {
    return "Execute Target File";
  }

  public String getUninstallStatusMessage( )
  {
    return "Execute Target File";
  }
  }
```

### Using InstallAnywhere Custom Code Actions in Uninstall

The Custom Code action framework provides a key method—`uninstall`—which enables you to customize the behavior or, add functionality to the InstallAnywhere uninstaller. If your deployment needs require any custom actions at uninstall time, you'll need to implement these tasks in a Custom Code action implementing the `uninstall` method. Additionally, this method enables you to specify an uninstall task equivalent to an action that occurs at install time. A single action can utilize install and uninstall methods, and can provide Install task and Pre and Post-Uninstall task options.

The uninstall method will only be called if the Custom Code Action is added to the Install, Pre-Uninstall, or Post-Uninstall task. If placed in Pre-Install or Post-Install the `uninstall` method in the Custom Code Action will not be called, and all code within the `uninstall` method will be ignored.

```
public void uninstall(UninstallerProxy up) throws InstallException
{
    System.out.println("This line will be displayed during uninstall");
}
```

The code snippet above simply displays text to standard out. This text will be displayed only if the uninstaller is run in debug mode.

### Debugging Custom Code

In order to debug your Custom Code, you can simply add System.out or System.err statements to display information within your code. This output will only be displayed if the installer or uninstaller is run in debug mode. See Chapter 11 for more detailed information on running installers and uninstallers in debug mode.

## Best Practices for Custom Code Development

Use a batch or shell script to combine all the tasks needed for your custom code development. A single script that compiles your code, packages it, moves files to your correct build location, then builds and executes your InstallAnywhere installer can save time, and make your development process much easier.

When developing and testing your custom code, it is recommended you place frequent Sytem.out and System.err statements within your code. Doing this enables you to easily identify possible problems within the code.

### Advanced Action Methods

`public long getAvailableDiskSpace( )`

A convenience method to get the amount of disk space available on the target system.

`public java.util.Vector getInstallBundles( )`

Returns a vector of Strings that describe the names of all install bundles (features) whose rules currently evaluate to true.

Passing the String name of one of the install bundles (features) contained in the returned Vector will cause that install bundle (feature) to be installed.

`public java.util.Vector getInstallSets( )`

Returns a vector of Strings that describe the names of all install sets (features) whose rules currently evaluate to true. Passing the String name of one of the install sets (features) contained in the returned Vector will cause that install set (feature) to be installed.

`public java.util.Vector getJavaVMList( )`

Returns a vector of Strings that contain the paths to all VMs found on the target system.

When searching for VMs, the installer will search along the end user's system PATH, and in the case of Windows systems, the system registry. This method returns all VMs found on the system.

`public long getRequiredDiskSpace( )`

A convenience method to get the amount of disk space required to install the selected product feature on the target system.

```
public boolean installBundledJRE (boolean installJRE)
                                                  throws CannotInstallJREException
```

Instructs the installer to either install the bundled JRE, if present, or not.

If the installer is instructed not to install the bundled JRE, the installer will attempt to choose a system virtual machine for any LaunchAnywhere executables that are to be installed.

`public boolean setChosenInstallSet(String installSet)`

Instructs the installer to install the components included in the specified install set (feature).

The install set is described by its full name as created in the InstallAnywhere Advanced Designer or as returned by the call to getInstallSets. Essentially, this call makes the specified install set the new default install set for the installer. Calls to this method will override prior calls made to this method and prior calls made to setChosenInstallBundles.

`public boolean setChosenInstallBundles(String installBundles)`

Instructs the installer to install the components included in the specified install bundle(s) (product component).

The install bundle(s) (features) are described by a comma-separated String containing the full name of all desired install bundles (features) as created in the InstallAnywhere advanced designer or as returned by the call to getInstallBundles. Calls to this method will override prior calls made to this method and prior calls made to setChosenInstallSet.

`public boolean setChosenInstallSet(String installSet)`

Instructs the installer to install the features included in the specified install set (feature).

The install set (feature) is described by its full name as created in the InstallAnywhere Advanced Designer or as returned by the call to getInstallSets. Essentially, this call makes the specified install set (feature) the new default install set (feature) for the installer. Calls to this method will override prior calls made to this method and prior calls made to setChosenInstallBundles.

`public boolean setJavaVM(String vmPath)`

Instructs the installer to use the system virtual machine described by the provided absolute path as the VM for all LaunchAnywhere executables installed by this installer.

`public void setJavaVMList(java.util.Vector newVMList)`

This method enables the list of VM paths to be set externally to the installer's normal mechanism for gathering and saving paths to system VMs.

This method enables the development of actions that are able to validate version information, etc. about a system VM. Setting the VM list prior to the display of the Choose VM step will change the VMs listed in the Choose VM step, as well as, the values returned by getJavaVMList.

## How to Write Custom Errors in the Installation Log

Custom Error provides access to the InstallAnywhere error logging and end-user installation log features to Custom Actions, Panels, and Consoles.

The `CustomError` object is obtained through via the InstallerProxy, CustomCodePanelProxy, and CustomCodeConsoleProxy objects by a request for the CustomError class, as follows:

Given an instance of `CustomCodePanelProxy` proxy,

```
CustomError error = (CustomError)proxy.getService(CustomError.class);
error.appendError("the file was not found", CustomError.ERROR);
error.setLogDescription("File Mover: ");
error.setRemedialText("Blank file was not found. Create a new text file and place
it in the TEMP directory.");
error.log( );
```

The code above will log the following text in the installation log.

## Summary

```
Installation: Successful with errors.
1      SUCCESS
0      WARNINGS
1      NONFATAL ERROR
0      FATAL ERRORS
Action Notes:
File Mover: Blank file was not found. Create a new text file and place it in the
TEMP directory.
Additional Notes: NOTE - Required Disk Space: 1,046,190; Free Disk Space:
3,818,479,616
Install Directory: -C:\Program Files\My_Product\
Status: SUCCESSFUL
Additional Notes: NOTE - Directory already existed
File Mover:- Status: ERROR
Additional Notes: ERROR - the file was not found
```

## Custom Code Development Exercises

### Exercise 20.1 Create a Custom Code Action

1. *Set InstallAnywhere variables.*
2. *Get InstallAnywhere variables.*
3. *Get Java properties.*
4. *Use a few methods provided by InstallerResources.*
5. *Log an error.*

### Exercise 20.2 Create a Custom Code Panel

1. *Get and Set InstallAnywhere variables.*
2. *Display a few components.*
3. *Use a few methods provided by GUIAccess.*

### Exercise 20.3 Create a Custom Code Console Action

1. *Create a simple Custom Code Console action.*
2. *Use a few methods provided by GUIAccess.*

### Exercise 20.4 Create a Custom Code Rule

1. *Get and Set InstallAnywhere variables*
2. *Create a scenario in which the value of an InstallAnywhere variable will set the rule to true or false.*

## How to Package and Execute Custom Code with an Installer

To create and use custom code in an installer:

1. *Add IAClasses.zip (found in the root InstallAnywhere directory) to the CLASSPATH.*

   The Java compiler will need to reference the classes in this file when compiling the code. `IAClasses.zip` is located in the root installation directory of InstallAnywhere.
2. *Create the action, panel, console, or rule. A good starting point is to use the Java source file templates found in the* `CustomCode/Templates` *folder inside the InstallAnywhere installation directory.*
3. *Compile the source files.*
4. *Decide which additional files and resources will be needed. For example, images, text files, or other resources may be required.*
5. *Create an unsigned Java Archive (JAR) file that contains the compiled class files and resource files. Signed archives will not function properly with InstallAnywhere's Custom Code API,*

**NOTE:** Make sure that the selected archive has full path information in it (each file in the archive should be stored with its proper package path). Without this JAR, Java will not be able to properly find the packaged class files.

6. *Add an action or rule that executes custom code (such as* **Execute Custom Code***).*
7. *Choose the JAR file created in the previous step*
8. *Type the fully qualified package and class name, such as:* `com.acme.MyAction`
9. *Test and debug the Action or Panel.*

   Since InstallAnywhere cannot be run from within an integrated development environment, the best way to debug the custom code is to use the Output Debug Information action, or to use `System.out.println` statements to print debug output to the console during testing.

**NOTE:** It is critical that the Custom Code JAR does not include the classes and resources from `IAClasses.zip` as this will seriously affect the behavior of the installer and uninstaller.

## Plug-Ins

Developers can register custom code as plug-ins with the InstallAnywhere Advanced Designer. This feature enables properly packaged custom code to be integrated into the design environment, where it will appear in the action palette under the plug-ins tab. Plug-Ins are stored within the `<InstallAnywhere>/plugins` folder (with InstallAnywhere 2008, you can add locations using **Edit > Preferences > Resources > Plugin Resource Paths**). The advantages of packaging custom code are:

- The developer can add them as regular actions without having to specify the JAR and class.
- Custom code usually utilizes InstallAnywhere Variables for parameters and return values. If a developer wants to execute custom code multiple times in a project, it often means the global scope of InstallAnywhere variables forces the developer to be very careful with their parameters and return values. Plug-ins have a local scope for parameters.

- Plug-ins are easily portable across development teams.

**NOTE:** Macrovision is interested in distributing plug-ins developed by InstallAnywhere developers. If you have written a plug-in that you think would be useful to other developers and you would like to share it, contact the Macrovision support team at http://support.macrovision.com.

## *Packaging Custom Code as a Plug-in*

1. *Package the custom code and all of its resources in a JAR (just like for regular custom code).*
2. *Create a properties file called* `customCode.properties`*. In this properties file will be specified all of the information InstallAnywhere needs to integrate the plug-in with the Advanced Designer. Place the properties file in the JAR with no stored path information (at the root level of the JAR).*

   The properties file MUST contain the following properties:

   - `plugin.main.class=<classname>`: this is the class that implements the proper member of the InstallAnywhere API for a custom code action, panel or console (such as a class that implements `com.zerog.ia.api.pub.CustomCodeAction`)
   - `plugin.name=<plug-in name>:` this is the name the plug-in will be displayed as in the plug-in tab of the InstallAnywhere Advanced Designer's action palette
   - `plugin.type=<action | panel | console>:` whether action, panel or console. This property helps InstallAnywhere determine when the plug-in can be used, and which icon to use to represent it in the Advanced Designer.

   The following properties MAY also be used:

   - `property.<propertname>=<propertydefault>`: This property tells the plug-in to populate the action's customizer with a property named <propertyname> and set to the default value of <propertydefault>
   - `plugin.icon.path=<relative path to .png or .jpg file in JAR>`: This property sets a custom 32 by 32 icon for the custom code plug-in in the Advanced Designer.
   - `plugin.available=<preinstall | install | postinstall | preuninstall | postuninstall>:`

     This property is a comma-separated value list to tell the Advanced Designer for which tasks to make the plug-in available.

   The following is an example of a properties file for a plug-in:

   ```
   plugin.main.class=com.zerog.ia.customcode.util.fileutils.ExtractToFile
   plugin.name=Extract to File
   plugin.type=action
   plugin.icon.path=myicon.gif
   plugin.available=preinstall,install,postinstall
   property.ExtractToFile_Source=path/to/file/in.zip
   property.ExtractToFile_Destination=$USER_INSTALL_DIR$$/$myfile.txt
   ```

3. *Additionally, plug-Ins can offer help to InstallAnywhere users on how to properly use the plug-in. Help is displayed in HTML, and is launched by the user pressing a button on the plug-ins customizer. The button only appears if a help file is provided in the plug-in. To utilize installer help:*

   a. Create a file called *`help.htm`*.
   b. Package it in the plug-in JAR (without stored path information).

4. *Place the properly packaged unsigned JAR (with the custom code, its resources and the properties file) in the* *`<InstallAnywhere>/plugins`* *directory.*
5. *Launch InstallAnywhere.*

   The plug-in will now be visible in the InstallAnywhere action.

# Chapter 21: Localizing and Internationalizing InstallAnywhere Installers

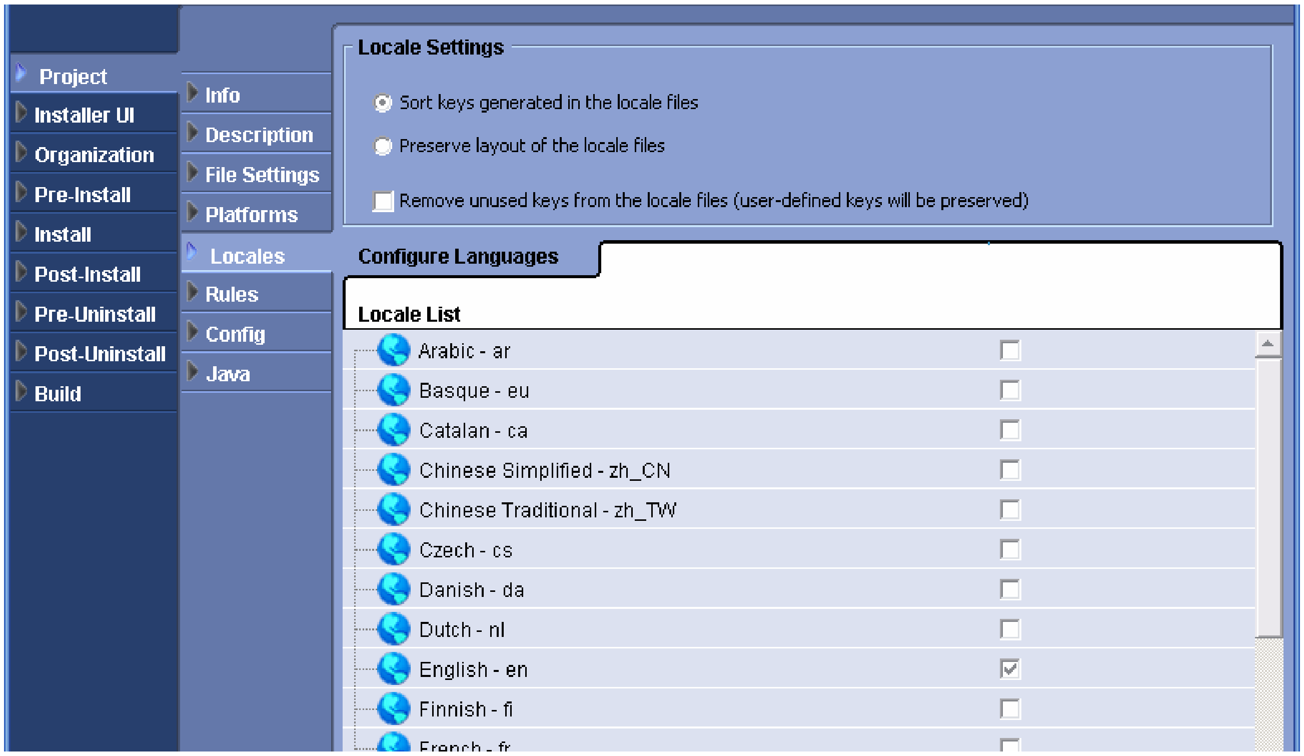
- **Dynamic and Static Text**
- **Localization and the Internationalized Designer**
- **Specific Localization Concerns**
- **Localizable Elements**

## Overview

Nearly every text string in an InstallAnywhere project can be localized. Translations of the text of built-in InstallAnywhere screens and dialog boxes are already provided. If developers want to further modify text strings by locale, the string files are output each time an installer is built, in a folder called “Project locales” which will be next to the build output folder. The files are named by locale code. For example, the default English (locale code `en`) locale file has the name `custom_en`.

These locale files contain the text strings grouped by the name of the action to which each belongs. Developers may alter the text strings and upon the next build of the installer the new localized text will be displayed with the action.

To generate multi-language installers, click the Project > Locales task in the Advanced Designer, and use the checkboxes to select the appropriate languages.



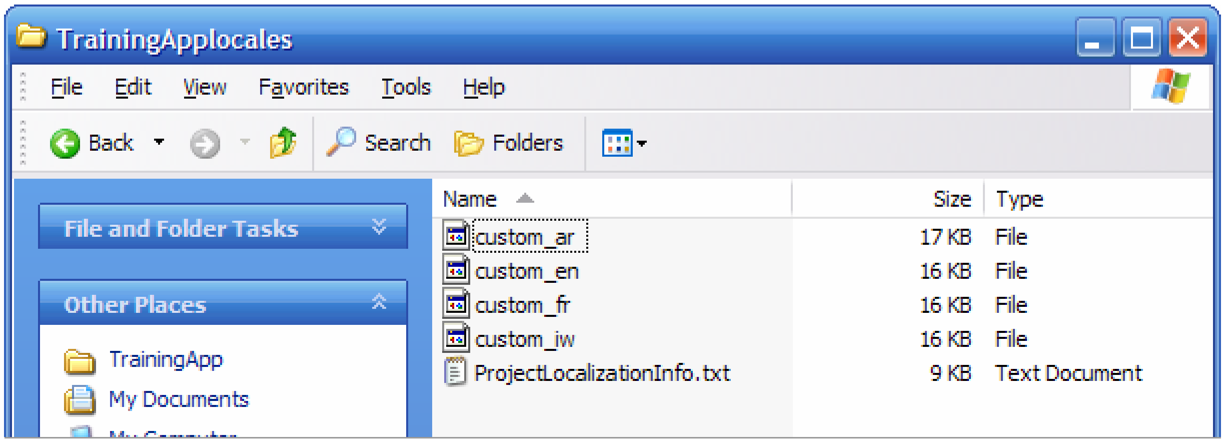
Locale Settings

[✓] Sort keys generated in the locale files
[ ] Preserve layout of the locale files
[ ] Remove unused keys from the locale files (user-defined keys will be preserved)

Locale List

| | |
|---|---|
| Arabic - ar | ☐ |
| Basque - eu | ☐ |
| Catalan - ca | ☐ |
| Chinese Simplified - zh_CN | ☐ |
| Chinese Traditional - zh_TW | ☐ |
| Czech - cs | ☐ |
| Danish - da | ☐ |
| Dutch - nl | ☐ |
| English - en | ✓ |
| Finnish - fi | ☐ |
| French - fr | ☐ |

When an installer project is first built, a folder called `<projectname>locales` is created in the same directory as the project file. For each locale selected in the Advanced Designer, there will be a file in this `<projectname>locales` folder. The locale files are generated as `custom_<localecode>`, so for English, which has a locale code of `en`, the name of the locale file will be `custom_en`.



| Name | Size | Type |
|---|---|---|
| custom_ar | 17 KB | File |
| custom_en | 16 KB | File |
| custom_fr | 16 KB | File |
| custom_iw | 16 KB | File |
| ProjectLocalizationInfo.txt | 9 KB | Text Document |

These files contain keys and values for all of the dynamic strings in the project, using the form *key=value*. The keys are generated by the name of the action, with a unique value to represent the unique instance of the action, and an additional parameter to signify which dynamic value of the action is being referenced. (Comments inside the locale files describe the format and contents of the file, along with a timestamp and information about the project associated with the file.) For example, the following are two key-value pairs:

```
InstallSet.a2ff402ca1cd.installSetName=Typical
InstallSet.a2ff402ca1cd.description=The most common application features will be
installed.  This option is recommended for most users.
```

The `ProjectLocalizationInfo.txt` file contains the mapping between the actions in the project and their keys in the locale files. Review the `ProjectLocalizationInfo.txt` file for any questions regarding to which action the key refers.

If a locale file includes any special (non-ASCII) characters, you should process the file with the Java SDK tool `native2ascii`, which converts special characters (such as “é” or “ñ”) into their platform-independent Unicode representations (such as “\u00e9” or “\u00f1”).

With InstallAnywhere 2008, you can use the options in the Locale Settings area of the Locales task (pictured above) to control whether to sort the contents of your locale files by key (the default), or to preserve the layout, white space, and comments you defined in your locale files.

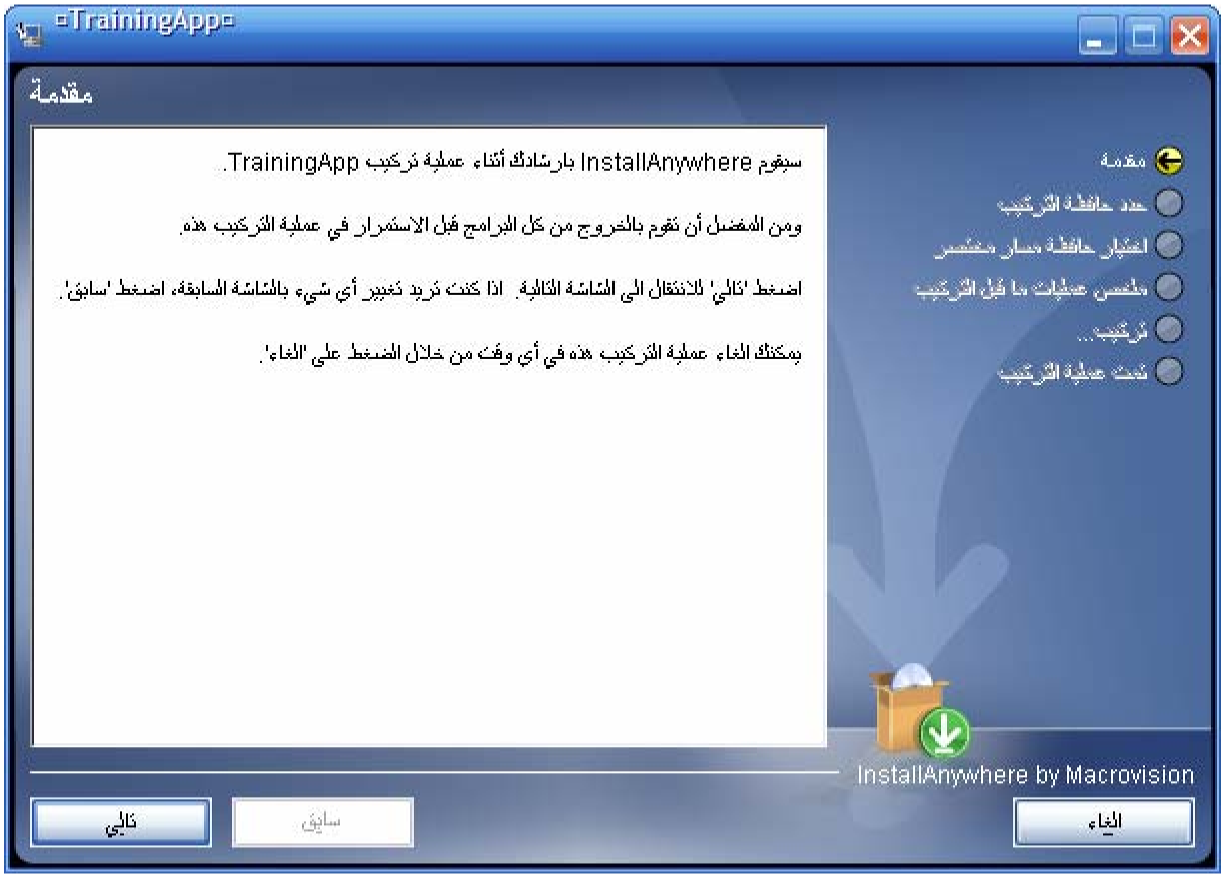
In addition, you can specify whether to remove unused entries from your locale files. (Unused keys can be caused by insertion and subsequent deletion of installer panels, for example.) Note that user-defined keys, such as those associated with custom code panels, will not be removed by setting this option.

For more information about the new locale options, see Macrovision Knowledge Base article **Q113630**.

## Bidirectional Text Support

InstallAnywhere 2008 introduces support for bidirectional text by supporting the Arabic and Hebrew locales, which display text right-to-left.

The following is the Arabic version of the Introduction panel.



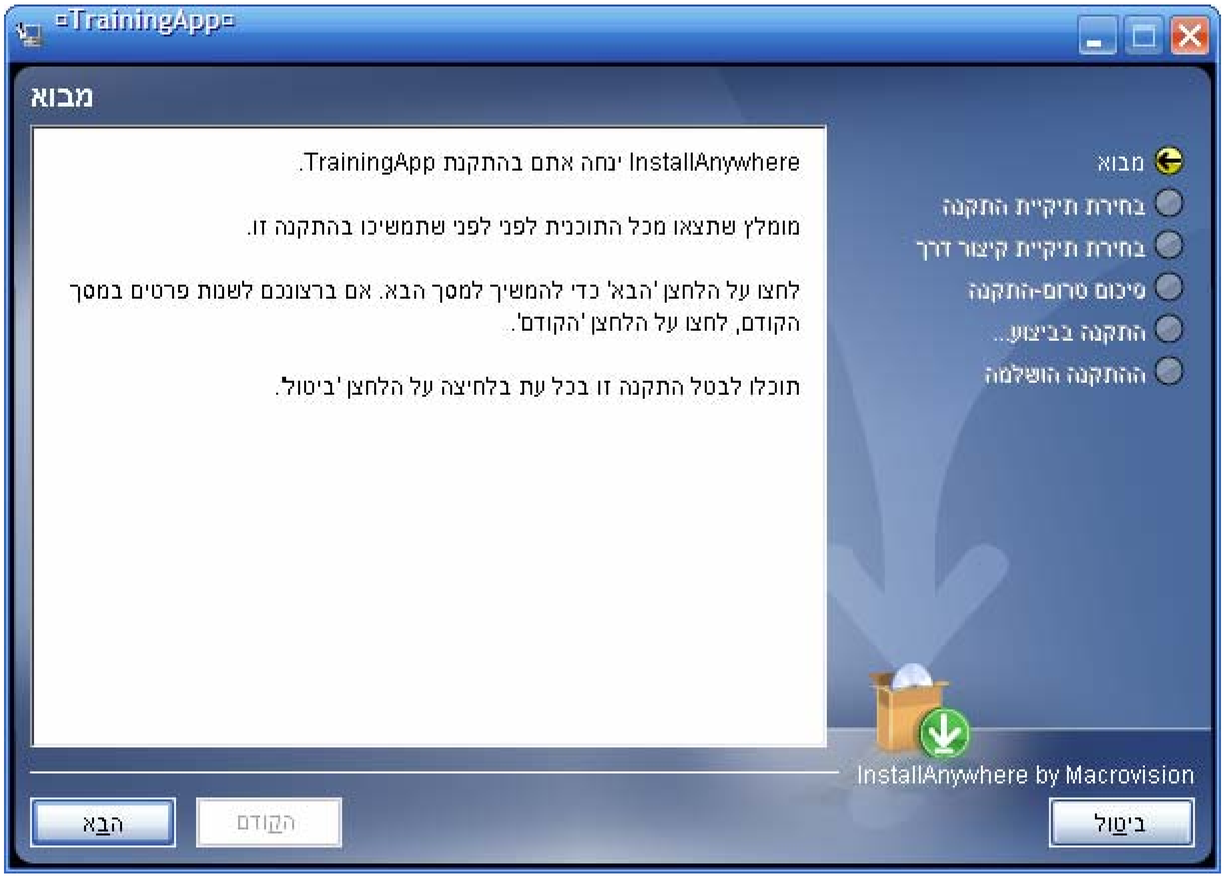
سيقوم InstallAnywhere بارشادك أثناء عملية تركيب TrainingApp.

ومن المفضل أن تقوم بالخروج من كل البرامج قبل الاستمرار في عملية التركيب هذه.

اضغط 'تالي' للانتقال الى الشاشة التالية. اذا كنت تريد تغيير أي شيء بالشاشة السابقة، اضغط 'سابق'.

يمكنك الغاء عملية التركيب هذه في أي وقت من خلال الضغط على 'الغاء'.

And the following is the Hebrew version of the same panel.



InstallAnywhere ינחה אתם בהתקנת TrainingApp.

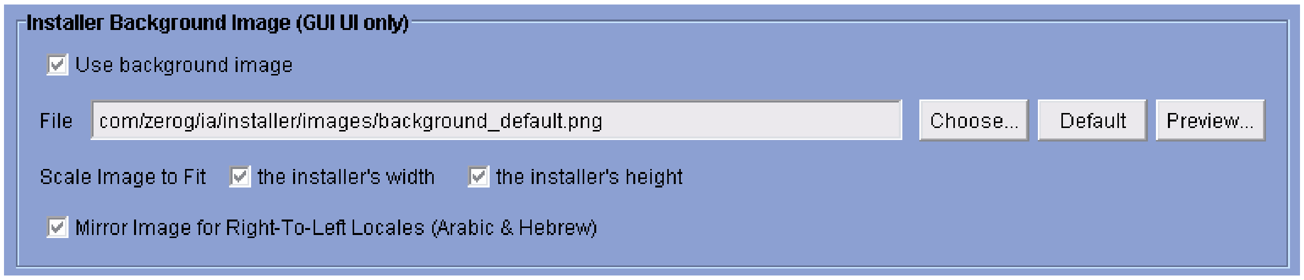
מומלץ שתצאו מכל התוכנית לפני לפני שתמשיכו בהתקנה זו.

לחצו על הלחצן 'הבא' כדי להמשיך למסך הבא. אם ברצונכם לשנות פרטים במסך הקודם, לחצו על הלחצן 'הקודם'.

תוכלו לבטל התקנה זו בכל עת בלחיצה על הלחצן 'ביטול'.

Note that right-to-left text is supported only in GUI mode, and not in console mode.

As illustrated in the following figure, the default installer behavior is to flip the UI background image when running on a right-to-left locale. You can override this behavior with the Mirror Image for Right-to-Left Locales setting in the general Look & Feel task.



Installer Background Image (GUI UI only)

☑ Use background image

File com/zerog/ia/installer/images/background_default.png

Scale Image to Fit ☑ the installer's width ☑ the installer's height

☑ Mirror Image for Right-To-Left Locales (Arabic & Hebrew)

## Dynamic and Static Text

Text that can be entered into the InstallAnywhere Advanced Designer is dynamic text. Dynamic text means that the developer can change its value. Text that cannot be edited by the Advanced Designer is referred to as static text. Standard items such as file choosers as well as text for actions and panels the developer is unlikely to wish to change have static text. Dynamic text is written to the locale files (located next to the project file). Translations of static text can be found in `<InstallAnywhere>\resource\i18nresources` directory. While developers are unlikely to need to change the static text, InstallAnywhere provides the developer with the option to change the static text.

Almost all dynamic text has default values in the Advanced Designer. These dynamic values also have default translations. Every string that is modified in the Advanced Designer needs to be localized by the installer developer if they want the translation to match. There are also actions that do not have defaults such as custom code and Get User Input panels. It is the developer's job to localize these strings as well.

## Localization and the Internationalized Designer

Developers can localize, not just the installers they create with the InstallAnywhere development environment, but the development environment itself.

The designer writes all changes of dynamic values into the locale file of the same language the Advanced Designer is running in. When using the French language variation, dynamic changes are written into the `custom_fr` file. Using the Advanced Designer is the correct way to modify this locale file. Changes to the locale files made outside of the Advanced Designer will be overwritten.

## Specific Localization Concerns

### Localizing Resources

The developer may find that they want to localize resources (such as License Agreements, side panels, billboards, and custom icons) for specific countries. Actions such as the License Agreement Panel and LaunchAnywhere serialize the paths and filenames to their resources as well as their dynamic strings to the locale files. The developer can then change these paths and the filenames. For example, to localize the License Agreement:

Make sure to include every resource in the Install task. Installers will not have access to resources not specified in this task.

Find the line in the locale file that contains the text `LicenseAgr.#.FileName`. Specify the filename of the file that contains the localized license agreement (for instance, `License_fr.html`). Do not type the fully qualified absolute pathname to the file—just the filename itself.

Find the line that contains the text `LicenseAgr.#.Path`. Specify the path name to the file that contains the localized license agreement (on the local file system).

### Localizing Custom Installer Labels

Custom labels that match the installer panels are not automatically localized. To match labels:

1. *Build the installer as you normally would.*
2. *Locate the installer's Locale directory.*
3. *Open the* `custom_en` *file in WordPad or another text editor.*
4. *Search for the* `Installer.1.installLabelsAsCommaSeparatedString` *variable.*

   This variable should contain the added installer labels.
5. *Copy and paste this variable into the other locale files.*
6. *Finally, provide translations for these labels in their respective files.*

## Localizing Custom Code

The InstallAnywhere API provides a simple means for localizing custom code actions, panels, and consoles. Here's an example of how to localize a `java.awt.Label` inside a custom code panel. The custom code panel's setupUI method should appear similar to the following:

```
public boolean setupUI(CustomCodePanelProxy ccpp)
{
    Label myLabel = new Label( );
    myLabel.setText(ccpp.getValue(MyCustomCodePanel.myLabel));
}
```

Every CustomCodePanelProxy, InstallerProxy, CustomCodeConsoleProxy, and UninstallerProxy provides access to the getValue method. This method takes a String as a parameter representing the key portion of the key-value pair as defined in InstallAnywhere's international resource files. Developers can create any name for the key that they would like, as long as it does not conflict with previously defined keys. Developers can even use a pre-existing key to obtain a string that has already been translated in InstallAnywhere's resource files.

To have the new locale keys to exist in every installer project, update the static text. To have the keys only in the current project, update the dynamic text. The dynamic text is regenerated every time the developer saves, so update the files every time the project is changed. This is another good reason to have the installer design done before starting the localization process.

## Best Practices for Localizing

Complete the installer design before translating the locale files. Changes to the installer design can affect the layout and contents of the locale files. If these finals change, it may require costly re-translation work.

Stick with the default text whenever possible. All default text is already translated, saving the team time and effort.

Test the installers on systems running in the foreign locale. This will help shake out any errors where the proper strings are not translated in the locale files.

Make sure every resource referenced in the locale files is included in the installer. This is especially true for license agreements, readme files, and other commonly translated documents.

InstallAnywhere's comprehensive locale support is just one of the features that stand out from other deployment solutions. InstallAnywhere Enterprise Edition offer support for 31 different locales, both single-byte "Western" locales and double-byte locales. Nearly every text string in your InstallAnywhere installer project can be localized, and translations of all of InstallAnywhere's default text are already provided.

## Changing Localized Text

It is not necessary to change the localized text in the installer unless:

- You've added or modified installer items that are displayed to the end user during installation (new components, features, license agreements, important information notes, and so forth) that do not have default translations.
- You would like to modify the provided default translation.
- Are using actions that return Install Panels (such as Choose Folder Panel or Get Password Panel).

### Modifying Localized Text

1. *Include any localized files (license agreements, graphics for billboards, etc.) in your installer. If these are not included, they won't be included with the installer and won't be available when end users run the installer. Including these files in your installer means that, InstallAnywhere will use them during the install process, and also install them onto the destination system. If you don't want these files installed, place them in the* Do Not Install Magic Folder*, and they won't be placed onto the destination system.*
2. *Build your installer. After this first build, a folder named [YourProjectName]locales will be created in the same folder as the* `.iap_xml` *project file.*
3. *There will be a series of files called* `custom_en`*,* `custom_fr`*, etc. in this locales folder—one for each language you chose to build for on the* ***Project > Locales*** *task in the Advanced Designer. These are language resource files, and contain localized text strings as well as pointers to filenames containing localized information to embed in your installer.*
4. *For each language that you want to customize, edit the appropriate language resource file. Use escaped Unicode to encode for these files. For example, if you want to specify a custom license agreement for French, edit the* `custom_fr` *file.*

   See Appendix G, "Language Codes", for a list of locale abbreviations.

**NOTE:** Make sure that your localized license agreements and important note files are added to the installer (using the Install task); otherwise, they will not be bundled into the installer.

### Changing Default Translations Provided in Language Packs

1. *Follow the directions for changing localized text.*
2. *Modify the language resource files located in the resource\i18nresources folder inside of your InstallAnywhere folder. These files contain the defaults for ALL strings, both the static defaults and the strings that are externalized to the locales.*

## Localizable Elements

Installers deployed to non-Latin systems require an international Java Virtual Machine.

### Localizing Items in the Installer

See the list below to determine the correct properties to modify in the language resource files. This list does not include properties from many new actions. For a complete list, contact Macrovision Support.

| Property | Definition |
|---|---|
| `Installer.#.ProductName` | Name of product displayed on installer title bar. |
| `Installer.#.RulesFailedMessage` | Message displayed if specified rules prevent the installer from running. |
| `Installer.#.ShortcutDestinationPathMacOS` | Path to where aliases are created during installation on Mac OS, relative to the end-user-selected alias folder chosen on the Choose Alias Location step. |
| `Installer.#.ShortcutDestinationPathWin32` | Path to where shortcuts are created during installation on Windows, relative to the end-user-selected shortcut folder chosen on the Choose Shortcut Folder step. |
| `Installer.#.ShortcutDestinationPathSolaris` | Path to where links are created during installation on Unix, relative to the end-user-selected links folder chosen on the |

| | |
|---|---|
| | Choose Link Location step. |
| `InstallSet.#.Description` | Description of one of the installer’s features. |
| `InstallSet.#.InstallSetName` | Name of one of the installer’s features. |
| `IntroAction.#.message` | Text to display on the installer’s Introduction step. |
| `Intro.#.stepTitle` | Title to display on the installer’s Introduction step. |
| `LicenseAgr.#.FileName` | Name of localized license agreement to be displayed as the installer is preparing itself. (Note that this is a filename only, and not a fully qualified pathname.) |
| `LicenseAgr.#.Path` | Path name to localized license agreement to be displayed as the installer is preparing itself. (Note that this is only a pathname and does not include the filename.) |
| `LicenseAgr.#.Title` | Title of License Agreement step in the installer. |
| `MakeExecutable.#.destinationName` | Name of the LaunchAnywhere Executable to be created on the destination computer. |
| `MakeRegEntry.#.Value` | Value to be written to the Windows registry. |
| `ShortcutLoc.#.macTitle` | Title of Mac OS X Choose Alias Location step in the installer. |
| `ShortcutLoc.#.SolarisTitle` | Title of Unix Choose Link Folder step in the installer. |
| `ShortcutLoc.#.Win32Title` | Title of Windows Choose Shortcut Folder step in the installer. |
| `Billboard.#.ImageName` | Name of billboard image file to be displayed as the installer is preparing itself. (Note that this is a filename only, and not a fully qualified pathname.) |
| `Billboard.#.ImagePath` | Path name to billboard image to be displayed as the installer is preparing itself. (Note that this is only a pathname and does not include the filename.) |
| `ChooseInstallSet.#.Title` | Title of Choose Install Set step in the installer. |
| `ChooseJavaVM.#.Title` | Title of Choose Java Virtual Machine step in the installer. |
| `CreateShortcut.#.DestinationName` | Name of the shortcut/alias/link to be created on the destination computer. |
| `human.readable.language.name` | The name of the language represented by the data in this resource file (as in English, Español, etc.). |
| `ImportantNote.#.FileName` | Name of text file to be displayed on the Important Note step of the installer. (Note that this is a filename only, and not a fully |

| | |
|---|---|
| | qualified pathname.) |
| `ImportantNote.#.Path` | Path name to text file to be displayed on the Important Note step of the installer. (Note that this is only a pathname and does not include the filename.) |
| `ImportantNote.#.Title` | Title of Important Note step in installer. |
| `InstallBundle.#.BundleName` | Name of component. |
| `InstallBundle.#.Description` | Description text describing component. |
| `InstallComplete.#.DisplayText` | Text to display on the Install Complete step of the installer. |
| `InstallComplete.#.Title` | Title of the Install Complete step in the installer. |
| `InstallDir.#.Title` | Title of the Choose Installation Directory step of the installer. |
| `Installer.#.InstallerName` | Name of the installer. |

# Appendix A. Standard IA Variables

InstallAnywhere includes a number of default variables that store information essential to the installation process. The following variables are standard InstallAnywhere variables.

| Variable | Definition | Status |
|---|---|---|
| `$/$ or $\$` | These represent platform-specific file separators, most useful to refer to paths in a platform-independent manner. These have the same value as the Java property file.separator. | This variable is read-only. |
| `$;$ or $:$` | These represent platform-specific path separators. | This variable is read-only. |
| `$CHOSEN_DIALOG_BUTTON$` | This variable is set to the return value as set by the end-user's choice in the Show Message dialog box action. For example, if the end-user chooses button 1, this variable will be set to 1. | |
| `$CHOSEN_INSTALL_FEATURE_`*n*`$` | If Choose Product Features is enabled, one $CHOSEN_INSTALL_FEATURE$ variable is created for each feature as given in the variable $CHOSEN_INSTALL_FEATURE_-NUM$. These variables hold the short name of a feature to be installed. For example, if $CHOSEN_INSTALL_-FEATURE_NUM$ equals 2, then two variables of this form are created: $CHOSEN_INSTALL_FEATURE_1$ and $CHOSEN_INSTALL_FEATURE_2$ | |
| `$CHOSEN INSTALL FEATURE LIST$` | This variable is a comma-separated list of all install features (short name) chosen for the installation. | |
| `$CHOSEN_INSTALL_FEATURE_NUM$` | If Choose Product Features is enabled, this variable holds the total number of components (as a string) that the end-user chooses to install. | |
| `$CHOSEN INSTALL BUNDLE <#>$` | This variable is deprecated. Use $CHOSEN_INSTALL_FEATURE_*n*$ | This variable is read-only. |
| `$CHOSEN_INSTALL_BUNDLE_LIST$` | This variable is deprecated. Use $CHOSEN_INSTALL_FEATURE_-LIST$ instead. | This variable is read-only. |
| `$CHOSEN_INSTALL_BUNDLE_-NUM$` | This variable is deprecated. Use $CHOSEN_INSTALL_FEATURE_-NUM$ instead. | This variable is read-only. |

| | | |
|---|---|---|
| `$CHOSEN_INSTALL_SET$` | If Choose Product Features is enabled, this variable holds the short name of the install set chosen by the end user. If the end user chose to customize the install, this variable holds the string CUSTOM. | This variable is can be set before install time. |
| `$CMD_LINE_ARGUMENTS$` | A special InstallAnywhere variable resolved by the launcher and not by the installer. If this variable is in the LAX property lax.command.line.args, it will resolve to the arguments sent to the LaunchAnywhere executable. | This variable is used by LaunchAnywhere. |
| `$COMMA$` | This resolves to a comma (,). | This variable is read-only. |
| $*componentname* DEPEND-ENCY_STATE_VARIABLE$ | If the dependency check passes, this variable will be an empty string. If the dependency check fails, the variable contains the Dependency Failed Message. | This variable is set at the beginning of the Install phase or after the Evaluate Dependencies action in Pre-Install. |
| $*componentname*_-MATCHED_KEY_FILE$ | This variable contains the location of where the key file of the dependency is installed. If the dependency check has failed, this variable contains an empty string. | This variable is set at the beginning of the Install phase or after the Evaluate Dependencies action in Pre-Install. |
| `$DEPENDENCY_FAILURES$` | This is a comma-separated list of Dependencies that have failed and are not installed. | This variable is set at the beginning of the Install phase or after the Evaluate Dependencies action in Pre-Install. |
| `$DEPENDENCY_SUCCESSES$` | This is a comma-separated list of Dependencies that have passed and are installed. | This variable is set at the beginning of the Install phase or after the Evaluate Dependencies action in Pre-Install. |
| `$DEPENDENCY_STATUS$` | Returns “success” or “failure”, depending on the entire dependency evaluation. If any of the dependencies in the installer fail, this variable is set to “failure”. | This variable is set at the beginning of the Install phase or after the Evaluate Dependencies action in Pre-Install. |
| `$DEPENDENCY_REPORT$` | This variable contains a report of all the dependencies that have failed. | This variable is set at the beginning of the Install phase or after the Evaluate Dependencies action in Pre-Install. |
| `$DEVELOPER DISK SPACE ADDITIONAL$` | This variable specifies an arbitrary additional string value, representing additional bytes that the Check Disk Space action adds to the computed required disk space for the installation. By default this variable has a value of zero. | Developers may set this variable. |
| `$DOLLAR$` | This resolves to $. | This variable is read-only. |

| | | |
|---|---|---|
| `$EMPTY_STRING$` | This variable represents a value of null. This is useful to determine whether or not any variables have been initialized. Variables that have not yet been initialized will have this as their value. | This variable is read-only. |
| `$EXTRACTOR_DIR$` | Full path to the directory containing the self-extractor executable (from where it was launched). | This variable is read-only. |
| `$EXTRACTOR_EXECUTABLE$` | Full path to the self-extractor executable (from where it was launched). | This variable is read-only. |
| `$FEATURE UNINSTALL LIST$` | Contains a comma-separated list of features the user chose to uninstall. | This variable is read-only. |
| `$FREE_DISK_SPACE_BYTES$` | The free disk space available on the destination install volume, as a string representing the free bytes, as determined by the Check Disk Space action. The variable gains its value immediately before the installation of any files or folder listed in the Install task. | This variable is read-only. |
| `$IA_CLASSPATH$` | The classpath as specified in the InstallAnywhere Designer environment. | Developers may set this variable in the Advanced Designer. |
| `$IA_INSTALL_LOG$` | Setting this variable will generate an XML-formatted installation log to the $USER_INSTALL_DIR$ location. This document will detail the installation along with warnings and errors. | This variable is read-only. |
| `$INSTALL_LOG_-DESTINATION$` | As the creation of the install log is the last action of an installation, this variable can be set anytime during pre-install, install, or post-install. The end-user input can change the installation log location. | Developers may set this variable. |
| `$INSTALL_LOG_NAME$` | This variable sets the name of the installation log. If not set, the installation log's name is based on the product name. | Developers may set this variable. |
| `$INSTALL_SUCCESS$` | This variable alerts the end user if the installation was successfully completed or if it failed. There are four possible values for this variable: SUCCESS, WARNING, NONFATAL_ERROR, and FATAL_ERROR. | This variable is read-only. |
| `$INSTALLER_LAUNCH_DIR$` | Full path to the installer's self-extractor. | This variable is read-only. |

| | | |
|---|---|---|
| `$INSTALLER_LOCALE$` | The locale as a string (see java.util.Locale.toString) that the end-user selected at the beginning of the installation. | This variable is read-only. |
| `$INSTALLER_TEMP_DIR$` | The path to a temp directory for use by an installer. This will be deleted when the installation has completed, assuming no items are in use. | This variable is read-only. |
| `$INSTALLER_TITLE$` | Title of the installer displayed in the title bar. | You may set this variable. |
| `$INSTALLER_UI$` | This resolves to the UI mode for the installer. | This variable is read-only, but developers may set it at the start. |
| `$JAVA_DOT_HOME$` | This is what the Java system property java.home will report. | This variable is read-only. |
| `$JAVA_EXECUTABLE$` | This is the path to the chosen Java executable. | This variable is read-only. |
| `$JAVA_HOME$` | This is the root of the Java installation. | This variable is read-only. |
| `$JDK_HOME$` | This is the path to the root of a JDK installation. The variable is set only if the chosen VM is a JDK. If it is not a JDK, this variable is blank. | This variable is read-only. |
| `$lax.nl.env.-`*`ENVIRONMENT_VARIABLE_-NAME$`* | (Windows and Unix only.) Access any system environment variable (for example, access PATH via $lax.nl.env.PATH$) by specifying the variable name as an all-uppercase string. These variables are resolved at application run time, when LaunchAnywhere executes. Developers can also access system environment variables using LaunchAnywhere properties. | This variable is read-only. |
| `$lax.nl.env.exact case.-`*`Environment Variable -Name$`* | (Windows and Unix only.) Access any system environment variable (for example, access Path via $lax.nl.env.exact_case.Path$) by specifying the variable name as a string of the exact case as it is defined in the environment. Note that these variables are resolved at application run time, when LaunchAnywhere executes. Developers can also access system environment variables using LaunchAnywhere properties. | This variable is read-only. |

| | | |
|---|---|---|
| $NEVER_UNINSTALLS_VM$ | Tells the uninstaller never to uninstall a Java Virtual Machine. This is useful when multiple projects and uninstallers share one virtual machine. | Set this variable to "true" to never uninstall a JVM. |
| $NULL$ | Equivalent to $EMPTY_STRING$. | This variable is read-only. |
| $PRODUCT_ID$ | Resolves to the value of Product ID in the Project > Description task. | This variable is read-only. |
| $PRODUCT_NAME$ | This is the product name. | Developers may set this variable. |
| $PRODUCT VERSION NUMBER$ | Resolves to the value of Product Version in the Project > Description task. | This variable is read-only. |
| $PROMPT USER CHOSEN - OPTION$ | This variable reflects the return value set by the end user's choice in the Show Message Console dialog box action. If the end user chooses Option 1, this variable is set to 1. | This variable is read-only. |
| $prop.*JAVA_PROPERTY*$ | Access any Java property through InstallAnywhere Variables. An example is $prop.os.name$, which returns the value of the os.name property. | This variable is read-only. |
| $REGISTER UNINSTALLER WINDOWS$ | This variable tells the uninstaller if it should register itself with Windows Add or Remove Programs. | Set this variable to "false" to not have the uninstaller register. |
| $REQUIRED_DISK_SPACE_ BYTES$ | The disk space required by the installer, as a string representing the required bytes, as determined by the Check Disk Space action. The variable gains its value immediately before the installation of any files or folder listed in the Install task. | This variable is read-only. |
| $RESTART_NEEDED$ | Tells the installer or uninstaller if the system needs to restart to complete the installation. | This variable is read-only. |
| $SHORTCUT_NAME$ | This variable resolves to "Shortcut" on Windows systems, "Alias" on Mac OS X systems, and "Link" on all other systems. | This variable is read-only. |
| $UNINSTALL_STATUS$ | The same as $INSTALL_STATUS$, but for the uninstaller. There are two possible values for this variable: SUCCESS or INCOMPLETE. | This variable is read-only. |

# Appendix B. InstallAnywhere-Provided Magic Folders

| Folder Name | InstallAnywhere Variable | Destination |
|---|---|---|
| User Installation Directory | `$USER_INSTALL_DIR$` | The installation folder, as specified by the end user. Developers can specify a default value for this variable in the Project Info screen in the Advanced Designer and choosing a location in the Default Install Folder area of the screen. |
| Programs Folder (Platform Default) | `$PROGRAMS_DIR$` | The default application directory on the destination system. (Program Files on Windows, in the Applications folder on Mac OS X, and the logged-in end user's home account on Unix.) |
| Programs Folder (32-bit) | `$PROGRAMS_DIR_32$` | The default application directory on a system. |
| Programs Folder (64-bit) | `$PROGRAMS_DIR_64$` | The default application directory on a 64-bit system. |
| Shortcuts | `$USER_SHORTCUTS$` | The folder specified by the end user as the shortcuts/links/aliases location. The value of this location can be changed by the end user if the Choose Alias, Link, Shortcut Folder action is turned on in the installer. Developers can specify a default value for this variable on a per-platform basis by selecting the Platforms task in the Advanced Designer. |
| System | `$SYSTEM$` | This variable represents the System folder on the target machine. |
| System Folder (64-bit) | `$SYSTEM_64$` | The default 64-bit System folder on a 64-bit system. |
| System Folder (32-bit) | `$SYSTEM_32$` | The default 32-bit System folder on a 32-bit system. |
| Desktop | `$DESKTOP$` | This variable represents the desktop on the target machine. This folder only resolves on Windows, Linux, and Mac systems. |
| Temp Directory | `$TEMP_DIR$` | This variable represents the temp directory on the target machine. When running the pure Java installer on Windows, `$TEMP_DIR$` will resolve the user's home directory. |
| Startup | `$STARTUP$` | The automatic startup folder for items that are launched automatically during operating system boot up. This folder only resolves on Windows systems. |
| Installation Drive Root | `$INSTALL_DRIVE_ROOT$` | The root directory on the volume where the installation is taking place. |
| Home Directory | `$USER_HOME$` | The home directory of the end user running the installer. For users who have already included |

| | | |
|---|---|---|
| | | the variable `$UNIX_USER_HOME$`, this variable will continue to function with the same definition as `$USER_HOME$`. |
| System Drive Root | `$SYSTEM_DRIVE_ROOT$` | The root directory of the system drive. |
| Java Home | `$JAVA_HOME$` | The home directory of the Java Virtual Machine to be used. |
| Windows | `$WIN_WINDOWS$` | The Windows directory (Windows systems only). |
| Start Menu | `$WIN_START_MENU$` | The Windows Start menu directory (Windows systems only). |
| Quick Launch Bar | `$WIN_QUICK_LAUNCH_BAR$` | The Quick Launch Bar on Windows. On Windows 2000 and newer, the location is relative to the UserProfile environment variable. |
| Do Not Install | `$DO_NOT_INSTALL$` | Doesn’t install the file on the target platform. Used for files (typically localized license agreements and graphics) used during installation but which don’t need to remain on the target system. |
| `USER MAGIC FOLDER_`*`n`* | `$USER_MAGIC_FOLDER_`*`n`*`$` | These variables are user-defined install-destination Magic Folders. They install to whichever directories their variable name has been set. To set these variables, use the Set InstallAnywhere Variable action.<br>Note that for Unix, if the leading “`/`” is not included in the path before the Magic Folder location, the result is `<installer location>/USER_MAGIC_FOLDER_`*`n`*. This is because the operating system assumes that any path not preceded by a “/” is below the current directory. |
| Programs Menu | `$WIN_PROGRAMS_MENU$` | The Windows Programs menu (in the Start menu). |
| All Users Start Menu | `$WIN_COMMON_START_MENU$` | The Windows All Users Start menu directory. |
| All Users Programs Menu | `$WIN_COMMON_PROGRAMS_-MENU$` | The Windows All Users Programs menu (in the Start menu). |
| All Users Startup | `$WIN_COMMON_STARTUP$` | The Windows All Users Startup folder in the Start menu (Windows systems only). |
| All Users Desktop | `$WIN_COMMON_DESKTOP$` | The Windows all-users Desktop folder. |
| Fonts | `$FONTS$` | The Fonts directory on Windows computers; typically `$WIN_WINDOWS$\Fonts`. |
| User Applications | `$MACX_USER_APPLICATIONS$` | The User Applications directory of the end user running the installer (Mac OS X only). |
| The Dock | `$MACX_DOCK$` | The Mac OS X Dock (for Shortcuts only). Other types of iles cannot be installed to the Dock. |
| `/usr/local/bin` | `$UNIX_USR_LOCAL_BIN$` | The `/usr/local/bin` directory (Unix computers only). |

| | | |
|---|---|---|
| `/usr/bin` | `$UNIX_USR_BIN$` | The `/usr/bin` directory (Unix computers only). |
| `/opt` | `$UNIX_OPT$` | The `/opt` directory (Unix computers only). |

**NOTE:** Developer-defined Magic Folders are not available in the Standard Edition.

# Appendix C. Actions

## *Install Actions*

| Action | Editions | Description |
|---|---|---|
| Create Alias, Link, Shortcut | E S | Create an alias (Mac OS X), symbolic link (Unix and Linux), or shortcut (Windows). |
| Create Alias, Link, Shortcut to DIM File | E | Create an alias (Mac OS X), symbolic link (Unix and Linux), or shortcut (Windows) to a file within a Developer Installation Manifest (DIM). |
| Create DIM Reference | E | Create a reference to a DIM. |
| Create Folder | E S | Create a new folder on the end user's system. If the folder already exists, it will not delete the existing folder. |
| Create LaunchAnywhere for Java Apps | E S | Create a launcher to start the installed Java application. See the expanded section on LaunchAnywhere below for more information. |
| Create Uninstaller | E S | Create the uninstaller and several additional files needed by the uninstaller. It is recommended that the uninstaller be installed into its own folder.<br>Install Anywhere automatically creates an uninstaller. Use the Create Uninstaller customizer to edit uninstaller settings. |
| Delete File | E | Delete a file from the end user's system. |
| Delete Folder | E | Delete a folder from the end user's system. |
| Deploy WAR/EAR Archive | E | Deploy a WAR or EAR archive to an application server. This action requires an Application Server host on the Organization > Hosts subtask. |
| Enable Update Notifications | E S | Include FLEXnet Connect with the installation, which enables you to communicate updates and other critical information to your end users. |
| Expand Archive | E S | Expand a ZIP file (.zip, .jar, .war, or .ear file) or decode a Mac Binary file (.bin) on the end user's system. |
| Install Archive | E S | Install a ZIP file (.zip or .jar file) on the target system. This action is not available from the Choose an Action dialog box, but its customizer appears automatically when a ZIP file is added. |
| Install File | E S | Install a file from the installer onto the end user's system. The action is not available from the Choose an Action dialog box, but its customizer appears automatically when files are added. |
| Install from Manifest | E | Install all of the files and folders specified in the manifest file on the end user's system. See the expanded section on Manifest files for more information. |
| Install HP-UX Depot | E | InstallAnywhere can install and uninstall HP-UX depot files through the Install HP-UX Depot action. The developer needs to specify the package name within the depot file, as they can contain multiple packages. In order to install multiple packages in the same depot, add one action for every package you want to install. This will not increase the size of the installer. |

| | | |
|---|---|---|
| Install Linux RPM | E | InstallAnywhere can install and uninstall Linux RPMs through the Install Linux RPM action. These RPMs can either be bundled with the installer, or pre-existing on the system. If the RPM is relocatable, and the Relocatable checkbox is set in the action customizer, the RPM will be installed to its location in the file tree. Additionally, the RPM can be set to ignore dependencies (similar to the `--nodeps` option for the command line RPM tool) and to force the installation (`--force`). |
| Install Merge Module | E | Install a Merge Module as if the Merge Module were run as a separate silent installer. |
| Install Solaris Package | E | InstallAnywhere can install and uninstall Solaris package files through the Install Solaris Package action. These packages can either be bundled with the installer, or pre-existing on the system. The developer must enter the name of the package. Additionally, InstallAnywhere supports bundling admin and response files for the packages with the installer. For more information on these files, consult the man pages for `pkgadd(1)`, `pkgask(1)`, and `admin(4)`. |
| Install SpeedFolder | E S | Dynamically pick up files from a folder at build time. All files will be installed as one single fast operation. TIP: Use SpeedFolders for large installations with many files, or builds that occur automatically. |
| Move File | E | Move or rename a file from one location to another location on the end user's system. |
| Move Folder | E | Move or rename a folder from one location to another location on the end user's system. |
| Run SQL Script | E | Runs an SQL script on a database server. This action requires an existing Database Server host on the Organization > Hosts subtask. |
| Set System Environment Variable | E | Set environment variables on the end user's system. Compatible with Windows and Unix only. Unix Bash, sh, ksh, zsh, csh, tcsh shells are supported. |

## *General Actions*

| **Action** | **Editions** | **Description** |
|---|---|---|
| Action Group | E | Adds a folder to the Action list in which you can group sets of InstallAnywhere actions. Available in Pre-Install, Post-Install, Pre-Uninstall, and Post-Uninstall steps. |
| Add Jump Label | E | Use this action to branch off the installation conditionally. By applying InstallAnywhere rules, developers may jump the end user to a later or earlier part of the installation, depending on the specifics of their system or install. Use this action in conjunction with the Jump to Target action. *Available during the Pre-Install and Post-Install sequences, but not within the files.* |
| Comment | E S | The Comment action enables developers to add a simple comment to the installer. |
| Copy File | E | Copy a file from one location to another location on the end |

| | | |
|---|---|---|
| | | user's system. |
| Copy Folder | E | Copy a folder from one location to another location on the end user's system. |
| Evaluate Dependencies | E S | This action evaluates the state of the dependencies that this installer may rely on. Evaluate Dependencies sets the following variables: $DEPENDENCY_SUCCESSES$, $DEPENDENCY_FAILURES$, $DEPENDENCY_REPORT$, and $DEPENDENCY_STATUS$. |
| Execute Ant Script | E | Execute Ant Script enables developers to execute scripts designed for the Apache Jakarta Project's Ant application. If this action is selected, Ant will be bundled with the application. This action should only be used by developers familiar with Ant. For more information, visit http://ant.apache.org. |
| Execute Command | E | Execute Command enables developers to execute a command as they would at any Command Line Interpreter. This action is useful for executing applications that are already installed on the system. The command line is entered just as it would be in the system's command line. |
| Execute Custom Code | E S | Execute Custom Code is designed to enable developers to extend the functionality of InstallAnywhere. InstallAnywhere's API is purely Java based and enables developers to do nearly anything that is possible in Java. Execute Custom Code action represents the non-interactive interface for this API. For more information, see the Custom Code section below. |
| Execute Script/Batch File | E | This action enables developers to enter the text to a script or batch file which will them be executed from within the installation. |
| Execute Target File | E S | Launches any executable or opens a document that is included in the installer. If the target is a document that has the appropriate application associations set up, then the document will be opened in the correct application. Available only during the installation of files and after files have been installed. |
| Execute Uninstaller | E | Runs an uninstaller that the developer specifies. Execute Uninstaller enables developers to set the uninstaller's UI mode and store the stdout, stderr, and exit code from that process. |
| Find Component in Registry | E | Find if a component exists on a system through the cross-platform registry, as well as discover existing component versions, their location, and if there are multiple instances of a particular component on the destination system. |
| Get Windows Registry Entry | E | InstallAnywhere allows developers access to information stored in the Windows Registry through this action. This action allows developers to retrieve the value, or check the existence of a key/value and store that information in InstallAnywhere variables to be used in the installation. |
| Jump to Target | E | Related to the "Add Jump Label" action seen earlier, this action allows developers to jump over or back to a specific point in an installation. When controlled by InstallAnywhere rules, this action gives developers a conditional method of moving non-linearly through an install. *Available during the Pre-Install and Post-Install sequence tasks.* |

| | | |
|---|---|---|
| Launch Default Browser | E S | This action allows developers to launch the user's default web browsers with the specified arguments. It can open a URL or a file on the system. *Available during the sequence tasks.* |
| Modify Text File - In Archive | E | The three modify text file actions are designed in to allow developers to alter text files on the installation system. These actions can be used to manage configuration files, or in the case of the "Multiple Files" option, change the line ending of a large number of files. |
| Modify Text File - Multiple Files | E | This action allows developers to alter text files on the installation system. These can be used to manage configuration files, or in the case of the "Multiple Files" option, change the line ending of a large number of files. |
| Modify Text File - Single File | E | See Modify Text File - Multiple Files. |
| Output Debug Information | E S | InstallAnywhere has comprehensive debugging built into the installer. By running in "Debug Mode" developers can diagnose many issues with installers. This action allows developers to output specific information to either the console, or a file. Developers can output the entire contents of the InstallAnywhere variable manager, the install tree, Java Properties, and other information related to the installation. |
| Output Text to Console | E S | This action outputs the text specified to the debug console. This is useful for measuring the progress of a non-interactive installer in silent or console mode, or the progress of a non-interactive portion of the installation. |
| Perform XSL Transform | E | This action allows developers to specify an XSLT and target for an Extensible Stylesheet Language transform. Predefined XSL Transforms can be found in `<InstallAnywhere>\resource\extras\presets`. |
| Perform XSL Transform - In Archive | E | The Perform XSL Transform - In Archive action works the same as the Perform XSL Transform, but does so for files in an archive. This action is useful for configuring web applications in WAR, EAR, and JAR files. |
| Query InstallShield Universal Software Information | E | This action returns information on InstallShield Universal software based on UUID, version, install location, and so on. |
| Register Windows Service | E | Start, stop, or pause a Windows Service on the end user's system. |
| Restart Windows | E S | This action will restart a Windows system. The system will reboot as soon as this action is reached, so it should be used carefully and only in conjunction with rules. |
| Set InstallAnywhere Variable - Multiple Variables | E S | The root of ANY InstallAnywhere installation, these actions allow developers to specify values for, and or create InstallAnywhere Variables. This action is used throughout the installation, and can be used to control nearly any aspect of the installation. |
| Set InstallAnywhere Variable - Single Variable | E S | See Set InstallAnywhere Variable - Multiple Variables. |
| Set Windows | E | Set multiple Windows registry keys, data, and values on the end |

| | | |
|---|---|---|
| Registry - Multiple Entries | | user's system. |
| Set Windows Registry - Single Entry | E S | Set an individual Windows registry key, data, and value on the end user's system. |
| Show Message Dialog | E | This action creates a modal dialog that requests end-user input. The message dialog box will appear over the currently displayed panel, and can be used to force the end user to return to the previous panel, exit the installer, or input information. When controlled by rules, it can be used as a data verification tool. |
| Start, Stop, Pause Windows Service | E | If the application is interacting with a Windows Service, the installer may need to manage that service. This action, when the installer is run with sufficient privileges, allows the installer to stop, start, or pause registered windows services. |
| Uninstall InstallShield Universal Software | E | This action allows developers to identify installed InstallShield Universal software and remove it. |

## *Panel Actions*

| **Action** | **Editions** | **Description** |
|---|---|---|
| Choose Alias, Link, Shortcut | E S | Added as part of the default project, this panel allows the end user to choose an installation location for "shortcuts" (Windows), "aliases" (Mac), and "links" (Unix). |
| Choose Features to Uninstall | E | This panel allows the end user to select which features they want to uninstall. *Available in the Pre-Uninstall task.* |
| Choose File | E | Choose File allows installers to request that the user select a file on certain criteria and set its result as an InstallAnywhere Variable. The variable can then be used later in the Install. |
| Choose Folder | E | Choose Folder allow developers to request that the user select a folder on certain criteria and set its result as an InstallAnywhere Variable that can be used later in the Install. |
| Choose Install Folder | E S | Part of the default project, This panel allows the user to choose the primary installation folder. It is not necessary that this panel is included, as without the panel, the installer will select the default specified in the Project > Platforms task. |
| Choose Install Set | E S | This panel allows developers to request that the end user choose an install set, or features to install. |
| Choose Java VM | E S | This panel allows developers to have the end user select the Java VM that will be used for any installed LaunchAnywhere Launchers. Developers can specify the type of VM that the end user should select, and the panel will search the system for an appropriate VM. |
| Choose Uninstall Type | | Allows the end user to select whether to install all or part of the application. *Available in the Pre-Uninstall task.* |
| Custom Code Panel | E | InstallAnywhere's Custom Code API enables developers to create custom panels where necessary. |
| Disk Space Check | E S | This action performs a disk space check on the installation destination system based on the end user's chosen install location and the end user's chosen feature. If there is not enough disk space to perform the install, then the installer will |

| | | |
|---|---|---|
| | | prompt the end user to free the required disk space or choose another install location. *Automatically added before files are installed. The action does not appear in the list.* |
| Display HTML | E S | The Display HTML panel allows developers to display HTML from an archived file or a specific URL on a panel during the installation. Use the General Settings tab to assign the HTML panel a title and select the source of the HTML.<br>A sample custom HTML panel is available in the Custom Code/Samples/HTMLPanelSample folder of the InstallAnywhere installation directory. |
| Display Message | E S | The Display Message Panel allows developers to simply display a text message to the end user during the installation. This can be useful for conveying information about installation choices that the end user has made. This panel is also particularly useful in debugging installer issue having to do with InstallAnywhere Variables. |
| Find File/Folder | E | This panel implements a search process that, depending on specifications made by the developer, will search portions of the file system for a specific named file, or a file matching a certain pattern. The end user can also choose a matching file. |
| Get Password | E | This panel allows developers to request a password from the end user. Developers can choose to validate the password against a list of specified passwords (enabling the index feature which allows different passwords to effectively unlock different features) or they can simply store the entered password in a variable (as when requesting a password to be used in a configuration routine). |
| Get Serial Number | E | Implementing InstallAnywhere’s built-in serial number verification and creation routines; this action/panel allows developers to add serial number functionality to the installer. Developers can choose to generate any number of serial numbers for any number of products. Serial numbers can represent unique products, or sets of products. The result of this action allows developers to create rules that can manage all aspects of the installation based on rights granted by the serial number the end user has entered. |
| Get User Input - Advanced | E | The older, smarter, and more capable brother of the Get User Input Panel, this action allows developers to get input from the user using multiple input types, and setting multiple variables. This action can have radio buttons, check boxes, text fields, and menus—all on the same panel. |
| Get User input - Simple | E | This action allows developers to request input from the end user. |
| Important Note | E S | The important note panel allows developers to display a text or HTML file without the radio buttons found on the license agreement panel. It is particularly useful for displaying Readme or errata type documents. |
| Install Complete | E S | This action displays information about the installation’s status to the end user. It also optionally displays if a restart is needed on Windows system. Available only after files have been installed. |

| | | |
|---|---|---|
| | | *Available in the Post-Install task.* |
| Introduction | E S | Part of the default project, this panel offers an introduction to the installation. |
| License Agreement | E S | This panel allows developers to display a license agreement to the end user. The end user must choose to accept the agreement in order to continue. Developers can set the default state of the radio buttons (Accept or Decline) and choose a file to use for a license agreement. The License Agreement Panel can also utilize HTML files. This option gives developers a degree of control over the text formatting and allows them to link to external documents. |
| Pre-Install Summary | E S | This panel summarizes information collected and evaluated prior to the installation of files. It allows the developer to customize what information is presented. It is included in the default InstallAnywhere Project. |
| Scrolling Message | E | This panel allows developers to enter long text in a message panel that includes scroll bars. This is particularly useful for instructions. |
| Uninstall Complete | E S | Displays information that the uninstaller has completed. *Available in the Post-Uninstall task.* |
| Uninstaller Introduction | E S | This panel offers an introduction to the installer. *Available in the Pre-Uninstall task.* |

## Console Actions

Console Actions (commonly called Consoles) are the means for requesting installer end-user input when using a command line interface. When the end user selects a console installation, console actions will be used instead of panel actions.

| Action | Editions | Description |
|---|---|---|
| Choose Features to Uninstall | E | This console allows the end user to select which features they want to uninstall. |
| Choose Install Folder | E | Chooses the primary installation location. |
| Choose Install Sets | E | This console allows developers to request that the end user choose an install set, or features to install. |
| Choose Java VM | E | This console allows developers to have the end user select the Java VM that will be used for any installed LaunchAnywhere Launchers. Developers can specify the type of VM that the end user should select, and the panel will search the system for an appropriate VM. |
| Choose Link Folder | E | This console allows the user to determine where to install Unix links. |
| Choose Uninstall Type | E | Allows the end user to select whether to install all or part of the application. |
| Custom Code | E | InstallAnywhere’s Custom Code API allows developers to create custom consoles where necessary. |
| Display Message | E | The Display Message console allows developers to simply display a text message to the end user during the installation. |
| Get Password | E | This console allows developers to request a password from the end user. |

| | | |
|---|---|---|
| Get Serial Number | E | This console allows end users to generate a list of serial numbers as well as request them from the end user. |
| Get User Input | E | This console allows developers to request input from the end user. |
| Install Complete | E | This console displays information about the installation's status to the end user. It also optionally displays if a restart is needed on a Windows system. Available only after files have been installed. |
| Install Failed | E | This console should be displayed when a console installer has generated an error. |
| License Agreement | E | This console allows developers to display a license agreement to the end .user. |
| Pre-Install Summary | E | This console summarizes information collected and evaluated prior to the installation of files. |
| Ready to Install | E | This console alerts the end user that the installer is about to install files. |
| Show Message Console 'Dialog' | E | Displays a message 'dialog' to the end user. |
| Uninstall Complete | E | Displays information that the uninstaller has completed. |
| Uninstaller Introduction | E | This console offers an introduction to the installer. |

## *Common Properties*

The following list contains common properties found in action customizers in InstallAnywhere.

| **Property** | **Description** |
|---|---|
| Comment | Sets the name of the action in the visual tree |
| Do not uninstall | Tells an action to not attempt to undo the results of the action at uninstall time |
| If file already exists on end user's system | Overrides the default behavior for how to resolve conflicts between installed files and pre-existing files |
| In Classpath | Puts the item on the classpath for all LaunchAnywhere executables installed |
| Installed File/Existing File | Determines whether the file is being installed or already exists on the end user's system |
| Override default Unix/Mac OS X permissions | Sets the file permissions to a specific value for this action |
| Path | Shows the path where the action will be installed |
| Show Indeterminate Dialog | Brings up an indeterminate progress bar to show progress to the end user while a external process is executing |
| Source | Shows the path where the item currently exists on the developer's system (displays the source path if source paths are being used) |
| Store process's exit code in | Sets the value of the InstallAnywhere Variable to the process exit code |
| Store process's `stderr` in | Sets the value of the InstallAnywhere Variable to the process standard error |
| Store process's `stdout` in | Sets the value of the InstallAnywhere Variable to the process's standard out |
| Suspend installation until | Pauses the installer until the launched process completes |

| |
|---|
| process completes |

## *Panel Action Settings*

Panel Actions (commonly called Panels) are the means for requesting user input through a graphical interface.

Graphic Installers may show the installation steps through a set of labels—words which represent the step. Installers may also display specific images for the steps. When Images is selected in the Installer UI > Look & Feel > Installer Panel Additions > Type of Additions to Installer Panels, the customizer for the panel in the Pre-Install and Post-Install task will enable the use of the Image Settings tab. If List of Installer Steps is selected the Label Settings tab will be enabled.

NOTE: These settings are unavailable to panel actions in the Uninstaller. Panel actions in the Uninstaller use the default values set in the Installer UI > Look & Feel task.

### Image Settings

Use panel image settings to choose a specific image to display on the chosen panel. Developers may choose to use the default panel image, display an image specific to that panel, or display no image at all.

### Label Settings

The Label Settings tab in the customizer enables developers to preview the labels and the icon images. The labels are highlighted, and marked as the installation progresses. The installer build process will auto populate the list based on the panel titles.

NOTE: Using the Installer UI > Look & Feel task's Installer Panel Additions tab and the Labels settings tab found on each individual panel's customizer, developers can assign multiple panels to the same label. Thus, if there are numerous steps, or if the installer has several panels for the same step the interface can be adjusted as needed.

To control label order, or to edit the content of the label, in the Installer UI > Look & Feel task's Installer Panel Additions tab use the Arrows and other control buttons found to the left of the list of panels.

### Help

Selecting Enable installer help in the Installer UI > Help subtask provides a Help feature for the installer program.

Selecting HTML allows greater formatting control of the help text. (You cannot apply HTML tags to the title.) To format the help text, use the HTML formatting tags. For example,

```
<B>MyHelp</B> <I>Information</I>
```

causes InstallAnywhere to display MyHelp *Information*.

Developers may either set a single help message, which they can define in this window, or customize help for each installer screen. To customize help for each installer screen, select Use different help text for each panel. Add the customized help in the Help tab of the action customizer at the bottom of the Pre-Install, Post-Install tasks.

## Additional Action Information

### LaunchAnywhere

A LaunchAnywhere Executable is an executable file that is used to launch a Java application on any LaunchAnywhere-compatible platform (All Windows, Unix platforms, and Mac OS X. LaunchAnywhere enables end users to double-click on an icon (Windows or Mac OS X) or type a single command (Unix) to start a Java application. The LAX is also in charge of configuring the Java application environment by setting the classpath, redirecting standard out and standard error, passing in system properties, environment variables, and command-line parameters, and many other options.

The launcher looks at a configuration file `<MyLauncherName>.lax` to determine how the launcher runs. This `lax` file is created during the installation, and is placed in the same location as the launcher.

A list of lax properties is located in Appendix I. LAX Properties.

### Manifest Files

Manifest files are text files which specify a list of files and directories. The manifest file has a certain format (listed below). The format specifies the file’s source, its destination (which is relative to the location of the action in the Visual Tree of the Install task), and optionally, which Unix file permission it should have and if it should be placed on the classpath. At build time, the file is analyzed and its contents are placed into the installer.

#### Manifest File Format

For files:

```
F,[SOURCEPATH]relative_path_to_source_file,./relative_path_to_destination_file
F,absolute_path_to_source_file,./relative_path_to_destination_file
```

To put files on the classpath:

```
F,absolute_path_to_source_file,./relative_path_to_destination_file,cp
```

To set a file's permissions on Unix:

```
F,[SOURCEPATH]relative_path_to_source_file,./relative_path_to_destination_file,755
```

For directories:

```
D,[SOURCEPATH]relative_path_to_source_dir[/],./relative_path_to_destination_dir[/]
D,absolute_path_to_source_dir[/],./relative_path_to_destination_dir[/]
```

Examples:

```
F,$IA_HOME$/path/to/source/file.txt,./destination/path/thisfile.txt
F,/absolute/path/to/source/file.txt,./destination/path/thisfile.txt,cp,655
D,$IA_HOME$/path/to/dir,./destination/path/dir
D,/absolute/path/to/dir,./destination/path/
```

# Appendix D. Build Arguments

A build properties file template named BuildProperties.xml can be found at

`<InstallAnywhere folder>/resource/build/BuildProperties.xml`

It provides a sample of all possible build settings and can be customized to suit your build requirements.

| | |
|---|---|
| -v | Print the InstallAnywhere product version:<br>`C:\Program Files\Macrovision\InstallAnywhere 2008>build.exe -v`<br><br>`InstallAnywhere 2008 Enterprise` |
| -p | Use the specified build properties file. |
| + | Add a platform to build (see the platform arguments below). |
| - | Remove a platform from build (see the platform arguments below). |

The following arguments modify the `add` and `remove` platforms arguments:

| | |
|---|---|
| `a,A` | AIX without VM option |
| `av,AV` | AIX with VM option |
| `h,H` | HP-UX without VM option |
| `hv,HV` | HP-UX with VM option |
| `j,J,o,O` | Pure Java option |
| `l,L` | Linux without VM option |
| `lv,LV` | Linux with VM option |
| `s,S` | Solaris without VM option |
| `sv,SV` | Solaris with VM option |
| `u,U` | Generic Unix without VM option |
| `n,N` | Named Unix without VM option |
| `nv,NV` | Named Unix with VM option |
| `w,W` | Windows without VM option |
| `wv,WV` | Windows with VM option |
| `x,X` | Mac OS X without VM option |
| `web` | Build Web installers |
| `cd` | Build CD-ROM installers |
| `opt` | Optimize by platform |
| `merge` | Build Merge modules |

# Appendix E. Exit Codes

## Resource Related

| Code | Status |
|---|---|
| cancelled1 | Missing resources, build abort cancelled by preferences |

## Project File Related

| Code | Status |
|---|---|
| 101 | Project load error |
| 102 | Project copy load error |
| 103 | Project file not found |
| 104 | Project file is read-only |
| 199 | Project file unknown error |

## Command Line Options Related

| Code | Status |
|---|---|
| 200 | Illegal build flag |
| 201 | Insufficient build flag |

## VM Pack Related

| Code | Status |
|---|---|
| 300 | VM Pack replaced |
| 301 | VM Pack not found |
| 302 | VM Pack illegal format |
| 399 | VM Pack unknown error |

## File Write Errors

| Code | Status |
|---|---|
| 400 | File write not found |
| 401 | File write busy |
| 402 | File write protected |
| 403 | File write error |
| 499 | File write unknown error |

## File Read Errors

| Code | Status |
|---|---|
| 500 | File read not found |
| 501 | File read busy |
| 502 | File read protected |
| 503 | File read error |
| 599 | File read unknown error |

## Other

| Code | Status |
|---|---|
| -1 | Other error/unknown error |
| 0 | No errors. Build completed successfully without errors or warnings |

| | |
|---|---|
| 666 | Insufficient rights in directory |
| 799 | Unknown internal error |

# Appendix F. Build Properties

## *Ant Build Integration*

Ant is a powerful Java-based build tool developed by the Apache Foundation's Jakarta Project. It can be used to control complex build tasks in Java and other development environments. Ant manages specific actions though "tasks", which can either be part of the core Ant distribution or available as extensions.

InstallAnywhere provides an Ant task to build installers from Ant. The InstallAnywhere Ant task is located in your InstallAnywhere application folder:

`<`*`InstallAnywhere`*`>/resource/build/iaant.jar`

To integrate the InstallAnywhere Ant task in an Ant project, you must set the classpath of the InstallAnywhere Ant task to the location of `iaant.jar`. Note that use of `iaant.jar` requires Java 1.4 or later.

Ant uses an XML file to specify the order of tasks for your build process. More information on Ant can be found on the Apache Foundation's Ant Project Web site (`http://ant.apache.org`).

### *Task Definition*

You add a task definition to your Ant project for the InstallAnywhere Ant task as follows:

```
<taskdef name="buildinstaller"
         classname="com.zerog.ia.integration.ant.InstallAnywhereAntTask"/>
    <classpath>
        <pathelement path="<InstallAnywhere>/resource/build/iaant.jar" />
    </classpath>
</taskdef>
```

### *Task Settings*

After defining the task, specify any parameters necessary for the build settings:

```
<buildinstaller
  IALocation="C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise"
  IAProjectFile="C:\Projects\myproject.iap_xml"
  additionalparameter=value>
  <configuration>
    <target platform="windows">
      <outputDir>win</outputDir>
      <buildWithNoVM>false</buildWithNoVM>
      <buildWithVM>true</buildWithVM>
      <bundledVM>SunJRE160_00i18nWin32.vm</bundledVM>
    </target>
    ...
  </configuration>
</buildinstaller>
```

Replace the `IAlocation` value with the absolute path to your own InstallAnywhere application folder.

Specify the path and file name of the project to build in the `IAProjectFile` parameter.

All other properties are optional. The parameters closely match the properties found in the BuildProperties.xml file described above. The following tables show the available parameters.

## Build Parameters

| Attribute | Description | Required |
|---|---|---|
| `IAProjectFile` | The location of the InstallAnywhere project that you want to build. | Yes |
| `IALocation` | The location where InstallAnywhere is installed. | No* |
| `propertiesfile` | The location of a `BuildProperties.xml` you can use. If used, all other attributes are ignored. | No |
| `failOnError` | Stop the build process if the command exits with a return code other than 0. Defaults to false. | No |

NOTES: If `IALocation` is not specified, the task will search for a copy of InstallAnywhere to run against. If InstallAnywhere is not installed in one of the default locations, then the task will check the InstallAnywhere product registry for a valid location.

## Platform Options

Platform options correspond to platform targets on the InstallAnywhere Build Targets tab. By default, the platform options you set in the InstallAnywhere Ant task supersede the corresponding settings in your InstallAnywhere project file and activate or deactivate all targets associated with the platform/VM combination you specify.

Note that Platform options do not activate or deactivate the targets you define in the Ant project's Configuration section. The InstallAnywhere Ant task will always build the targets you define in the Configurations section regardless of platform options defined here or in the InstallAnywhere project file.

| | |
|---|---|
| BuildLinuxWithVM | true/false |
| BuildLinuxWithoutVM | true/false |
| LinuxVMPackLocation | Path |
| BuildHPUXWithVM | true/false |
| BuildHPUXWithoutVM | true/false |
| HPUXVMPackLocation | Path |
| BuildAIXWithVM | true/false |
| BuildAIXWithoutVM | true/false |
| AIXVMPackLocation | Path |
| BuildSolarisWithVM | true/false |
| BuildSolarisWithoutVM | true/false |
| SolarisVMPackLocation | Path |
| BuildNamedUNIXWithVM | true/false |
| BuildNamedUNIXWithoutVM | true/false |
| NamedUNIXVMPackLocation | Path |
| NamedUNIXTitle | String |
| BuildWindowsWithVM | true/false |
| BuildWindowsWithoutVM | true/false |
| WindowsVMPackLocation | Path |
| BuildUNIXAll | true/false |
| BuildMacOSX | true/false |
| WantAuthenticationMacOSX | true/false |
| WantAuthenticationMacOSXShowGUI | true/false |
| BuildPureJava | true/false |
| OverrideAllPlatformSettings | true/false (see note below) |

Note that when you set OverrideAllPlatformSettings to true, Ant overrides all the platform settings in the InstallAnywhere project with the platform options provided in the Ant task. For example, if your InstallAnywhere project targets Windows and Linux systems and your Ant task adds a Mac OS X target (BuildMacOSX="true"), the Ant task builds installers for Windows, Linux, and Mac OS X. However, if you also set OverrideAllPlatformSettings to true, Ant builds an installer for Mac OS X only.

## Build Options

The following build options specify your build distribution settings.

| | |
|---|---|
| BuildCDROMInstaller | true/false |
| BuildWebInstaller | true/false |
| BuildMergeModule | true/false |
| BuildReadOnlyMergeModule | true/false |
| BuildOutputLocation | Path |
| OptimizeCDROMInstaller | true/false |
| OptimizeWebInstaller | true/false |
| OptimizeMergeModule | true/false |
| AutoPopulateLabels | true/false |
| AutoCleanComponents | true/false |

## Installer Options

Installer options affect the behavior of the installers the project builds.

| | |
|---|---|
| InstallerStdErrRedirect | Path |
| InstallerStdOutRedirect | Path |
| InstallerValidVMList | String |
| InstallerInitialHeapSize | Int |
| InstallerMaxHeapSize | Int |
| UNIXDefaultUI | silent/console/gui |
| WindowsDefaultUI | silent/console/gui |

In order to redirect InstallAnywhere's build process output to Ant's log, you must specify a redirector as described in the Ant documentation (Exec task).

## Configuration Elements

Installer options affect the behavior of the installers the project builds.

| | |
|---|---|
| configuration | Contains targets that append additional build targets to those already defined in the InstallAnywhere project file. |
| target | Contains the settings to define a single build target. Targets must include a platform setting. |
| platform | Specifies the target platform. Possible values include *windows*, *linux*, *macosx*, *solaris*, *aix*, *hpux*, *unix*, *unixwithvm*, and *java*. |
| buildWithNoVM | Sets whether or not this target includes a bundled VM. Use *true* to build an installer without a bundled VM; otherwise, use *false*. |

| | |
|---|---|
| buildWithVM | Sets whether or not this target includes a bundled VM. Use *true* to build an installer with a bundled VM; otherwise, use *false*. |
| bundledVM | Specifies the VM to bundle with the installer. |

## Example

```
<taskdef name="buildinstaller"
         classname="com.zerog.ia.integration.ant.InstallAnywhereAntTask">
  <classpath>
    <pathelement path="c:\ant\lib\ext\iaant.jar"/>
  </classpath>
</taskdef>

...

<buildinstaller
  IALocation="C:\Program Files\Macrovision\InstallAnywhere 2008 Enterprise"
  IAProjectFile="C:\Projects\myproject.iap_xml"

  BuildLinuxWithoutVM="true"
  BuildWindowsWithoutVM="true"
  BuildWebInstallers="true"
  OptimizeWebInstallers="true"
  InstallerStdErrRedirect="C:\console.txt" >

  <configuration>
    <target platform="macosx">
      <buildWithNoVM>true</buildWithNoVM>
    </target>
  </configuration>
</buildinstaller>
```

# Appendix G. Language Codes

| *Language* | *Locale Code* | *Language Group* |
|---|---|---|
| Arabic | ar | Eastern |
| Catalan | ca | Western |
| Czech | cs | Eastern |
| Danish | da | Western |
| German | de | Western |
| Greek | el | Eastern |
| English | en | Western |
| Spanish | es | Western |
| Basque | eu | Western |
| Finnish | fi | Western |
| French | fr | Western |
| French (Canada) | fr_CA | Western |
| Hebrew | iw | Eastern |
| Hungarian | hu | Eastern |
| Indonesian | id | Eastern |
| Italian | it | Western |
| Japanese | ja | Eastern |
| Korean | ko | Eastern |
| Dutch | nl | Western |
| Norwegian | no | Western |
| Polish | pl | Western |
| Portuguese | pt | Western |
| Portuguese (Brazil) | pt_BR | Western |
| Russian | ru | Eastern |
| Slovak | sk | Western |
| Slovenian | sl | Western |
| Swedish | sv | Western |
| Thai | th | Eastern |
| Turkish | tr | Eastern |
| Chinese (Simplified) | zh_CN | Eastern |
| Chinese (Traditional) | zh_TW | Eastern |

# Appendix H. Localizable Elements

Installers deployed to non-Latin systems require an international Java Virtual Machine.

**Localizing Items in the Installer**

See the list below to determine the correct properties to modify in the language resource files. This list does not include properties from many new actions. For a complete list, contact Macrovision Support.

| Property | Definition |
|---|---|
| `Installer.#.ProductName` | Name of product displayed on installer title bar. |
| `Installer.#.RulesFailedMessage` | Message displayed if specified rules prevent the installer from running. |
| `Installer.#.ShortcutDestinationPathMacOS` | Path to where aliases are created during installation on Mac OS, relative to the end-user-selected alias folder chosen on the Choose Alias Location step. |
| `Installer.#.ShortcutDestinationPathWin32` | Path to where shortcuts are created during installation on Windows, relative to the end-user-selected shortcut folder chosen on the Choose Shortcut Folder step. |
| `Installer.#.ShortcutDestinationPathSolaris` | Path to where links are created during installation on Unix, relative to the end-user-selected links folder chosen on the Choose Link Location step. |
| `InstallSet.#.Description` | Description of one of the installer's features. |
| `InstallSet.#.InstallSetName` | Name of one of the installer's features. |
| `IntroAction.#.message` | Text to display on the installer's Introduction step. |
| `Intro.#.stepTitle` | Title to display on the installer's Introduction step. |
| `LicenseAgr.#.FileName` | Name of localized license agreement to be displayed as the installer is preparing itself. (Note that this is a filename only, and not a fully qualified pathname.) |
| `LicenseAgr.#.Path` | Path name to localized license agreement to be displayed as the installer is preparing itself. (Note that this is only a pathname and does not include the filename.) |
| `LicenseAgr.#.Title` | Title of License Agreement step in the installer. |
| `MakeExecutable.#.destinationName` | Name of the LaunchAnywhere Executable to be created on the destination computer. |
| `MakeRegEntry.#.Value` | Value to be written to the Windows registry. |

| | |
|---|---|
| `ShortcutLoc.#.macTitle` | Title of Mac OS X Choose Alias Location step in the installer. |
| `ShortcutLoc.#.SolarisTitle` | Title of Unix Choose Link Folder step in the installer. |
| `ShortcutLoc.#.Win32Title` | Title of Windows Choose Shortcut Folder step in the installer. |
| `Billboard.#.ImageName` | Name of billboard image file to be displayed as the installer is preparing itself. (Note that this is a filename only, and not a fully qualified pathname.) |
| `Billboard.#.ImagePath` | Path name to billboard image to be displayed as the installer is preparing itself. (Note that this is only a pathname and does not include the filename.) |
| `ChooseInstallSet.#.Title` | Title of Choose Install Set step in the installer. |
| `ChooseJavaVM.#.Title` | Title of Choose Java Virtual Machine step in the installer. |
| `CreateShortcut.#.DestinationName` | Name of the shortcut/alias/link to be created on the destination computer. |
| `human.readable.language.name` | The name of the language represented by the data in this resource file (as in English, Español, etc.). |
| `ImportantNote.#.FileName` | Name of text file to be displayed on the Important Note step of the installer. (Note that this is a filename only, and not a fully qualified pathname.) |
| `ImportantNote.#.Path` | Path name to text file to be displayed on the Important Note step of the installer. (Note that this is only a pathname and does not include the filename.) |
| `ImportantNote.#.Title` | Title of Important Note step in installer. |
| `InstallBundle.#.BundleName` | Name of component. |
| `InstallBundle.#.Description` | Description text describing component. |
| `InstallComplete.#.DisplayText` | Text to display on the Install Complete step of the installer. |

# Appendix I. LAX Properties

The properties that LaunchAnywhere can modify are defined as follows:

| Property | Definition |
|---|---|
| `lax.application.name` | The name of the application that the launcher executes. |
| `lax.class.path` | The classpath for the application. By default, set to `$IA_CLASSPATH$` (the classpath specified in the InstallAnywhere Designer environment).<br><br>When specifying the classpath, use either forward or backward slashes (‘`/`’ or ‘`\`’) within a path. Use either colons or semicolons to separate multiple paths—LaunchAnywhere substitutes the proper characters for the installation platform at run time. |
| `lax.command.line.args` | A list of arguments passed to the application’s main method. These are specified exactly the same way as they would be on the command line. For example, to invoke the application as:<br><br>`java myApp arg1 arg2`<br><br>set this property to<br><br>`arg1 arg2`<br><br>Be sure to place quotes around any arguments that have spaces.<br><br>When it is necessary to pass in an argument that is known only at install time (for example, the installation directory), use an InstallAnywhere variable. |
| `lax.dir` | The path to the directory containing the LaunchAnywhere native launcher. If you are specifying a path in Windows, make sure to use escaped backslashes, as in C:\\Program Files\\OfficeSuite.exe. |
| `lax.java.compiler` | The JIT compiler being used for execution of this application. This property is run-time only: The lax.java.compiler property cannot be set in the LaunchAnywhere Properties or Uninstaller Properties dialog box. |
| `lax.main.method` | The name of the application’s starting method that this LaunchAnywhere Executable invokes. |
| `lax.main.class` | The class to be launched by this LaunchAnywhere Executable. This class must contain a method with the name defined in the `lax.main.method` property. |

| | |
|---|---|
| `lax.nl.current.vm` | The full path to the VM executable to be used. If the LaunchAnywhere executable cannot find the specified VM, it searches the system for the VMs in `lax.nl.valid.vm.list`. |
| `lax.nl.env.`*`variable_name`* | Windows and Unix only. Access any system environment variable (for example, access PATH with `$lax.nl.env.PATH$`) by specifying the property name as an all-lowercase or all-uppercase string. These properties are resolved at application run time, when LaunchAnywhere executes. You can also get access to system environment variables via InstallAnywhere variables. |
| `lax.nl.env.exact_case.`*`variable_name`* | Windows and Unix only. Access any system environment variable (for example, access Path via `$lax.nl.env.exact_case.Path$`) by specifying the property name as a string of the exact case as it is defined in the environment. Note that these properties are resolved at application runtime, when LaunchAnywhere executes. You can also get access to system environment variables via LaunchAnywhere variables. |
| `lax.nl.env.path` | The system PATH for the computer this application is running on. This property is run-time only; the `lax.nl.env.path` property cannot be set using the LaunchAnywhere Properties or Uninstaller Properties dialog box. |
| `lax.nl.java.compiler` | This property defines the name of the JIT that your application should use. Set this option to no value (leave the value blank) to use the default JIT. You can also specify the name of a specific JIT by defining it in this property. Set to “off” if no JIT is to be used. |
| `lax.nl.java.launcher.main.class` | The class that contains the main method called by LaunchAnywhere. |
| `lax.nl.java.launcher.main.method` | The name of the main method called by LaunchAnywhere. |
| `lax.nl.java.option.additional` | LaunchAnywhere writes the value of this property to the command line verbatim. Java VM properties or settings not directly supported by current LAX configuration properties can be included as part of the command line used to invoke Java. For example, to pass a custom variable as part of the command line, set `lax.nl.java.option.additional` to `-Dmyvariable=value`. |
| `lax.nl.java.option.check.source` | Set to on or off to tell the VM to verify bytecodes. |
| `lax.nl.java.option.debugging` | Turns on debugging in the Virtual Machine so that an application can be debugged. |
| `lax.nl.java.option.garbage.-` | Whether to have a low-priority background thread |

| | |
|---|---|
| `collection.background.thread` | that does garbage collection. Set to on or off. |
| `lax.nl.java.option.garbage.-collection.extent` | Set to either of the following values:<br>`min`: Garbage-collect everything except classes.<br>`full`: Garbage-collect everything. |
| `lax.nl.java.option.java.heap.-size.initial` | Defines the initial heap size for the installer that will be invoked. This number is always specified in bytes, not in kilobytes or megabytes, and is analogous to the VM parameter `-ms` or `Xms`. The default is 16777216 (16 MB). |
| `lax.nl.java.option.java.heap.-size.max` | Defines the maximum heap size in bytes for the installer that will be invoked. This number is always specified in bytes, not in kilobytes or megabytes, and is analogous to the VM parameter `-mx` or `Xmx`. The default is 50331648 (48 MB). |
| `lax.nl.java.option.verbose` | Defines the level of content in output messages:<br>`gc`: Output garbage collection messages.<br>`normal`: Output all normal verbose messages.<br>`all`: Output all normal and garbage collection messages.<br>`none`: Do not output any verbose messages. |
| `lax.nl.java.option.verify.mode` | Sets when Java will verify classes for security and errors. Values can be remote, all, or none. |
| `lax.nl.message.vm.not.loaded` | The message to show the end user in a dialog box if no VM can be found. |
| `lax.nl.valid.vm.list` | The list of VMs that this LaunchAnywhere Executable will allow the Java application to be run against. The value for this property can be any space-delimited combination of the following:<br>`ALL` (any VM)<br>`JDK` (any Java JDK)<br>`JRE` (any Java JRE)<br>`1.4.*`<br>`1.5+`<br><br>The value of this property will also override the value listed in `lax.nl.current.vm` if the VM listed in that property is not of a valid type. The order of the valid VM list specifies the precedence in which VMs found on the system should be chosen if a valid VM is not listed in `lax.nl.current.vm`. |
| `lax.resource.dir` | The platform name in exact case. |
| `lax.root.install.dir` | The root directory of the entire installation (same as `$USER_INSTALL_DIR$`). |
| `lax.stderr.redirect` | The location of your application's `stderr` output. Set to null to suppress, console to write to a console window, or any file name to output to a file. Note: If you are specifying a path name to a Windows file, make sure to use escaped backslashes (e.g., `c:\\myfolder\\output.txt`) |

| | |
|---|---|
| `lax.stdout.redirect` | The location of your application's `stdout` output. Set to null to suppress, console to write to a console window, or any file name to output to a file. If you are specifying a path name to a Windows file, make sure to use escaped backslashes (e.g., `c:\\myfolder\\output.txt`) |
| `lax.user.dir` | The working directory for your application. The default value is . (period). Leave as-is to set the working directory to the directory where the LaunchAnywhere Executable resides. To override the default behavior, specify an absolute or relative path. (A relative path is relative to the LaunchAnywhere Executable location.) |
| `lax.version` | The version number of the LaunchAnywhere executable. |
| `LISTPROPS` | This property lists all system properties available to the Java application (it can take any value). You must redirect stdout and stderr to see the results of this output. |

# Appendix J. Installation Planning Worksheet

Product Name: ________________________________ Product Version: ____________________________

## Target Platforms

____ Mac OS X

____ Windows

____ AIX

____ HP-UX

____ Linux

____ Solaris

____ Other Unix:

____ Other Java-Enabled Platforms:

## Installer Target

____ Technical End user

____ Non-Technical End user

## Deployment Media

____ Web

____ CD-ROM/DVD

____ Merge Module

## Application Type

____ Native Application

____ Java Application

____ .NET

## Java Specific Options

Java Virtual Machine(s) Version Required: ______________________________________________________

__________________________________________________________________________________________

__________________________________________________________________________________________

__________________________________________________________________________________________

__________________________________________________________________________________________

__________________________________________________________________________________________

## *Installation Needs*

Location(s) on target system, where files are to be installed:

Mac OS X ____________________

Windows ____________________

AIX ____________________

HP-UX____________________

Linux ____________________

Solaris ____________________

Other Unix ____________________

Other Java-enabled platforms ____________________

## *List Configuration that Must Be Done to the Target Platform*

____________________

____________________

____________________

____________________

____________________

____________________

## *What Information Must Be Collected from the End User?*

____________________

____________________

____________________

____________________

____________________

____________________

____________________

## *Team Development Options*

Will this project be managed by more than one developer? ___Yes ___ No

If Yes, then what Source Paths will be defined for file maintenance?

| | |
|---|---|
| Name: ____________________________ | Description:__________________________________________ |
| ____________________________ | __________________________________________ |
| ____________________________ | __________________________________________ |
| ____________________________ | __________________________________________ |
| ____________________________ | __________________________________________ |
| ____________________________ | __________________________________________ |
| ____________________________ | __________________________________________ |

## *Uninstall Options*

Will this installation require any special uninstall options? ___ Yes ___ No

If Yes, specify uninstall options:

Pre-Uninstall

______________________________________________________________________

______________________________________________________________________

______________________________________________________________________

______________________________________________________________________

______________________________________________________________________

Post-Uninstall

______________________________________________________________________

______________________________________________________________________

______________________________________________________________________

______________________________________________________________________

______________________________________________________________________

# Index