

# **InstallAnywhere® 2008 Training Manual**

# InstallAnywhere 2008 Training Manual

**Part Number:** TRNIA09-1007

**Product Release Date:** 18-March-2008 / Printed in the United States

## Copyright Notice

Copyright © 2001–2008 Macrovision Europe Ltd. and/or Macrovision Corporation. All Rights Reserved.

This product contains proprietary and confidential technology provided by and/or owned by Macrovision Europe Ltd., UK, and/or Macrovision Corporation of Santa Clara, California, U.S.A. Any use, copying, publication, distribution, display, modification, or transmission of such technology in whole or in part in any form or by any means without the prior express written permission of Macrovision Europe Ltd. and/or Macrovision Corporation is strictly prohibited. Except where expressly provided by Macrovision Europe Ltd. and/or Macrovision Corporation in writing, possession of this technology shall not be construed to confer any license or rights under any of Macrovision Europe Ltd. and/or Macrovision Corporation's intellectual property rights, whether by estoppel, implication, or otherwise.

ALL COPIES OF THE TECHNOLOGY & RELATED INFORMATION, IF ALLOWED BY MACROVISION CORPORATION, MUST DISPLAY THIS NOTICE OF COPYRIGHT AND OWNERSHIP IN FULL.

## Trademarks

Macrovision, AdminStudio, DemoNow, DemoShield, FLEXenabled, FLEXlm, FLEXnet, FLEXnet Certified, FLEXnet Connector, FLEXnet Manager, FLEXnet Publisher, Globetrotter, Hawkeye, InstallFromTheWeb, InstallShield, InstallShield Developer, InstallShield DevStudio, InstallShield Professional, It All Starts Here, OneClickInstall, Package For The Web, QuickPatch, ReadySell, RipGuard, SafeCast, Trymedia Systems, and Zero G Software are registered trademarks or trademarks of Macrovision Corporation in the United States of America and/or other countries. All other brand and product names mentioned herein are the trademarks and registered trademarks of their respective owners.

## Restricted Rights Legend

The software and documentation are “commercial items,” as that term is defined at 48 C.F.R. §2.101, consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.2702, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.2702-1 through 227.7202-4, as applicable, the commercial computer software and commercial computer software documentation are being licensed to U.S. government end users (A) only as commercial items and (B) with only those rights as are granted to all other end users pursuant to the terms and conditions set forth in the Macrovision Corporation standard commercial agreement for this software. Unpublished rights reserved under the copyright laws of the United States of America.

February 2008

# Contents

---

<b>1</b>	<b>Introduction . . . . .</b>	<b>1</b>
	<b>Multiplatform Installation . . . . .</b>	<b>2</b>
	<b>Introduction to InstallAnywhere . . . . .</b>	<b>2</b>
	<b>Requirements . . . . .</b>	<b>3</b>
	Authoring Environment Requirements . . . . .	3
	<b>Supported Operating Systems . . . . .</b>	4
	Target Environment . . . . .	4
	<b>Supported Operating Systems . . . . .</b>	5
	<b>Supported Java Virtual Machines . . . . .</b>	5
	<b>Editions . . . . .</b>	<b>6</b>
	Enterprise Edition . . . . .	6
	Standard Edition . . . . .	6
	<b>The InstallAnywhere End-user Experience . . . . .</b>	<b>7</b>
<b>2</b>	<b>Starting a Project Using the InstallAnywhere Project Wizard . . . . .</b>	<b>15</b>
	<b>Building an Installer Using the Wizard . . . . .</b>	<b>16</b>
	<b>Creating a New Project . . . . .</b>	<b>16</b>
	<b>Setting Project Information . . . . .</b>	<b>18</b>
	<b>Installing Tasks . . . . .</b>	<b>19</b>
	Add Files . . . . .	19
	Choose Main Class . . . . .	20
	Setting the Classpath . . . . .	21
	<b>Building the Installer . . . . .</b>	<b>22</b>
	<b>Testing the Project . . . . .</b>	<b>23</b>

<b>3</b>	<b>Introduction to the Advanced Designer . . . . .</b>	<b>25</b>
	<b>Installation Planning . . . . .</b>	<b>25</b>
	<b>Installation Goals . . . . .</b>	<b>26</b>
	<b>Working with Advanced Designer . . . . .</b>	<b>26</b>
	<b>Defining Installer Projects and the Product Registry . . . . .</b>	<b>27</b>
	Product Registry . . . . .	27
	Installer Identification and Version . . . . .	28
	<b>File Settings: Timestamps and Overwrite Behavior . . . . .</b>	<b>29</b>
	Installed File Timestamps . . . . .	30
	Default Overwrite Behavior . . . . .	30
	<b>Platforms . . . . .</b>	<b>32</b>
	<b>Locales . . . . .</b>	<b>35</b>
	<b>Rules Before the Pre-Install Task . . . . .</b>	<b>35</b>
	<b>Creating Debug Output . . . . .</b>	<b>35</b>
	Installer Debug Output . . . . .	35
	<b>Virtual Machines . . . . .</b>	<b>36</b>
	Optional Installer Arguments . . . . .	36
	Java . . . . .	36
	<b>Quick Quiz . . . . .</b>	<b>37</b>
<b>4</b>	<b>Building Releases . . . . .</b>	<b>39</b>
	<b>Build Targets . . . . .</b>	<b>39</b>
	VM Packs . . . . .	41
	VM Selection . . . . .	42
	<b>Distribution . . . . .</b>	<b>43</b>
	Web Installers . . . . .	44
	CD-ROM/DVD Installers . . . . .	45
	Burning CD-ROM Installers . . . . .	45
	<b>Merge Modules and Templates . . . . .</b>	<b>47</b>
	<b>Build Log . . . . .</b>	<b>47</b>
<b>5</b>	<b>Basic Installer Customization . . . . .</b>	<b>49</b>
	<b>Customizing the Installer Look and Feel . . . . .</b>	<b>50</b>
	<b>Installer UI Modes . . . . .</b>	<b>51</b>
	<b>Splash Screens . . . . .</b>	<b>51</b>
	<b>GUI Panel Additions . . . . .</b>	<b>52</b>
	<b>Background Images . . . . .</b>	<b>53</b>
	<b>Frame UI Settings . . . . .</b>	<b>54</b>
	<b>Billboards . . . . .</b>	<b>54</b>
	<b>Help . . . . .</b>	<b>54</b>
	<b>Conditional Logic . . . . .</b>	<b>55</b>
	<b>Quick Quiz . . . . .</b>	<b>57</b>

<b>6</b>	<b>Installer Organization . . . . .</b>	<b>59</b>
	<b>Install Sets, Features, and Components . . . . .</b>	<b>59</b>
	Install Sets . . . . .	59
	Features . . . . .	60
	Components . . . . .	61
	<b>Using the Organization Task . . . . .</b>	<b>61</b>
	Install Sets . . . . .	62
	Features . . . . .	65
	Components . . . . .	66
	Types of Components . . . . .	67
	<i>Shared Components</i> . . . . .	68
	<i>Component Dependencies</i> . . . . .	68
	<b>Organizing Your Features and Components . . . . .</b>	<b>69</b>
	Best Practices for Components . . . . .	69
	Best Practices for Features . . . . .	70
	Best Practices for Install Sets . . . . .	70
	<b>Adding Files to Your Project . . . . .</b>	<b>71</b>
	Adding Individual Files . . . . .	71
	Magic Folders . . . . .	74
	Adding Directories with SpeedFolders . . . . .	75
	Specifying Source Files with Manifest Files . . . . .	78
	<b>Quick Quiz . . . . .</b>	<b>80</b>
<b>7</b>	<b>Introduction to Advanced Actions and Panel Actions . . . . .</b>	<b>81</b>
	<b>Adding an Action . . . . .</b>	<b>82</b>
	Action Availability by Task . . . . .	84
	General Actions . . . . .	85
	Action Groups . . . . .	86
	Pre-Install/Uninstall and Post-Install/Uninstall Actions . . . . .	86
	<b>Examples of Common Actions . . . . .</b>	<b>87</b>
	Set InstallAnywhere Variable Action . . . . .	87
	Display Message Panel . . . . .	90
	Set System Environment Variable . . . . .	92
	<b>Display HTML Panel . . . . .</b>	<b>93</b>
	<b>Installer Panel Additions . . . . .</b>	<b>95</b>
	Panel Images . . . . .	96
	Panel Labels . . . . .	96
	Using Jump Actions and Logic . . . . .	97
	<b>Application Servers and Database Servers . . . . .</b>	<b>97</b>
	<b>Common Properties . . . . .</b>	<b>100</b>
	<b>Panel Action Settings . . . . .</b>	<b>101</b>
	Image Settings . . . . .	101
	Label Settings . . . . .	102
	Help . . . . .	103

LaunchAnywhere Executable .....	104
<b>Quick Quiz .....</b>	<b>105</b>
<b>8 Applying Basic and Intermediate Development Concepts .....</b>	<b>107</b>
<b>Building the Installer .....</b>	<b>108</b>
<b>Debugging InstallAnywhere Installers .....</b>	<b>109</b>
During Installer Development .....	109
Debugging a Win32 Installer .....	110
Debugging a Unix/Linux Installer .....	111
Debugging a Mac OS X Installer .....	111
Debugging a Pure Java Installer .....	111
Debugging LaunchAnywhere Executables .....	112
Reviewing Debug Information .....	113
Using Output Debug Information Actions .....	114
Debugging Using the Display Message Panel .....	114
<b>9 Source and Resource Management .....</b>	<b>115</b>
<b>How Source Paths Work .....</b>	<b>116</b>
<b>Adding Source Paths .....</b>	<b>117</b>
InstallAnywhere Preferences .....	117
System Environment Variables .....	118
<b>Updating the Location of Files and Resources .....</b>	<b>119</b>
<b>Managing Source Files .....</b>	<b>120</b>
<b>The Resource Manager .....</b>	<b>120</b>
<b>Adding Source Path Management Capability to Your Installer Project .....</b>	<b>121</b>
Enabling/Disabling Source Paths .....	121
Default Source Paths .....	121
Adding Source Paths .....	122
Preferences Menu .....	122
Set System Environment Variables .....	122
Using Source Paths in Your Project .....	123
Switching Access Path Locations .....	123
Quick Quiz .....	124
<b>10 Advanced Installer Concepts .....</b>	<b>125</b>
<b>Console Installers .....</b>	<b>125</b>
<b>Silent Installers .....</b>	<b>130</b>
Using a Response File .....	130
Configuring Variables Used in Response Files .....	131
<b>Uninstallation .....</b>	<b>133</b>
Feature-Level Uninstallation .....	135
Uninstaller Integration with the Target System .....	136
Uninstaller for Multiple Products .....	139

<b>11</b>	<b>Advanced Organizational Concepts . . . . .</b>	<b>141</b>
	<b>Integrating Find Component in Registry Action . . . . .</b>	<b>141</b>
	<b>Merge Modules and Templates . . . . .</b>	<b>142</b>
	Merge Modules . . . . .	143
	Templates . . . . .	145
	Advanced Topic: Importing ISMP Manifests . . . . .	145
	Merge Module Types . . . . .	148
	<i>Design-time Merge Modules</i> . . . . .	148
	<i>Build-time Merge Modules</i> . . . . .	148
	<i>Install-time Merge Modules</i> . . . . .	148
	<i>Dynamic Merge Modules</i> . . . . .	149
	Creating Merge Modules and Templates . . . . .	149
	<i>Build Options</i> . . . . .	149
	<i>Merge Module Size</i> . . . . .	149
	<i>Creating Merge Modules as Read Only</i> . . . . .	150
	<i>Advertised Variables</i> . . . . .	150
	Adding Advertised Variables . . . . .	150
	Adding Merge Modules . . . . .	150
	<b>Importing a Design Time Merge Module . . . . .</b>	<b>151</b>
	Merge Module Customizer . . . . .	153
	<b>InstallAnywhere Collaboration and DIMs . . . . .</b>	<b>153</b>
	Creating a DIM . . . . .	154
	Unit Tests . . . . .	159
	Consuming a DIM in an InstallAnywhere Project . . . . .	160
	<b>FLEXnet Connect . . . . .</b>	<b>163</b>
	<b>Quick Quiz . . . . .</b>	<b>164</b>
<b>12</b>	<b>Integrating InstallAnywhere with Automated Build Environments . . . . .</b>	<b>165</b>
	<b>InstallAnywhere Command-Line Build Facility . . . . .</b>	<b>165</b>
	Builder Arguments . . . . .	166
	<b>Digitally Signed Installers . . . . .</b>	<b>167</b>
	<b>Ant Build Integration . . . . .</b>	<b>168</b>
	Task Definition . . . . .	168
	Task Settings . . . . .	169
<b>13</b>	<b>Custom Code . . . . .</b>	<b>171</b>
	<b>Writing Custom Code . . . . .</b>	<b>172</b>
	<b>Custom Code Actions . . . . .</b>	<b>173</b>
	Custom Code Action Example . . . . .	173
	Writing the Action Code . . . . .	173
	Compiling and Packaging the Action Code . . . . .	175
	Adding the Custom Action to Your Project . . . . .	175
	Packaging Custom Code as a Plug-in . . . . .	178

Adding Action Help . . . . .	180
Executing External Scripts and Executables via Custom Code Action . . . . .	180
Using Custom Code Actions in Uninstall . . . . .	182
<b>Accessing Properties and Variables . . . . .</b>	<b>183</b>
Custom Variables and Response Files . . . . .	185
Variables and Proxy Classes . . . . .	185
<i>substitute</i> . . . . .	185
<i>getVariable</i> . . . . .	186
<b>Example: Displaying the Current Time . . . . .</b>	<b>186</b>
<b>Custom Code Rules . . . . .</b>	<b>188</b>
Custom Code Rule Example . . . . .	189
<b>InstallShield MultiPlatform Services . . . . .</b>	<b>190</b>
Compiling and Packaging Code That Uses ISMP Services . . . . .	191
Adding ISMP Code to Your Project . . . . .	191
File Service . . . . .	192
Security Service . . . . .	193
System Utility Service . . . . .	193
Advanced Topic: Differences Between Services in ISMP and IA . . . . .	194
Advanced Topic: Querying the ISMP Registry . . . . .	195
Debugging Custom Code . . . . .	196
<b>Advanced Action Methods . . . . .</b>	<b>197</b>
<b>Writing Custom Errors in the Installation Log . . . . .</b>	<b>199</b>
<b>Working with Java Native Interface (JNI) . . . . .</b>	<b>200</b>
JNI Example 1: Basic Text Display . . . . .	201
<i>Write and Compile the Java Class</i> . . . . .	201
<i>Use javah to Create a Header</i> . . . . .	201
<i>Creating the Native Method</i> . . . . .	202
<i>Running the Program</i> . . . . .	202
JNI Example 2: Calling a Windows DLL from a Custom Code Action . . . . .	203
Building the Distribution Archive . . . . .	204
Accessing the Library at Run Time . . . . .	205
Potential Problems Using JNI . . . . .	206
Additional Resources . . . . .	207
Quick Quiz . . . . .	208
<b>14    Custom Panels and Consoles . . . . .</b>	<b>209</b>
<b>User Input Panels . . . . .</b>	<b>209</b>
Using the Simple Get User Input Panel . . . . .	211
Variables and the Get User Input Panel . . . . .	212
Using the Advanced Get User Input Panel . . . . .	213
<b>Custom Code Panels . . . . .</b>	<b>215</b>
Default Custom Code Panel . . . . .	215
Creating a Custom Code Panel . . . . .	216
Packaging, Adding, and Testing the Custom Code Panel . . . . .	217

Adding Controls to the Custom Code Panel.....	219
Laying Out the Custom Code Panel .....	220
Working with Variables .....	222
Buttons and Action Listeners .....	224
<i>Example: Custom Code Panel</i> .....	226
<b>Custom Code Consoles.....</b>	<b>229</b>
Custom Code Console Actions.....	229
Additional Console Controls.....	231
Custom Code Console Action Example.....	232
<b>Quick Quiz .....</b>	<b>235</b>
<b>15 Localizing and Internationalizing Installers.....</b>	<b>237</b>
<b>Bidirectional Text Support.</b> .....	<b>240</b>
<b>Dynamic and Static Text.</b> .....	<b>241</b>
<b>Localization and the Internationalized Designer.</b> .....	<b>241</b>
<b>Specific Localization Concerns</b> .....	<b>242</b>
Localizing Resources .....	242
Localizing Custom Installer Labels .....	242
Localizing Custom Code .....	243
Best Practices for Localizing .....	243
Changing Localized Text .....	244
Modifying Localized Text.....	244
Changing Default Translations Provided in Language Packs .....	244
<b>Localizable Elements</b> .....	<b>245</b>
Localizing Items in the Installer .....	245
<b>A Installation Planning Worksheet .....</b>	<b>247</b>
<b>Target Platforms</b> .....	<b>247</b>
<b>Installer Target.</b> .....	<b>247</b>
<b>Deployment Media</b> .....	<b>248</b>
<b>Application Type.</b> .....	<b>248</b>
<b>Java Specific Options</b> .....	<b>248</b>
<b>Installation Needs</b> .....	<b>248</b>
<b>List Configuration that Must Be Done to the Target Platform</b> .....	<b>249</b>
<b>What Information Must Be Collected from the End User?</b> .....	<b>249</b>
<b>Uninstall Options.</b> .....	<b>250</b>

<b>B Exercises . . . . .</b>	<b>251</b>
<b>Chapter 2 Exercises . . . . .</b>	<b>252</b>
Creating the OfficeSuite Project . . . . .	252
<i>Setting Project Information</i> . . . . .	254
<i>Installing Tasks</i> . . . . .	255
<i>Building the Installer</i> . . . . .	258
<i>Testing the Project</i> . . . . .	259
Creating the TrainApp Project . . . . .	260
<b>Chapter 3 Exercises . . . . .</b>	<b>265</b>
Building an Installer with the Advanced Designer . . . . .	265
<i>Creating a New Project</i> . . . . .	265
<i>Pre-Install Actions</i> . . . . .	266
<i>Defining the Installation Tasks</i> . . . . .	266
<i>Adding a LaunchAnywhere Executable to the Install Task</i> . . . . .	267
<i>Post-Install Actions</i> . . . . .	269
<i>Testing</i> . . . . .	271
Examining the TrainApp Project in the Advanced Designer . . . . .	272
<b>Chapter 4 Exercises . . . . .</b>	<b>273</b>
Building the TrainApp Project . . . . .	273
<b>Chapter 5 Exercises . . . . .</b>	<b>276</b>
Exploring Look and Feel . . . . .	276
<i>Exploring the Installer UI Tasks</i> . . . . .	277
<i>Making Additions to the GUI Installer Panels</i> . . . . .	278
<i>Selecting Billboards</i> . . . . .	279
<i>Exploring the Help Options</i> . . . . .	281
<i>Rebuild the Project</i> . . . . .	281
Using Installer Rules . . . . .	281
Using Rules to Control Visual Elements . . . . .	282
Managing Installer Flow Based on End-User Input . . . . .	284
<i>Step One: Retrieving End-User Input</i> . . . . .	285
<i>Step Two: Applying the End User's Choice</i> . . . . .	286
Changing the Splash Screen . . . . .	287
Changing the Background Image . . . . .	288
<b>Chapter 6 Exercises . . . . .</b>	<b>289</b>
Using the Basic Installer Organization . . . . .	289
Magic Folders . . . . .	290
Examining the TrainApp Project Organization . . . . .	292
Working with SpeedFolders . . . . .	293
<b>Chapter 7 Exercises . . . . .</b>	<b>294</b>
Using Panels in Pre-Install . . . . .	294
Using Install Task Actions . . . . .	295
Creating Installer Logic Using Jump Labels and Actions . . . . .	297
Modifying TrainApp.properties at Run-time . . . . .	297

<b>Chapter 9 Exercises .....</b>	<b>299</b>
Creating Source Paths .....	299
<i>Managing Resources in the InstallAnywhere Project File</i> .....	299
<b>Chapter 10 Exercises .....</b>	<b>300</b>
Building a Console-Enabled Installer Using OfficeSuite .....	300
Building a Silent-Mode Installer .....	301
Installing TrainApp in Silent Mode .....	302
Hiding TrainApp from Add or Remove Programs .....	303
<b>Chapter 11 Exercises .....</b>	<b>304</b>
Creating Merge Modules .....	304
<b>Chapter 12 Exercises .....</b>	<b>304</b>
Building TrainApp from the Command Line .....	304
<b>Chapter 13 Exercises .....</b>	<b>305</b>
Adding FirstCustomCodeAction to TrainApp .....	305
Setting Variables from Custom Code .....	305
Using the InstallerResources Class .....	305
Logging Errors from Custom Code .....	307
Creating a Custom Rule .....	307
<b>Chapter 14 Exercises .....</b>	<b>308</b>
Adding a Get User Input Panel to TrainApp .....	308
Interacting with a Custom Panel's Navigation Buttons .....	309
<b>Chapter 15 Exercises .....</b>	<b>310</b>
Localizing the TrainApp Installer .....	310
<b>C     Index .....</b>	<b>313</b>

## Contents

# Introduction

---

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 is 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 file-types as a single unit. This is one deployment option, 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 (for example, 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 distributions). 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 a multitude of 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, easy-to-use 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.

## Introduction to 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 versions 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.

InstallAnywhere offers two authoring environments:

- Project Wizard
- Advanced Designer

The Project Wizard guides you through creating a new installer, using a simple step-by-step interface for describing your project, linking to files, defining shortcuts and icons, and building releases.

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



**Note:** 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.

## Authoring Environment Requirements

Refer to the following hardware and software requirements for the InstallAnywhere Authoring Environment.

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

## Supported Operating Systems

InstallAnywhere runs on the latest versions of these operating systems, fully updated with the most recent patches and service packs:

- 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, 5.3, and 6.1 (Power/PowerPC)



**Note:** 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 for any other platform or language. Localizations for 31 languages are included with Enterprise Edition. Localizations for 9 languages are included with Standard Edition.

## Target Environment

Refer to the following hardware and software requirements for the InstallAnywhere Target Environment.

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

## Supported Operating Systems

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 Server (32-bit and 64-bit x86, Itanium 2, and AMD-64)
- Windows Server 2008 (32-bit and 64-bit)
- 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 AMD-64)
- SUSE Linux 9 and 10 (x86 and PowerPC)
- Mac OS X 10.2, 10.3, 10.4, and 10.5 (Intel and PowerPC)
- Solaris 9 and 10 (SPARC, x86, and AMD-64)
- HP-UX 11i (Itanium 2 and PA-RISC)
- AIX 5.2, 5.3, and 6.1 (Power/PowerPC)
- i5/OS (OS/400) on System i—V5R3 and V5R4 (Enterprise Edition only)
- 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 JVM can be bundled with an installer, ensuring that the target system meets the minimum requirements for both the installers and your applications.



**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 or EAR deployment actions to a J2EE container
- 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 flexibility for customizing 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, refer to: <http://www.macrovision.com/products/installation/installanywhere.htm>

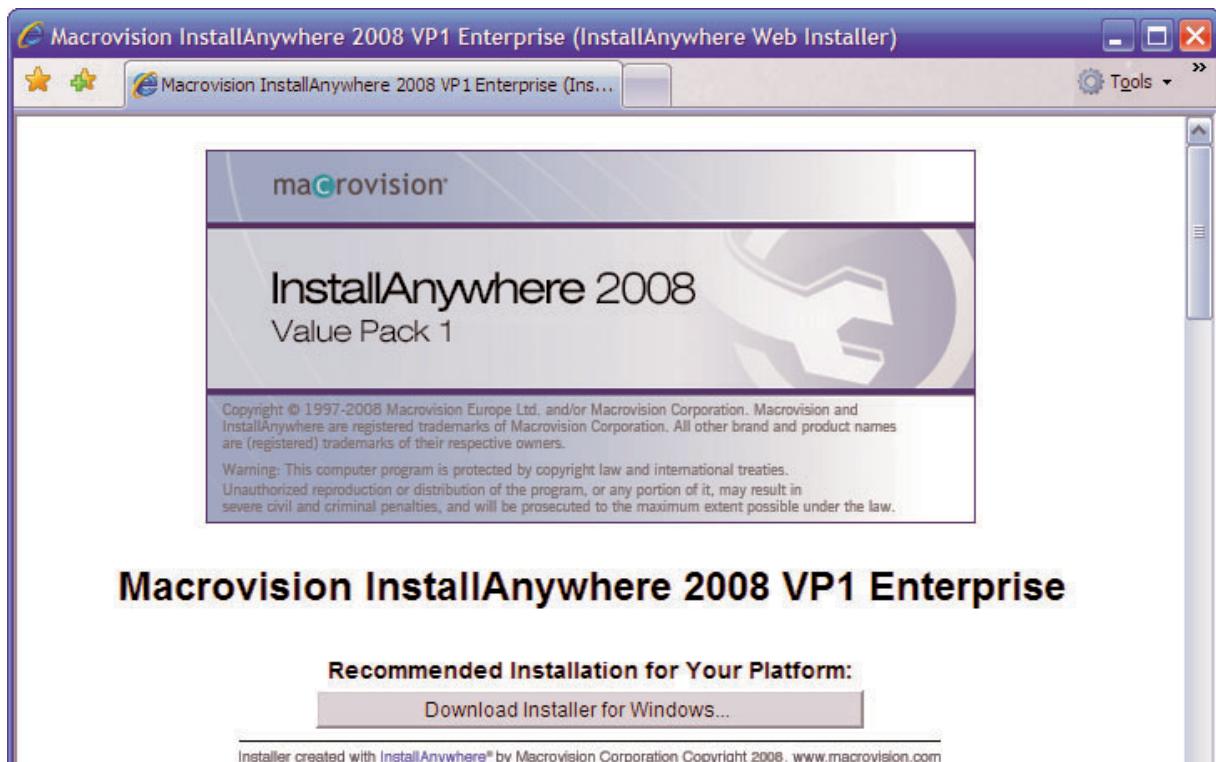
# The InstallAnywhere End-user Experience

The InstallAnywhere application is installed using an installer built with InstallAnywhere. In this example, the installer is called “InstallAnywhere 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 is the focus of this section.

The InstallAnywhere installer is built using InstallAnywhere, and it makes use of many of the InstallAnywhere features that are covered in the exercises.

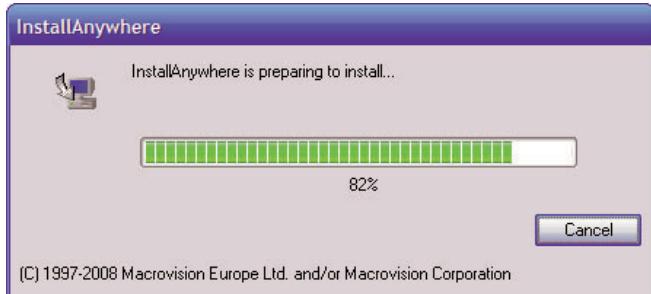
The first objective is simplicity. When your end-user visits your download page, you want to present a “single-click install.” This is accomplished using the InstallAnywhere Web Installer Applet. Since InstallAnywhere automatically creates HTML pages configured with the InstallAnywhere Web Installer Applet, not only is the end user’s experience simple, but your work is done for you. This is covered later in the chapter.

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.



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

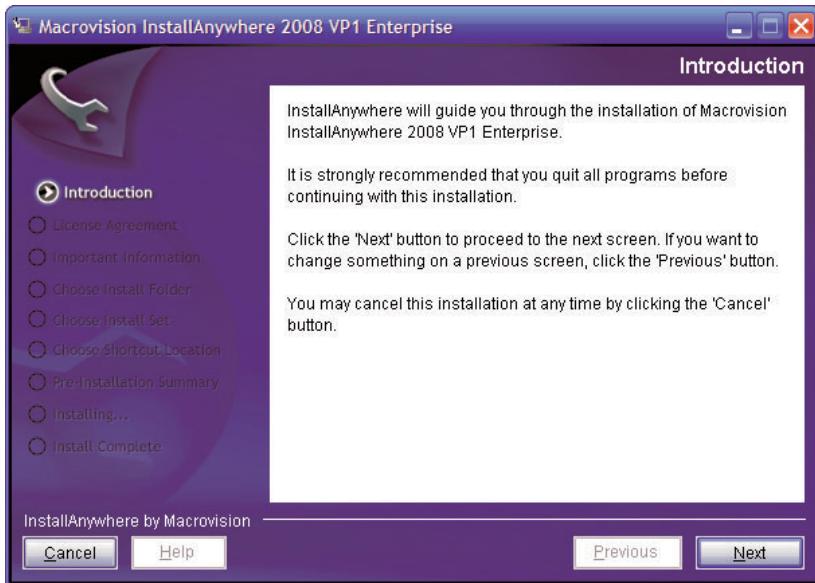
The progress bar displays the percent of the action complete.



**Figure 1-2:** InstallAnywhere Installer Progress

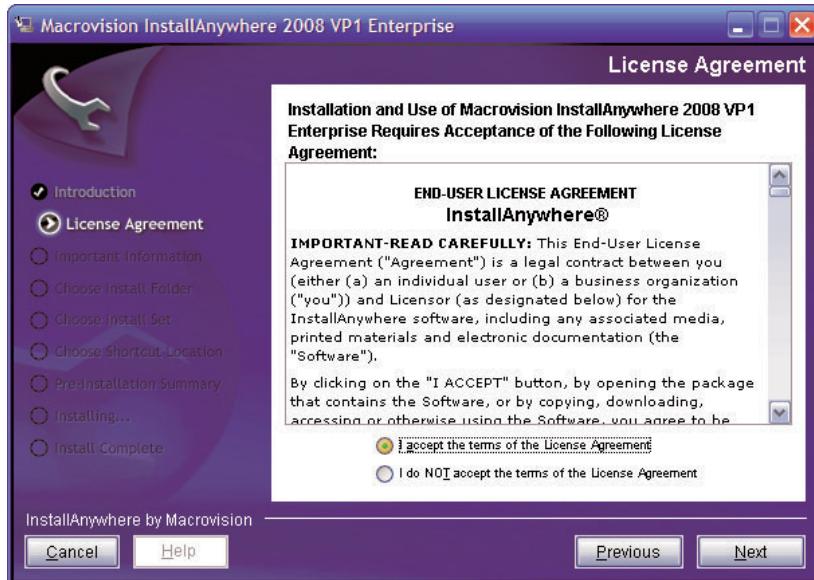
InstallAnywhere's installer utilizes some of InstallAnywhere's advanced user interface (UI) customization options. For example, note the background images and the dynamic list of installation steps used in the installer. These features are available when you use InstallAnywhere's advanced GUI user interface.

The left pane of the Installer can present a list of steps that the installer will perform.



**Figure 1-3:** Informational Pages

These steps are updated as the installation progresses. In the right pane the Installer can present information and accept information from the end user.



**Figure 1-4:** Licensing Agreement

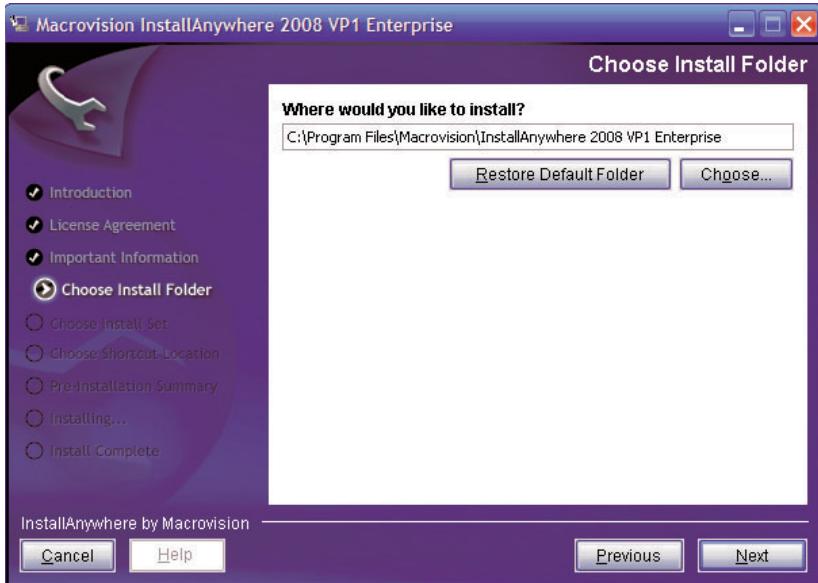
With the InstallAnywhere installer, a license agreement is presented. The user must accept before the installation can continue.

A varying level of detail and information can be packaged in the installer, even HTML and hyperlinks. For example, the InstallAnywhere installer uses this panel to display the product's release notes.



**Figure 1-5:** HTML From Within the Installer

The installer can request installation configuration information from the end user, such as the installation location.



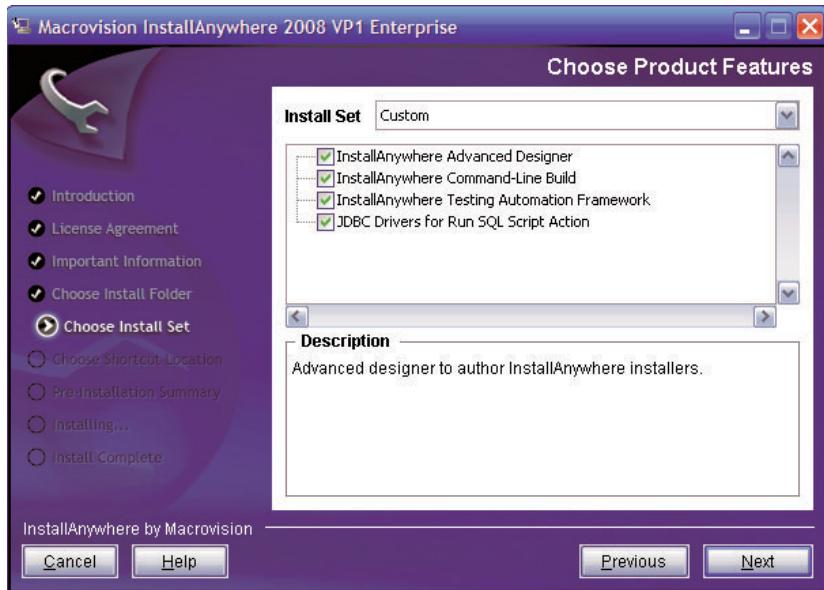
**Figure 1-6:** Choosing an Installation Path

The installer also allows the user select an install set, which is a pre-packaged group of product features.



**Figure 1-7:** Choosing an Install Set

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.



**Figure 1-8:** Choosing Product Features

If the user selects the **Custom** install option, the installer prompts the user with a list of individual product features that can be installed separately.

InstallAnywhere supports the use of DIMs as a way for product developers to collaborate with installation developers.



**Figure 1-9:** Installing InstallAnywhere Collaboration

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 11.

The InstallAnywhere installer requests that the user choose a location for shortcuts to be installed.



**Figure 1-10:** Choosing a Shortcut Location

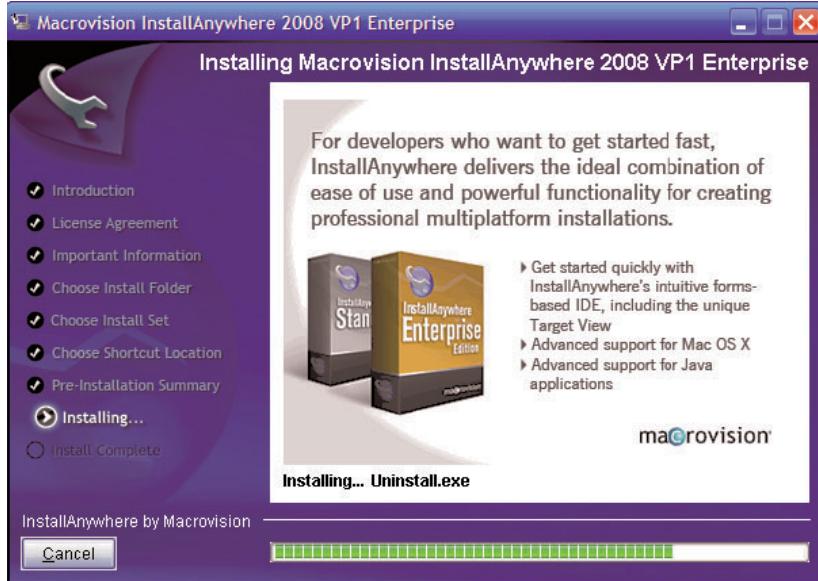
When the installation is done on a Unix system, the same panel would reflect links rather than shortcuts, and on a Macintosh system the user would choose where to install aliases.

The InstallAnywhere installer displays a summary of information gathered in the installation so far. This summary includes disk space calculations and user choices.



**Figure 1-11:** Pre-installation Summary

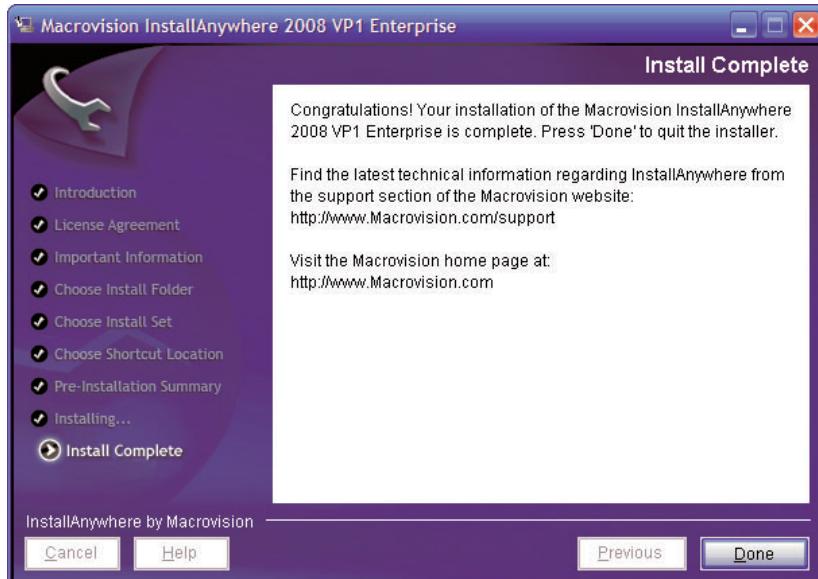
Now the installation of files begins.



**Figure 1-12:** Present Billboards, Animated Graphics, and Progress Bar

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, Macrovision-product graphics are displayed.

When the file installation is complete, the installer presents a panel indicating that the install has completed successfully. If the installation encountered any problems, a list of errors appears.



**Figure 1-13:** Install Complete

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.



# 2

## Starting a Project Using the InstallAnywhere Project Wizard

---

InstallAnywhere has two authoring modes: the Project Wizard and the Advanced Designer. While the Advanced Designer gives you greater control over an installer project, the Project Wizard makes the process easier by making choices for you and guiding you through the process.

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 automatically finds the main class for a Java application.

InstallAnywhere opens displaying the first frame of the Project Wizard, unless the default preference has been changed using the **Preferences** command from 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:

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

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

# Building an Installer Using the Wizard

The following tutorial teaches how to build an installer for a sample Java application, called “OfficeSuite for Java”, which is included in the **InstallAnywhere** directory. Building this installer is covered in the following tasks:

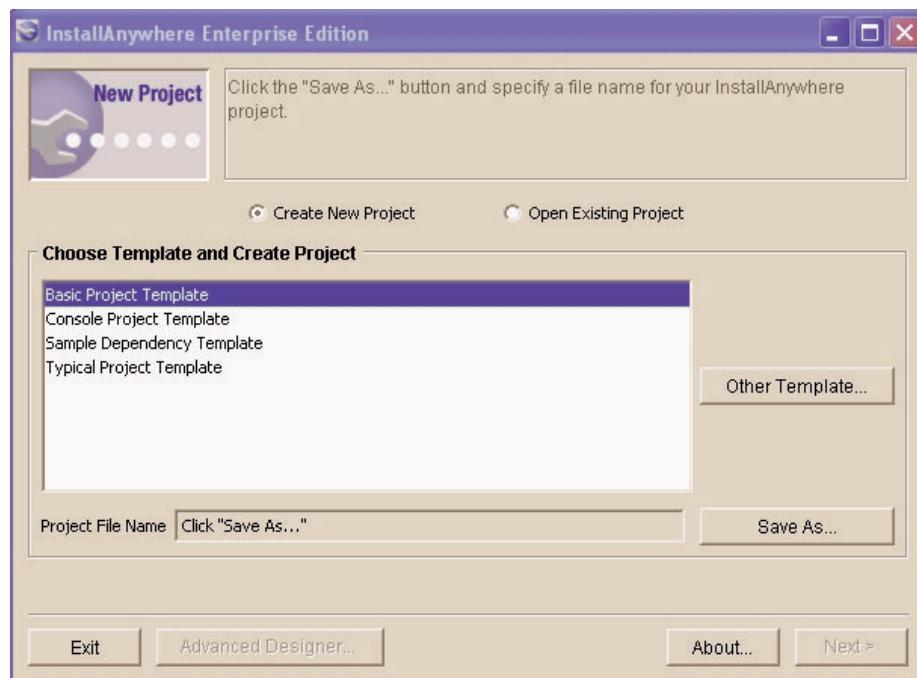
- Creating a New Project
- Setting Project Information
- Installing Tasks
- Building the Installer
- Testing the Project

## Creating a New Project

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.

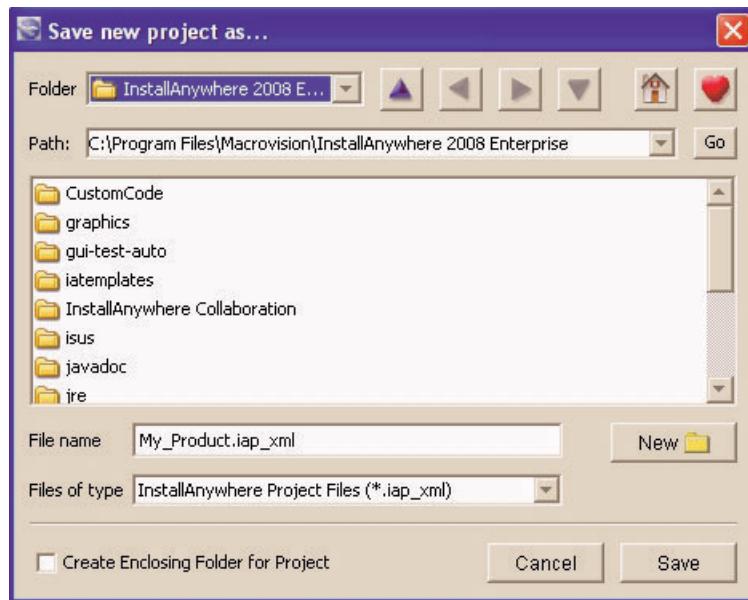
To create a new project:

1. Launch **InstallAnywhere**.
2. On the initial screen, the **Create New Project** option should already be selected.



**Figure 2-1:** Create New Project Window

3. 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 this can be changed.

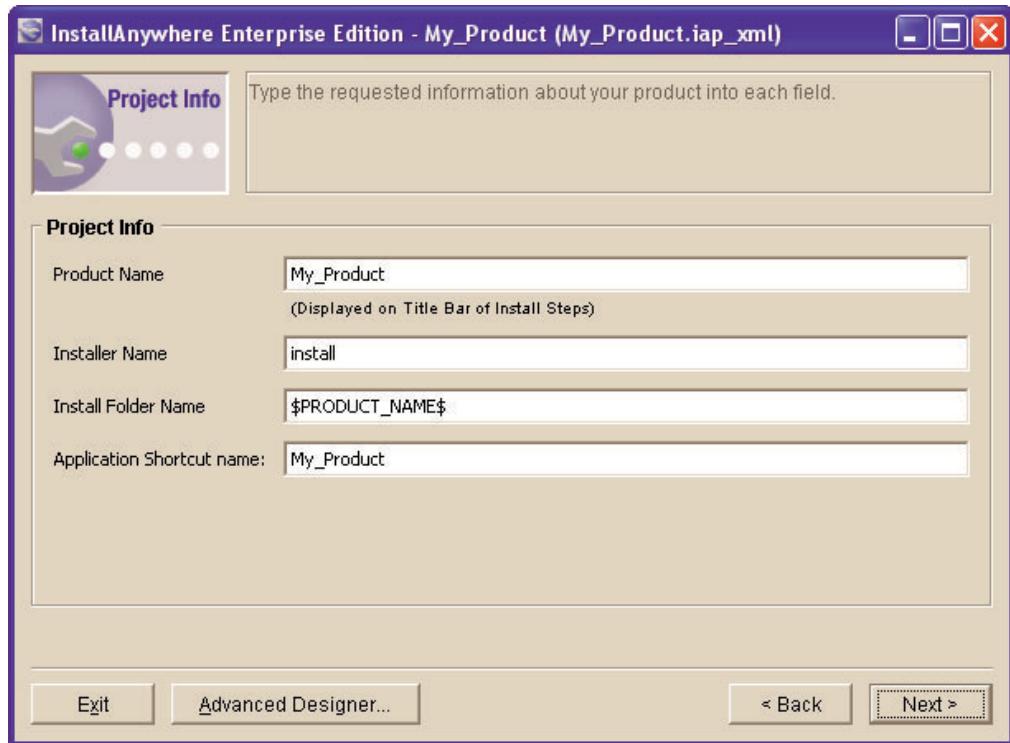


**Figure 2-2:** Save New Project Window

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

# Setting Project Information

Setting the project information 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.



**Figure 2-3:** Setting Project Info Using the Wizard

1. Enter the information in the appropriate text boxes. For this tutorial, refer to the following table:

Heading	Value
<b>Product Name</b>	OfficeSuite
<b>Installer Name</b>	OfficeSuite
<b>Install Folder Name</b>	OfficeSuite
<b>Application Shortcut Name</b>	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.

2. Click **Next** to move forward in the Project Wizard.

# Installing Tasks

This section consists of several steps. First add files to the project, choose the main Java class for starting the project, and set the classpath that the project uses.

## Add Files

1. Click **Add Files**. The **Add Files to Project** dialog box appears.

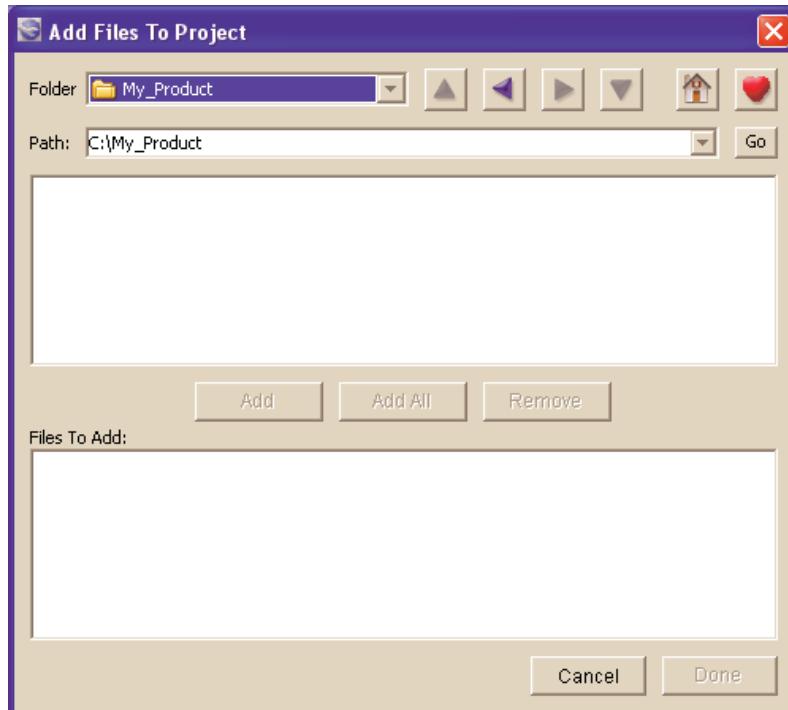


Figure 2-4: Adding Files to the Project Using the Wizard

2. Browse through the list to find the `OfficeSuiteSourceFiles` folder, located within the `InstallAnywhere` installation directory.
3. Click **Add All** to add the `ImagesAndDocs` and `OfficeSuite2000` directories, which are inside the `OfficeSuiteSourceFiles` folder. These files appear in the **Files to Add** list.

This type of file linking adds a static list of files to your project: any files you add later 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.

4. 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.

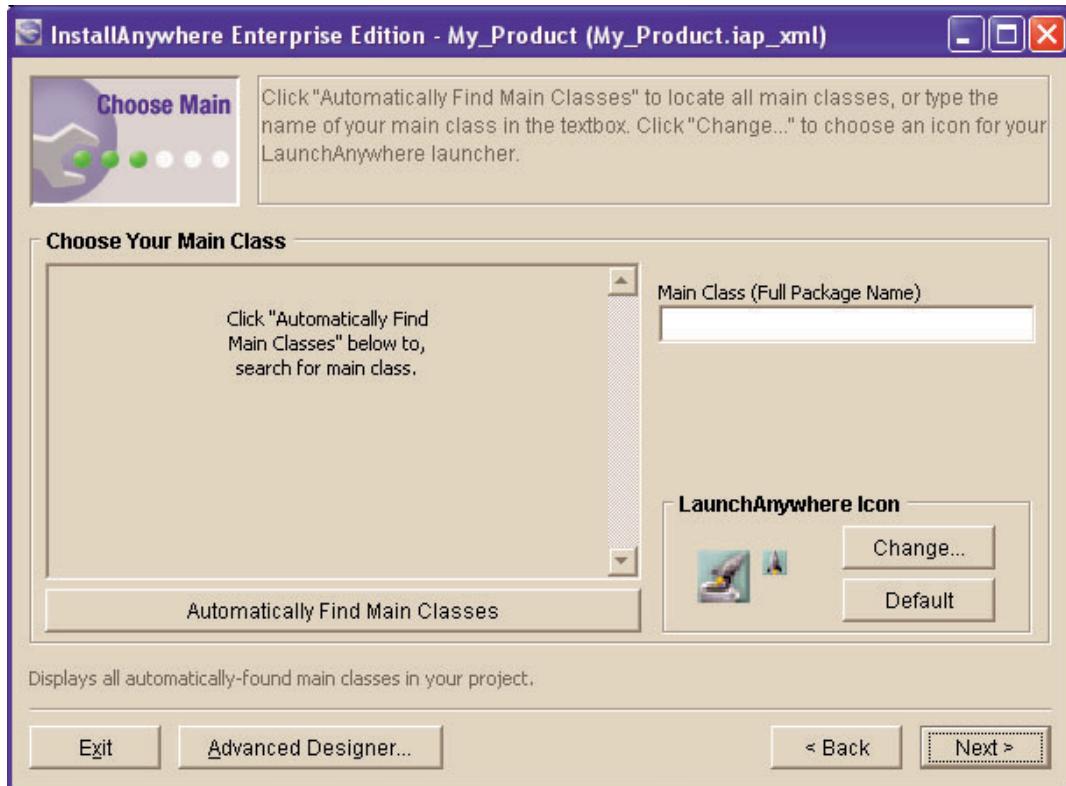
5. Click **Next** to move to the next step in the Project Wizard.

## 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.



**Figure 2-5:** Choosing a Starting Class

1. Click **Automatically Find Main Classes** at the bottom of the screen.
2. Select the main class.
3. Specify a custom icon for the LaunchAnywhere executable by clicking **Change** and choosing a 32-by-32 or a 16-by-16 pixel GIF for the application icon. Navigate to the `Image and Docs` directory and choose `OfficeIcon.gif`.

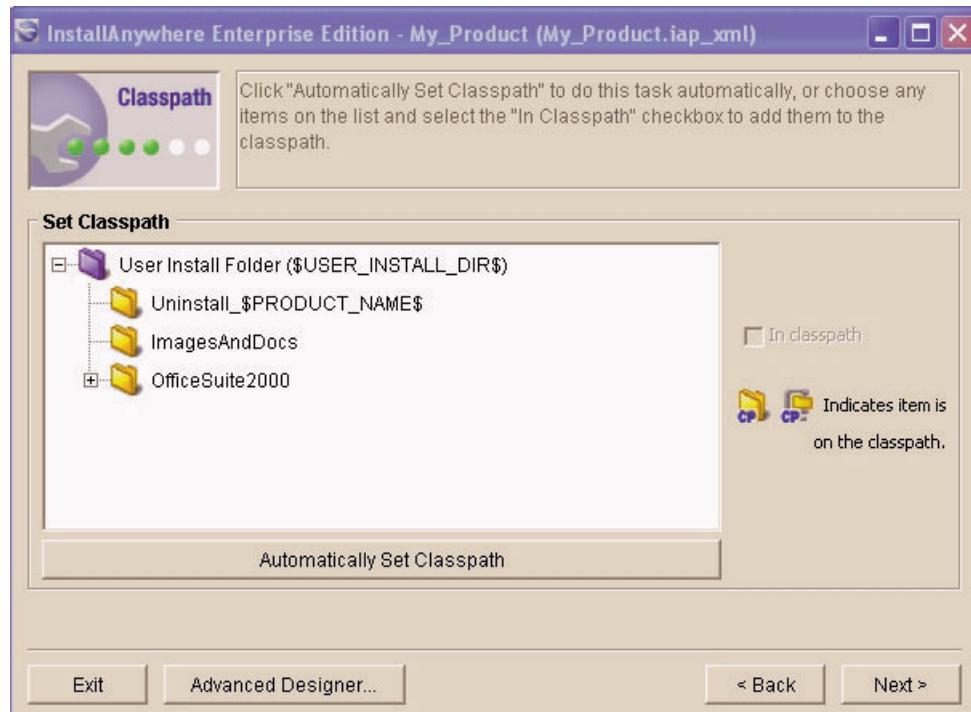


**Note:** Windows-only .ico files are not supported.

4. Click **OK** to confirm and close the dialog box. The icon appears on the main screen.
5. Click **Next** to move to the next step in the Project Wizard.

## Setting the Classpath

1. **Automatically Set Classpath** configures a Java application's classpath, which is a list of directories and .jar files containing classes used by the application.



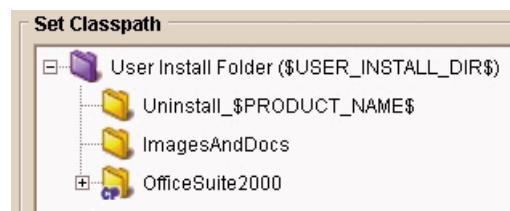
**Figure 2-6:** Automatically Set Classpath Window

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.

2. 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.



**Figure 2-7:** File List After Automatically Setting Classpath

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

# Building the Installer

The first several items on the **Build Installer** screen, from **Mac OS X** through **Unix (All)**, represent installers that can be double-clicked on their respective platforms.

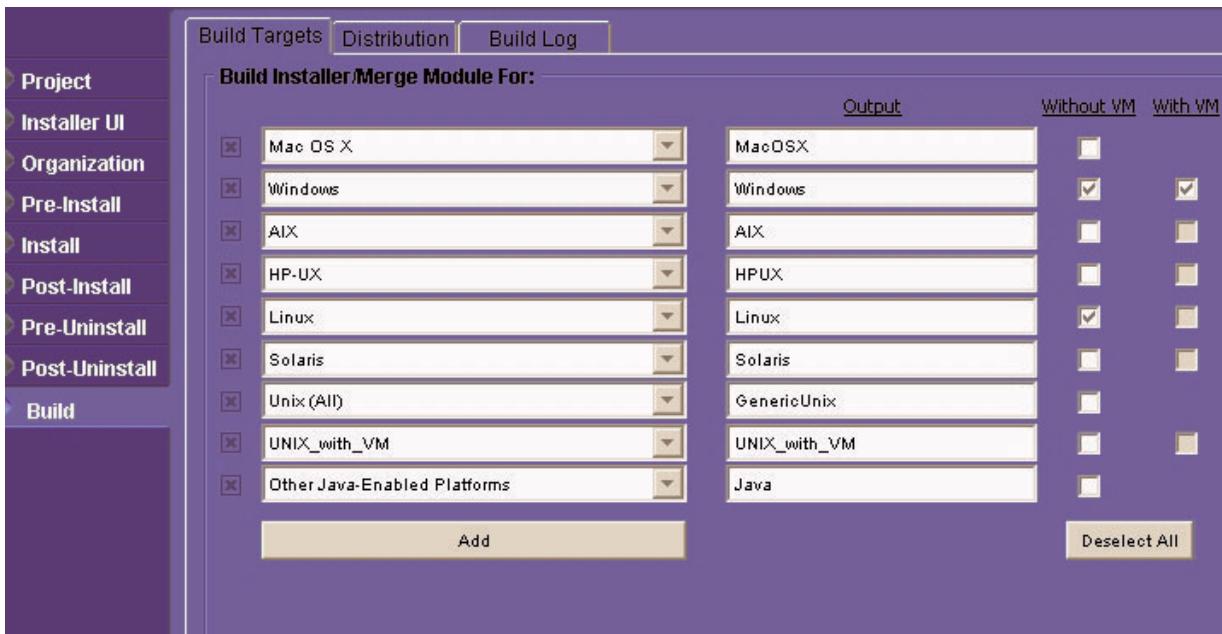


Figure 2-8: Build Installer Options

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. You 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 allows end users to choose the appropriate installer for their system.

1. Choose the desired destination platforms.
2. Click **Build**.

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

# Testing the Project

Now that an installer is built, it is important to test the project to verify that it functions as desired. To test the project:

1. Click **Try It**.
2. After deploying the sample installer:
  - On Windows, go to the OfficeSuite program group and choose OfficeSuite.
  - On Unix, change directories where the program was installed and enter “OfficeSuite”.
  - On Mac OS X, double-click the OfficeSuite icon on the desktop.
3. After launching OfficeSuite for Java, quit by selecting **Exit** from the **File** menu.

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



**Tip:** On Windows, hold down the Control (Ctrl) key while the installer launches to see the debug output.

4. 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 are located in the `Build_Output` directories found in the same folder as the `.iap_xml` project file.

The `Build_Output` folder, also contains the `Web_Installers`, and `CDROM_installers`. From within each of these sub-directories, choose the platform to test. For the CDROM installer, transfer the entire contents of the `CDROM_Installers` sub-directory.



# 3

# Introduction to the Advanced Designer

---

This chapter contains information on:

- Installation Planning and Goals
- 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

## 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 A, "Installation Planning Worksheet."

# Working with Advanced Designer

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.

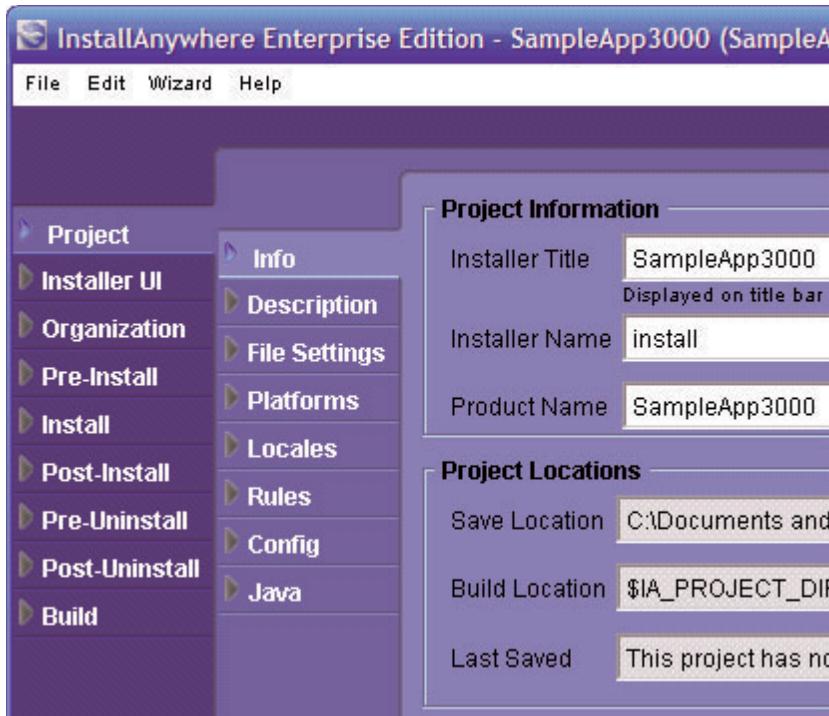


Figure 3-1: Advanced Designer

The following table provides Advanced Designer menu descriptions.

**Table 3-1:** Advanced Designer Menu Description

Name	Description
<b>Project</b>	Settings related to your specific project. These include general settings, file settings, and localization settings.
<b>Installer UI</b>	Set the look and feel for the installer by adding background images, billboards, and other graphical elements.
<b>Organization</b>	Manage Install Sets, Features, Components, and Merge Modules.
<b>Pre-Install</b>	An ordered sequence of panels and actions that occur before file installation.
<b>Install</b>	Manage the file installation tree and installation-time actions.
<b>Post-Install</b>	An ordered sequence of panels and actions that occur after file installation.
<b>Pre-Uninstall</b>	An ordered sequence of panels and actions that occur before file uninstallation.
<b>Post-Uninstall</b>	An ordered sequence of panels and actions that occur after file uninstallation.
<b>Build</b>	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, refer to the Advanced Designer tutorial.

## 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.

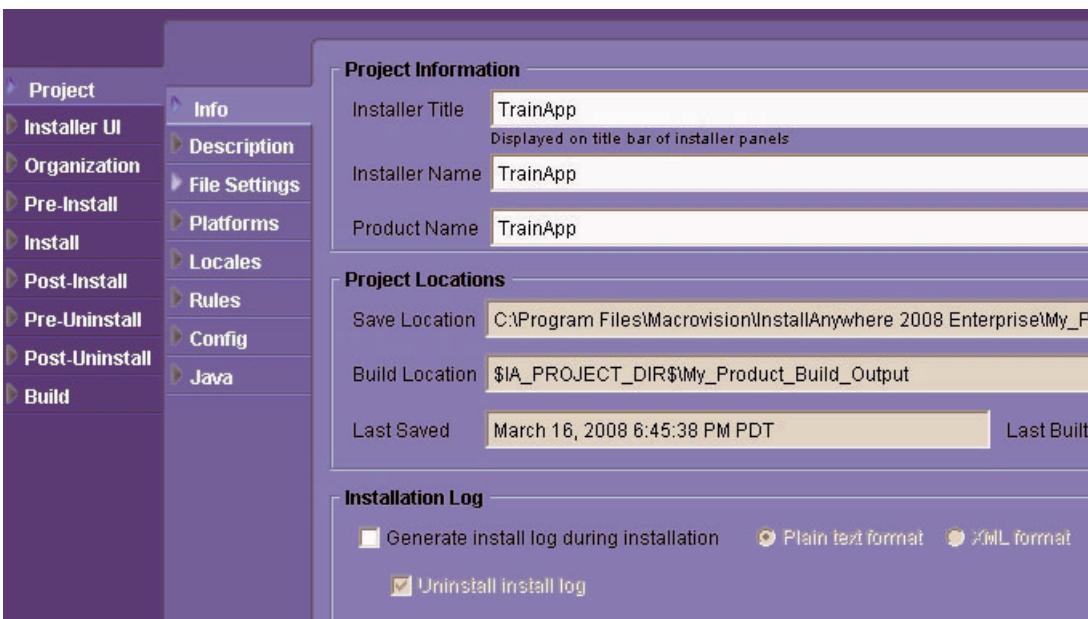
## Installer Identification and Version

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. You can control the creation of 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.



**Figure 3-2:** Project Information and 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.

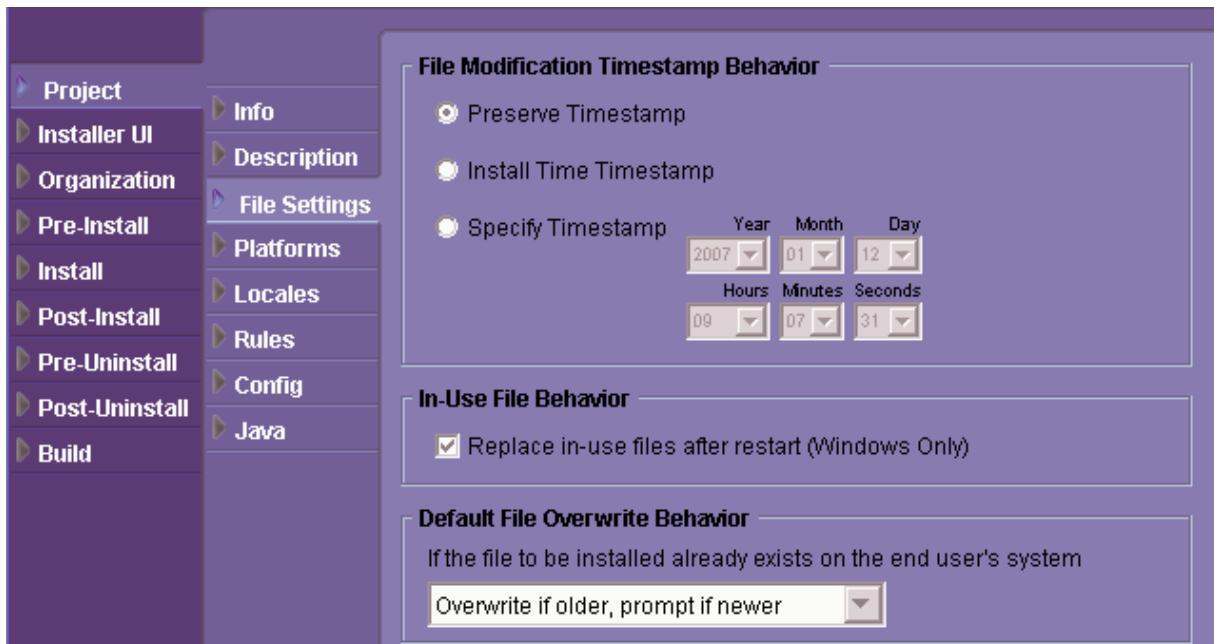


Figure 3-3: File Settings and Modifications

## 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 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.



**Note:** The file's 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 have the same name 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 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 and prompt if newer. The options for determining overwrite behavior may be prompted or set as a default. If the file-overwrite options are set to prompt the user, a sample prompt appears similar to the following figure..

**Table 3-2:** Overwrite Behavior

Overwrite Option	Select this option to:
<b>Always overwrite</b>	Install files without giving the user the choice whether to overwrite files which currently exist on the computer.
<b>Never overwrite</b>	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.
<b>Overwrite if older, do not install if newer</b>	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.

**Table 3-2:** Overwrite Behavior

Overwrite Option	Select this option to:
<b>Overwrite if older, prompt if newer</b>	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.
<b>Prompt if older, do not install if newer</b>	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.
<b>Always Prompt User</b>	Prompt the user whenever an installation file exists on the target computer.

If the file-overwrite options are set to prompt the user, a sample prompt appears similar to the following figure.

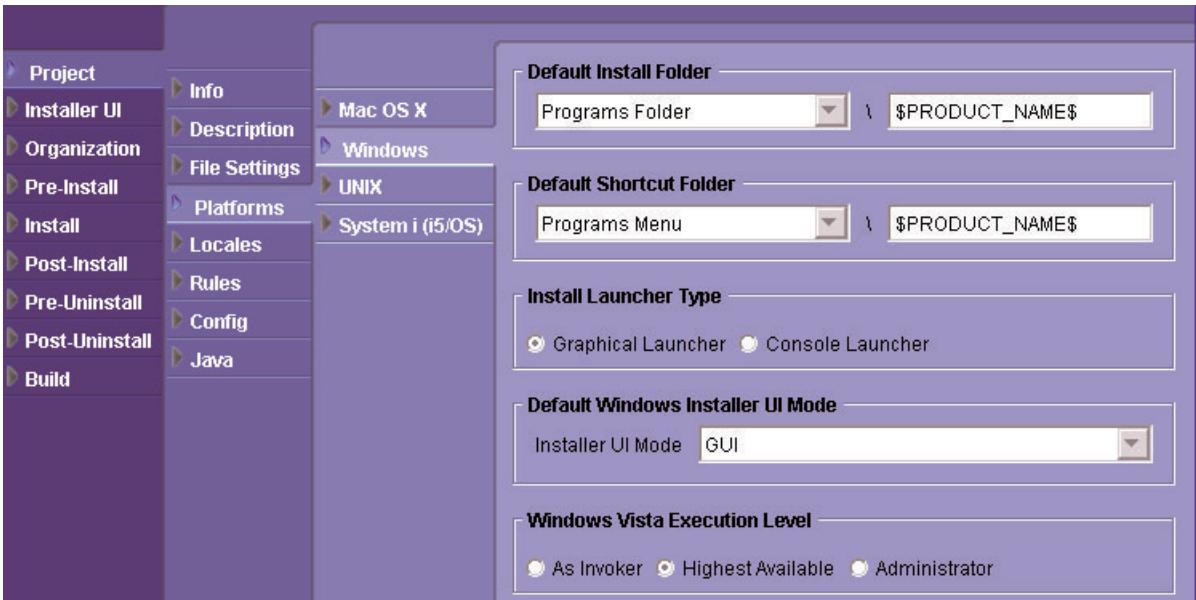


**Figure 3-4:** Overwrite Sample Prompt

# 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.



**Figure 3-5:** Platform Support

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.

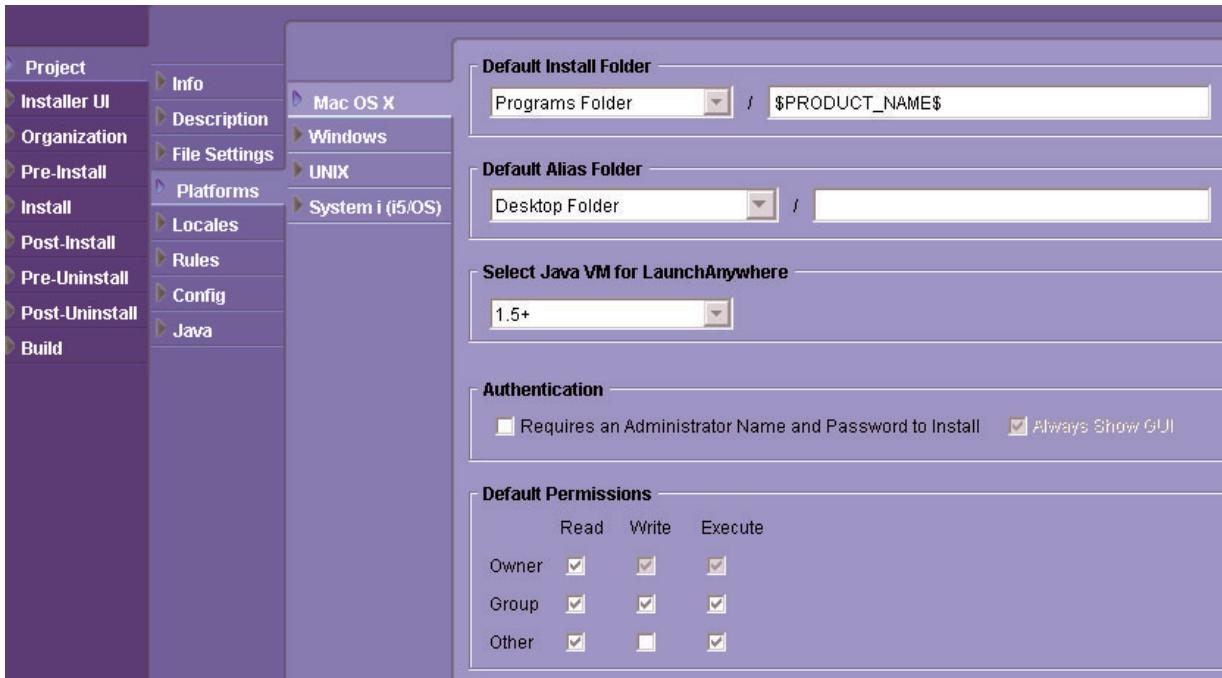
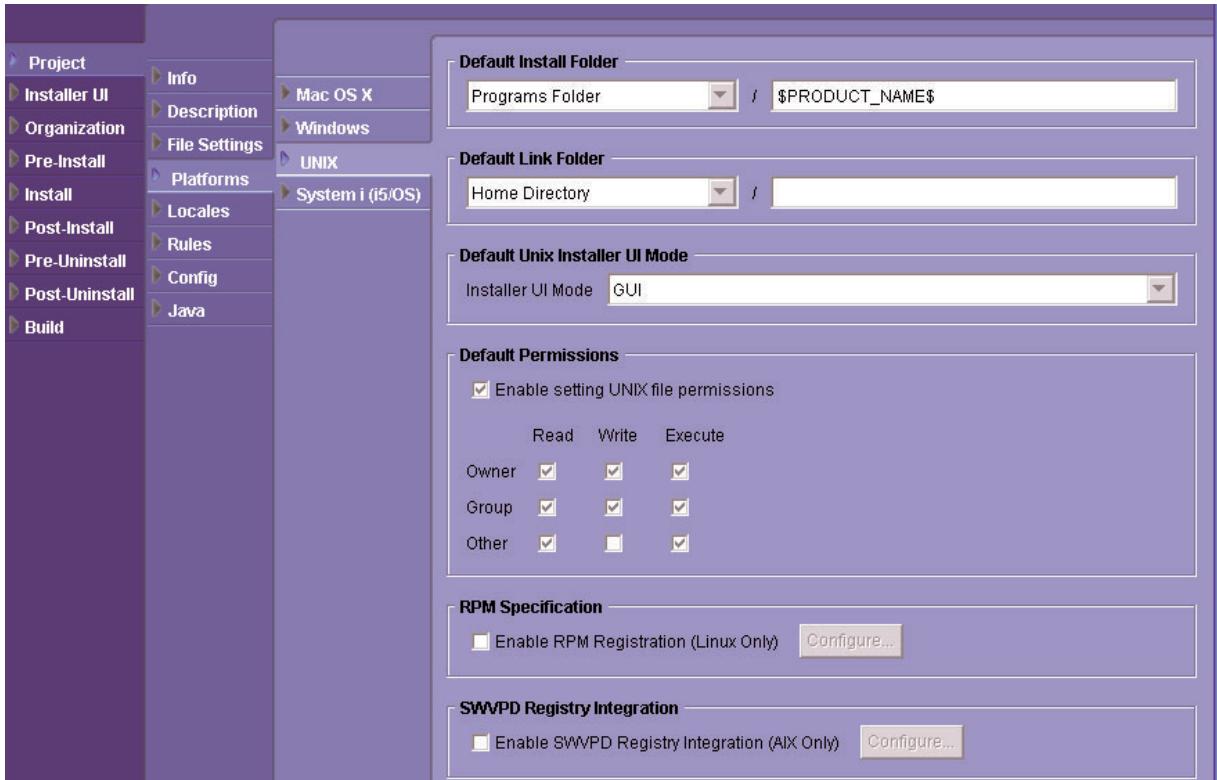


Figure 3-6: Mac OS X Support

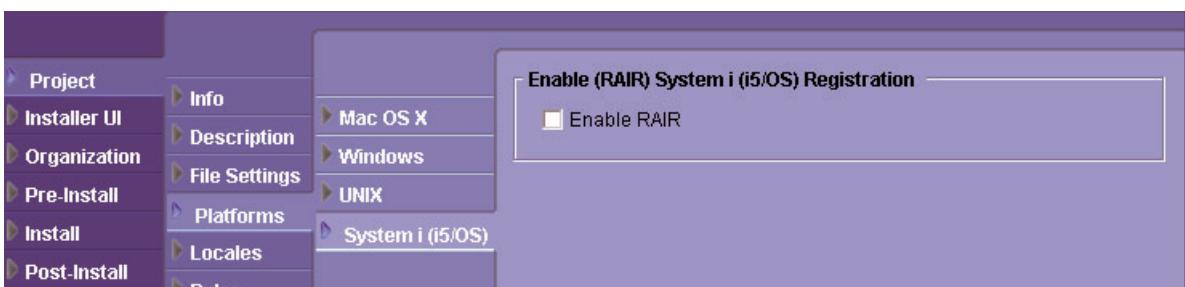
The Unix task shows similar settings. For Linux target systems, you can enable RPM (Red Hat Package Management) support. RPM is a package of installation tools that InstallAnywhere installers will use in the Linux environment and other Unix environments. The RPM feature enables the installer to interact with and make entries into the RPM database.

Beginning with InstallAnywhere 2008 Value Pack 1, you can additionally specify to integrate with the AIX Software Vital Product Database (SWVPD).



**Figure 3-7:** Unix Support

Beginning with InstallAnywhere 2008 Value Pack 1, you can also specify to integrate with the Registered Application Information Repository (RAIR) on i5/OS systems.



**Figure 3-8:** System i (i5/OS) Support

## 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, refer to “Localizing and Internationalizing InstallAnywhere Installers.”

## Rules Before the Pre-Install Task

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.

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, refer to the `gui-test-auto` subdirectory of the InstallAnywhere distribution.

For more information on debugging, refer to Chapter 13.

# 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, `1ax.n1.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.

With InstallAnywhere, you 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.

For additional information on custom code, refer to Chapter 13.

## 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



# 4

## Building Releases

---

Once you have defined the properties, file links, behavior, and other related information of your installation project, you can build releases for testing and eventual duplication and distribution. This chapter describes how to build releases from the graphical InstallAnywhere environment.

For additional information on automated tools for building your InstallAnywhere projects, refer to Chapter 12.

## Build Targets

In the **Build** task, you define the target operating systems for your installer as well as defining the form for distributing the installer. In **Build > Build Targets**, you specify the desired target operating systems. You also indicate whether to provide a VM (Java Virtual Machine) with the installer to provide greater ease of use for the end user. For any build target that includes a VM, you should select the VM to bundle in the **VM to Bundle with Installer** list.

Starting with InstallAnywhere 2008 (Enterprise Edition), you can define *dynamic build targets*: using the **Build Targets** tab (pictured in the following figure), 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.

To create a build target, click **Add** at the bottom of the list of existing targets; to delete a build target, click the **X** icon to the left of the target.



Figure 4-1: Build Targets Tab

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 the build is taking place, a progress indicator similar to the following is displayed.

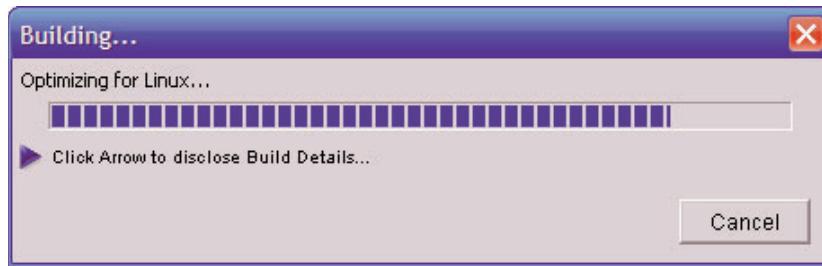


Figure 4-2: Build-progress Indicator

While InstallAnywhere provides options for many flavors of Unix, it also enables the creation of 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.

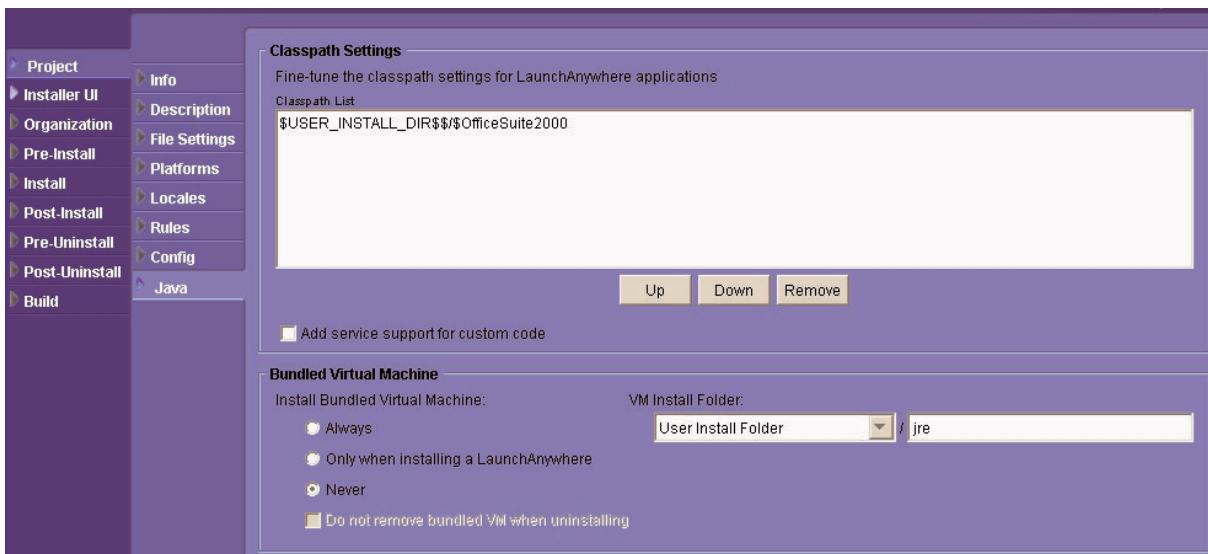
## VM Packs

VMs that you bundle with a target (for the sake of target systems that may not have an appropriate VM installed) are implemented as VM packs. InstallAnywhere ships with some VM packs for you to bundle, and the Macrovision web site provides additional 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 files, which are .zip or .jar archives that contain the Java VM and a vm.properties file. (The vm.properties file contains display and platform information about the VM pack.) 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**.

In addition to the default VM packs available to InstallAnywhere, you can create your own. For information on creating a VM pack, see the InstallAnywhere help topic “Creating VM Packs.”

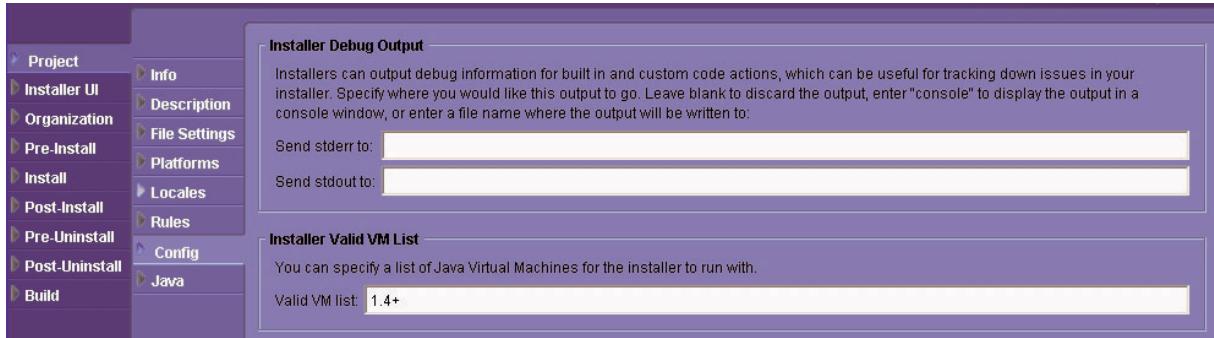
In the **Bundled Virtual Machine** area of the **Project > Java** task, you can specify whether a VM you bundle with your release image should be installed on a target system (as opposed to being temporarily installed for the sake of the running installer).



**Figure 4-3:** Bundled Virtual Machine Options

## VM Selection

For releases with which you have not bundled a VM, you can specify which Java versions can be used with the installer, using the **Valid VM List** setting in the **Project > Config** task.



**Figure 4-4:** Specifying a Valid VM List

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 > Config** tab of the Advanced Designer.

The values for these settings 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 enables the installer or application to run only against the JRE 1.4.2\_02. A value of “JDK\_1.5.0\_06” enables 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, they will in effect be combined with an OR operator. In other words, a VM is valid if it matches any of the given expressions.

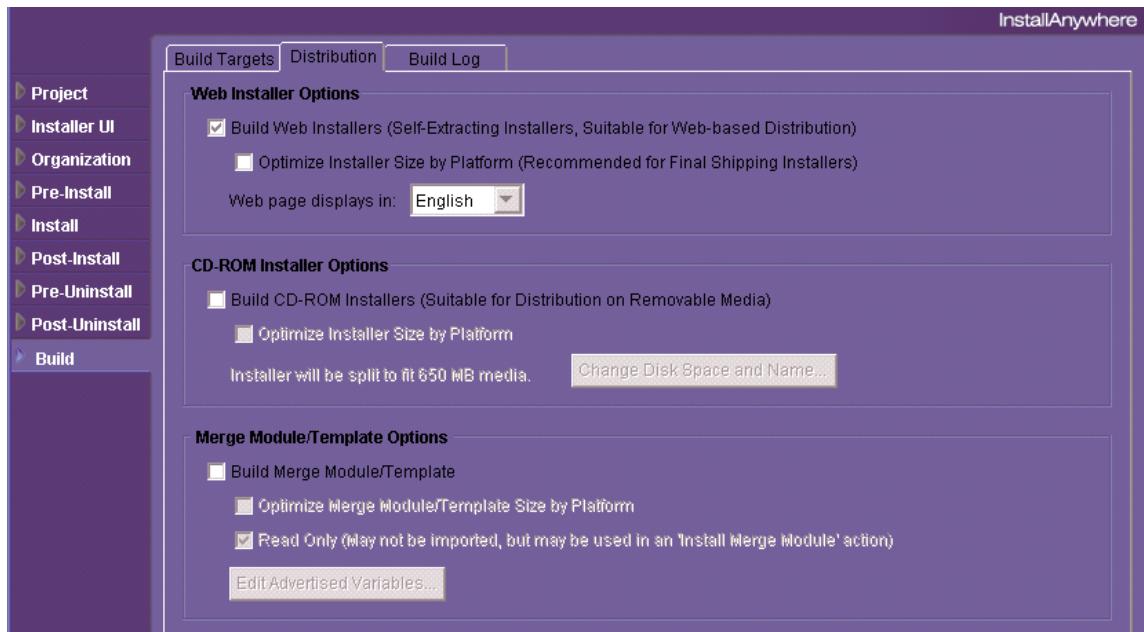
The optional JDK or JRE specifies which type of VM is valid. If specified, it must be followed by an underscore character. You can only specify one or the other.

The version number can have varying degrees of precision; however, it is recommended that you have 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).

# Distribution

In the **Distribution** subtask, you define the form of the release image to be distributed. You 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.



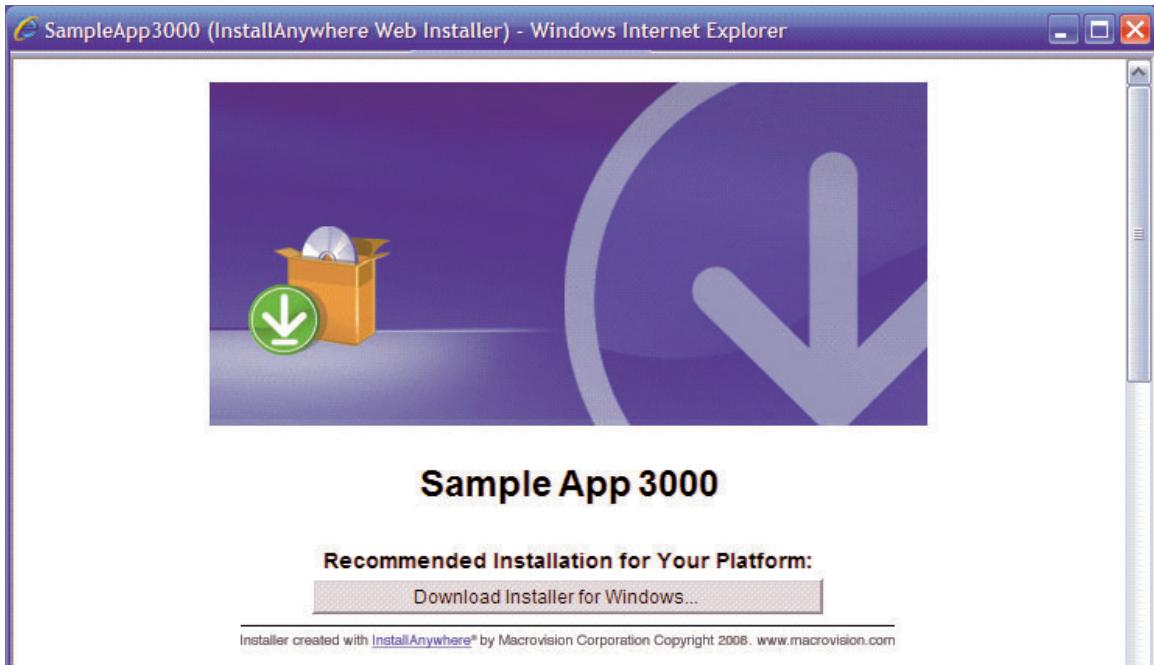
**Figure 4-5:** Distribution Tab

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). You can also select in which language to build the target web page.

The web page presented to the user appears similar to the following.



**Figure 4-6:** Sample Web Installer

You can test the web installer by clicking **Try Web Installer** at the bottom of the **Build** task after a successful build.

## CD-ROM/DVD Installers

CD-ROM/DVD installers consist of multiple files meant to be burned 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 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 enables you 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

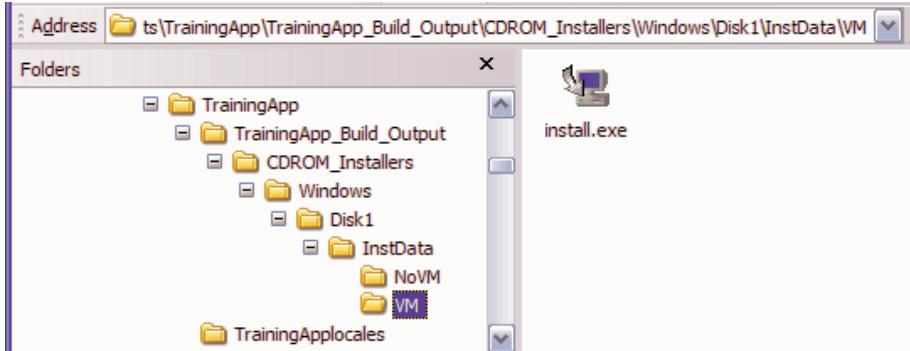
The directory structure for CD-ROM installers is:

```
Platform1/Disk1/InstData/  
    |-...  
    |-MediaId.properties  
    |-Resource1.zip  
  
Platform1/Disk2/InstData/  
    |-MediaId.properties  
    |-Resource2.zip  
  
Platform1/Diskn/InstData/  
    |-MediaId.properties  
    |-Resource $n$ .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 output may appear in Windows Explorer as follows:



**Figure 4-7:** Build Output in Windows Explorer

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 be:

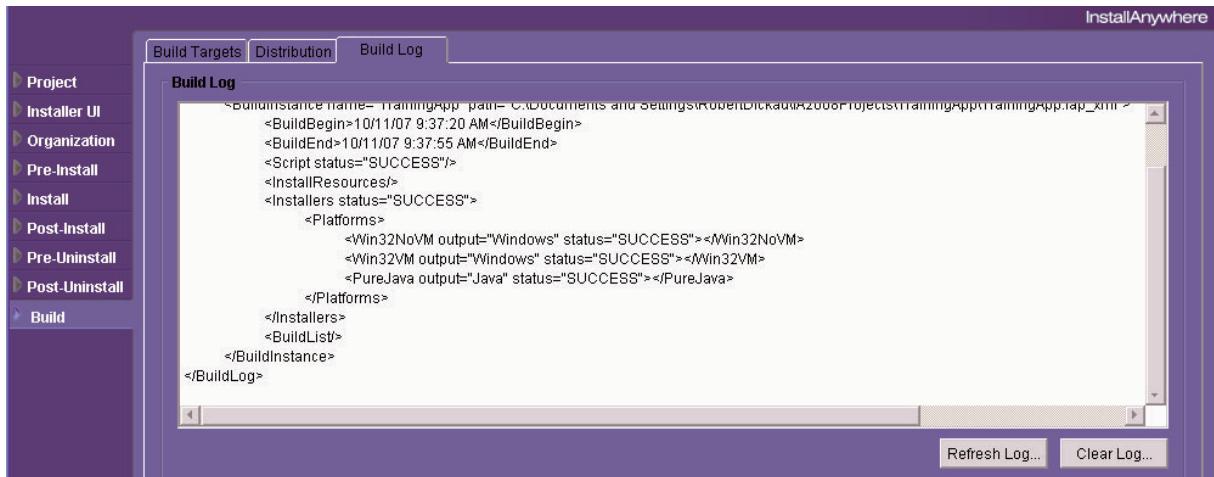
```
<ISO CD NAME>  
    |-Disk1  
    |-Disk2
```

# Merge Modules and Templates

You can also build *merge modules* in this task. Merge modules enable you to create installers that can easily be integrated into other InstallAnywhere installers. More information on Merge Modules and Templates is available in Chapter 11.

## 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, and click **Delete Log** to remove the log.



**Figure 4-8:** Build Log

**Chapter 4: Building Releases**

*Build Log*

# 5

## Basic Installer Customization

---

This chapter contains information on:

- Customizing the Installer Look and Feel
- Installer UI Modes
- Splash Screens
- GUI Panel Additions
- Background Images
- Frame UI Settings
- Billboards
- Help
- Conditional Logic
- 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.

This chapter covers some of the many 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 the Installer Look and Feel

One of the keys to a professional looking installer is the appearance of the installer itself. In most cases, you will want the installer to reflect the image of your product or company. InstallAnywhere enables you 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.

There are numerous ways to customize and modify the Installer:

- **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:** Additions to **GUI Installer Panels** option enables you to display a list of steps or an image along the left side of the installer's panel.
- **Background Images:** the 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.

# Installer UI Modes

Defined by the **Allowable UI Modes** setting, InstallAnywhere installers can run in several different modes defined by the end-user interface. Refer to the following table for additional information.

**Table 5-1:** UI Run Modes

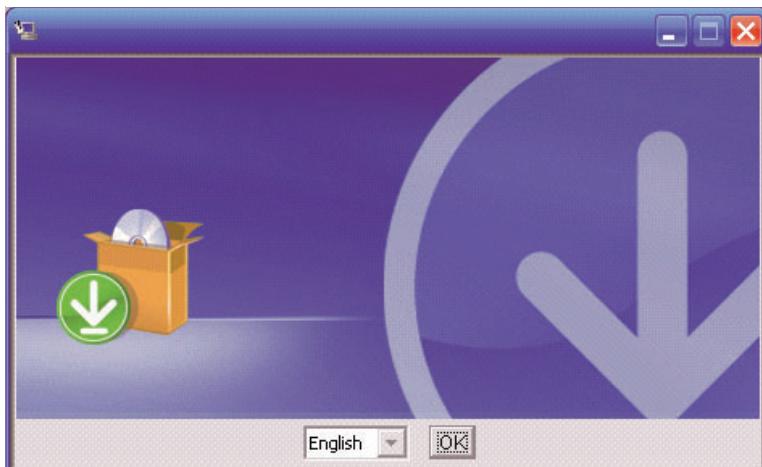
Type	Description
<b>GUI</b>	The GUI mode uses Swing, an end-user interface toolkit provided by the Java Foundation Classes. InstallAnywhere's GUI installer interface provides a rich end-user experience including background graphics, rendered HTML, and alpha transparency for graphics used in the installer. The InstallAnywhere installer itself is an example of a GUI installer.
<b>Console</b>	Console mode provides a TTY or terminal-style interface that can enable 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.)
<b>Silent</b>	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.

The default splash screen appears similar to the following figure.

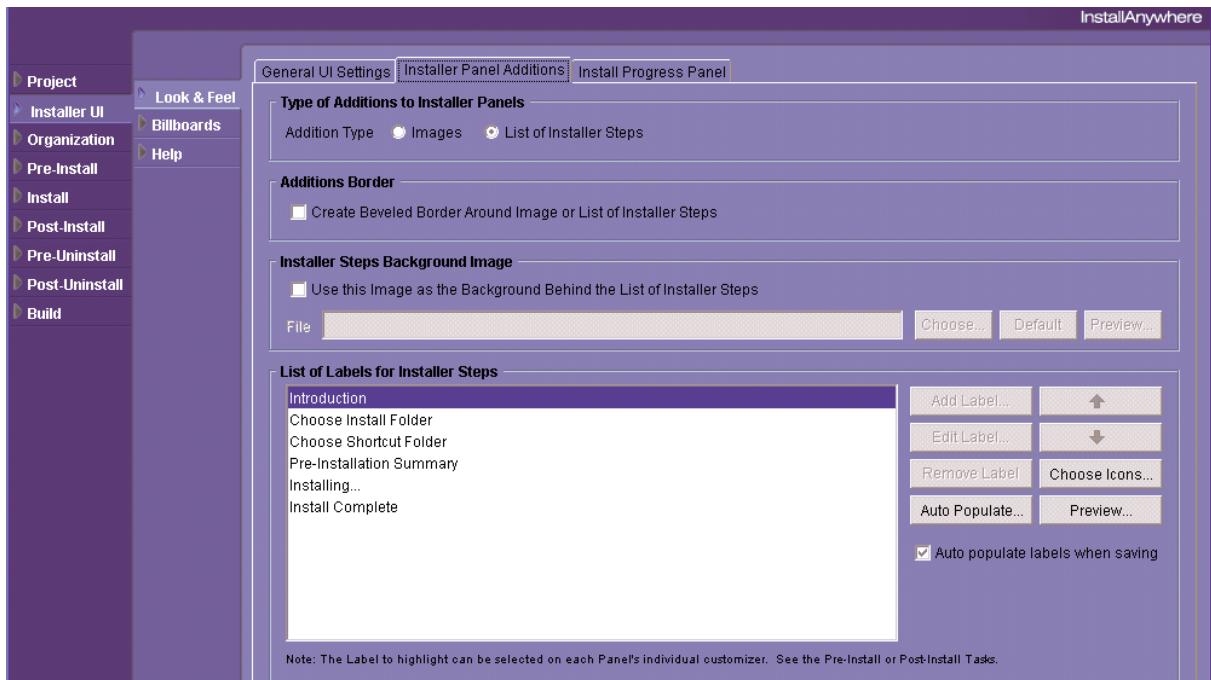


**Figure 5-1:** Default Splash Screen

# GUI Panel Additions

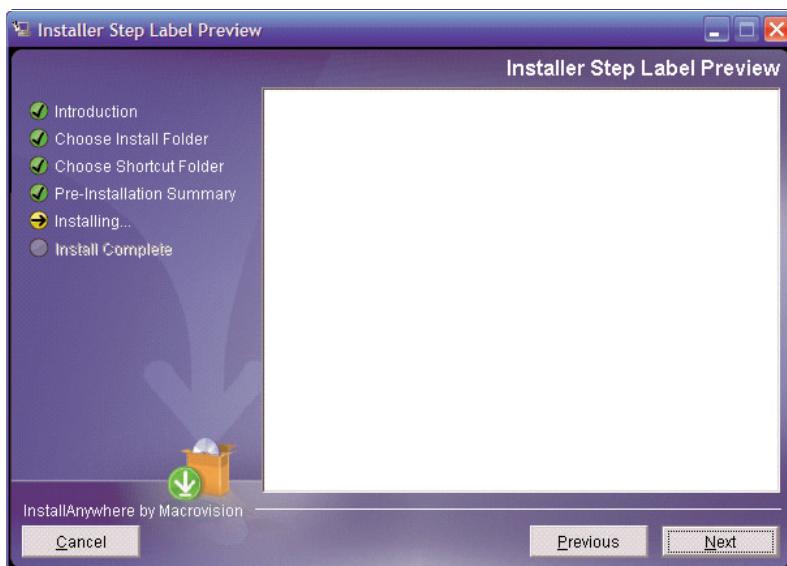
The **Additions to GUI Installer** panels option allows you to display a list of steps or an image along the left side of the installer's panel.

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



**Figure 5-2:** Installer Look & Feel: Panel Additions

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

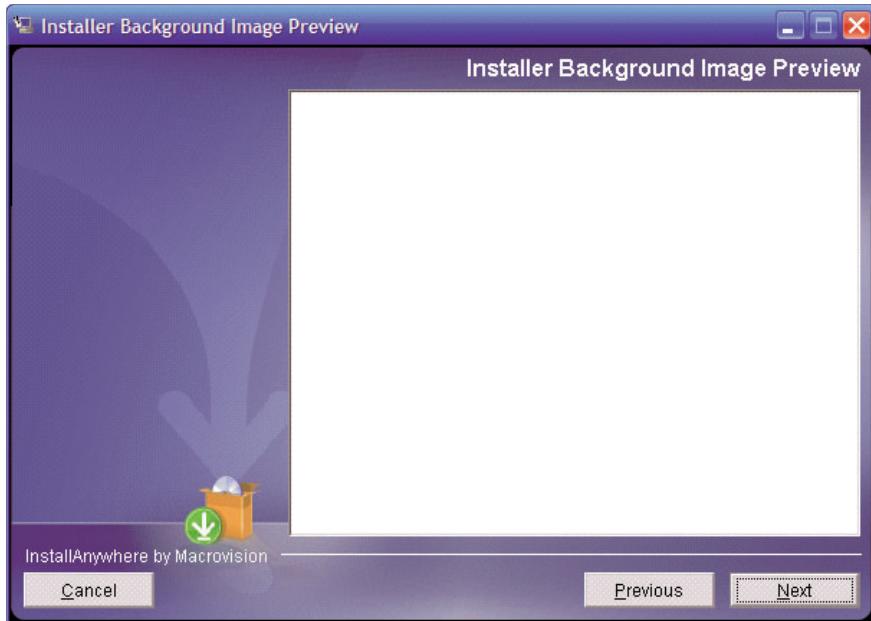


**Figure 5-3:** Installer Modifications

# 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:

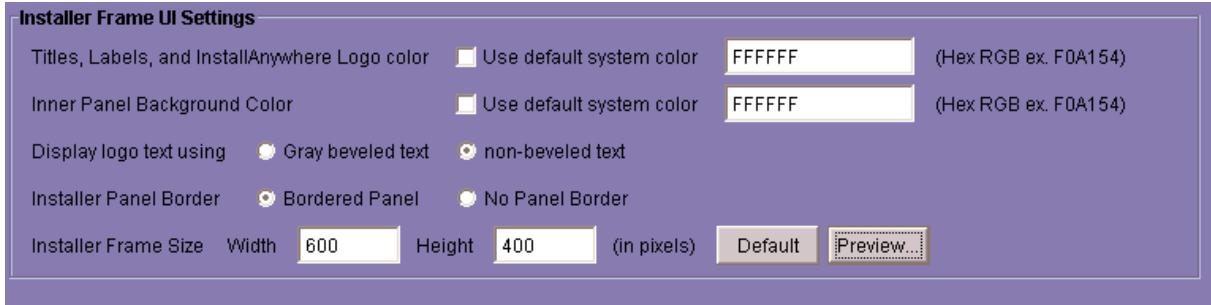


**Figure 5-4:** Default Background Image

In addition, you can specify the reverse the image when the installer runs using a right-to-left locale (Arabic or Hebrew). For more information about right-to-left locales, refer to Chapter 15.

## 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.



**Figure 5-5:** Frame User Interface Settings

## 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.

# Conditional Logic

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

The following segment covers 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 are covered 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 number of preset rules included in the InstallAnywhere Standard Edition (S) and Enterprise Edition (E), listed in the following table.

**Table 5-2:** Pre-set Rules

Name	Editions	Description
<b>Check File/Folder Attributes</b>	E	This rule enables you to check the attributes of a file or directory that already exists on the target system. The rule enables you to check if the object exists, whether it is a file or a folder/directory and whether it is readable and/or writable.
<b>Check If File/Folder Exists</b>	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.
<b>Check Platform</b>	E S	This rule enables 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.
<b>Check System Architecture</b>	E S	This rule enables 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.
<b>Check User-Chosen Language</b>	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.
<b>Compare File Modification Timestamp</b>	E	This rule enables you to compare the timestamp of an existing target file in order to make an overwrite decision.
<b>Compare InstallAnywhere Variables</b>	E S	This rule enables 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.
<b>Evaluate Custom Rule</b>	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.

**Table 5-2:** Pre-set Rules

Name	Editions	Description
<b>Match Regular Expression</b>	E	The Match Regular Expression rule enables 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 useful expressions, and web applications that can be used to verify the validity of your expression on the World Wide Web.

# 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:**

\$prop.user.name\$      equals

**Operand 2:**

jim

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

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



# 6

## Installer Organization

---

The data you install on the target system are organized in a hierarchical structure. This structure contains all of the install sets, features, components, files, and other data to be installed on a target system. In this chapter you will learn about the different levels of design that make up your installation project, and see the different ways to add files to your installation.

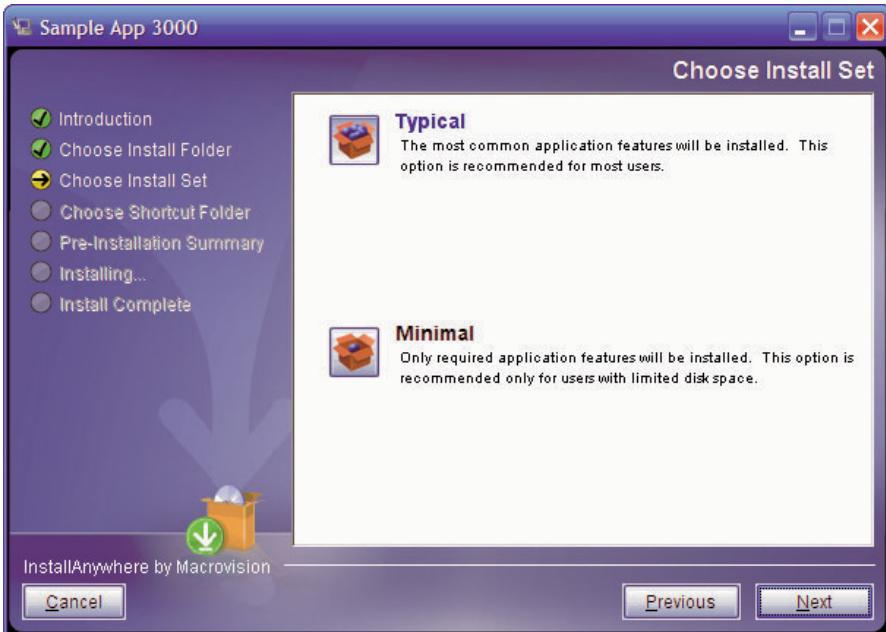
## Install Sets, Features, and Components

The basic organizational elements of your installation project are install sets, features, and components. The following sections describe these three levels of design.

### Install Sets

*Install sets* are the broadest organizational concept within InstallAnywhere. Install sets are a set of product features which represent high-level, easily selectable, installation options. These sets are generally options such as “Typical”, “Minimal”, and “Custom”; or “Client Only” and “Client and Server.”

End users select the desired install set using the Choose Install Set panel (pictured below) or console.

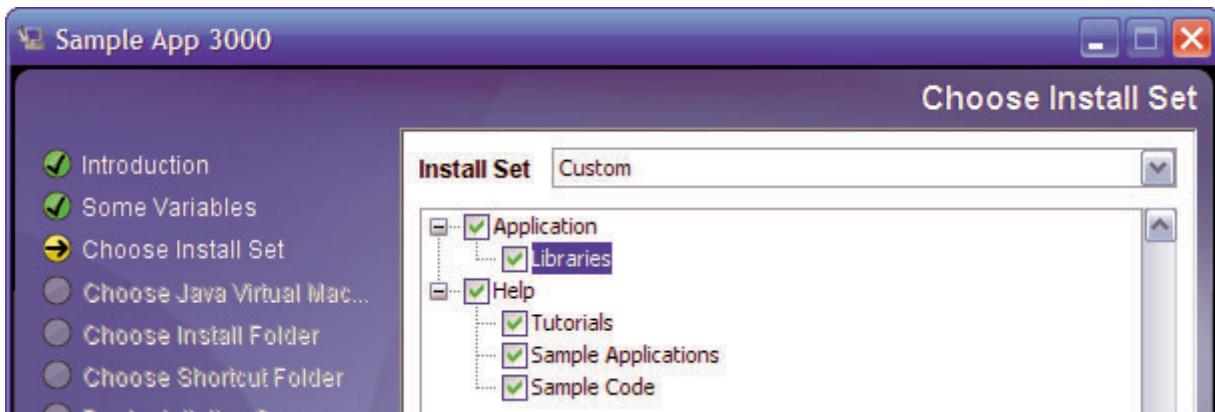


**Figure 6-1:** Choosing Install Set

## Features

*Features* are logical groupings of capabilities in your product. Features are effective when you want to give end users a high degree of control over what product data to install. Features are intended to identify distinct parts of the product so the end user may choose whether to install each one. It is up to you to define the logical grouping of components into features by assigning components to the features.

You can arrange features into a tree made up of top-level features, subfeatures, sub-subfeatures, and so forth. (There must be at least one feature in your project.) The following figure shows a tree of features and subfeatures as they would appear to the end user at run time: the top-level features are called "Application" and "Help", and the "Help" feature is made up of sub-features called "Tutorials", "Sample Applications", and "Sample Code".



**Figure 6-2:** Adding Specific Features

Each feature can belong to one or more install sets. Features are displayed to the end user if you enable the Custom install set, as described later in this chapter; a feature is the smallest separately installable piece of your project from a user's standpoint. When a user selects an install set other than Custom, all of the features in that install set will be installed on the target system.

As described later in this chapter, you can assign files directly to features, or you can assign files to components and then assign the components to features.

## Components

*Components* are the smallest piece of an installation handled by the installer. From your perspective as the installation developer, components 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 (compared to features). Common components may be shared between multiple installers, multiple versions of a product, or multiple products. For example, two products in a suite might have several shared components. Moreover, by adding rules to particular components, you shield users from some of the underlying complexity of a product installation.

Components are uniquely identified so that you can 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 you can search for a particular version of a component on a system in order to determine whether the latest version has been installed to a particular location.

Just as an install set is made up of one or more features, each feature contains one or more components. Similarly, 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 given file or action may belong to only one component. All installers must have at least one component, and can have as many as needed.



**Note:** For most installations, you will not need to manipulate the components in any way. Components are automatically generated based on the way that you assign files to features and install sets.

## Using the Organization Task

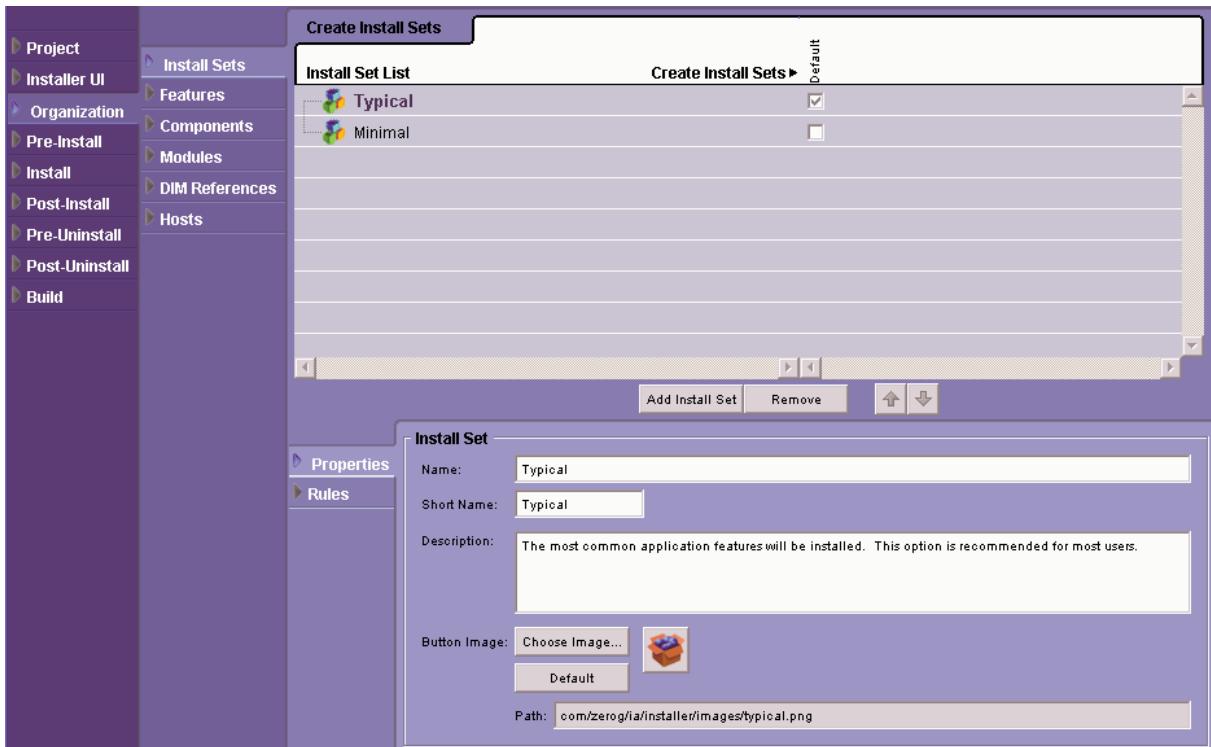
The **Organization** task is where you arrange install sets, features, components, merge modules, and DIMs. (Merge Modules and DIMs are discussed in Chapter 11.) Install sets and features support multiple levels of installation options for the end user of the installer. Components are the smallest element 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 you add a component in the **Organization > Components** subtask, files and actions can be assigned to that component after the files and actions are added in the **Install** task.

## Install Sets

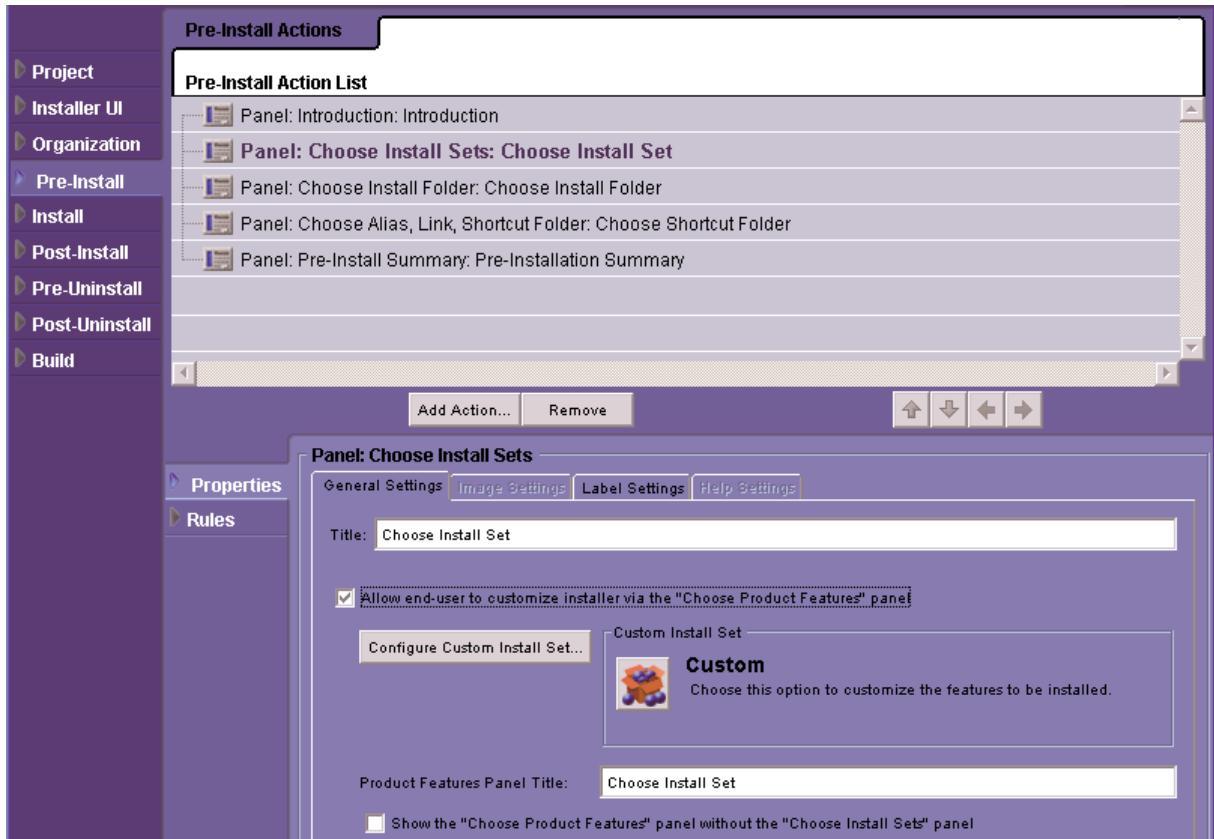
The **Organization > Install Sets** subtask is where you add, name, remove, and order install sets in your project. In the **Install Set List** area, you define which install set to use as the default by selecting the **Default** check box for the desired install set. The following figure shows the default install set settings: the project contains Typical and Minimal install sets.



**Figure 6-3:** Install Set List

To add, remove, or reorder install sets in your project, use the buttons under the **Install Set List** area. The customizer for each install set enables you to define each set's display name, description, and icon. These characteristics are displayed to the end user in the **Choose Install Set** panel, as pictured earlier in this chapter.

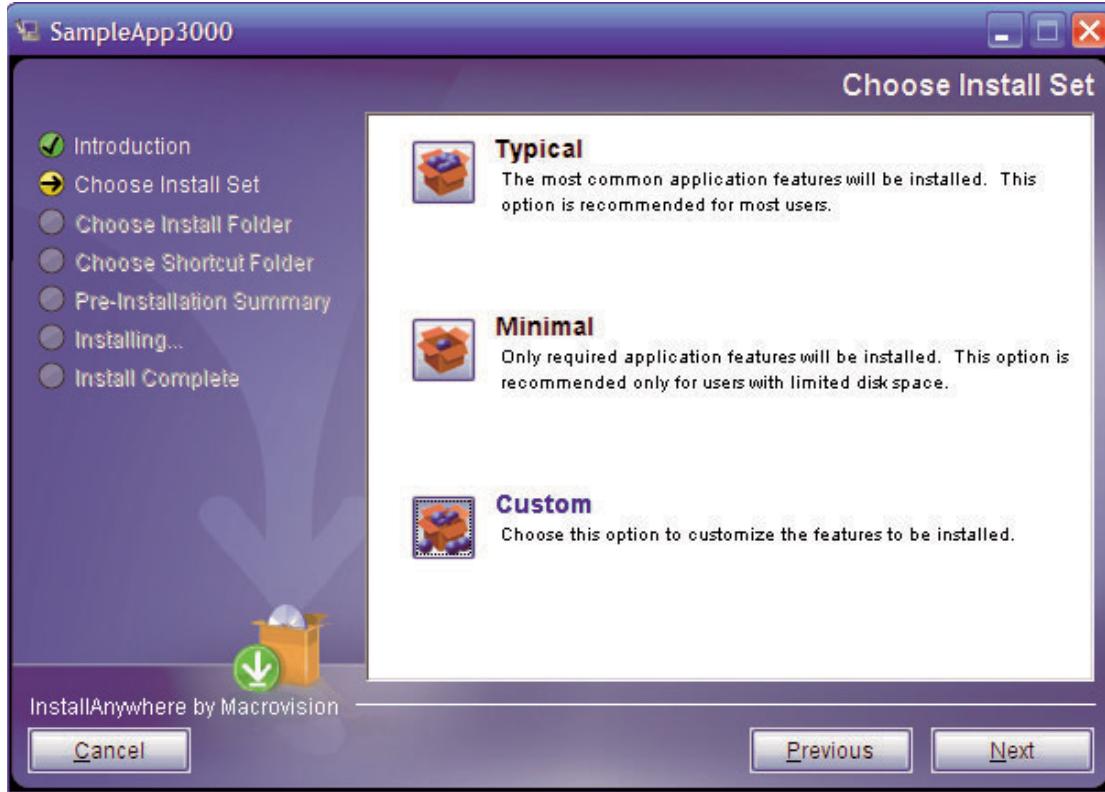
In addition to defining your own install sets, you can enable the Custom install set, using the customizer of the **Choose Install Sets** panel. After adding the **Choose Install Sets** panel to your **Pre-Install** task, select the check box labeled “Allow end-user to customize installer...”



**Figure 6-4:** Allow End-user to Customize Installer Menu

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

If you enable this option, at run time a user selecting the **Custom** install set can manually select which features to install. At run time, the **Choose Install Sets** panel containing the **Custom** install set appears as follows.



**Figure 6-5:** Custom Install Set

The feature-selection panel in which the user selects which individual features to install is pictured earlier in this chapter.

You can associate one or more rules with an install set, by selecting Rules in the customizer and adding the desired rules. If the rules on an install set evaluate to false, the install set will not be displayed.

## Features

The **Organization > Features** subtask is where you add, name, remove, or order features, as well as assign features to install sets. 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.

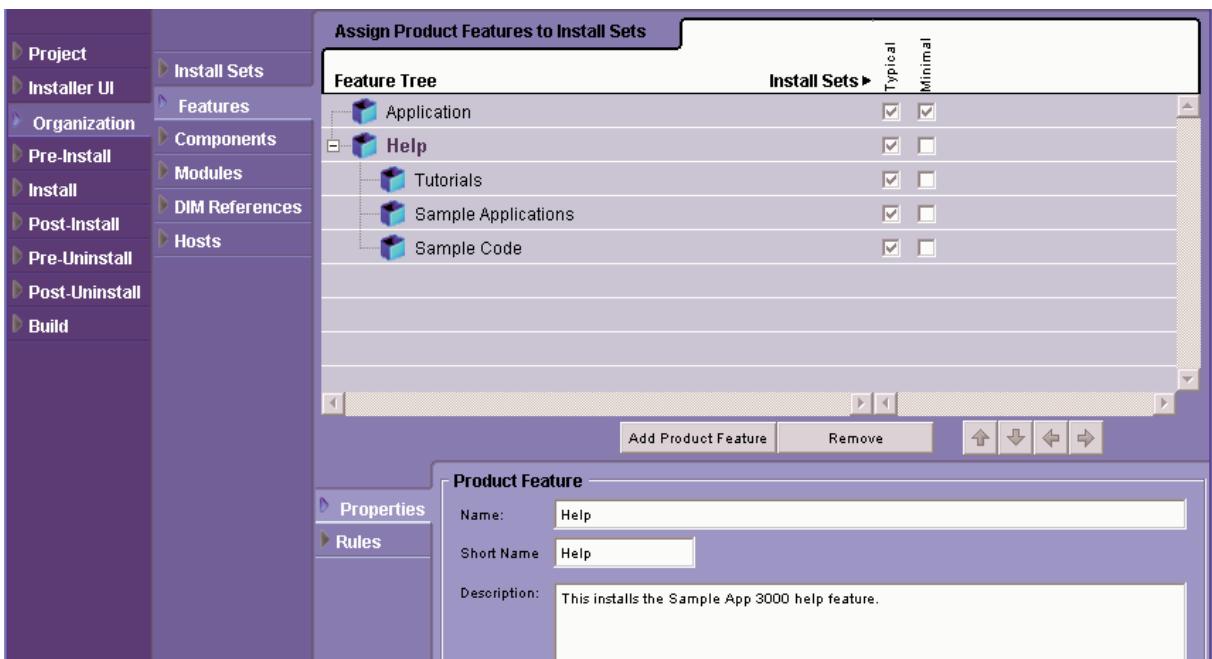


Figure 6-6: Feature Tree

Similar to install sets, you can associate rules with a feature by selecting **Rules** in the customizer and adding the desired rules. The rules for features are evaluated before the features are installed. If the rules on the feature evaluate to false, the feature will not be displayed.

## Components

In the **Components** task, you add and remove components, set component properties, and associate components with features. To specify the features associated with a component, select the check boxes corresponding to the desired features.

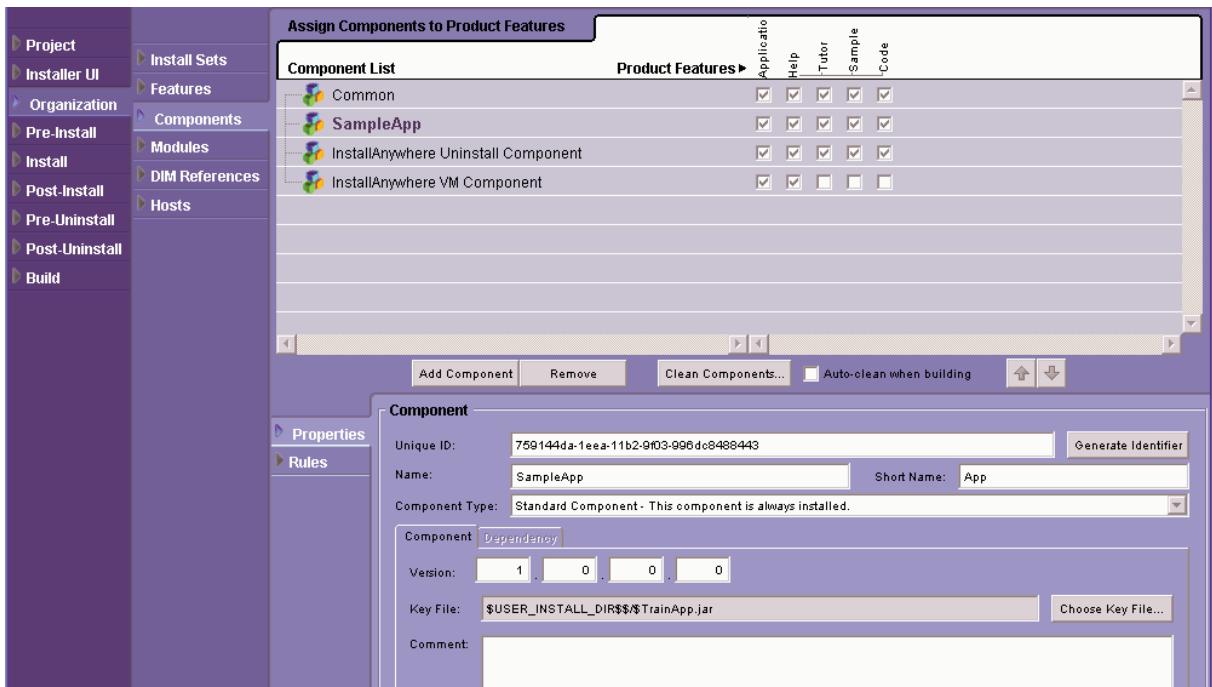


Figure 6-7: Component List

The customizer for each component is where you specify the component's unique ID and version information. This information is written to the InstallAnywhere product registry. The **Unique ID** value is a UUID that must be different from the UUID of any other component, except for a different version of the "same" component. The **Find Component in Registry** action enables you to detect a component based on its UUID and version.

A component's *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.

Rules can be associated with a component, again by selecting **Rules** in the customizer and adding the desired rules. The rules for a component are evaluated before the component is installed.

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

## Types of Components

The three types of components InstallAnywhere supports are the following:

- **Standard Component**—A component that will always be installed, regardless of previous installations or dependencies. The uninstaller for the product with which a standard component is installed will remove the standard components.
- **Shared Component**—A component that will be installed if it has not already been installed on the system. A shared component can be made available to other applications or installations. At uninstall, a shared component will be removed only if no other installed application references it. The uninstaller for the last product referencing the shared component will uninstall it.
- **Dependency**—Instead of installing the component, defining a dependent component will require that the specified component has already been installed on the system. Dependencies are not elements of your install per se, but rather are prerequisite requirements that the installer will enforce.

You define a component's type in the component's customizer, using the **Component Type** setting in the component's customizer, as illustrated in the following figure.

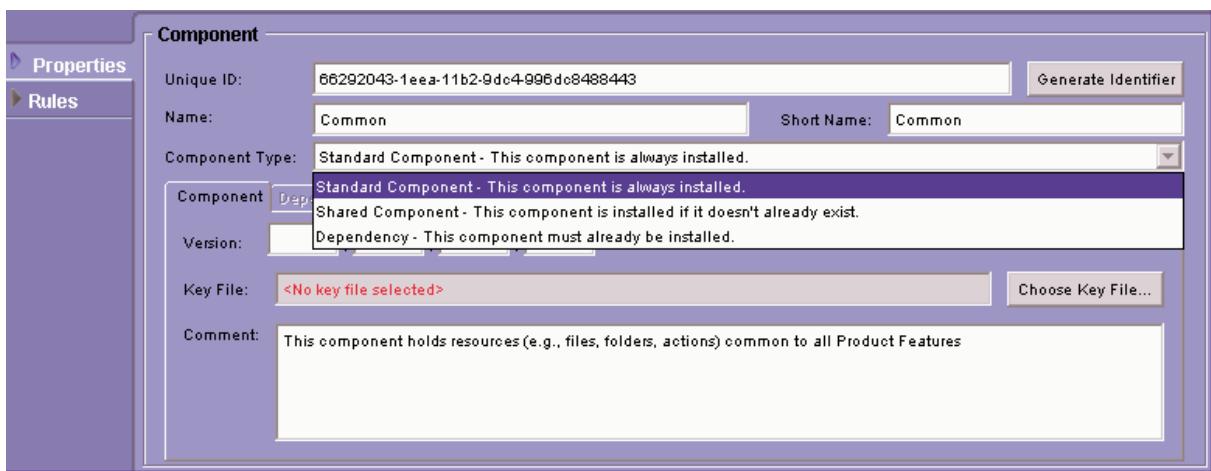


Figure 6-8: Component Type Customizer

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 prerequisite dependencies for your package, and can even enable multiple applications to share common components across a system.

## Shared Components

InstallAnywhere's shared component functionality 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. 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.

## Component Dependencies

Component Dependencies enable you to specify requirements for your installation that may, or may not be included in your package. For example, if your installation requires that 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.

In the customizer for a component of type **Dependency**, you specify the Unique ID of the desired component, along with any location or version restrictions.

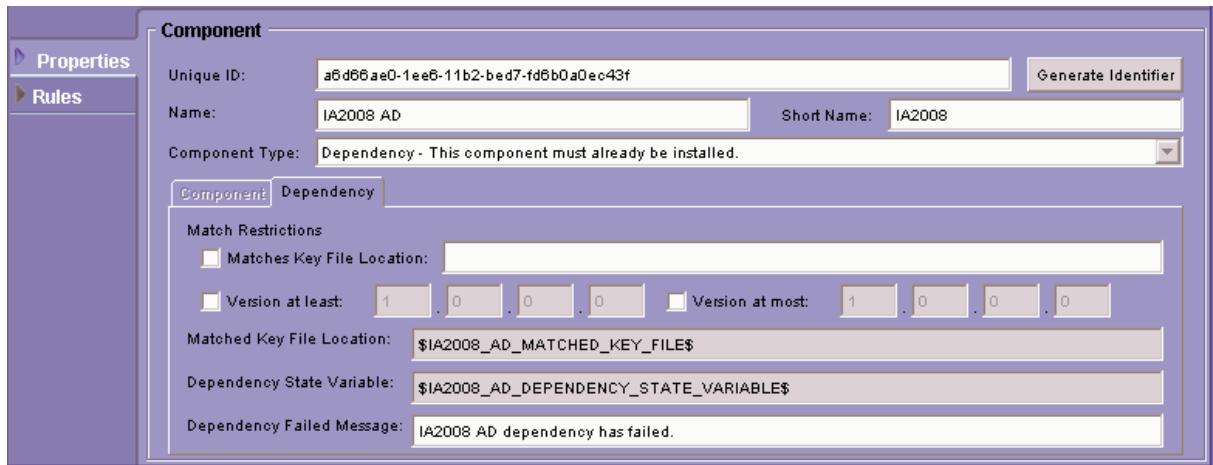


Figure 6-9: Component Dependency Tab

If the dependency is satisfied, the property listed in the customizer (`$component_MATCHED_KEY_FILE$`) will contain the location of the component's key file. If the dependency is not met, the property `$comp_DEPENDENCY_STATE_VARIABLE$` contains the string defined in the **Dependency Failed Message** setting.

If the installer needs a component that should already be installed on the target system, another option is to use the **Find Component in Registry** action to locate that component. This action searches for the component by using its UUID. The installer can compare versions and locate the greatest version found. The installer can also search for the key file. The component's location can then be used as an installation location by the installer.

# Organizing Your Features and Components

Components are the lowest level of organization in an installer. Each product must have at least one component, and 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 enable you to plan for future releases, suite installers, and other uses of software elements in your deployment plan.

InstallAnywhere automatically creates 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 component-based software, you should manually manage the creation of components.

## Best Practices for Components

When using components, first determine and organize which components to add. There are a few recommended best-practices 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, it is recommended 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 in 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 dependent 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 display only approximately four install sets 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.

# Adding Files to Your Project

This section describes the different ways in which InstallAnywhere supports adding files to your project.

## Adding Individual Files

You add individual files to your project using the **Install** task. The **Visual Tree** in the **Install** task displays your project's files arranged by destination folder. Instead of displaying hard-coded destinations, destinations are represented by Magic Folders such as `$USER_INSTALL_DIR$`, which represents the main product installation directory on a target system. Magic folders are further described in the following section.

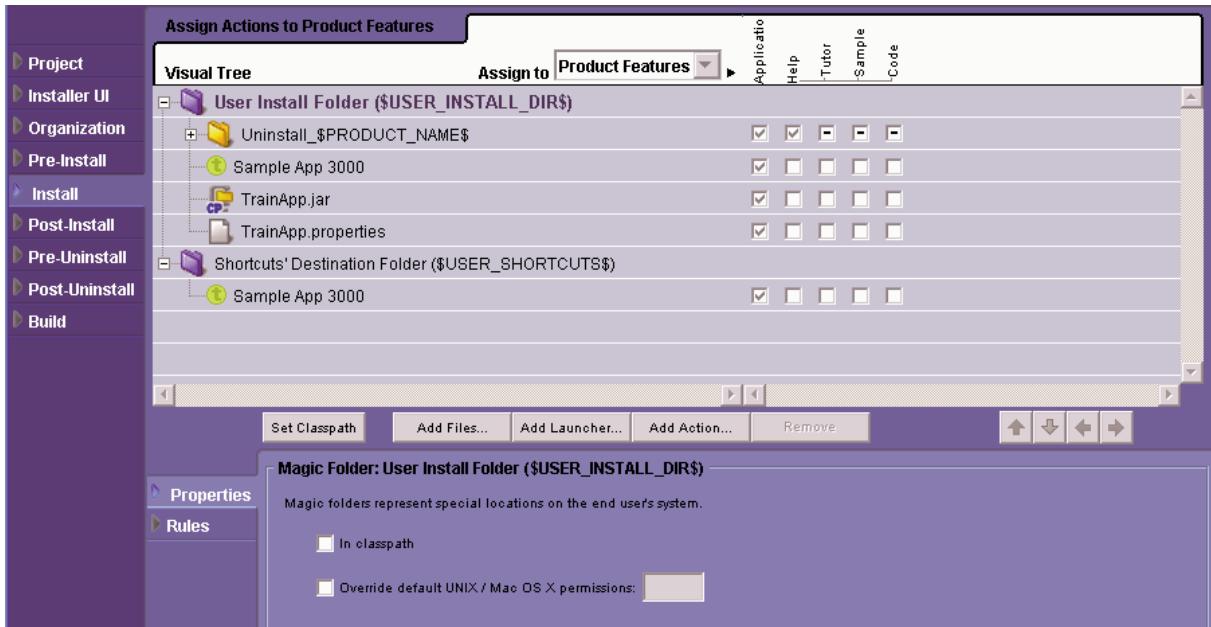
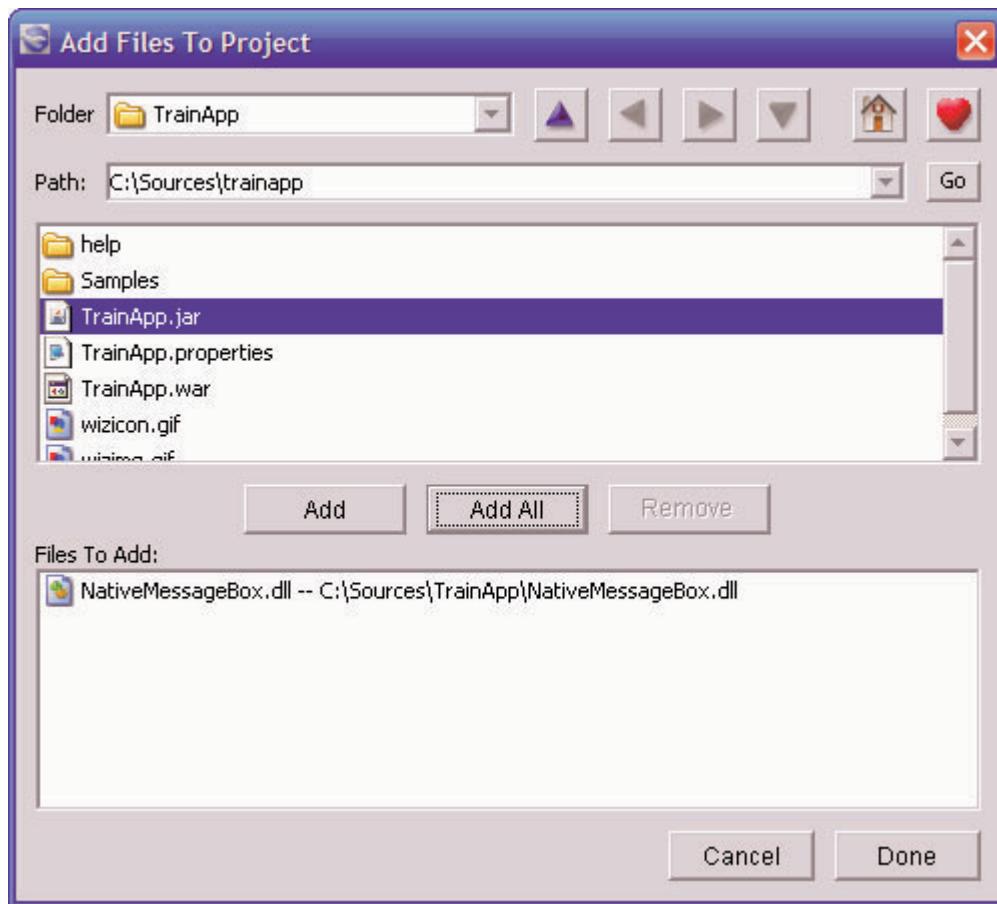


Figure 6-10: Magic Folders Menu

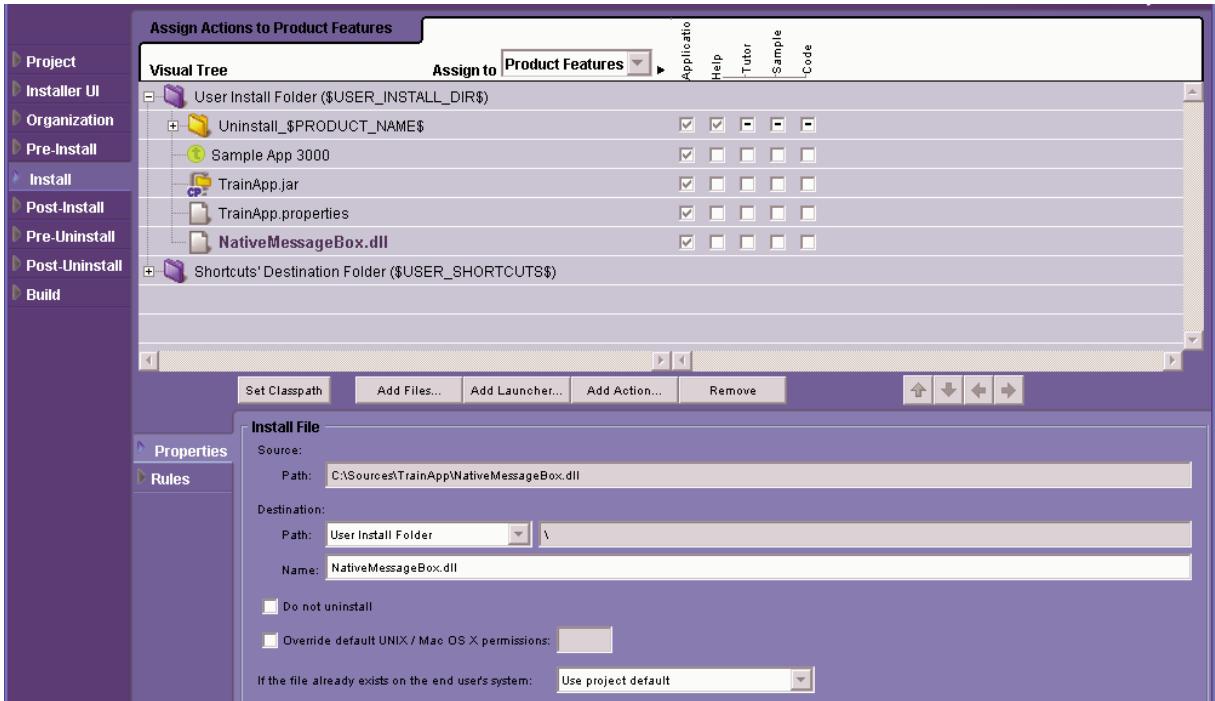
To control whether to associate the files you add to components or to associate them with features (and therefore have InstallAnywhere define components for you), select the desired setting from the **Assign to** list at the top of the **Install** task.

To add a file, click **Add Files** under the **Visual Tree** area, and use the **Add Files to Project** panel to select one or more files.



**Figure 6-11:** Adding Files to a Project

After you click **Done**, the file appears in the **Visual Tree** under the specified directory.



**Figure 6-12:** Product Features Visual Tree

In the file's customizer, you can modify the destination path by selecting the new target directory from the **Path** list. The **Path** list contains descriptions all of the Magic Folders, which are described in the following section. You can also modify the file name to use after installation, whether the file should be uninstalled, and whether to override the default permissions on a Unix-style target system. In addition, you can optionally specify custom file-overwrite behavior.

## Magic Folders

Magic Folders represent a specific destination 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) on a Windows system.

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 using pre-defined Magic Folders, InstallAnywhere 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.

For a list of predefined Magic Folders, refer to the InstallAnywhere help topic "Magic Folders and Variables".

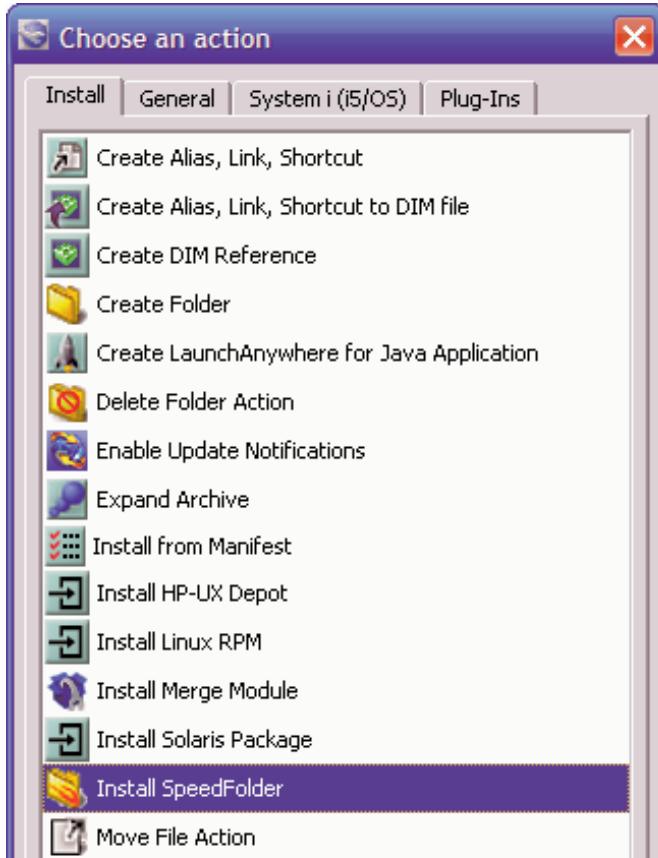
## Adding Directories with SpeedFolders

Using SpeedFolders will greatly increase the speed and memory efficiency of your installer. Similar to folders that you add 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.

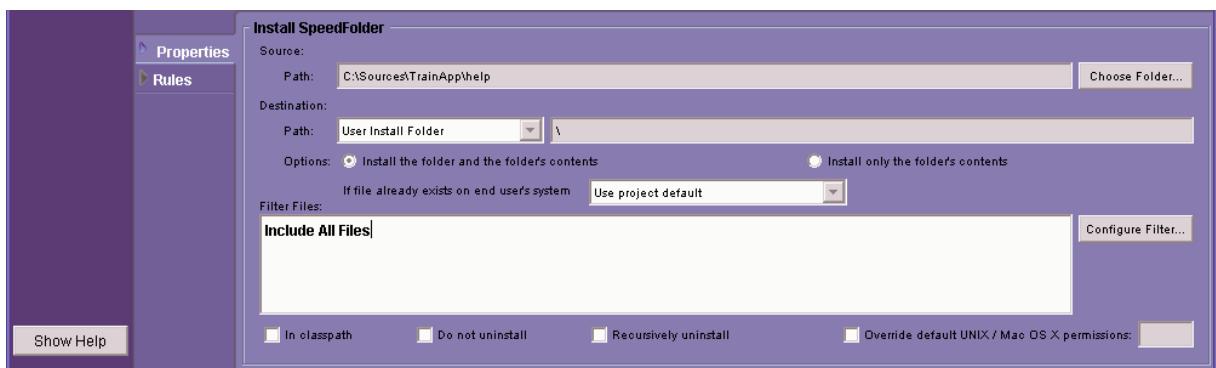
SpeedFolders have filters that allow inclusion or exclusion of files that meet particular naming criteria. Individual files or folders in a SpeedFolder cannot be assigned to different components, nor can SpeedFolders be converted into traditional directories, or traditional directories into SpeedFolders.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

To add a SpeedFolder to the **Install** task, click **Add Action** and select **Install SpeedFolder**.



**Figure 6-13:** Installing a SpeedFolder

After you click **Add**, the customizer for the **Install SpeedFolder** action is where you specify the source and destination locations, along with overwrite and uninstallation behavior.



**Figure 6-14:** Install SpeedFolder Properties

By clicking **Configure Filter**, you can specify files or folders to include and exclude. By default, all of the files in the directory tree will be included; selecting the **Preview** or **TreeView** tab on the **Filter Files** dialog box displays the files currently included in the SpeedFolder.

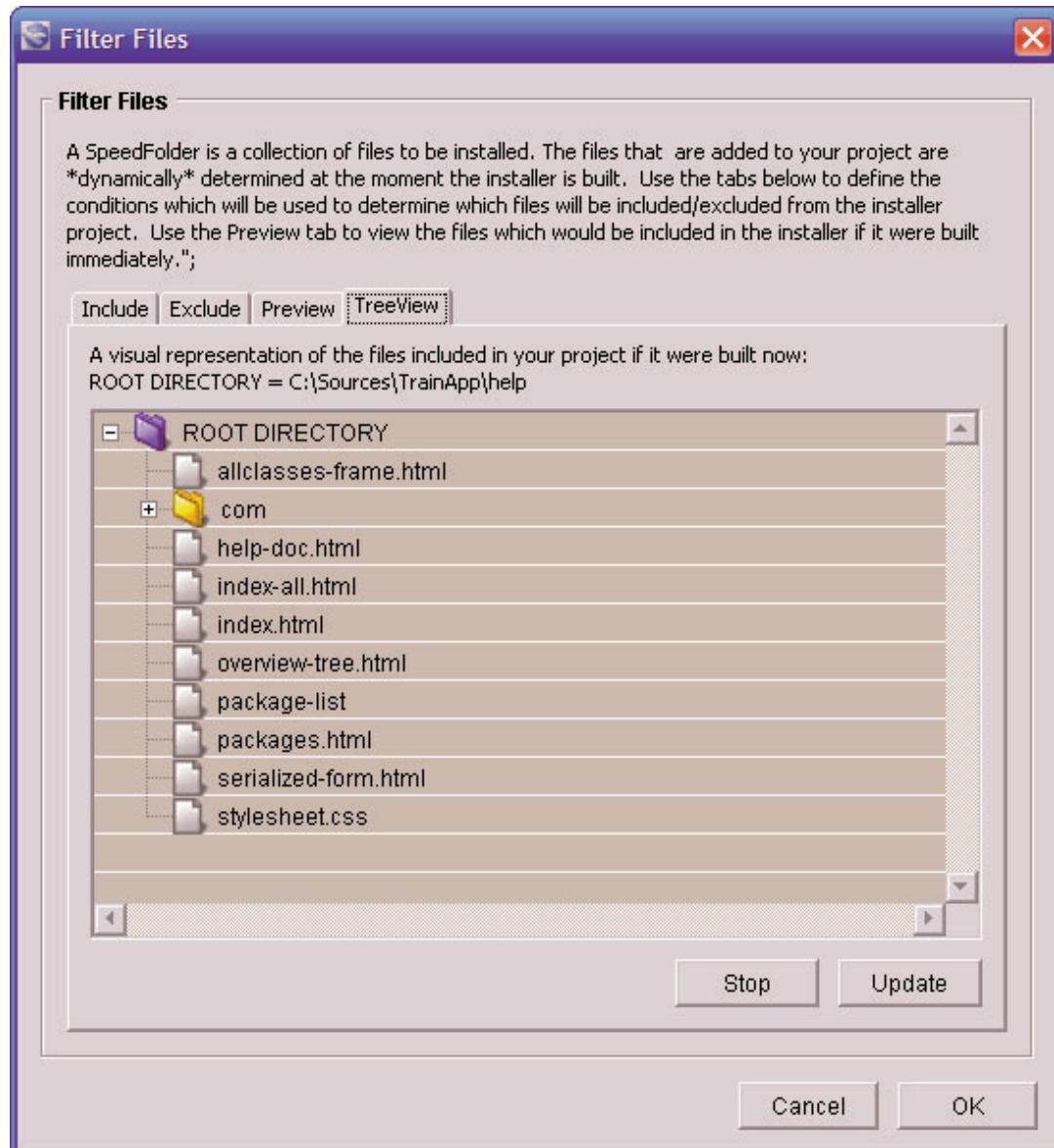
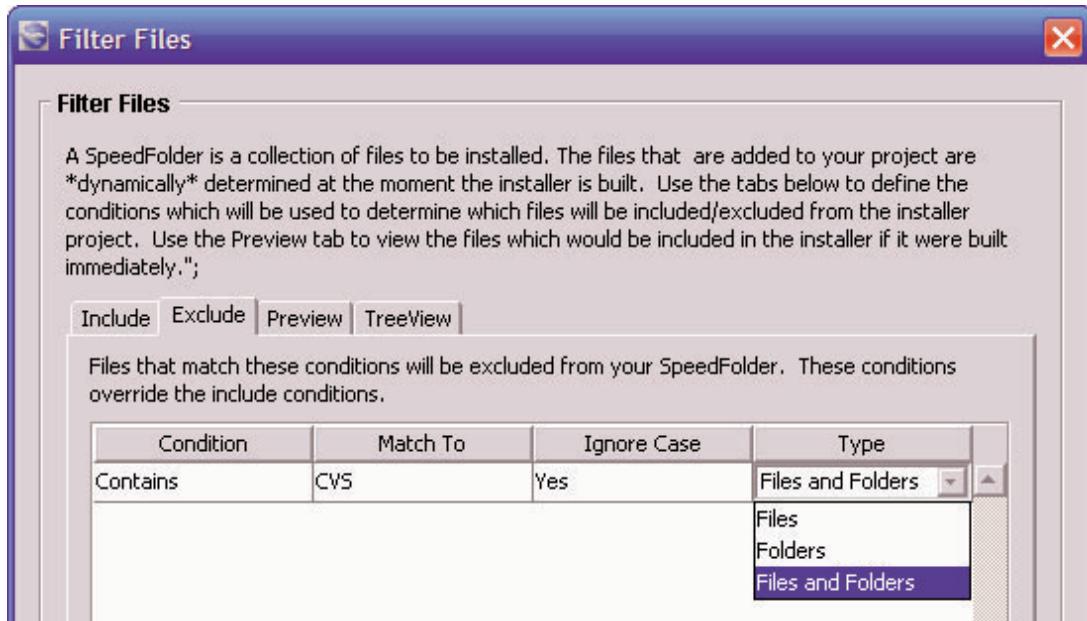


Figure 6-15: Filter Files Tree View

In the **Include** and **Exclude** tabs, you specify files and directories to include and exclude. For example, the following figure shows how to exclude the CVS directories left by the Concurrent Versions System.



**Figure 6-16:** Filter Files Include and Exclude Tabs

At build time, the current set of files in the source directory that pass the filter definitions will be included in your installation media and installed on the target system.

## Specifying Source Files with 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
```

## 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

# Introduction to Advanced Actions and Panel Actions

---

This chapter covers the use of some of InstallAnywhere's more advanced and more useful installer *actions*. These actions represent operations performed by the installer or uninstaller.

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 and 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.

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.

Requests for user input that appear inside the graphical installer wizard are *panel actions*. 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.

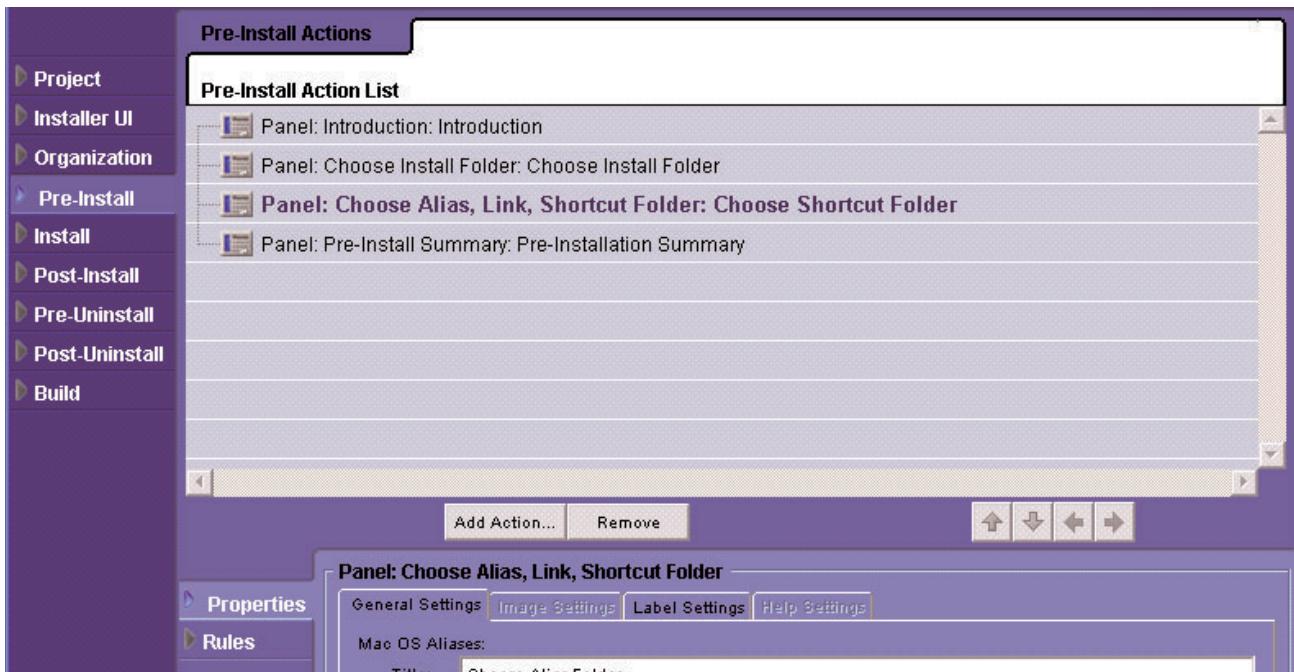
*Console actions* are displays of information and requests for user input used during a command-line installation. Custom code can also be integrated with the InstallAnywhere Designer and will appear as a *plug-in action*.

*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.

# Adding an Action

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 by using 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.

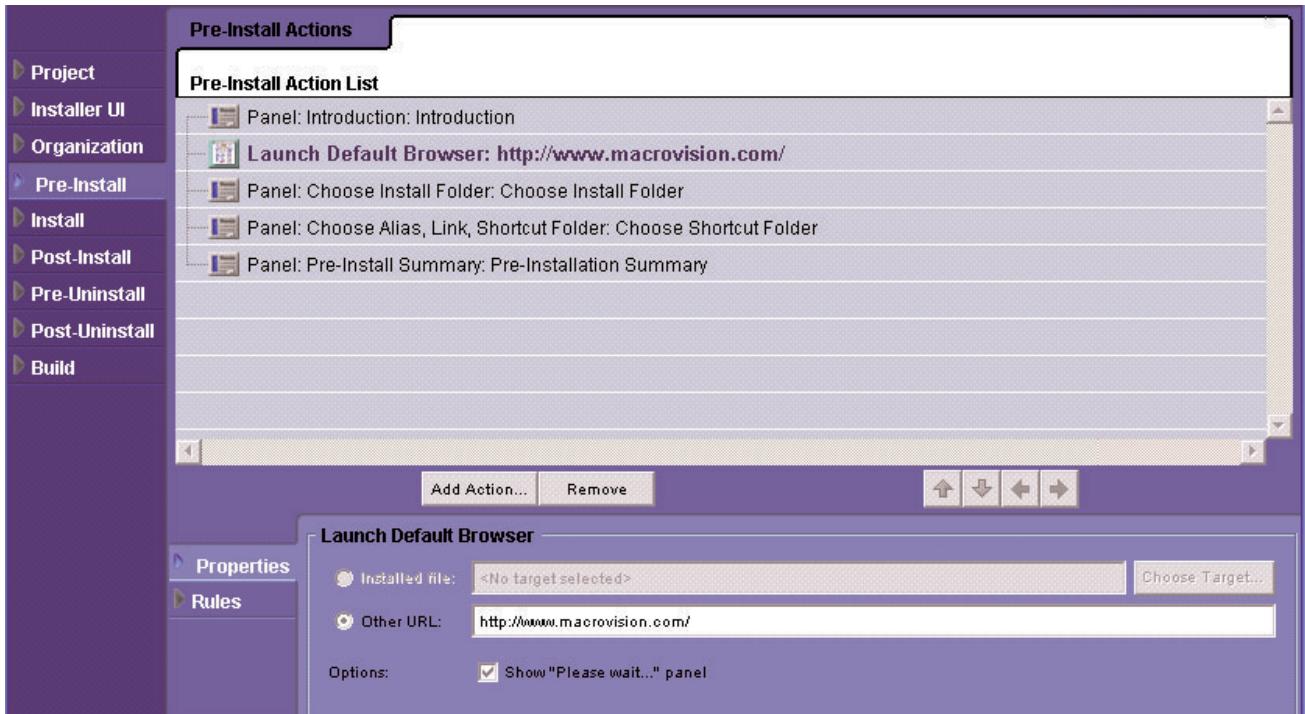


**Figure 7-1:** Pre-Install Action List

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. You can modify the action in the customizer. The **Properties** area enables you to add file locations and variables to the action, as well as detail how you 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 figure.



**Figure 7-2:** Pre-Install Action List

The **Rules** area enables you 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**, indicating 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. In the figure below, the first two actions do not have rules associated with them, with the second two actions have rules added.

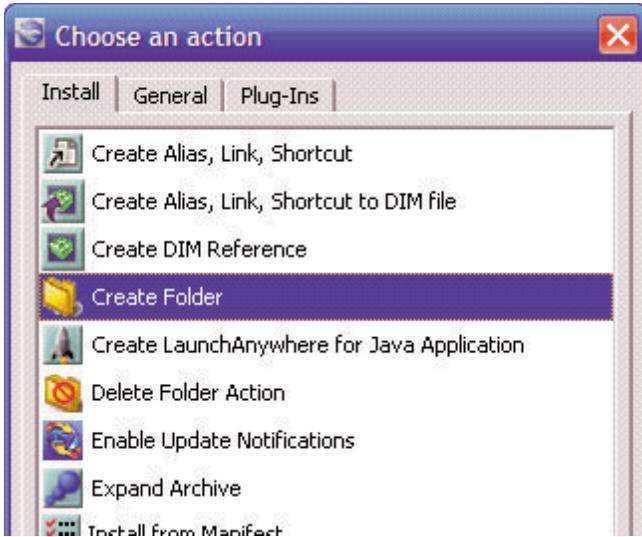


**Figure 7-3:** Actions with Associated Rules

## 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.



**Figure 7-4:** Choosing an Action

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

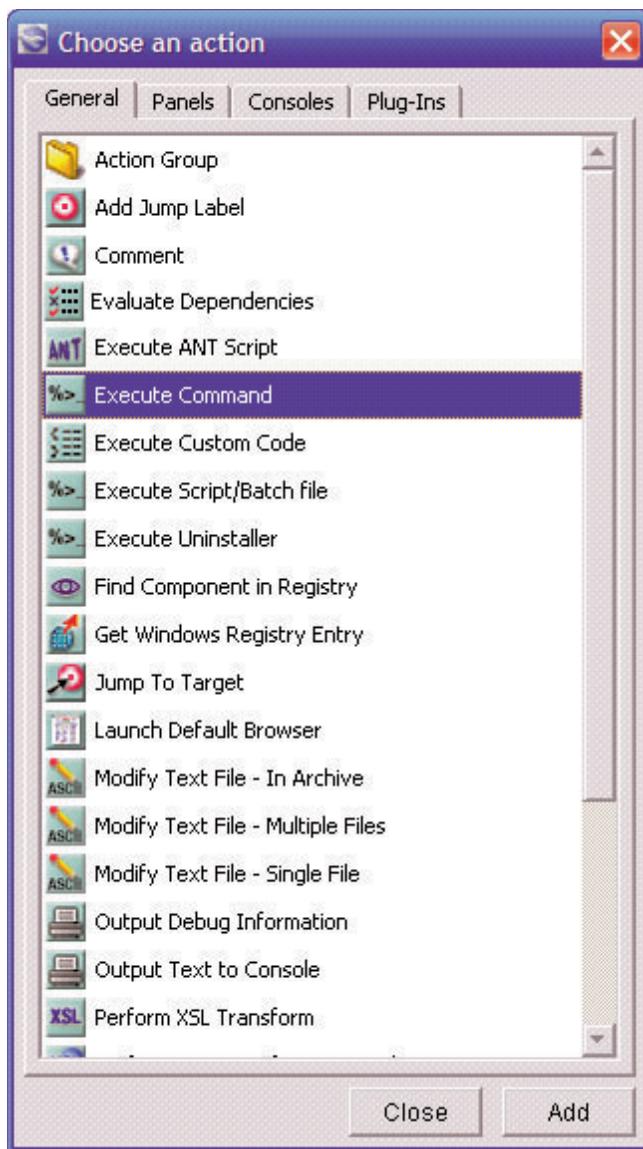


**Note:** The **Plug-ins** tab will appear when a custom code plug-in has been added.

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. You will examine some common actions in this section.

## General Actions

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



**Figure 7-5:** General Actions

## 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.



**Figure 7-6:** Action Groups

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 generally 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. It is common to require that information is input at this stage, and automate later configurations based on that input. This enables your user to make one set 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, refer to the InstallAnywhere help library.

Some commonly used actions are the following:

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

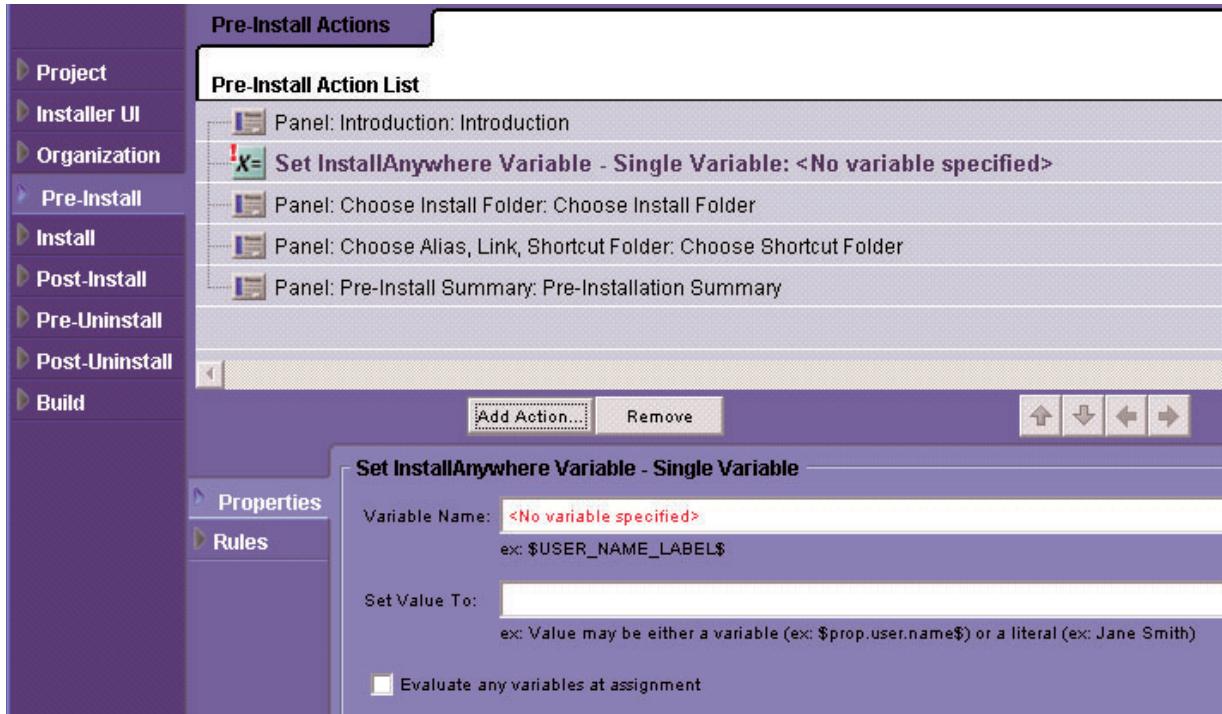
## Set InstallAnywhere Variable Action

Most installation programs must set the values of InstallAnywhere variables. 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 **Set InstallAnywhere Variable—Single Variable** action (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.

For a list of built-in variables, refer to the InstallAnywhere help topic "Standard InstallAnywhere Variables".

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** and click **Add**. Once the action is added, click **Close**. The new action should appear as follows.



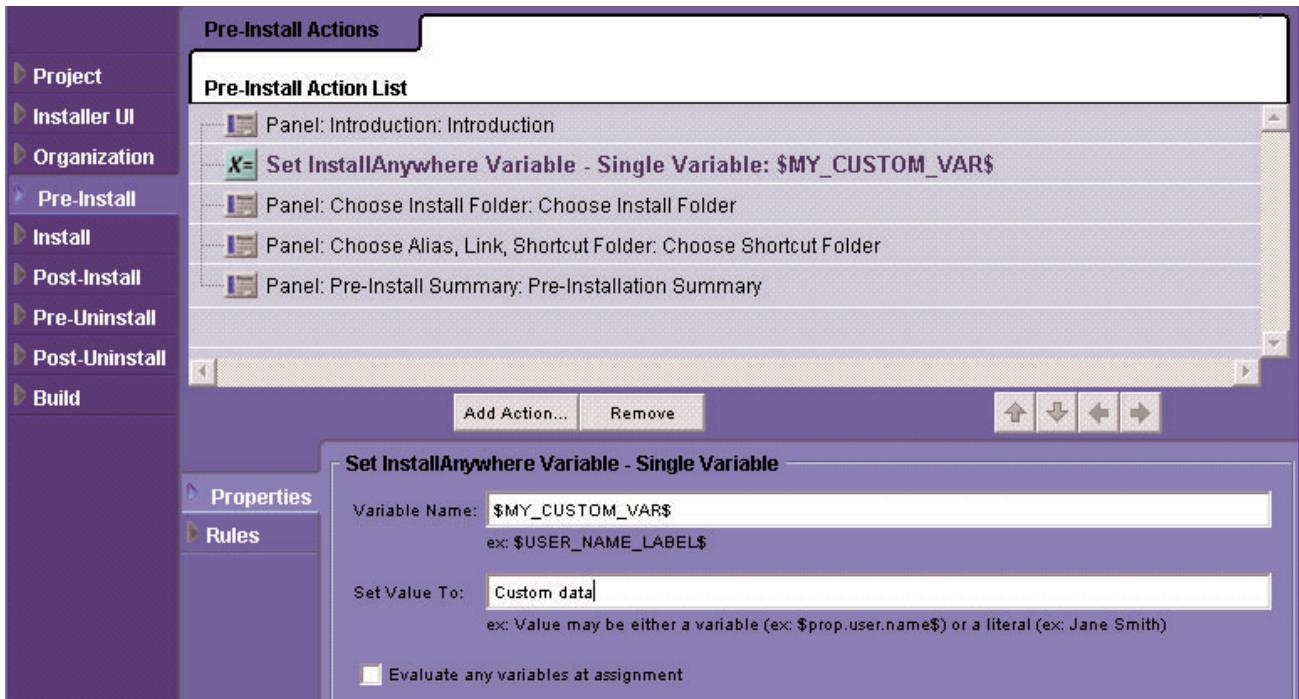
**Figure 7-7:** Pre-install Actions List

If necessary, you can move the action up and down in the list by clicking the arrow buttons.



**Note:** 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.



**Figure 7-8:** Adding a Variable Name

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 and select **Panel: Display Message**. Click **Add** to include the action and **Close** to exit. The new action should appear as follows.

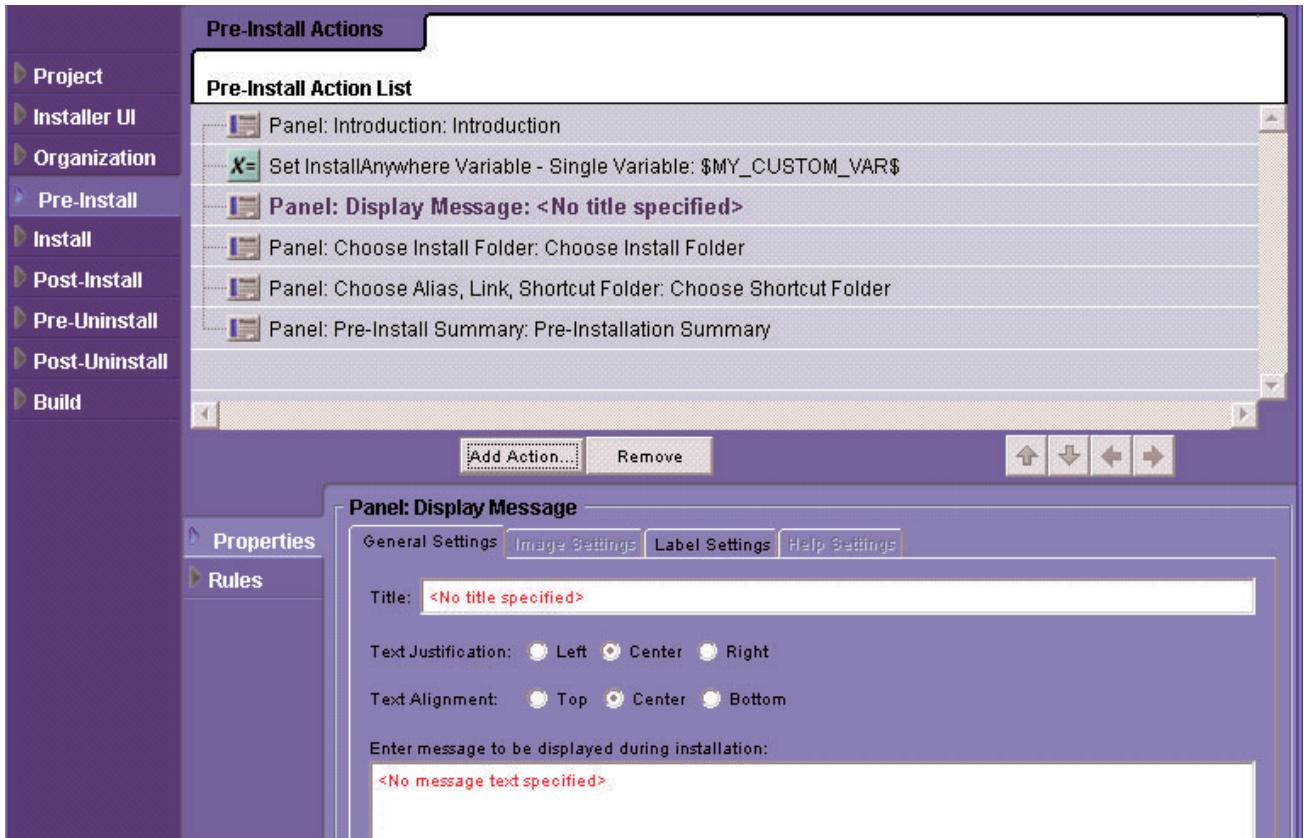


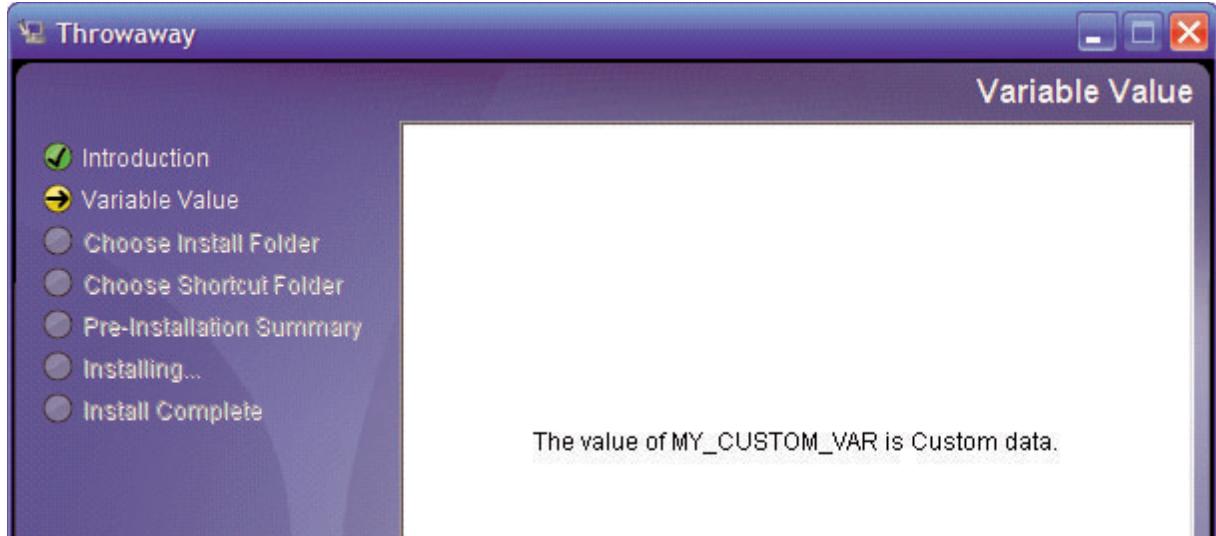
Figure 7-9: Customizing Display Messages

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\$**.

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



**Figure 7-10:** Variable Value Display

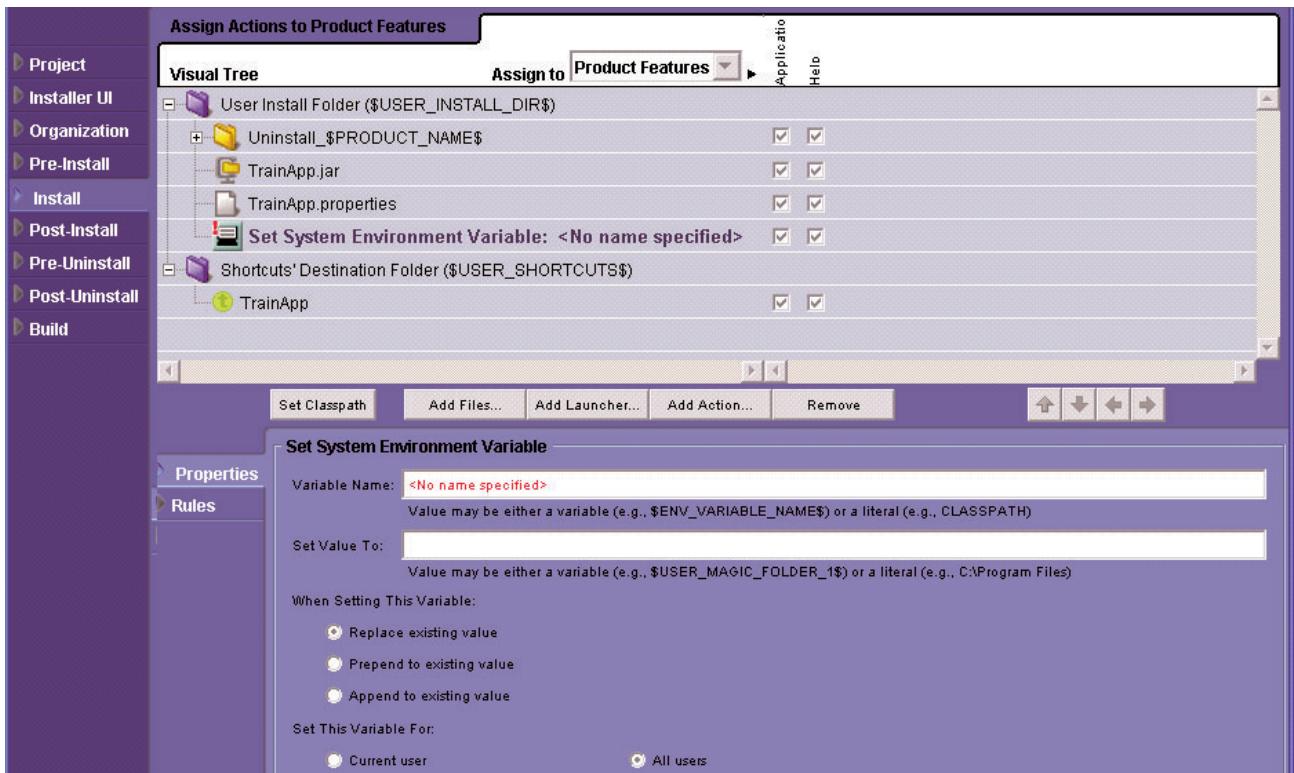
You can use the **Label Settings** tab of the panel's customizer to specify the text that appears in the left-hand list of steps on the panel at run time.

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, and click **Add Action**. Select **Set System Environment Variable** and click **Add**. Once the action is added, click **Close**. The empty action appears similar to the following.



**Figure 7-11:** Setting System Environment Variables

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.

# Display HTML Panel

In addition to displaying a simple message using the **Display Message** panel, you can display HTML content using the **Display HTML Panel** action. Using this panel, you can display HTML pages and images from a local archive or from an external URL.

For example, you add the panel to the **Pre-Install** task the same way you add other panels, using the **Add Action** button, if necessary using the up-arrow and down-arrow buttons to schedule the action relative to other actions.



Figure 7-12: The Display HTML Panel in the Pre-Install Task

The customizer for the **Display HTML** panel enables you to specify the settings common to all panel actions, such as the panel title and the label settings.

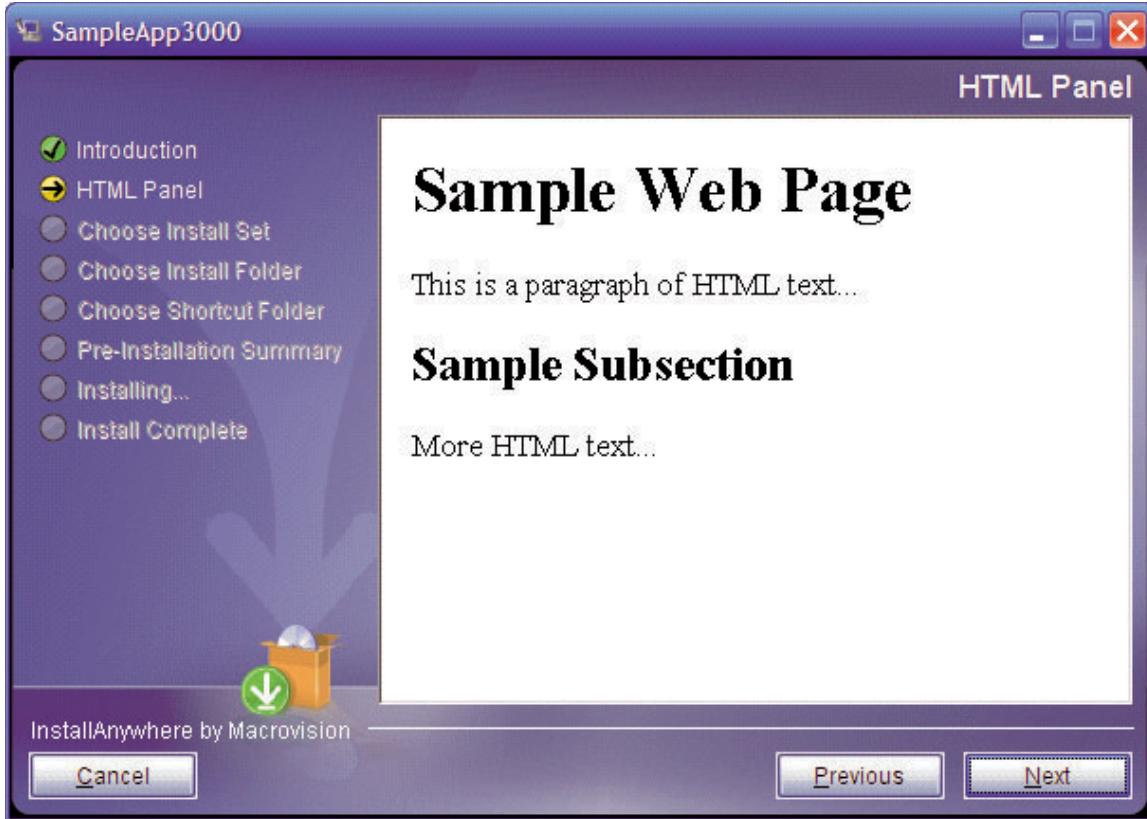


Figure 7-13: Display HTML Properties

In addition, you specify the source of the HTML pages for the **Display HTML** panel. If you select **Path**, you will be prompted to browse for a .zip or .jar file containing the HTML pages, images, and style sheets you want to display. After selecting the archive file, enter the name of the file within the archive to display in the **Initial Page** setting. The HTML content you display can include hyperlinks, either within the archive you selected or existing externally.

For an example archive containing HTML content, view the contents of the `CustomCode/Samples/HTMLPanelSample` subdirectory of your InstallAnywhere installation directory.

If you select **Existing URL**, simply enter the external URL of the HTML page to display.



**Figure 7-14:** Example of a Display HTML Panel

Your HTML content can also include forms, and the user's input into the form can be passed to InstallAnywhere variables. To use this functionality, select the **Read Form Elements as InstallAnywhere Variables** check box in the **Display HTML** customizer.

In the HTML content, include a `<form>` element with `<input>` fields, and at run time the user input will be included in InstallAnywhere variables. For example, the following HTML excerpt includes an input field called `username`.

```
<form name="register">
<p>Please enter your user name:<br>
<input type="text" size="50" name="username"
       value="(your name here)" class="textField">
</p>
</form>
```

At run time, the **Display HTML** panel appears similar to the following:



**Figure 7-15:** HTML Panel Preview

To reference the data entered by the user, you can use the expression `$fieldname$`, where `fieldname` is the value of the `name` attribute in the HTML input field. In the previous example, the data would be stored is accessible using the expression `$username$`.

## 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, if needed).

Previous sections covered 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 **Installer UI > 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.

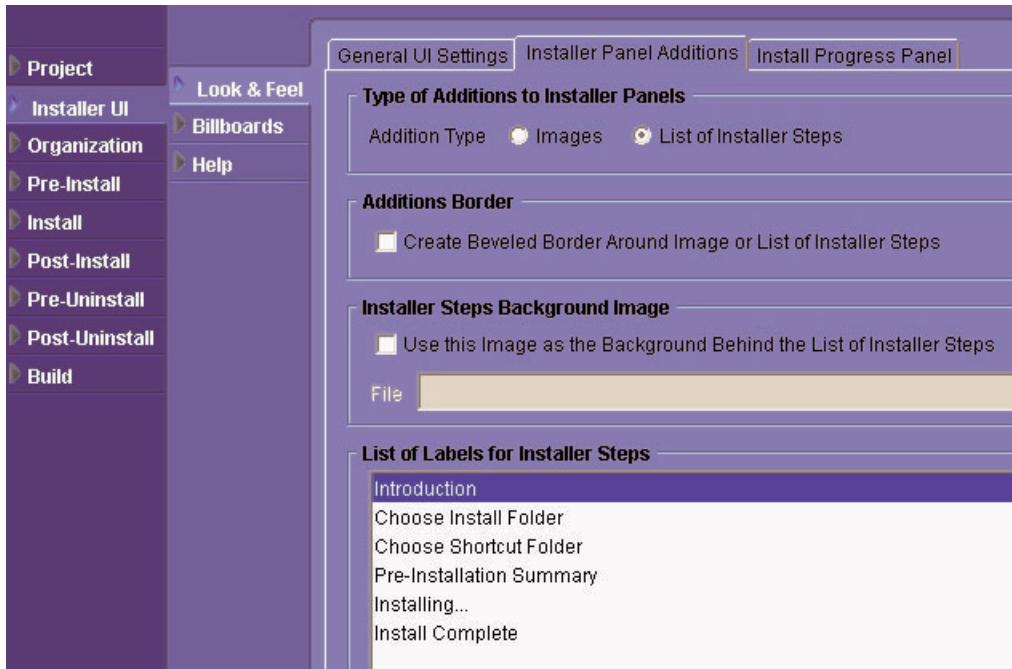


Figure 7-16: Installer Panel Additions Tab

## Panel Labels

The **Panel Labels** 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.

## 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 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 enables you to jump to the target specified. Be sure to add conditional rules to it or your end user will constantly be jumping.

## 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.



**Figure 7-17:** Adding Hosts

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

For additional information on application server, database server, and operating system hosts, refer to the InstallAnywhere help topics.

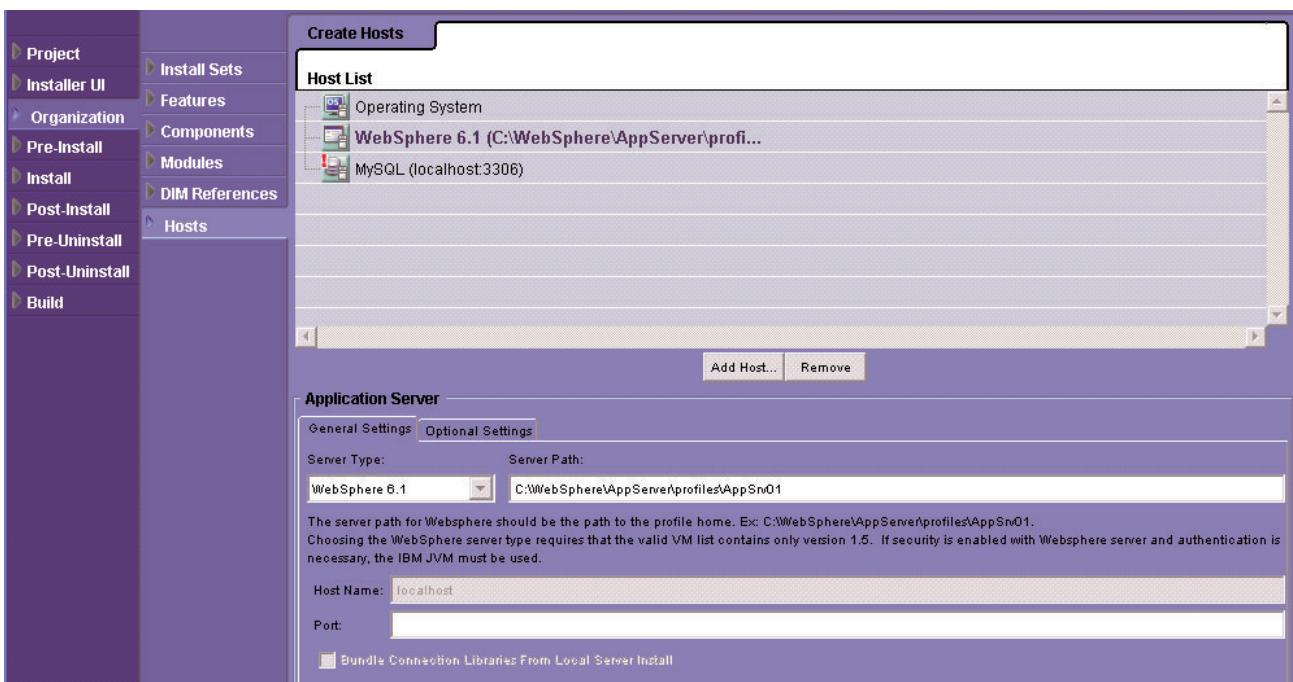


**Figure 7-18:** Choosing a New Host

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, etc.), and different server types can have different general and optional settings.

In addition, 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.

Refer to the InstallAnywhere help library for detailed information about the various host properties.

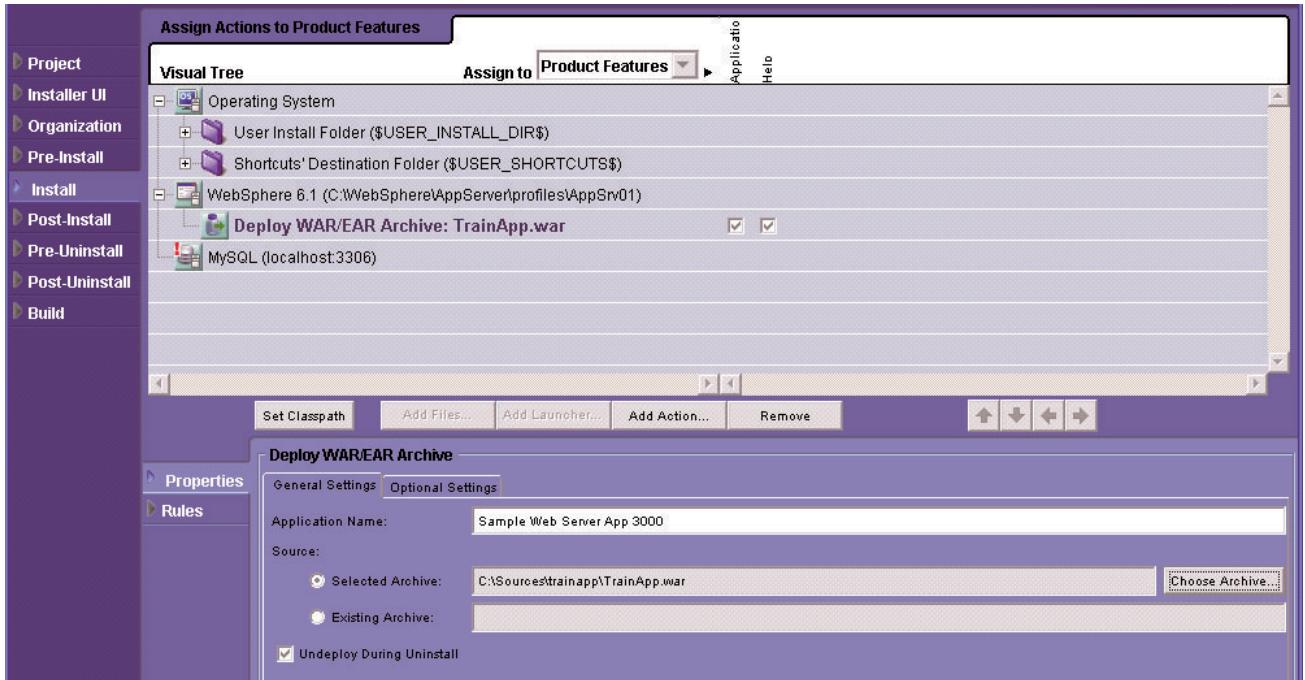


**Figure 7-19:** Configuring Hosts

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, refer to the 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 Server** host, or add a **Run SQL Script** action to a **Database Server** host.

To add these actions, open the **Install** task of your project. The new hosts appear in the **Visual Tree**, along with the standard **Operating System** host.



**Figure 7-20:** Deploying EAR/WAR Archives

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; 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, refer to the InstallAnywhere help library.

# Common Properties

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

**Table 7-1:** Common Properties

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



**Figure 7-21:** Selecting Type of Additions to Installer Panel

## Image Settings

Use panel image settings to choose a specific image to display on the chosen panel. You 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 allows you 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.



Figure 7-22: Label Settings Tab on the Post-Install Task



**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. To format the help text, use the HTML formatting tags—but note that HTML tags cannot be applied to the title. For example:

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

causes InstallAnywhere to display **MyHelp Information**.

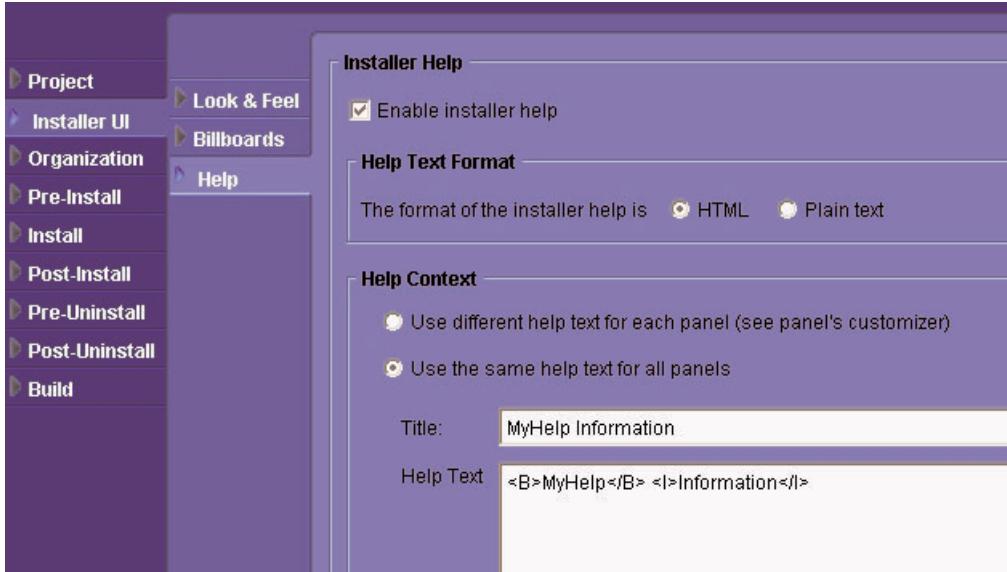


Figure 7-23: HTML Tags in Installer Help

When you click the **Preview** button, help is displayed similar to the following figure.

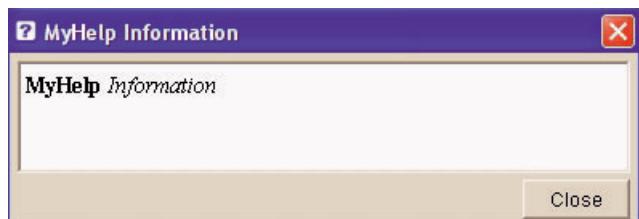


Figure 7-24: Help Output Preview

You may either set a single help message, which you 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.

## LaunchAnywhere Executable

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 specifications provided. 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.

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.

For a list of variables you can use to configure a LaunchAnywhere executable, refer to the InstallAnywhere help topic "LAX Properties".

## 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



# 8

## Applying Basic and Intermediate Development Concepts

---

This chapter contains a single, unstructured exercise using what is covered in the previous chapters. Specifically, you will:

- Build an installer with specific features, actions, and panels
- Use the debugging features at various stages of development

After completion of the installer project, you will test and debug the installer utilizing InstallAnywhere built-in debugging features. While debugging is typically a post-production task, it can be done at any point in the development process. Keep in mind that post-development suggestions are useful when debugging a customer problem or other post-development issues.

# Building the Installer

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 according to the following guidelines:

**Table 8-1:** Installer Guidelines

Name	Value
<b>Magic Folders</b>	Your installer must include at least one magic folder, other than the core USER_INSTALL_DIR and SHORTCUTS.
<b>LaunchAnywhere</b>	Your installer must contain at least one LaunchAnywhere Launcher.
<b>InstallAnywhere Variables</b>	Your installer must display an understanding and reasonable management of InstallAnywhere variables.
<b>InstallAnywhere Rules</b>	You must implement rules that control installer behavior and install-path options.
<b>Implement the following actions and panels</b>	<ul style="list-style-type: none"><li>• Panel: Choose Alias, Link, or Shortcut</li><li>• Panel: Choose File or Choose Folder</li><li>• Panel: Choose Install Folder</li><li>• Panel: Display Message</li><li>• Panel: Get Password</li><li>• Panel: Install Failed</li><li>• Panel: Install Success</li><li>• Panel: Install Summary</li><li>• Panel: Introduction</li><li>• Panel: Show License Agreement</li></ul>
<b>Implement the following InstallAnywhere actions</b>	<ul style="list-style-type: none"><li>• Add Comment</li><li>• Create LaunchAnywhere for Java Application</li><li>• Create Alias, Link, or Shortcut</li><li>• Install File</li><li>• Install Folder</li><li>• Install Uninstaller</li><li>• Output Text to Console</li><li>• Create Alias, Link, or Shortcut</li><li>• Execute Target File</li><li>• Get Windows Registry</li><li>• Set InstallAnywhere Variable</li><li>• Set Windows Registry Entry - Single</li><li>• Show Message Dialog</li></ul>

2. When you have completed your installer project, test your installer. Does it meet your expectations?

For additional information on testing, refer to the debugging sections in this chapter.

# Debugging InstallAnywhere Installers

Using the installer you've just constructed, you will 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; certain debugging mechanisms are more effective during installer development, and others work well later in the process, such as when end user has a problem with the installer.

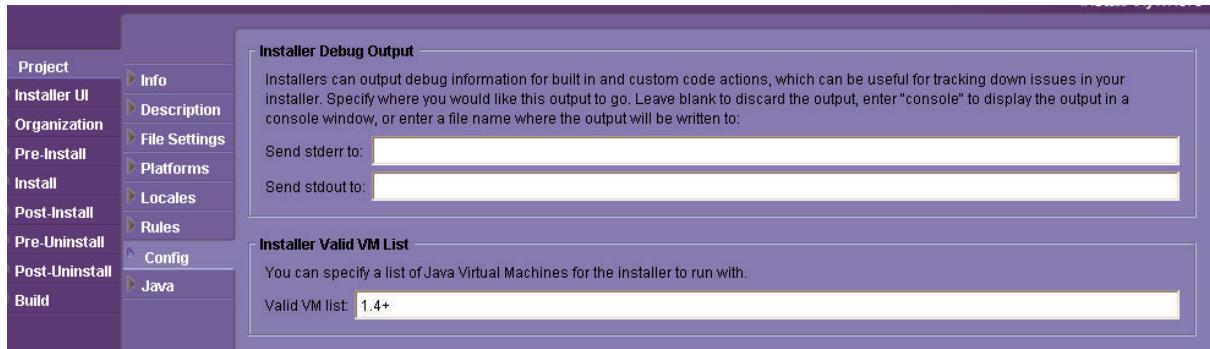
Most InstallAnywhere installers utilize Macrovision's LaunchAnywhere technology. Along with many convenient features for end users (double-clickable launchers, native-like user experience), LaunchAnywhere launchers provide many built-in debugging features.

## During Installer Development

InstallAnywhere provides a project-specific debugging feature that enables 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**.



**Figure 8-1:** Installer Debug Output Menu

Advanced Designer has two fields within the **Installer Debug Output** section. In these fields, you can enter a path to the text file that the installer places 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 system-specific paths. This will enable you use one entry for multiple platforms.

2. Set the output files:

Send stderr to: /tmp/outputfile.txt  
Send stdout to: /tmp/outputfile.txt

These settings direct the output to `/tmp/outputfile.txt` on a Unix or derivative system, and to `C:\tmp\outputfile.txt` on a Windows system. You will need to verify 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.txt`) will contain most, if not all, the information needed to debug InstallAnywhere installations.



**Note:** Because these files will not be uninstalled, it is 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.

## 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 until a console window appears. Before exiting the installer, copy the console output to a text file for later review.

On some Windows systems, 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.

1. Launch the installer and enable it to extract the necessary files.
2. When the **Preparing to Install** window appears, go to the Windows `temp` directory and search for a folder which starts with an “I” followed by many digits (for example, “I1063988642”).
3. Ensure it is the most recent directory by sorting the directories by their “modified” date. Open the directory, and search for a file named `sea_loc`.
4. Delete the `sea_loc` file.
5. Return to the installer and click **OK**.
6. At the first opportunity, cancel the installation.
7. Return to the directory inside the `temp` directory, where the file `sea_loc` was deleted. Search for the directory named `windows`. In this folder, there should be an `.exe` file (most likely `install.exe`). You should also see another file with the same name, but it will have a `.lax` extension.
8. 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
```

9. After these changes have been made, save the file and launch the `.exe`.

When the installation is complete, it will produce the `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 a Unix command line, you need to enter one of the following (based on which shell) at the command line prior to executing the installer:

```
export LAX_DEBUG=true
setenv LAX_DEBUG true
LAX_DEBUG=true
```

or

```
set LAX_DEBUG
```

Other options may be available for specific Unix shells.

Once this is set, run the installer. This redirects 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 need to set the variable `LAX_DEBUG=file`. Once you launch the installer, a file called `jx.log` containing debug output is 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

The following methods are available 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 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.

This procedure has 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 appears similar to the following truncated sample:

```
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 shows 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 you 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 the Display Message Panel

It is often desirable to debug some portions of an installation during installer development. One simple feature of InstallAnywhere Enterprise Edition enables you to add a display message panel that can display specific InstallAnywhere variable values.

For example:

1. Add a rule that states: `Install Only If $prop.os.name$=Solaris`  
The install continues, and the action assigned this rule does not execute.
2. So, add a display message panel. The message is: `The prop.os.name is: $prop.os.name$`

When you run the installer, the value of `prop.os.name` is “SunOS” and not “Solaris”; you can reformulate the rule to match the proper name.

# 9

# Source and Resource Management

---

In today's software development environment, it is rare that a developer is working in a vacuum, without interacting with other developers and departments. It is also rare that the coding effort for a project will be the responsibility of a single developer, from the planning stages to the customer desktop.

Development occurs in tightly integrated teams, often working in parallel, sharing the ownership of components and files. A core team tweaks the product while those responsible for release and deployment work at creating the deployment packages.

This chapter covers how to effectively manage the availability of these source files and resources, focusing on the topics:

- How Source Paths Work
- Managing Source Files
- The Resource Manager
- Creating Source Paths

# How Source Paths Work

Source paths enable developers to reference file resources using variable paths instead of absolute paths. This allows the sharing of 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.

Source paths will automatically be substituted for the most complete path possible.

For example, if you have two source paths defined as:

```
$SOURCEPATH1$ = 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:

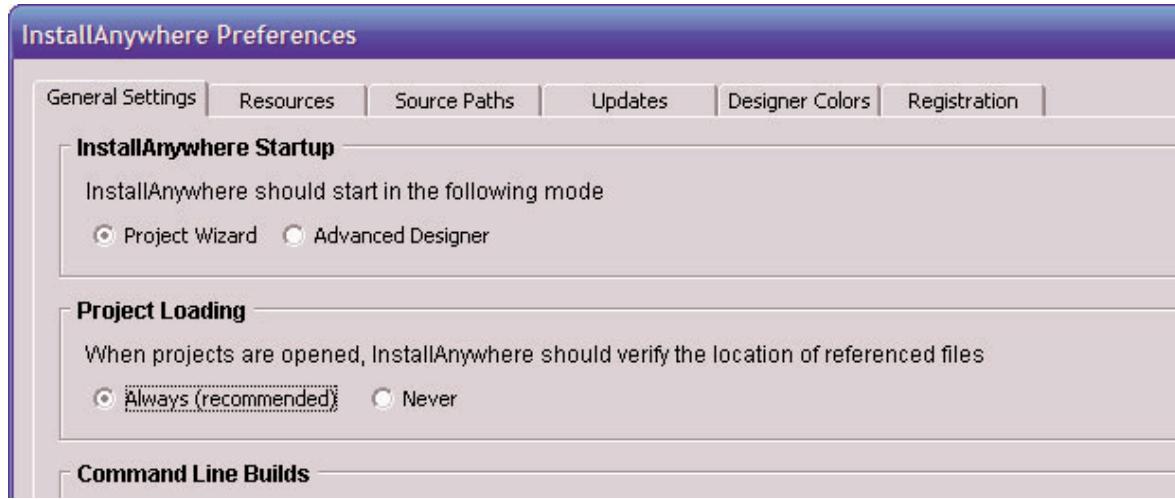
- `$IA_HOME$` is the location on the system where InstallAnywhere is running. A common location may be C:\Program Files\InstallAnywhere.
- `$IA_PROJECT$` is the location on the system where the InstallAnywhere project is located.
- `$USER_HOME$` is the User Home folder.

# Adding Source Paths

Source paths are added through the use of source path variables. This can be done in Advanced Designer or using environment variables.

## InstallAnywhere Preferences

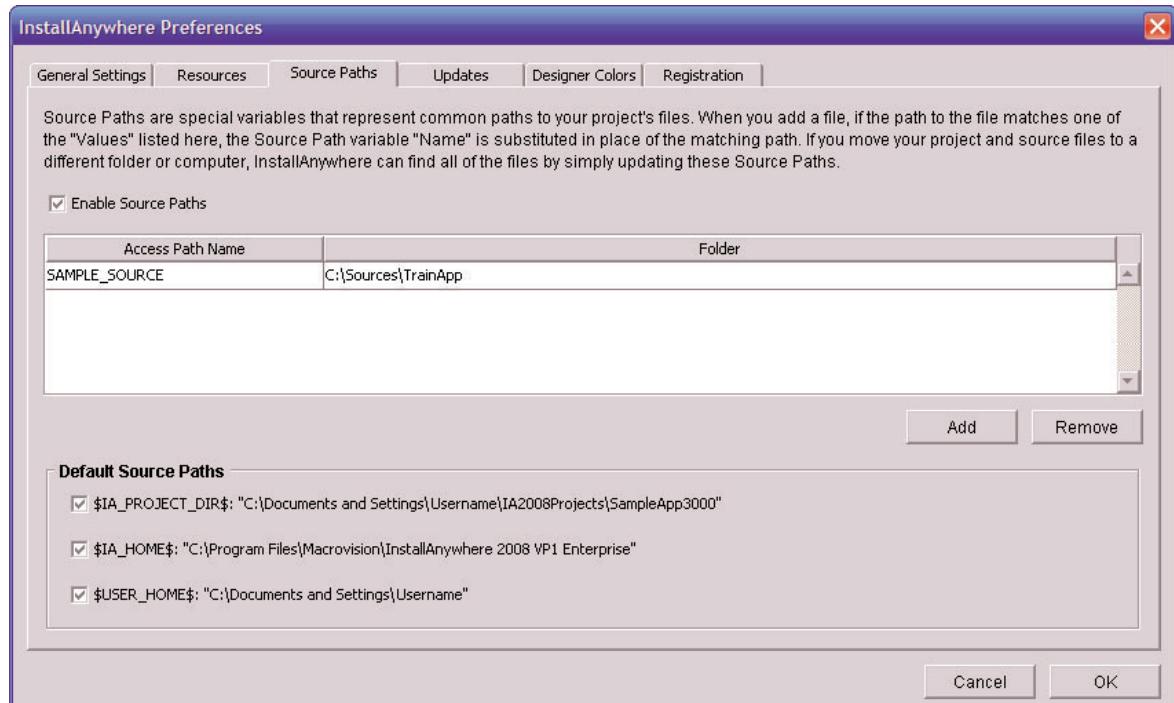
1. From the **Edit**, select **Preferences** to open the InstallAnywhere Preferences window.



**Figure 9-1:** InstallAnywhere Preferences Window

2. Select the **Source Paths** tab.
3. Click **Add** to create a new entry.

4. Enter the **Access Path Name** name, such as **RESOURCE**, in the table. Do not include the dollar sign (\$) around the path name. It will be added automatically when it is used.



**Figure 9-2:** Adding an Access Path Name and Folder

5. Click under **Folder** in the table. A button will appear labeled **Choose Folder**. Click the button to select the target location for the Source Path Variable (for example, c:\resources\test.txt).

## System Environment Variables

1. Access the environment variables.
  - On a Windows system, right-click **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.
2. Add an environment variable for the source path, prepended with **IA\_PATH\_** tag.

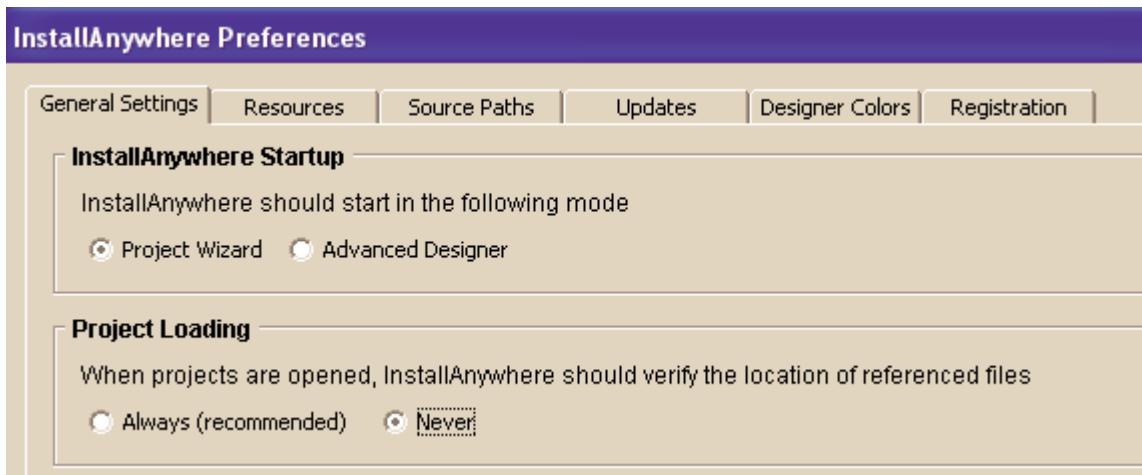
For example, to set the source path **SOURCE\_PATH**, set the environment variable **IA\_PATH\_SOURCE\_PATH**.

# 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, set the **Project Loading** preference found on the **General Settings** tab to **Never**.



**Figure 9-3:** Project Loading Setting

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 Manager settings can be found on the **General Settings** tab on 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, so if two paths are defined as:

```
$LONGPATH$ = D:\temp\dir\source  
$SHORTPATH$ = D:\temp
```

The file D:\temp\dir\source\hello.txt becomes

```
$LONGPATH$\hello.txt and not $SHORTPATH$\dir\source\hello.txt
```

## Enabling/Disabling Source Paths

To enable or disable any of these Source Paths, click **Edit** and **Preferences**. From the **Source Paths** tab, select or clear the option you would like.

## Default Source Paths

There are three default source paths that exist in any project; these default source paths cannot be changed or edited. They are:

- **\$IA\_HOME\$** is the location on your system where you are running InstallAnywhere. A common location may 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. Variables are defined in the **Preferences** menu, in the `PathManager.properties` file, or at the operating system level.

### Preferences Menu

1. From the **Edit** menu, select **Preferences** and click the **Source Paths** tab.
2. Click **Add** to include a new variable entry to the table.
3. Enter the **Access Path Name**, such as **RESOURCE**, in the text box.
4. Enter the **Folder** in the space allotted. For example, `c:\resources\test.txt`, or browser to the file location.



**Note:** Do not type dollar signs (\$) around source paths when you add them into Preferences.

### Set System Environment Variables

1. Access your environment variables. On Windows, right-click on **My Computer** on the Desktop, choose **Properties**, choose **Advanced**, and click **Environment Variables**.
2. Add source paths. These are stored in the same format as source paths created in the **Preferences** menu 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.

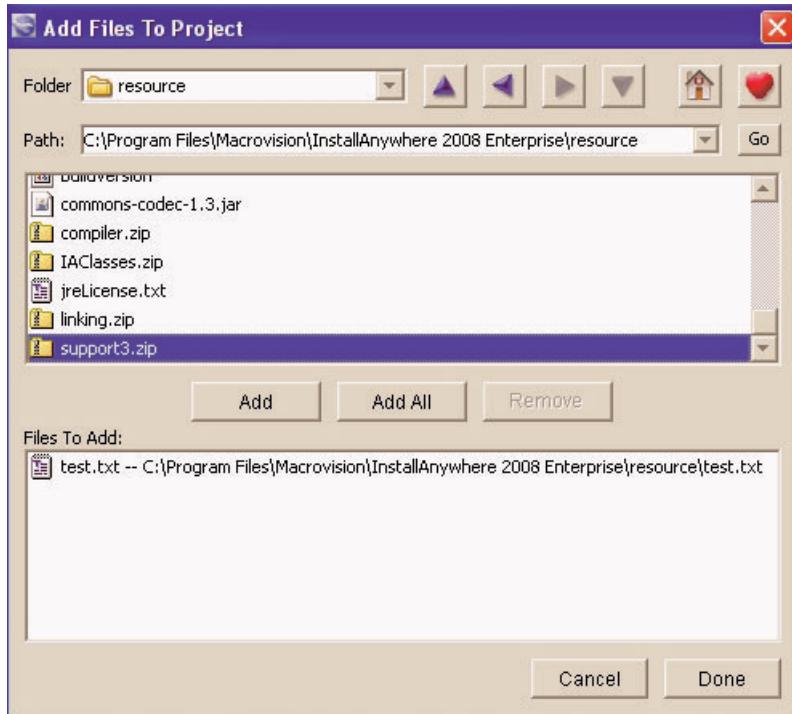


Figure 9-4: Adding Files From the Install Task

## 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.

## 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

# 10

## Advanced Installer Concepts

---

In a contemporary environment, an end user is likely to find a number of heterogeneous systems. They will be installing and uninstalling with different platforms, different operating environments, and different interaction environments. InstallAnywhere helps you meet the needs of these end users by presenting installer options that can run in any number of environments.

In Chapter 5, you were introduced to some commonly used installer concepts and tools. Beyond the basic installer wizard, graphical user tools, and variable-based rules, InstallAnywhere offers additional options for you to use when building an installer.

This chapter describes additional installation-related concepts, including:

- console-mode, for command-line-driven installations
- silent installations, for working with limited or no user interaction
- customized uninstallers, to control the removal of single or multiple products

### Console Installers

In enterprise-level environments, it is not uncommon for end users to install applications to servers and other remote systems. In these cases, a rich graphical user interface (GUI) 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 a silent installation that requires no end-user interaction.

InstallAnywhere Enterprise Edition supports both console-mode 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 using the default settings you defined in the project, 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 installation mode that best meets their needs.

Console-mode installers enable your end users a non-graphical user interface structured to enable interaction through text only. Console mode is intended to add support for non-graphical environments such as those common on so called “headless” Unix systems. 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.

To enable console mode for your project, you must select the **Console** check box in the **Installer UI > Look & Feel** task.



Figure 10-1: Allowable UI Modes

Additionally, to enable console mode for a Windows installer, choose the **Console Launcher** option for the **Install Launcher Type** in the task **Project > Platforms > Windows**, pictured in the following figure.

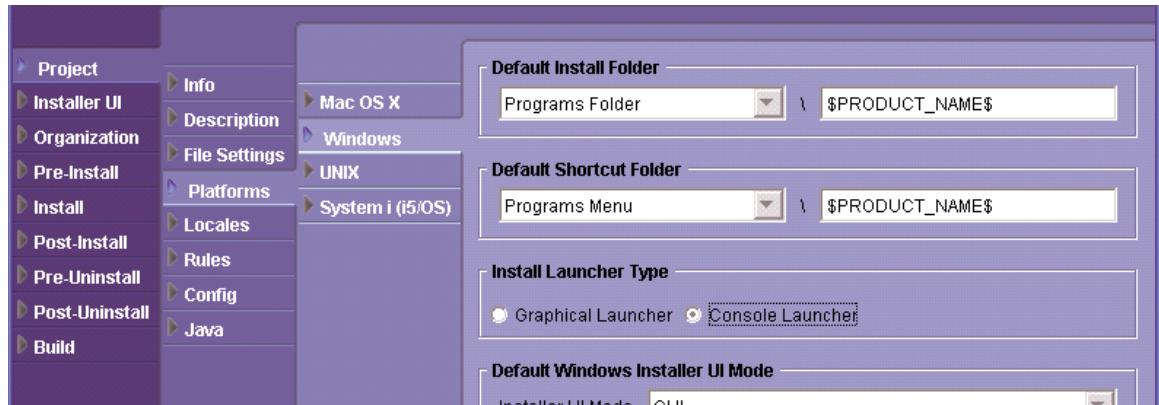


Figure 10-2: Install Launcher Type

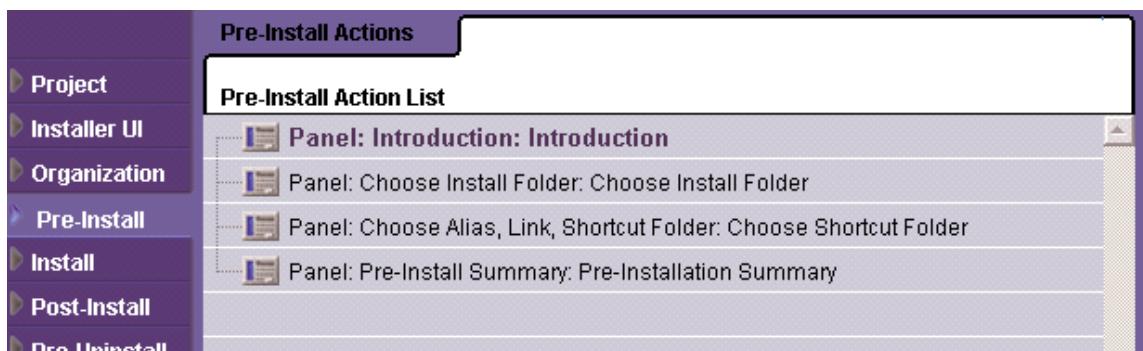
To run an installer in console mode, the user runs the installer with the `-i console` argument, as in the following:

```
install -i console
```

**Note:** If console mode has not been enabled for your project, the installer displays an error message that begins, “*Installer User Interface Not Supported*”.

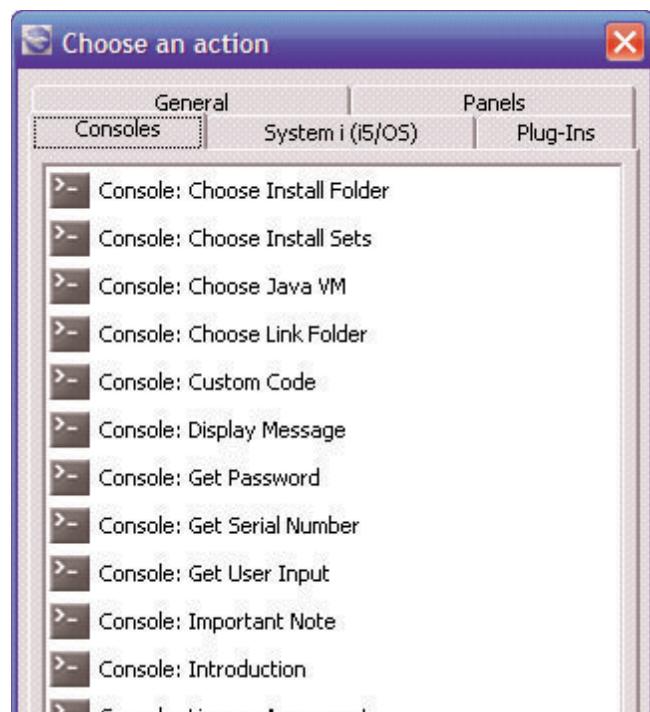
InstallAnywhere does not automatically provide console alternatives for panels you have added to your installer. You must provide consoles for each panel that you want displayed during console mode. In general, InstallAnywhere console actions provide parity with panels provided in the graphical mode.

For example, the **Pre-Install** task normally includes graphical panels such as **Introduction**, **Choose Install Folder**, and **Pre-Install Summary**.



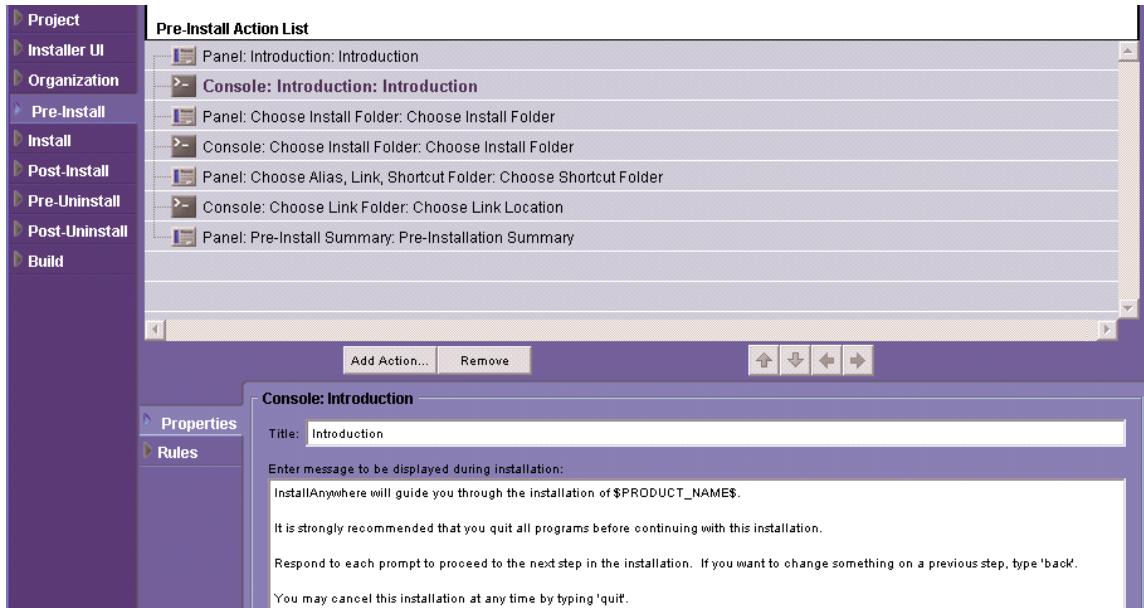
**Figure 10-3:** Pre-Install Action List

To add console equivalents to these graphical panels, begin by clicking the **Add Action** button and selecting the **Consoles** tab.



**Figure 10-4:** Consoles Tab

You then select the desired consoles and click **Add**, which places the console actions in the **Pre-Install** task. It is not necessary to insert the console actions next to their graphical panel equivalents, but doing so can help with the overall organization. Another approach is to group the panels and console actions separately.



**Figure 10-5:** Adding Console Actions

As with graphical panels, the customizer for a console action enables you to modify the text and behavior of the action in console mode. The default text of a console action is generally similar to the text used by the graphical panel, though slightly reworded to reflect the text-only mode. For example, where panel text might refer to **Previous** and **Next** buttons, the console action mentions typing "back" to return to a previous step.

At run time, the console actions display the text you specified in the respective customizers. For example, the console version of the **Introduction** panel appears as follows:

```
Introduction
-----
InstallAnywhere will guide you through the installation of SampleApp.

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

Respond to each prompt to proceed to the next step in the installation. If you
want to change something on a previous step, type 'back'.

You may cancel this installation at any time by typing 'quit'.

PRESS <ENTER> TO CONTINUE:
```

Similarly, the console version of the **Choose Link Location** panel appears similar to the following:

```
Choose Link Location
-----
Where would you like to create links?

-> 1- Default: C:\Documents and Settings\User\Start Menu\Programs\SampleApp
    2- In your home folder
    3- Choose another location...

        4- Don't create links

ENTER THE NUMBER OF AN OPTION ABOVE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:
```

Console-mode enables text to be output to the console line-by-line. It does not support formatted text, clearing the screen, or positioning the cursor.



**Tip:** Running the installer in console mode sets the InstallAnywhere variable `INSTALLER_UI` to the value "CONSOLE". If necessary, you can test the value of `INSTALLER_UI` to determine whether any action should run.

## 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, displaying only progress information, is available on Windows and Mac OS X.) As with console mode, you must enable silent mode for your project in the **Look & Feel** subtask.

To perform a silent installation from the command line, a user runs the following command:

```
install -i silent
```

This command runs the silent installer with all of the project's default settings. As with console mode, the user will see an error message if the project does not support silent mode.

## Using a Response File

To override the default settings in a silent installation, a user can point to a *response file*. A response file contains the values of InstallAnywhere variables to use during a silent installation.

To generate a response file, a user runs the installer with the **-r** switch. You can also use the setting **Always Generate Response File** in the **Project > Info** subtask. When the installer runs, it records end-user choices in a file called `installer.properties`, stored in the same directory as the installer. Sample contents of a response file are the following.

```
# Wed Mar 05 12:46:16 CST 2008
# Replay feature output
# -----
# This file was built by the Replay feature of InstallAnywhere.
# It contains variables that were set by Panels, Consoles or Custom Code.

#Choose Install Folder
#-----
USER_INSTALL_DIR=C:\\Program Files\\SampleApp3000

#Choose Shortcut Folder
#-----
USER_SHORTCUTS=C:\\Documents and Settings\\All Users\\Start Menu\\Programs\\SampleApp3000
```

Apart from comments (lines that begin with the hash sign #), the response file contains entries of the form:

```
PROPERTY_NAME=Value
```

A user can manually create a response file or modify property values inside a recorded response file as appropriate.

When a user deploys an installation, the installer executable looks for a file called `installer.properties` or `installername.properties`, and if such a file is present reads the property names and values and uses them as values during the installation. To specify a different file name or location, the user can specify the desired response file with the `-f` switch, as in:

```
./install.bin -i silent -f /usr/tmp/SampleResponseFile.properties
```

The response file can specify the user-interface mode by setting `INSTALLER_UI=SILENT` (for example) in the response file. This negates the need for using the additional `-i` switch to the installer executable.

## Configuring Variables Used in Response Files

In some cases it is undesirable to store property values in a response file generated with the `-r` switch. In the **Project > Info** task, you can configure variables to encrypt or exclude from a response file.

For example, suppose you have a simple **User Input** panel that asks the user to provide a sensitive passphrase.

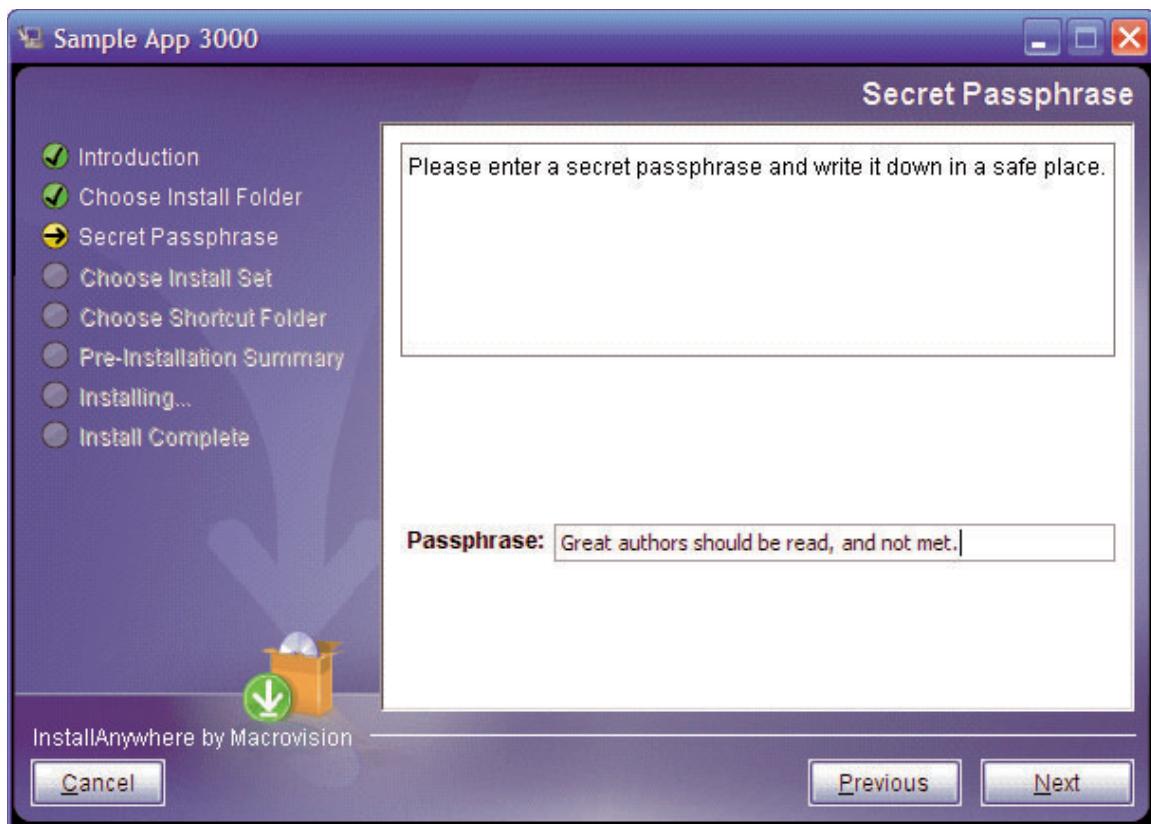
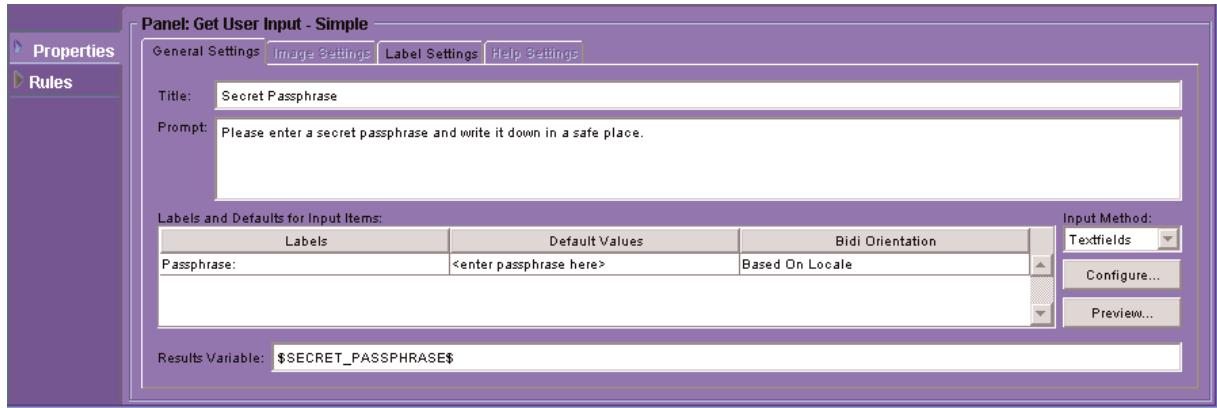


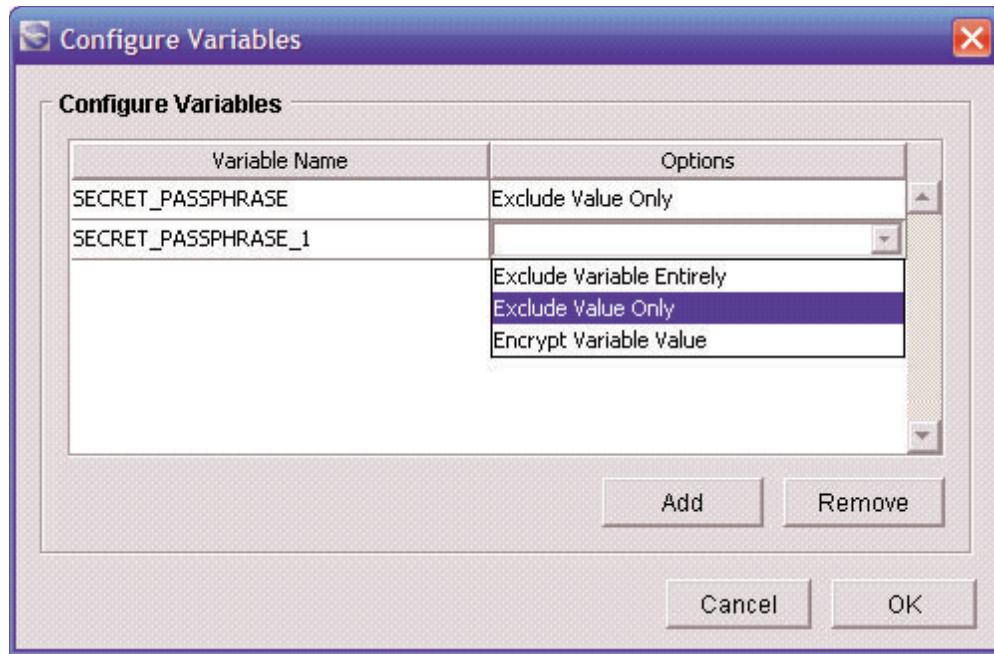
Figure 10-6: Passphrase Example

In the InstallAnywhere project, the panel is configured to store the user input in InstallAnywhere variables with the base name \$SECRET\_PASSPHRASE\$, which means the variables SECRET\_PASSPHRASE and SECRET\_PASSPHRASE\_1 both expose the sensitive data.



**Figure 10-7:** Input Variable Properties

To prevent the value from being included in the response file, begin by clicking the **Configure** button in the **Project > Info** task, which displays the **Configure Variables** dialog box. Click **Add** to add the names of variables to encrypt or exclude from the response file, along with the desired behavior (whether to exclude the entire variable entry from the response file, exclude only the value, or encrypt the variable's value).



**Figure 10-8:** Adding Options to Variables

If you select **Exclude Value Only**, a user who generates a response file by running `install -r` receives the following entries for the corresponding panel:

```
#Secret Passphrase
-----
SECRET_PASSPHRASE=
SECRET_PASSPHRASE_1=
```



**Note:** For additional information on user input panels, refer to Chapter 14.

## Uninstallation

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 customize the uninstaller in the same way you can customize the installer. In particular, you use the **Pre-Uninstall** and **Post-Uninstall** tasks to modify the panels and actions performed during uninstallation.



**Figure 10-9:** Pre-Install and Post-Install Tasks

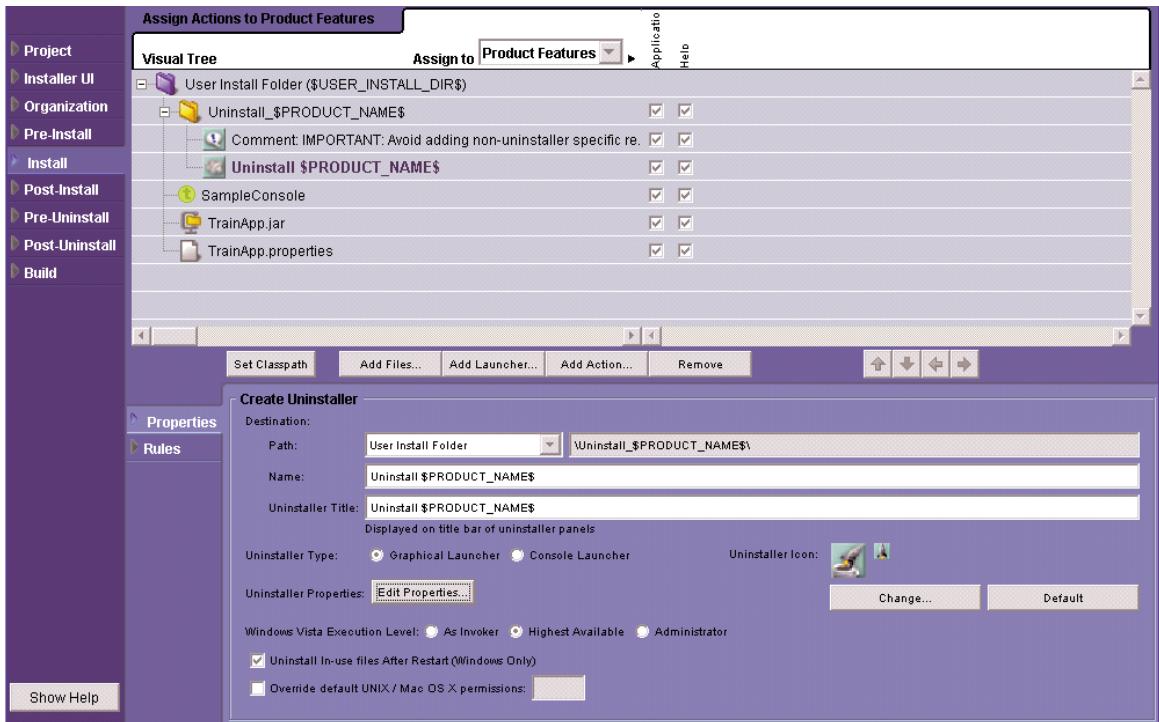
The uninstaller is similar to the installer. It is a collection of panels, consoles, and actions. It keeps track of what the installer has done, and contains a record of every action run during install time. All **Pre-Uninstall** panels, actions, and consoles run first; then the uninstall functionality of actions in the **Install** task are called; and lastly the **Post-Uninstall** actions are run. In addition, the product information (product, feature, and component information) is removed from the InstallAnywhere registry.



**Note:** You can disable integration with the InstallAnywhere registry by deselecting the **Do not update the product registry** check box in the **Project > Info** task.

If your installer supports console mode, you should add console equivalents to the graphical panels displayed during uninstallation.

You can customize the appearance, location, and some of the behavior of the uninstaller using its customizer in the **Install** task.



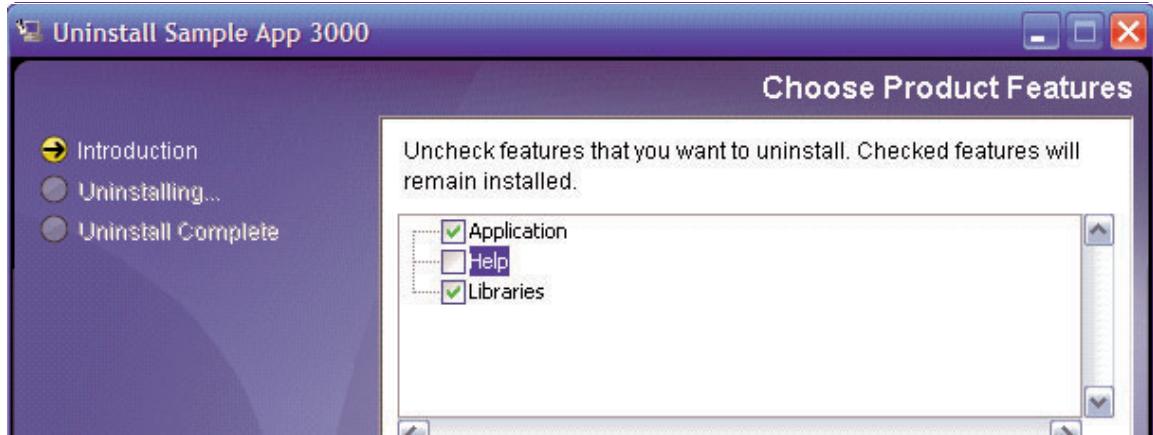
**Figure 10-10:** Customizing the Uninstall

By default, the uninstaller executable is called `Uninstall_ProductName`, and is located in an `Uninstall_ProductName` subdirectory of the installation location `$USER_INSTALL_DIR$`.

## Feature-Level Uninstallation

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.



**Figure 10-11:** Choosing Features to Uninstall

In the settings for the **Choose Features to Uninstall** panel, you can specify to use the opposite behavior, that installed features are displayed un-selected, and selecting a feature causes it to be uninstalled.

A feature-level uninstallation 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 Integration with the Target System

InstallAnywhere automatically creates an uninstaller for the project, which can be launched 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.

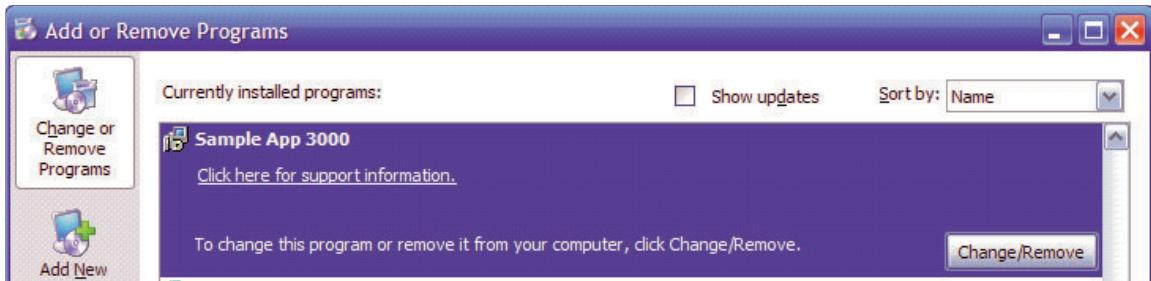


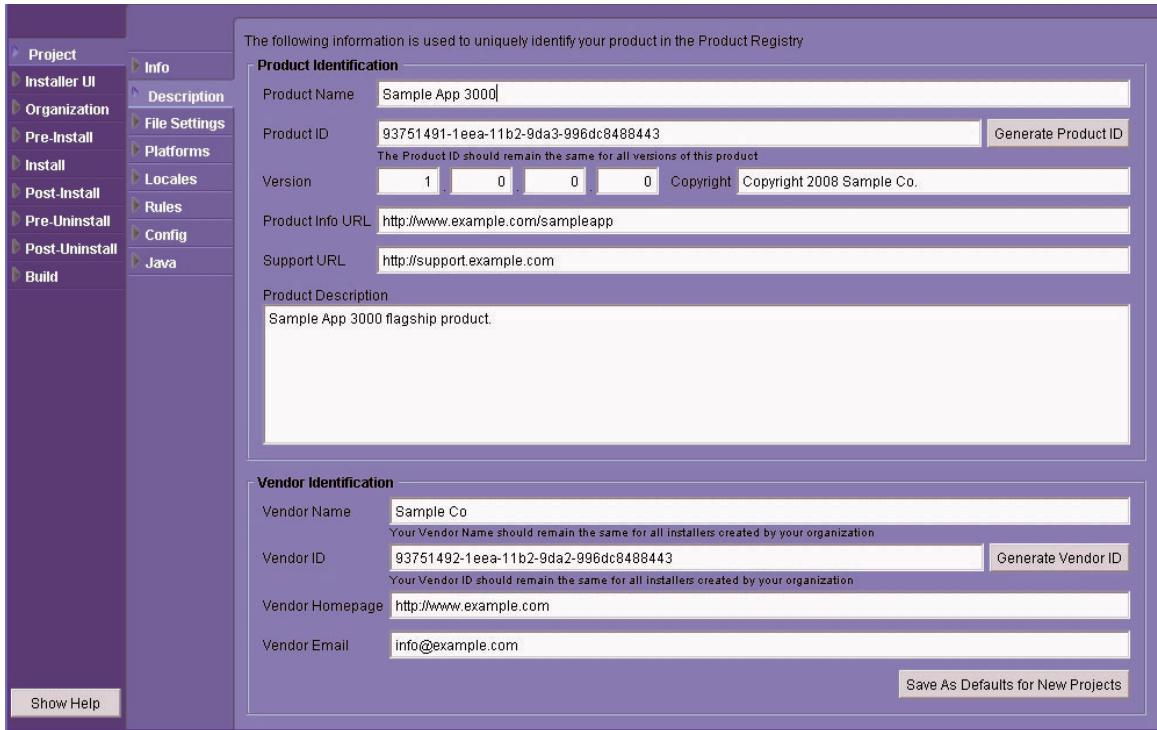
Figure 10-12: Add or Remove Programs

If the user clicks the “Click here for support information” link in the **Add or Remove Programs** entry, the following **Support Info** panel is displayed.



Figure 10-13: Support Information Panel

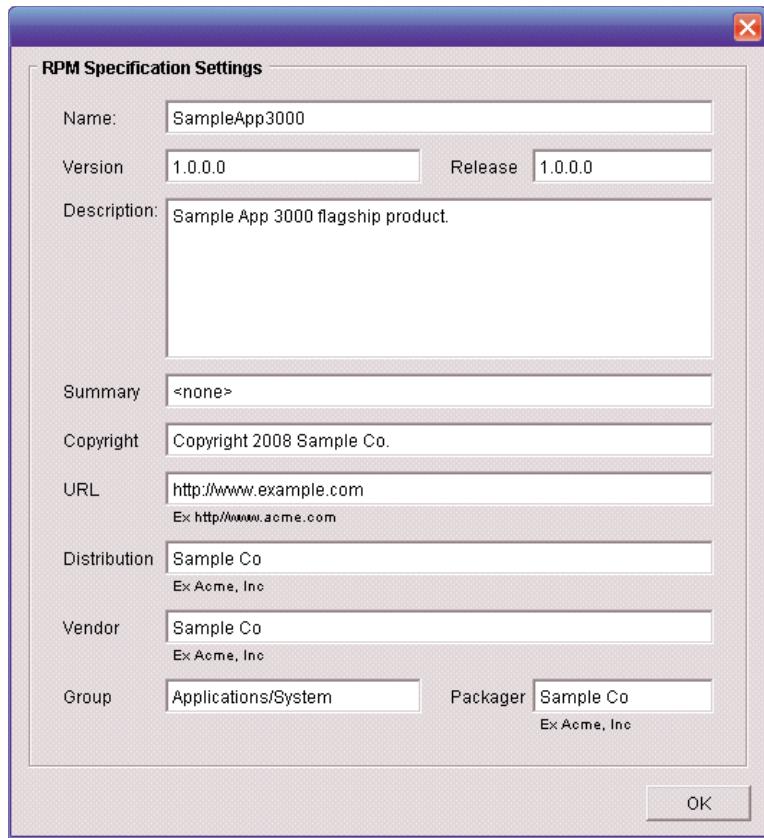
The details are populated based on the settings you specify in the **Project > Description** task.



**Figure 10-14:** Product Identification Task

Similarly, you can optionally specify to integrate your product information with the RPM database on a Linux user's system. As described in Chapter 3, you can specify the integration in the **Project > Platforms > UNIX** task.

If you select to enable RPM registration, the **Configure** button enables you to specify the information to store in the RPM database.



**Figure 10-15:** RPM Specification Settings

Likewise, InstallAnywhere 2008 Value Pack 1 introduces support for integration with the SWVPD registry on AIX target systems, and RAIR support on System i (i5/OS) systems.

## 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 each project in the “suite” have the same Product ID value. 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. For 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"



# 11

## Advanced Organizational Concepts

---

This chapter contains information on:

- Integrating Find Component in Registry Action
- Merge Modules and Templates
- Importing a Design Time Merge Module
- Creating Merge Modules
- Collaboration and DIMs
- 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. The following figure shows the customizer for the **Find Component in Registry** action.

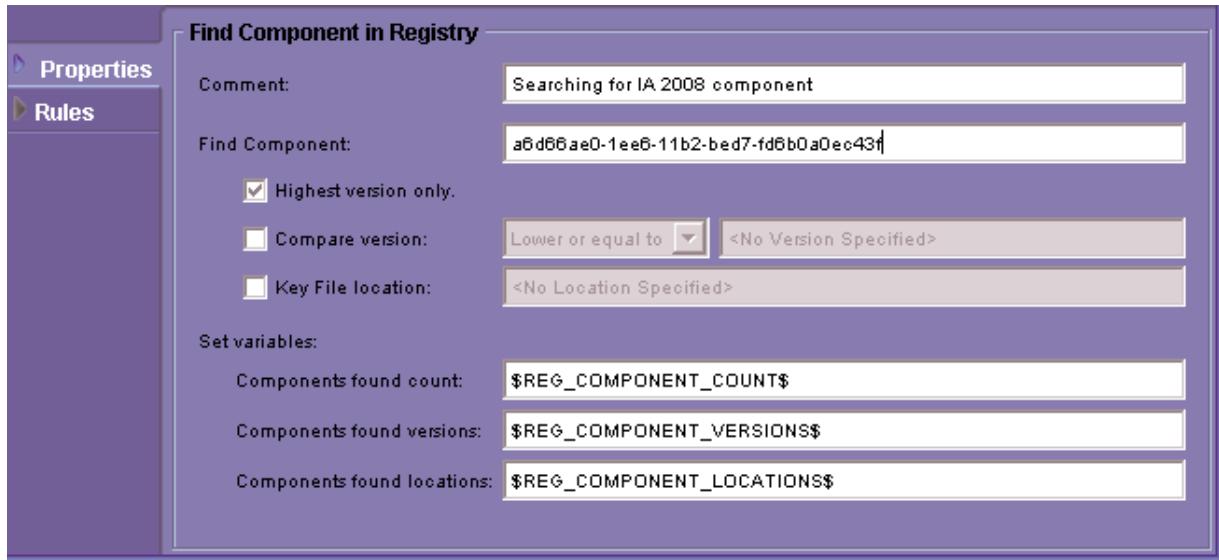


Figure 11-1: Find Component in Registry Menu

## 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.
- 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.

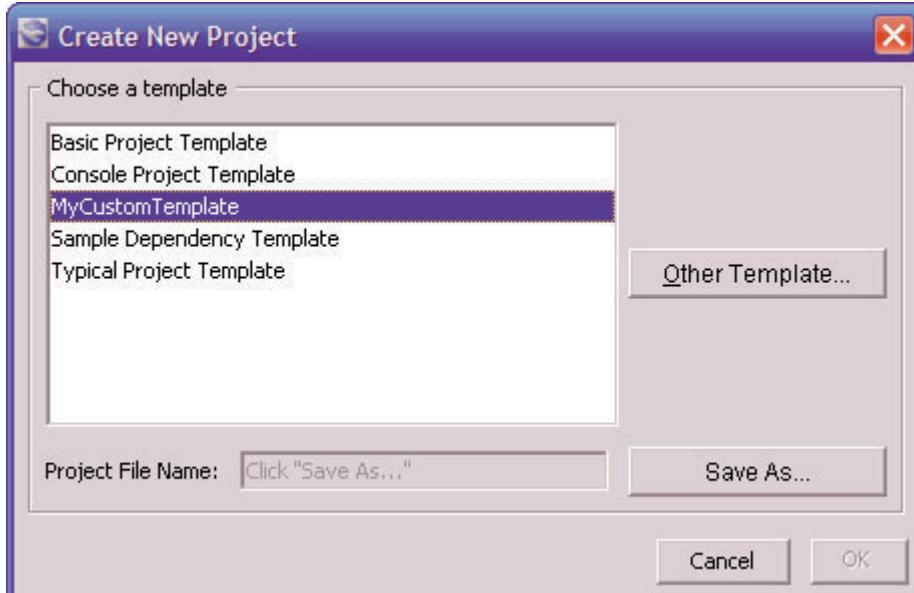
Other important facts about Merge Modules:

- **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` directory 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.



**Figure 11-2:** Creating a New Project from Template

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.

## Advanced Topic: Importing ISMP Manifests

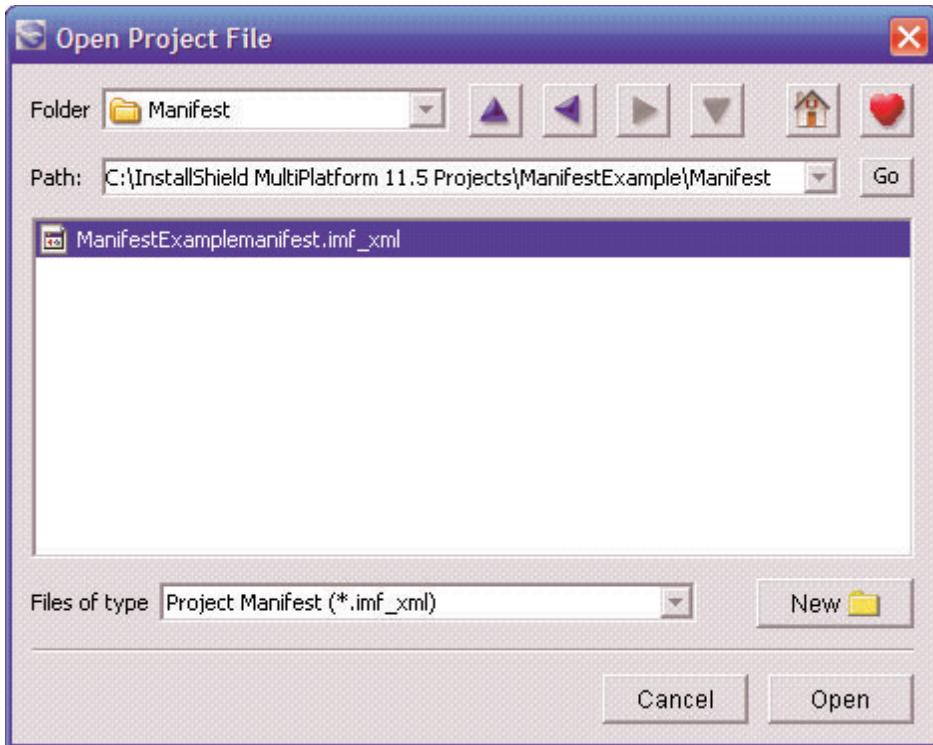
To assist with migration from InstallShield MultiPlatform to InstallAnywhere, you can export an ISMP project manifest using ISMP 11.5 Service Pack 1, and then import the ISMP manifest to create a new InstallAnywhere project.



**Note:** To download the ISMP Project Manifest Export Plug-in installer, see Macrovision Knowledge Base article Q112375.

When you export an ISMP project manifest, the result is an XML-format file with the extension .imf\_xml, along with DIM files that describe the ISMP project data. These project manifest files are different from InstallAnywhere file and directory manifests.

To import the ISMP manifest into InstallAnywhere, pull down the **File** menu and select **Open**, select **Project Manifest (\*.imf\_xml)** from the **Files of type** list, and then browse for the ISMP project manifest.



**Figure 11-3:** Opening a Project Manifest

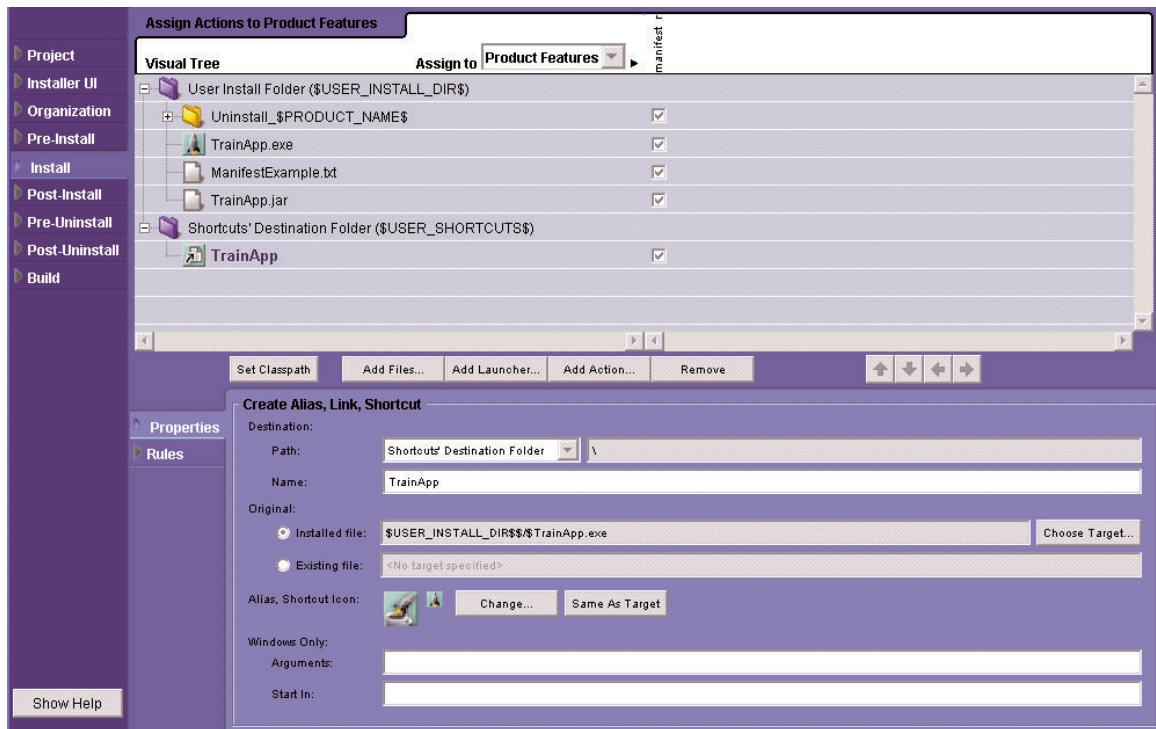
When you select the ISMP manifest file and click **Open**, you will be prompted whether the components referenced in the ISMP manifest should remain as DIMs, using the following prompt.



**Figure 11-4:** Confirmation Dialog Box

If you select **No**, the component data will be imported directly into your project, instead of remaining in DIM format.

When the import is complete, the data from the ISMP manifest is converted to InstallAnywhere format, and you can modify individual actions and data using each converted action's customizer.



**Figure 11-5:** Action Customizer

Not all data can be translated directly from ISMP to InstallAnywhere. For example, ISMP conditions are not translated to InstallAnywhere rules (though there may be direct equivalents between the two systems), and custom code and custom dialog boxes are not translated. For information about custom code and custom panels in InstallAnywhere, as well as the ability to call ISMP services from InstallAnywhere actions, refer to Chapters 13 and 14.

# 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 Modules 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 do not 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, refer to 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.



**Figure 11-6:** Import Settings

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.

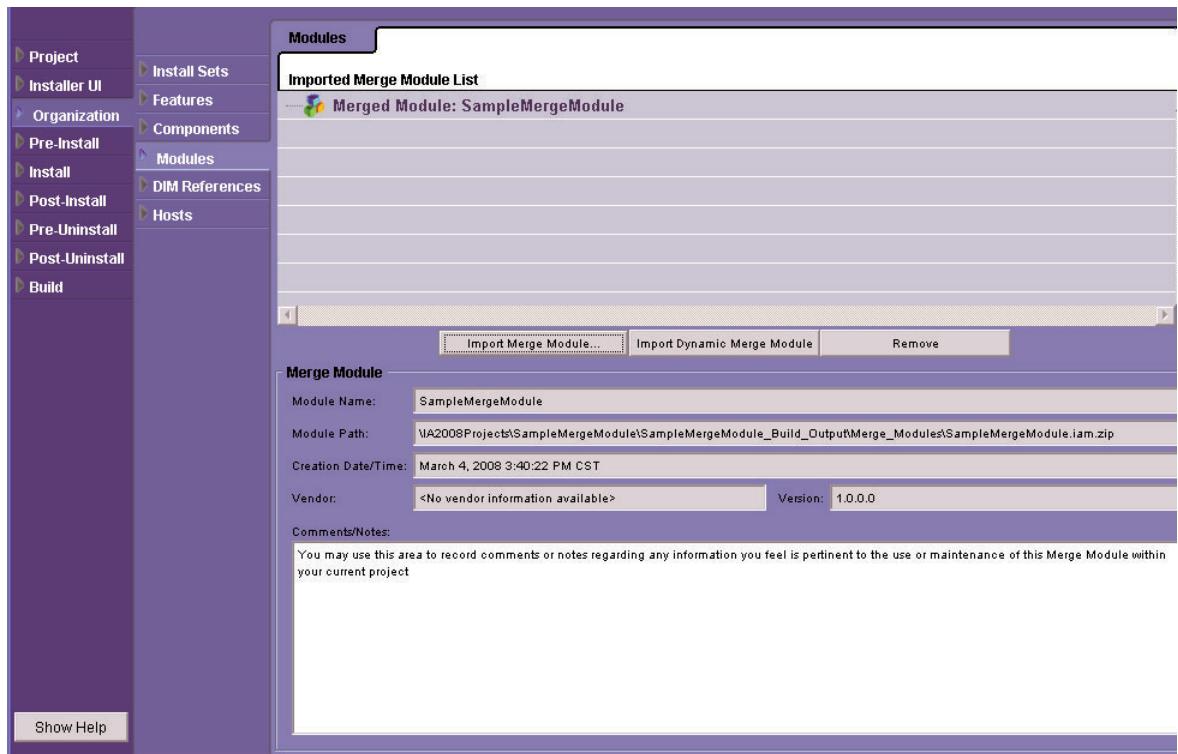


Figure 11-7: Imported Merge Module List

5. Click OK.

The merged project appears under the **Features** task. 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.



Figure 11-8: Panels and Actions in the Feature Tree

## Merge Module Customizer

The Merge Module customizer will be populated automatically with the information about the Merge Module. You may add comments and notes in the text box provided.

## InstallAnywhere Collaboration and DIMs

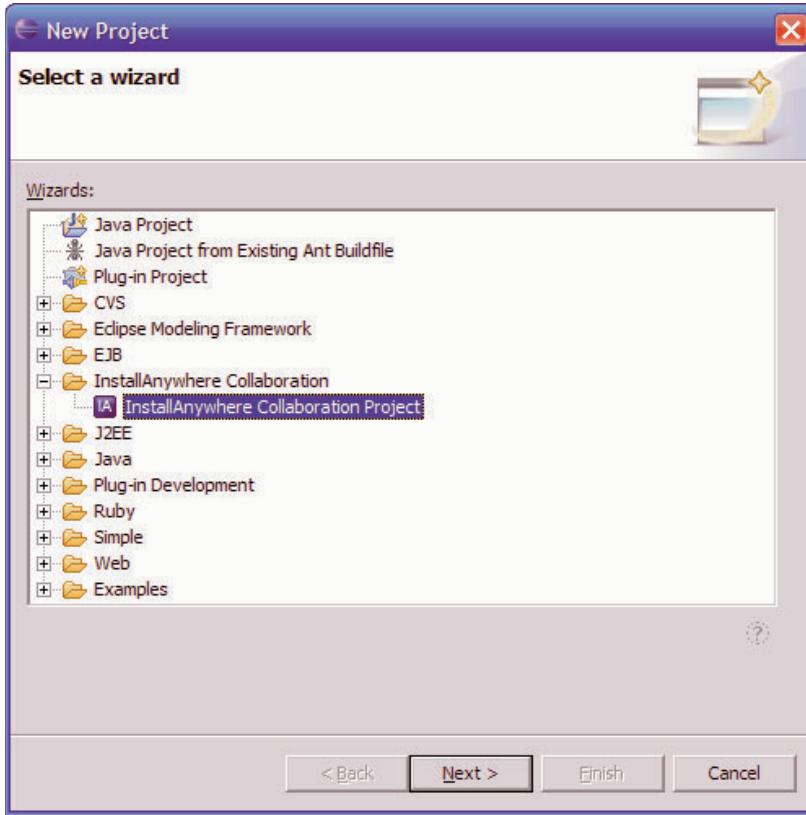
InstallAnywhere Collaboration supports software development teams by enabling software application developers to collaborate with release engineers on the design, development, and deployment of their software subsystems. InstallAnywhere Collaboration is a standalone product that plugs in to Eclipse.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

When you install InstallAnywhere on a system that has Eclipse installed, you can optionally install a copy of InstallAnywhere Collaboration. For information about obtaining additional copies of InstallAnywhere Collaboration, visit the Macrovision web site at <http://www.macrovision.com>.

## Creating a DIM

To create a DIM, a developer opens Eclipse and creates a new InstallAnywhere Collaboration project.

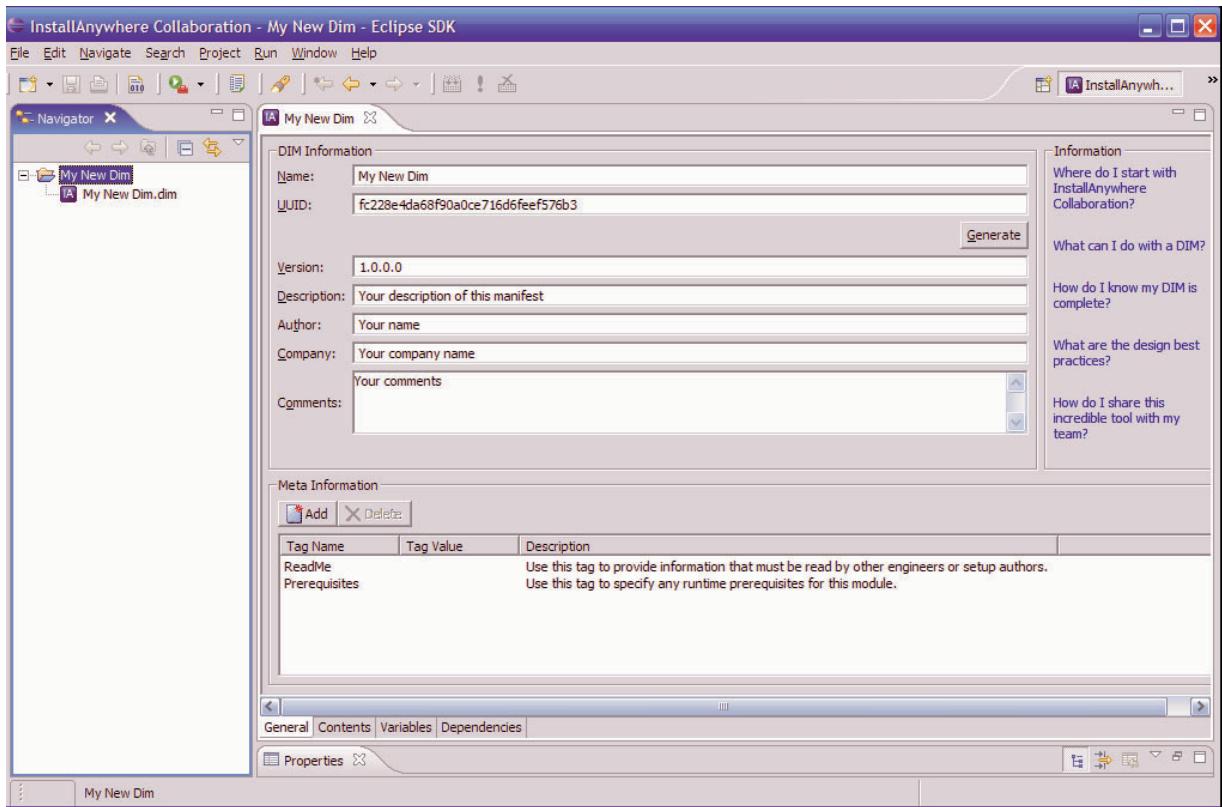


**Figure 11-9:** New Project Window

If appropriate, an InstallAnywhere Collaboration project can contain more than one DIM. By default, creating a new project creates a single DIM.

In Eclipse, the DIM settings are exposed in the following four views, which can be accessed at the bottom of the environment:

- General
- Contents
- Variables
- Dependencies



**Figure 11-10:** New DIM Tab

The Eclipse figures displayed here use the InstallAnywhere Collaboration perspective. To specify this perspective, click the **Open Perspective** toolbar button, select **Other**, and select **InstallAnywhere Collaboration**.

In the **General** view, you specify global information regarding the DIM. Each DIM has a required name, UUID, and version. There are also optional **Description**, **Author**, **Company**, and **Comments** settings that are stored in the .dim file but that do not affect the target system.

In this view you can specify *meta tags*, which are arbitrary named strings that are displayed in the InstallAnywhere environment. Meta tags are made up of a name, value, and description. The values of meta tags are displayed in the InstallAnywhere environment when you consume the DIM, but the values are not used as installation data. (For run-time data, you can create run-time variables, as described later in this section.) You can use meta tags to convey instructions concerning the DIM to the installation developer.

The **Contents** view is where you add installation data such as files (files to install or changes to ASCII-file or XML-file contents), registry information, and environment-variable strings.

To specify files to install, for example, right-click the File System icon and select **New File Set**.

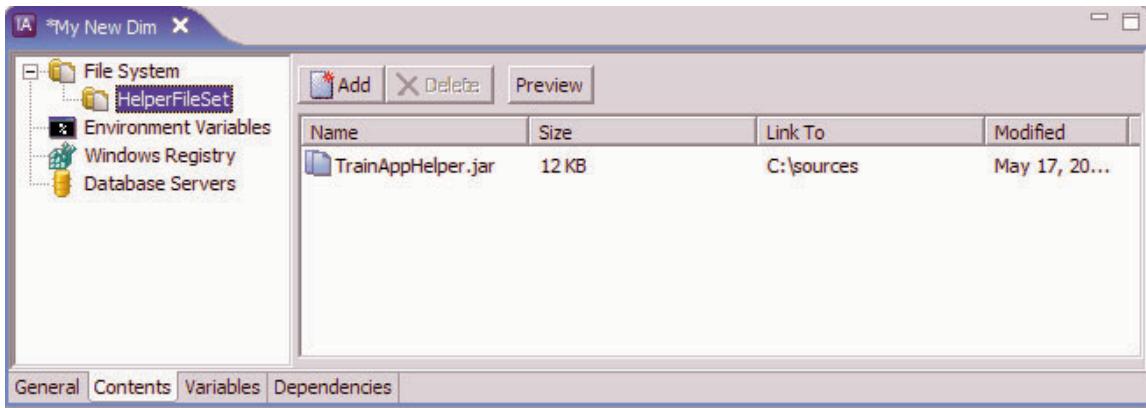


Figure 11-11: Contents View

To add files or directories to the file set, click the **Add** button and select **Add File**, **Add Dynamic File Source**, or **Add Empty Directory**, followed by browsing for the desired item.

To specify properties of the file set as a whole, select the file set and modify the **Properties** list that appears beneath the **Contents** view.

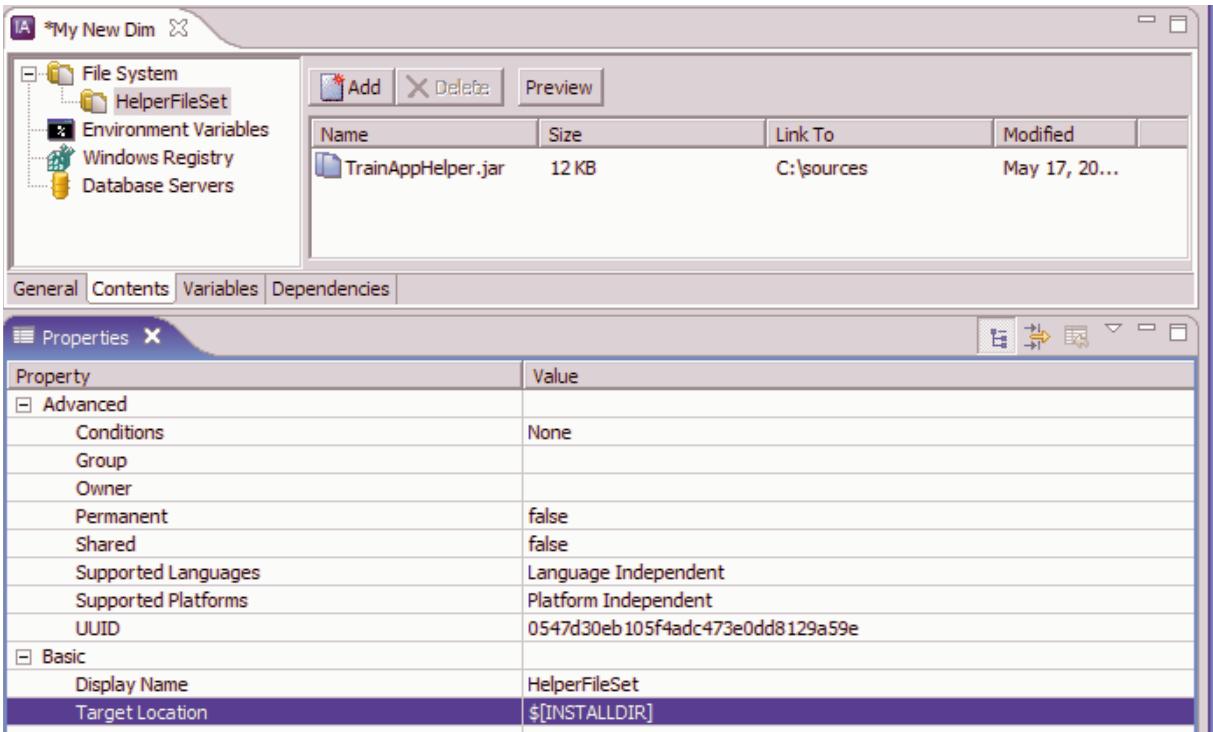
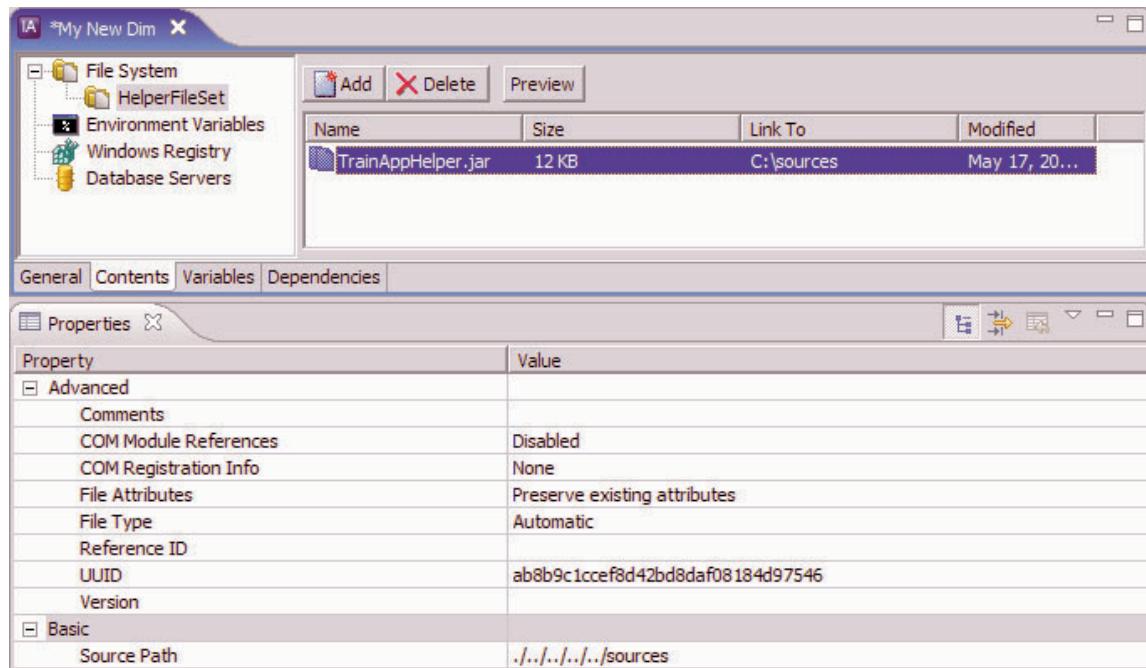


Figure 11-12: Contents View Properties

Because of the installer-independent nature of DIMs—they can also be used by InstallShield Windows—the system variables that represent destinations on the target system use a different syntax from the one used by InstallAnywhere. For example, the overall destination of the project that consumes the DIM is

represented as `[$INSTALLDIR]`. The **Target Location** property of a file set provides a drop-down list of special system variables, such as `[$ProgramFilesFolder]`, `[$OSSystemFolder]`, `[$TempFolder]`, and so forth.

In addition, you can specify conditions and supported languages and platforms for a file set, as well as properties such as file attributes and COM-registration information. Another consequence of the installer-independent nature of DIMs is that there are some properties that can be specified in a DIM that have no effect when you consume the DIM in an InstallAnywhere project. The following figure shows the **Properties** list for an individual file.

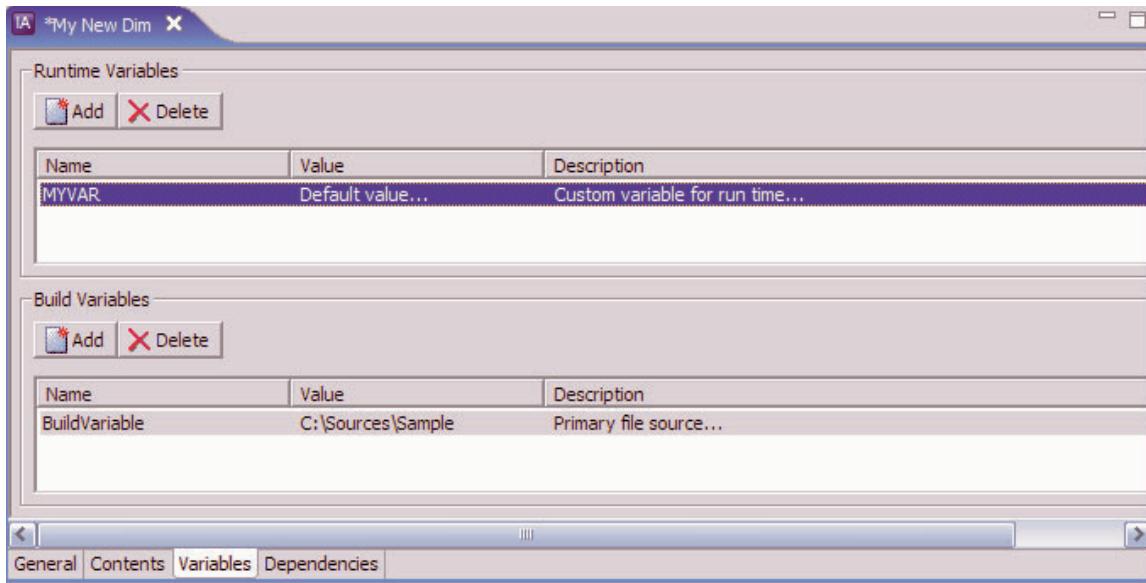


**Figure 11-13:** Specifying Values in Contents View

Other types of data that can be used by a DIM include registry data from .reg files and environment variables. For information regarding the creation of these types of data, see the InstallAnywhere Collaboration help library.

There are two kinds of variables used in DIMs, *run-time variables* and *build variables*. A run-time variable is available at run time, using the syntax `[$VARNAME]` (similar to an InstallAnywhere variable). For example, a run-time variable called `MYVAR` can be expressed as `[$MYVAR]` in a file set's **Target Location** value, or the expression `[$MYVAR]` can be used in an environment variable value.

A DIM build variable is similar to an InstallAnywhere source path, representing the source path of a file at build time. Like InstallAnywhere source paths, DIM build variables are not available at run time.



**Figure 11-14:** Variables View

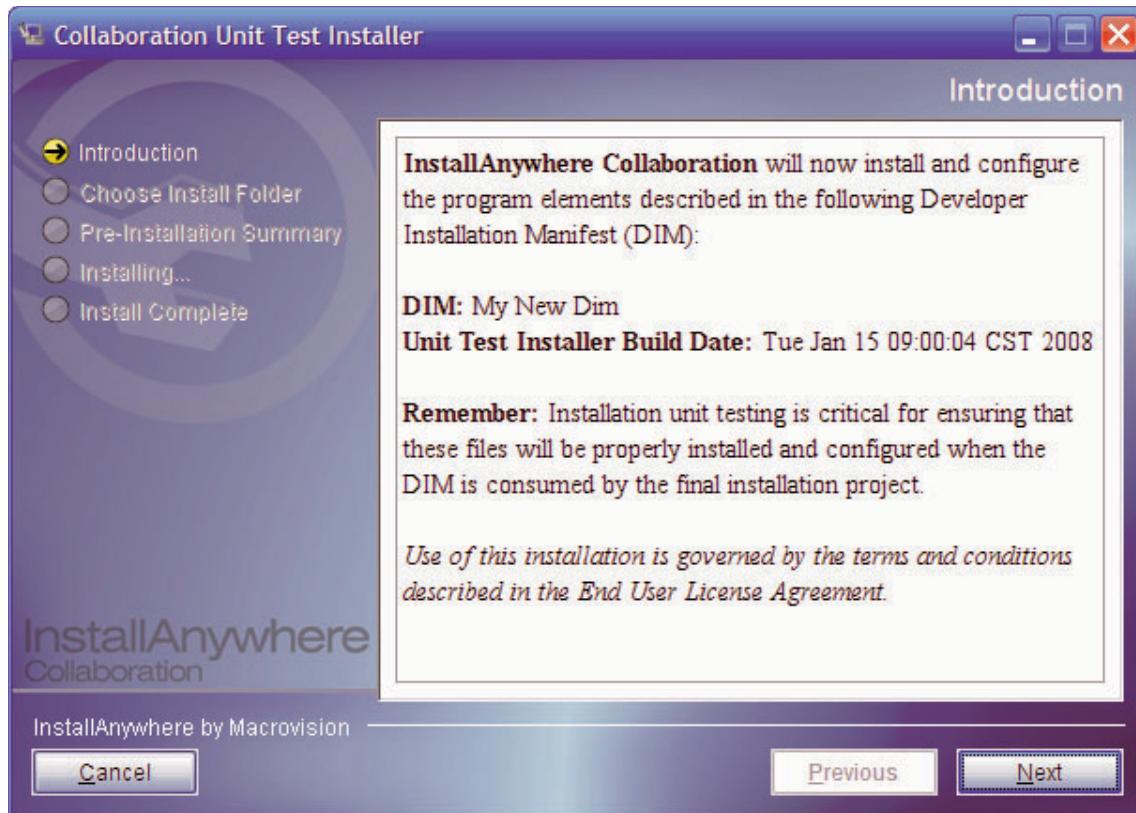
InstallAnywhere Collaboration uses either relative or absolute paths to source files. A recommended procedure is to treat DIMs as source code, meaning that they should be included in your version control system.

Data in the **Dependencies** view of a DIM project is currently unused by InstallAnywhere.

## Unit Tests

Unlike working with an InstallAnywhere merge module, there is no build process for a DIM: you simply click **Save**, and the DIM maintains references to the data that will be consumed by the installation. To rename the DIM, use the **Save As** command.

To test the DIM, build a unit test of the DIM. With Eclipse, the unit test is a simple InstallAnywhere installation that installs the DIM data. To build the unit test, click the **Build Unit Test** toolbar button. To run the unit test, click **Run Unit Test**.



**Figure 11-15:** Collaboration Unit Test Installer

To uninstall the data installed by the unit test, use the uninstaller .jar file in the installation directory.

## Consuming a DIM in an InstallAnywhere Project

You can use the **Organization > DIM References** task in the Advanced Designer to consume DIMs created by your developers. To add a DIM reference, click the **Add DIM Reference** button and browse for the desired .dim file. The new **DIM Reference** icon appears in the **DIM References** task.

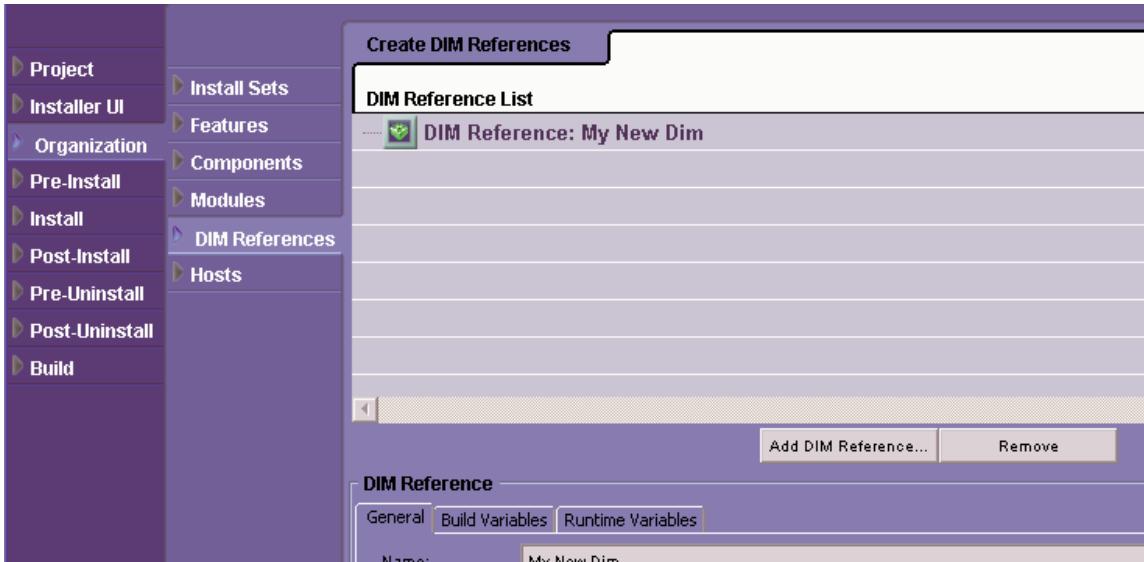
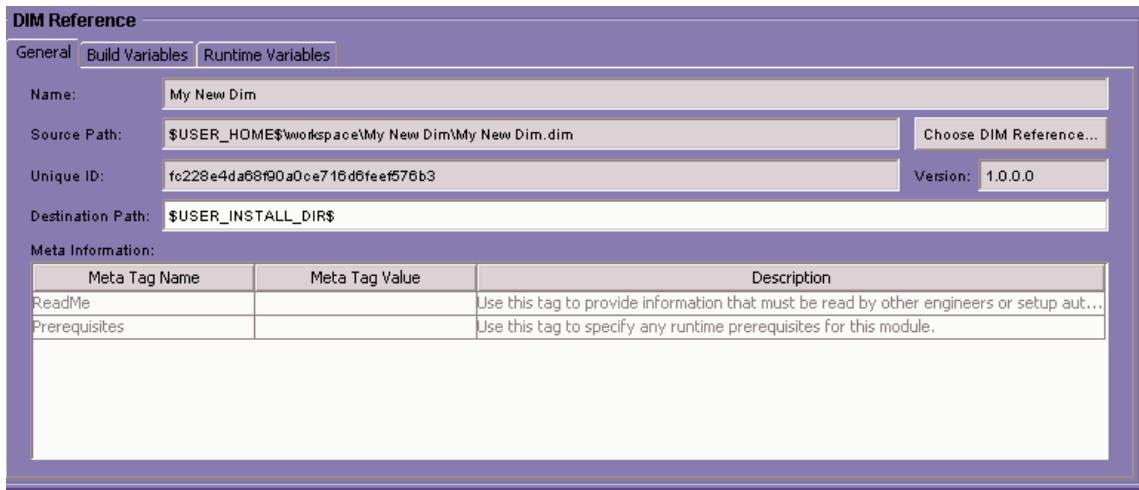


Figure 11-16: DIM References

The DIM's customizer organizes the settings into **General**, **Build Variables**, and **Runtime Variables** tabs. In the **General** tab, you see the global DIM information set by the developer, such as the DIM name, version, UUID, and destination path.

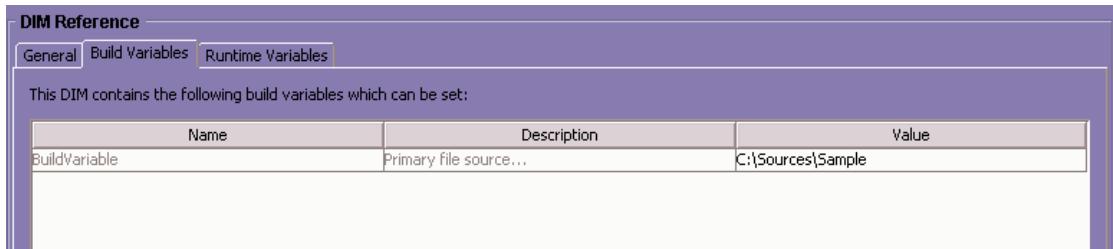
Apart from the destination path, the DIM settings in the General category are read-only. The value you specify in the **DIM's Destination Path** setting is used as the DIM's `[$INSTALLDIR]` value.

In the **Meta Information** section of the customizer, you can also view the meta tags defined by the developer. These tags can contain special instructions for using the DIM, and therefore it is recommended that you read the meta tags before building the project.



**Figure 11-17:** Build and Run-time Variables

In the **Build Variables** and **Runtime Variables** tabs, you see the build variables and run-time variables that the developer defined in the DIM.



**Figure 11-18:** Build Variables

DIMs are also exposed in the **Install** task of your project. As with other actions in the **Install** task, you can assign the DIM reference to one or more product features or components.

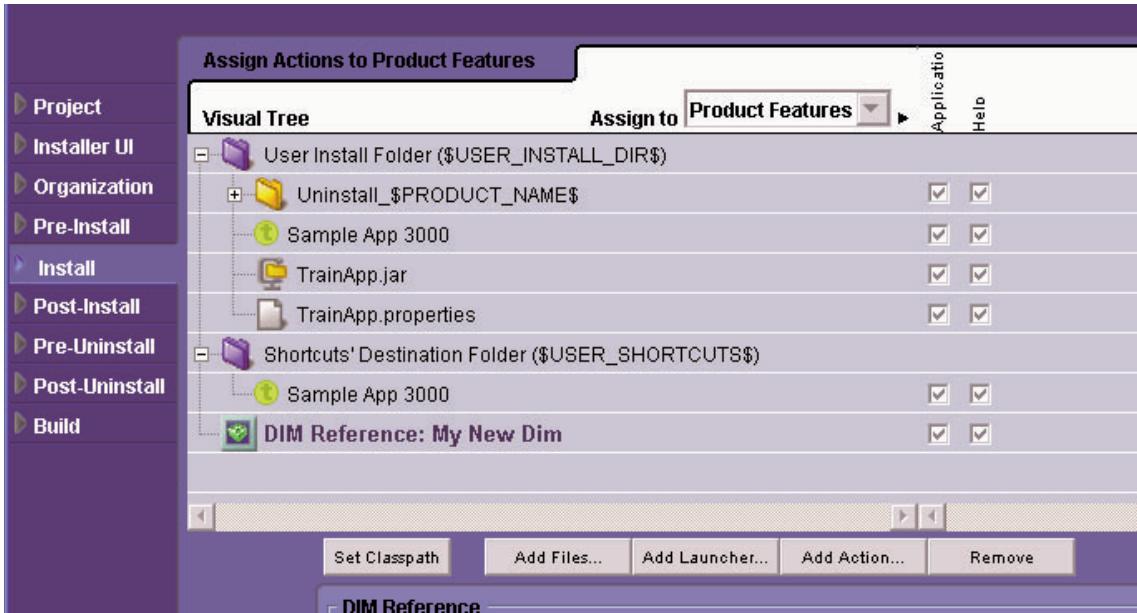


Figure 11-19: Viewing DIMs From the Install Task

DIMs do not contain shortcuts, as it is typically the responsibility of the installation developer to specify where shortcuts should be placed on a target system. This helps to ensure that all the product shortcuts are created in the same directory of the **Programs** menu, for example. To create a shortcut to a file inside a DIM, you can use the **Create Alias, Link, Shortcut to DIM File** action.

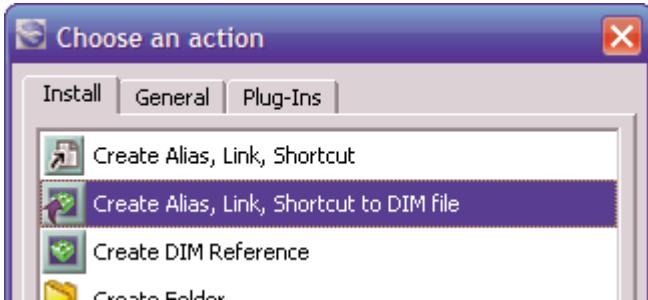


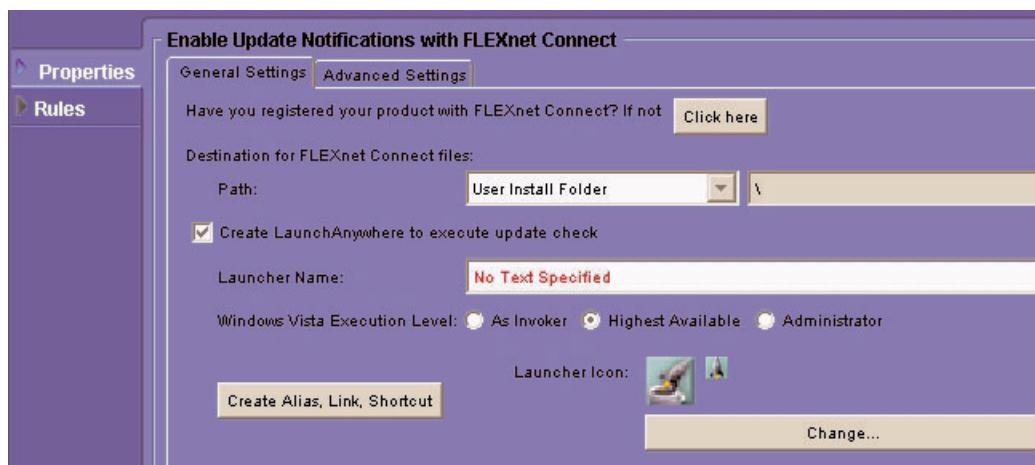
Figure 11-20: Choosing the Shortcut Action

In the customizer for the **Create Alias, Link, Shortcut to DIM File** action, you browse for the target file inside the DIM.

Once the DIMs are in place, you can build your installation project, and the DIM data will be included in your InstallAnywhere installation program.

# 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.



**Figure 11-21:** Enable Update Notification

# 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
  
- 2.** Which Merge Module Type will be included in the Master uninstaller?
  - A. Design Time
  - B. Build Time
  - C. Install Time
  
- 3.** 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
  
- 4.** 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 | 4. D

# 12

## Integrating InstallAnywhere with Automated Build Environments

---

This chapter contains information on:

- InstallAnywhere Command-Line Build Facility
- Digitally Signed Installers
- Ant Build Integration

InstallAnywhere includes 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 make some modifications to the project without utilizing the Advanced Designer, build your project from a “headless” system, 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 the graphical Advanced Designer interface.

The Command-Line Build facility comes in the form of a second executable called `build` (or `build.exe` on Windows systems). This executable takes as an argument the full path to an InstallAnywhere `.iap_xml` project file, and by default will simply build the installer as specified in the project file. You can run `build` or `build.exe` with the `-?` switch to display a usage message and some sample commands.

For example, to build a SampleApp project, you might use a command line similar to the following from the command prompt or a batch file on a Windows build system:

```
"C:\Program Files\InstallAnywhere\build.exe" "C:\Projects\SampleApp3000.iap_xml"
```

In this case, InstallAnywhere builds your project with the settings stored in the project file from the last build or save of the project.



**Note:** 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.

While the command-line build is taking place, InstallAnywhere displays the values of Java system properties being used for the build, along with progress and status information about the different steps involved in the build. When the build is complete, the command-line builder displays a breakdown by time of the various build stages.

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 systems, you can use the command `echo %ERRORLEVEL%`. For a list of specific exit codes, refer to the InstallAnywhere help library topic “Exit Codes”.

There are times when you may wish to build an installer with settings different from those stored in the project file. Naturally, you could open the project file with the graphical environment, 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.

## Builder Arguments

As before, to build releases for a `SampleApp` project with the previously saved build settings, you can run the following command:

```
build SampleApp.iap_xml
```

The build executable accepts switches to add or remove different platforms from the current build, with or without included bundled VM packs. To add a platform, use the argument `+z`, where `z` is the abbreviation for a particular platform; and to remove a platform use the argument `-z`.

Examples of platform abbreviations are:

- `w` for Windows without a VM pack
- `ww` for Windows with a VM pack
- `l` for Linux without a VM pack.

For a complete list of platform abbreviations, run `build -?` or refer to the InstallAnywhere help topic “Command-Line Build Options”.

Using the add-platform and remove-platform arguments to build installers for the `SampleApp` project, overriding the project’s defined platforms to build for Mac OS X and Linux only (turning all others off), you would run a command similar to the following:

```
build SampleApp.iap_xml +x +l -j -s -u -w
```

Other arguments to the build executable include those for specifying what media type to build (`cd`, `web`, and `merge`); whether to optimize by platform (`opt`). InstallAnywhere 2008 Value Pack 1 introduces switches for specifying a working directory for the build (`-d`), credentials to access a System i (i5/OS) machine (`-i`), and pointing to a license server for floating licenses (`-ls`).

As an alternative to specifying arguments to the build executable on the command line, you can store the settings in a build-properties file, and use the `-p` switch to point to the desired build-properties file. The build-properties file is XML-based, with elements corresponding to command-line switches to the build executable. A sample build-properties file that specifies to build only the CD-ROM media for Windows platforms (with no VM) is the following:

```
<build>
  <OverrideAllPlatformSettings>true</OverrideAllPlatformSettings>
  <BuildWindowsWithoutVM>true</BuildWindowsWithoutVM>
  <BuildCDROMInstaller>true</BuildCDROMInstaller>
</build>
```

Assuming the settings are stored in `SampleAppBuildProps.xml`, to build the installer for a `SampleApp` project, overriding the saved build settings for that project with a build properties file:

```
build SampleApp.iap_xml -p SampleAppBuildProps.xml
```

A build-properties file template called `BuildProperties.xml` is provided in:

```
<InstallAnywhere>/resource/build/BuildProperties.xml
```

The `BuildProperties.xml` file provides a sample of all possible build settings and can be customized to suit your build requirements.

## Digitally Signed Installers

Newer versions of Microsoft Windows include security features which can warn users that they are about to run “unsigned code”, meaning that the operating system is unsure 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 digitally 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 will also need `signcode.exe`, the Microsoft tool which enables you to integrate your key and certificate into the signature and to sign your installer.

If digital signatures are not already in use at your organization, you will need to download `signcode.exe` from 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` options at <http://msdn.microsoft.com>

# 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 through "tasks", which can either be part of the core Ant distribution or available as extensions.

InstallAnywhere provides an Ant task that enables you to build installers from Ant. The InstallAnywhere Ant task is located in your InstallAnywhere application directory:

```
<InstallAnywhere>/resource/build/iaant.jar
```

To integrate the InstallAnywhere Ant task in an Ant project, you must set the class path 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\SampleApp.iap_xml"
    additionalparameters="values..." >

    <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 in the previous section.

For a complete list of build parameters and other options, refer to the InstallAnywhere help topic “Ant Build Integration.”



# 13

## Custom Code

---

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, you can create your own custom components using the Custom Code API.

InstallAnywhere offers an open Application Programming Interface (API) which enables you to write Java code that can run within InstallAnywhere's architecture.

Using the API also provides access to additional functionality in InstallAnywhere, such as its unique variables and resource loading features. You 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:

**Table 13-1:** Types of Custom Code

Custom Code Type	Description
<b>Rules</b>	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.
<b>Actions</b>	These actions run within InstallAnywhere's action framework, alongside the default InstallAnywhere actions.
<b>Panels</b>	These panels run within InstallAnywhere's graphical interface during the installation process. You can use this mechanism to add panels to the Installer not provided in InstallAnywhere's default panels.
<b>Consoles</b>	These actions run within InstallAnywhere's console interface during the installation process. You can use this mechanism to add custom console elements to the Installer.

For additional information on Panels and Consoles, refer to Chapter 14.

InstallAnywhere ships with a collection of 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.

## Writing Custom Code

The steps you follow when using custom code are:

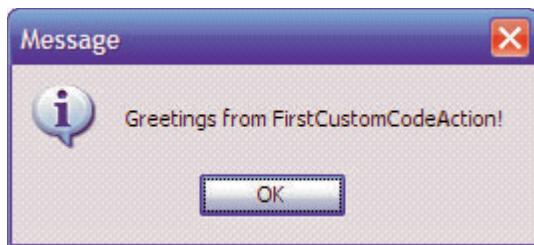
1. Write and compile your custom Java code.
2. Package the custom class and any required resources into a **.jar** file.
3. Execute the class in your InstallAnywhere project.

# 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). In fact, Custom Code Actions enable you to perform nearly any action possible with Java.

## Custom Code Action Example

Suppose you want your installation program to display an arbitrary message to the user in a simple dialog box, as shown in the following figure. The message box uses the `showMessageDialog` method of the Java Swing `JOptionPane` class.



**Figure 13-1:** Custom Code Message Box

This example is designed to illustrate the technique of creating a custom code action. The implementation does not yet handle console mode or silent mode correctly.

## Writing the Action Code

First, you must write the source for the custom code action class. For this example, you will write a class called `FirstCustomCodeAction`, with its source in a file called `FirstCustomCodeAction.java`. Custom code actions must extend the class `com.zerog.ia.api.pub.CustomCodeAction`, so the general outline of the `FirstCustomCodeAction` class is as follows:

```
import com.zerog.ia.api.pub.*;

public class FirstCustomCodeAction extends CustomCodeAction
{
    // ...
}
```

To integrate the custom code with the running installation, you must override the `install` method of the `CustomCodeAction` class. The `install` method has the following signature:

```
public void install(InstallerProxy ip) { ... }
```

Furthermore, to display a status message on the installer's progress bar while your action is taking place, you must implement the `getInstallStatusMessage` method, which returns the string message to display.

```
public String getInstallStatusMessage( )
{
    return "Installing FirstCustomCodeAction...";
}
```

There are similar methods for specifying the behavior and status message of the action during uninstallation. The signatures of the `uninstall` and `getUninstallStatusMessage` methods appear as follows:

```
public void uninstall(UninstallProxy up) { ... }

public String getUninstallStatusMessage( )
{
    return "Uninstalling FirstCustomCodeAction...";
}
```

At installation time, the InstallAnywhere framework checks each custom code action for an `install` method with that specific signature, and executes the code in it.

The completed source file `FirstCustomCodeAction.java` appears as follows.

```
import com.zerog.ia.api.pub.*;
import javax.swing.*;

public class FirstCustomCodeAction extends CustomCodeAction
{
    public void install(InstallerProxy ip)
    {
        JOptionPane.showMessageDialog(null,
            "Greetings from FirstCustomCodeAction!");
    }

    public void uninstall(UninstallerProxy up)
    {
        // do nothing during uninstallation
    }

    public String getInstallStatusMessage( )
    {
        return "Installing FirstCustomCodeAction...";
    }

    public String getUninstallStatusMessage( )
    {
        return "Uninstalling FirstCustomCodeAction...";
    }
}
```

## Compiling and Packaging the Action Code

To compile the custom code action, the InstallAnywhere archive *IAClasses.zip* must be on the compiler class path. A command to compile the class appears similar to the following:

```
javac -classpath "...IA\IAClasses.zip" FirstCustomCodeAction.java
```

You must then package the resulting .class file into a .jar or .zip archive. Using the *jar* command from the Java SDK, for example, you would run a command similar to the following:

```
jar cvf FirstCustomCodeAction.jar FirstCustomCodeAction.class
```

Once you have the code packaged into a .jar or .zip file, you can execute it using an Execute Custom Code action.

If your custom class is inside a package, you must place the class file in a subdirectory that matches the package name. For a custom class whose full name is **com.macrovision.training.AnotherCustomAction**, you would copy the class file into the subdirectory structure **com/macrovision/training**.

## Adding the Custom Action to Your Project

Next, you can specify to execute your custom code by adding an Execute Custom Code action to one of your project's tasks. For example, the following figure shows a new **Execute Custom Code** action in the **Install** task.

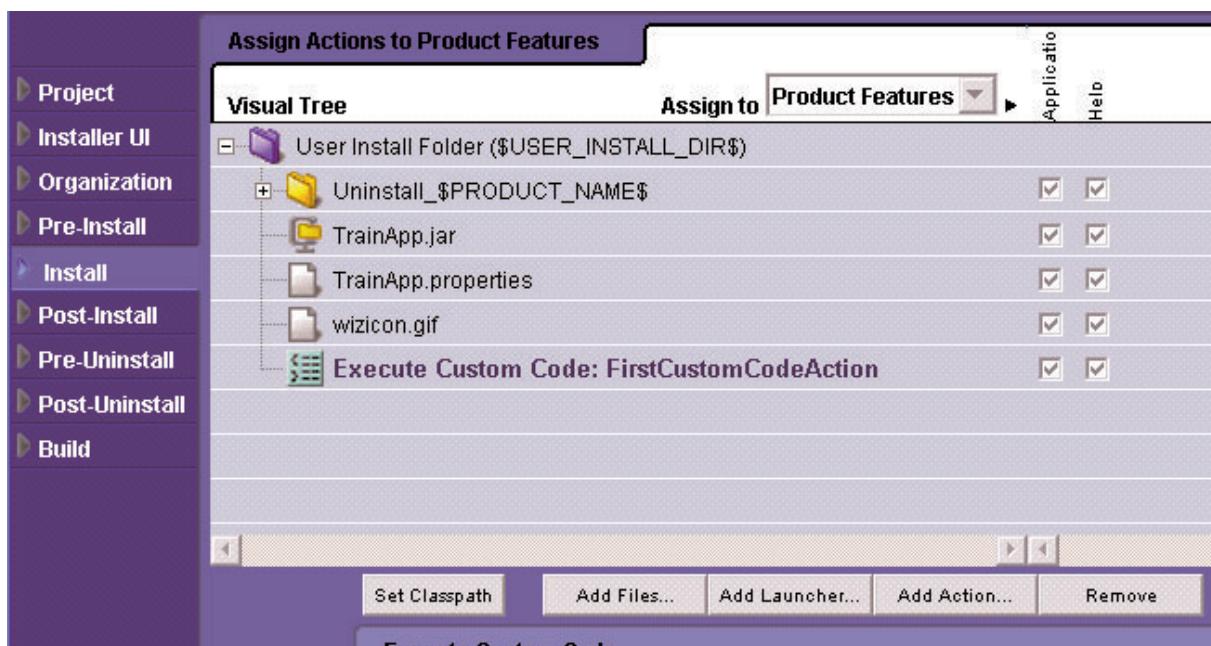
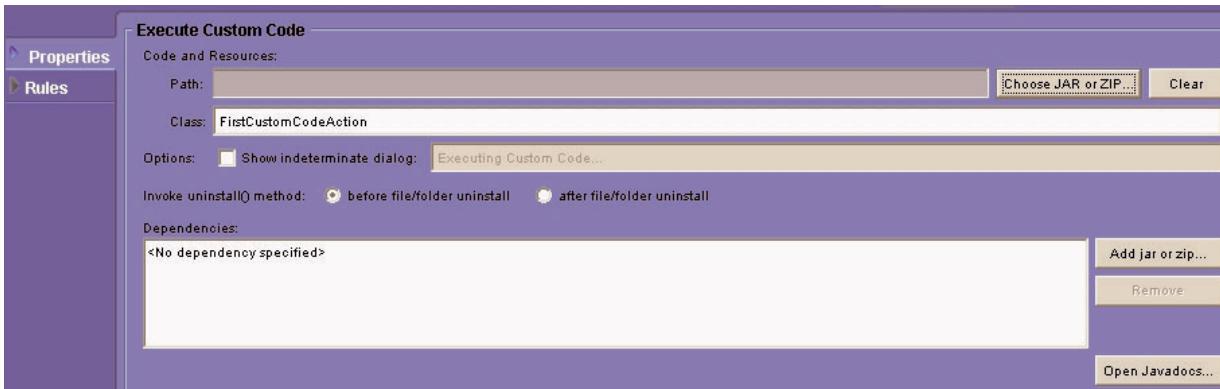


Figure 13-2: Execute Custom Code Action

In the customizer for the **Execute Custom Code** action, you browse for the .jar or .zip file you created earlier (which populates the **Path** setting), and then enter the full class name in the **Class** field.

If the custom class is part of a package, enter the fully qualified class name, such as `com.macrovision.training.ActionName`.

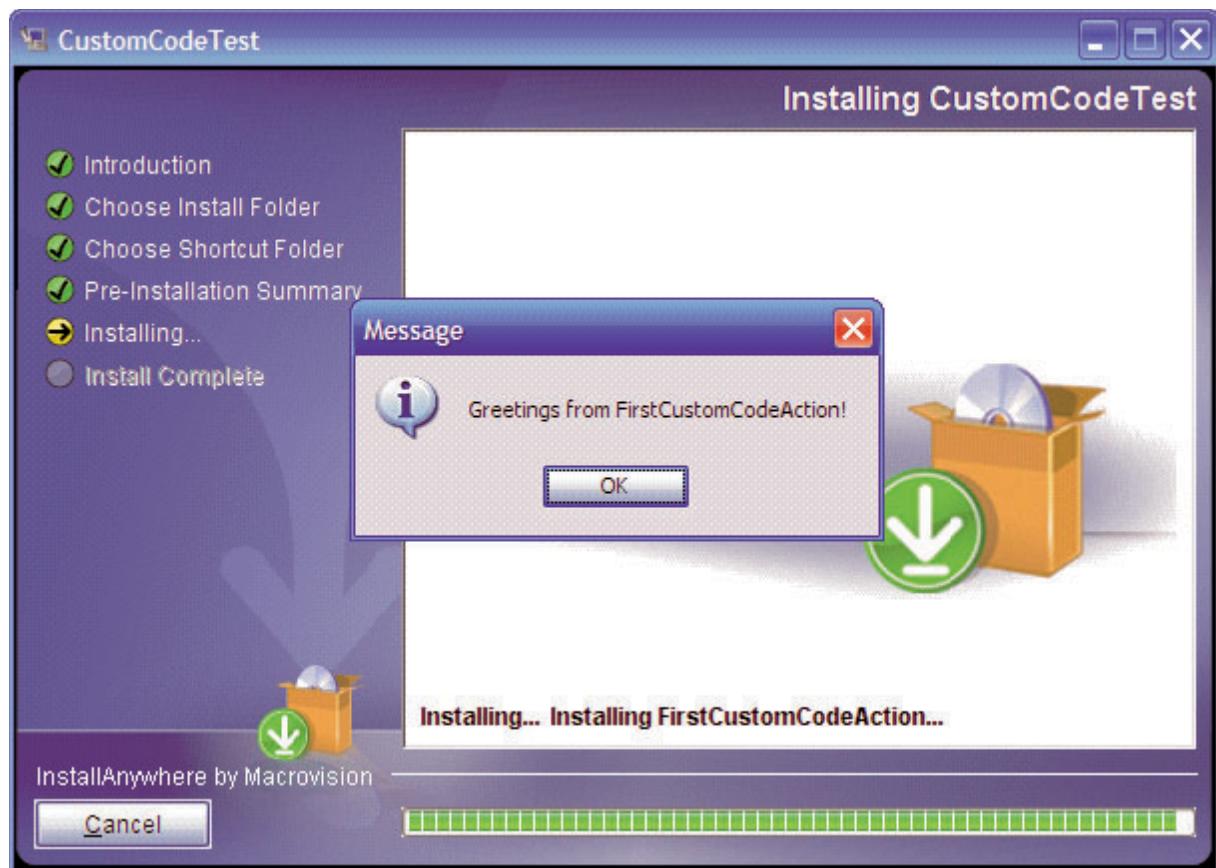


**Figure 13-3:** Execute Custom Code Resources

You can optionally specify whether to show the “indeterminate” progress dialog, which shows a constantly moving progress bar; and specify whether to call the `uninstall` method of your action before or after files and folders are removed. Moreover, if the action contains any dependencies, you can browse for the .jar or .zip files containing the dependent classes.

Do not specify `IAClasses.zip` as a dependency.

Finally, when you build and run your project, the message box from the custom class is displayed while the **Install** task is taking place.



**Figure 13-4:** FirstCustomCodeAction at Run-time



**Note:** The text from the **getInstallStatusMessage** method is displayed in the background window.

## Packaging Custom Code as a Plug-in

In addition to executing custom code with the Execute Custom Code action, you can package frequently used custom classes as *plug-ins*, which enable you and other developers to insert your code into a project without needing to specify the class name or archive.

To package a custom class as a plug-in, you must create a properties file that describes the plug-in, and then copy it into the root level of your .jar file. The properties file should be named `customCode.properties`, and must contain the following entries:

- **plugin.main.class=classname**—The `classname` value should be the full name of your action class, such as `FirstCustomCodeAction` or `com.macrovision.training.MyAction`.
- **plugin.name=DisplayName**—The `DisplayName` value should be the human-readable display name for your action, to be displayed in the Insert Action panel.
- **plugin.type=action or panel or console**—This value should be set to the code's type.

Additional optional entries include:

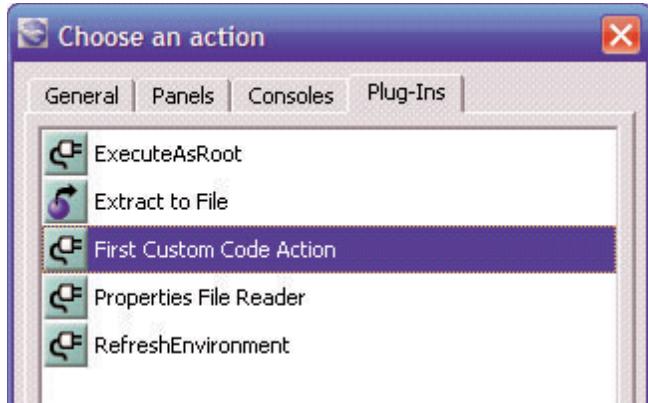
- **plugin.icon.path=icon-path**—This value should be set to the relative path within the .jar file to a 32-by-32-pixel .png or .jpg file to be displayed as the icon for the plug-in.
- **property.propertyname=propertydefault**—This value specifies a property you can set in the customizer used by the plug-in.
- **plugin.available=preinstall | install | postinstall | preuninstall | postuninstall**—This value should be a comma-separated list of tasks in which the plug-in can be placed.

For this example, a sample `customCode.properties` file you would place in `FirstCustomCodeAction.jar` would appear similar to the following:

```
plugin.main.class=FirstCustomCodeAction
plugin.name=First Custom Code Action
plugin.type=action
```

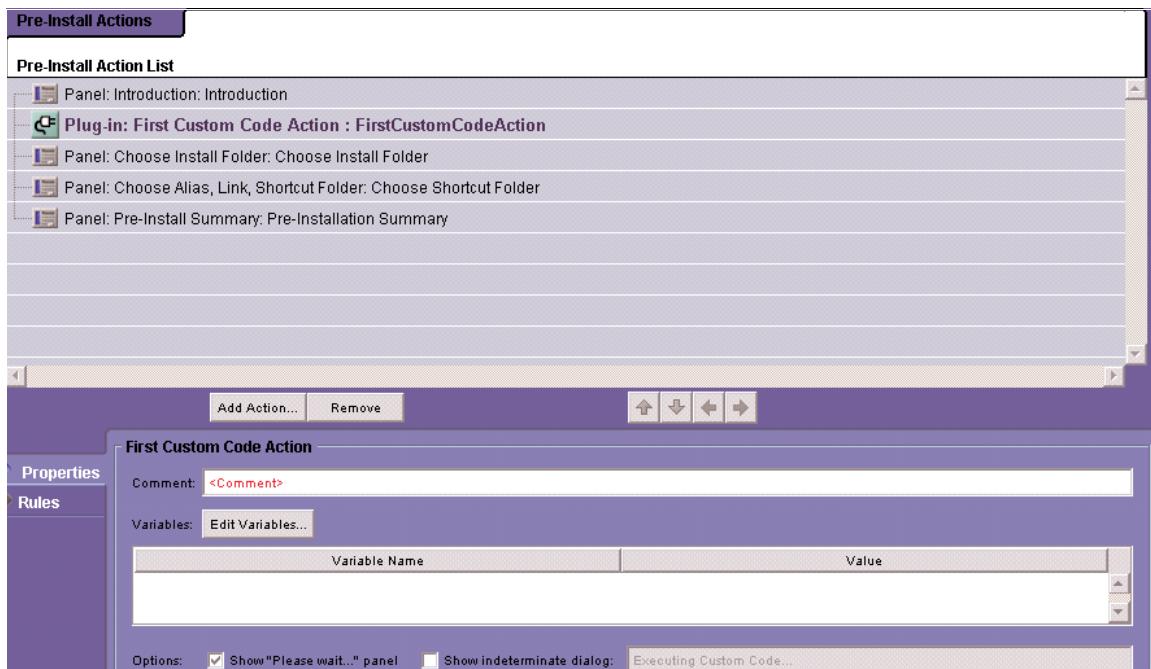
Finally, you should place the .jar file in the `plugins` subdirectory of the InstallAnywhere installation.

After placing the .jar file in the `plugins` directory and restarting InstallAnywhere, you can insert the custom code action using the same **Add Action** functionality you use for built-in actions; the **Plug-Ins** tab of the **Choose an action** panel lists **First Custom Code Action** (along with other sample actions that ship with InstallAnywhere):



**Figure 13-5:** Inserting a Custom Code Action as a Plug-in

The action then appears in the selected task, and the customizer for the action exposes options for editing variable values and displaying status information.



**Figure 13-6:** Custom Code Properties

## Adding Action Help

To provide development-time help for your action, you can add a file called `help.html` to the root of your `.jar` file. If a file with that name is present, a button labeled **Help With Plug-in** is displayed at the bottom of the action's customizer; and if a developer clicks that button, your help text from `help.html` is displayed.

## 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 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 will 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.

# Accessing Properties and Variables

Instead of hard-coding data used by your custom code actions, it is useful to have action parameters that can be specified at design time. For example, in the previous **FirstCustomCodeAction** example, it may be desirable to allow setting the display-message using a parameter in the action's customizer.

To enable a property called message to contain the message displayed to the user at run time, you can add the following line to the `customCode.properties` file inside the custom .jar file.

```
property.message=Your message here...
```

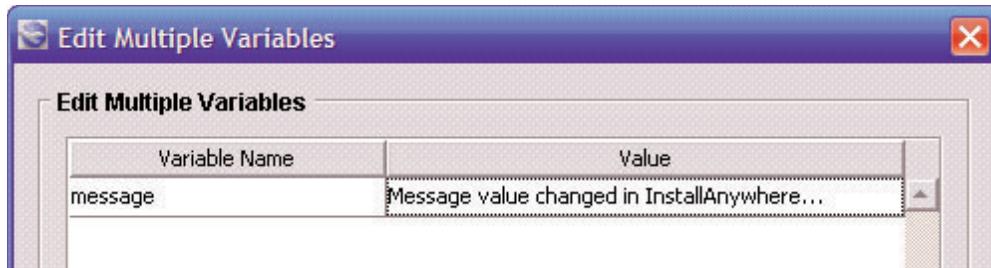
Inside the code, you can use the `substitute` method of the `InstallerProxy` object passed to the `install` method:

```
public void install(InstallerProxy ip)
{
    JOptionPane.showMessageDialog(null, ip.substitute("$smessage$"));
}
```

The argument to the `substitute` method is a string containing embedded expressions of the form `$property$`, and the return value is a string with the embedded expressions expanded to the values of the specified InstallAnywhere properties.

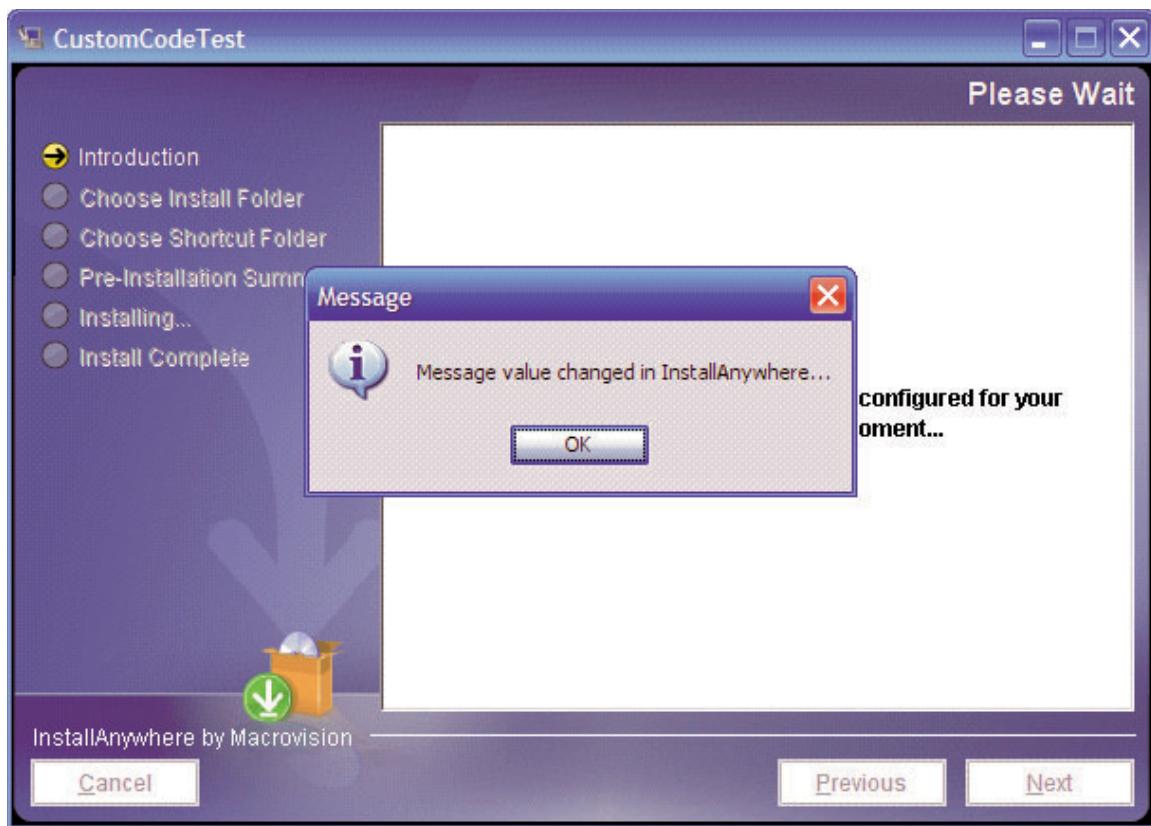
At design time, the presence of the `property.message` entry in your `customCode.properties` file causes the `message` variable and its default value to appear in the variables list.

The developer-user of your action can double-click the variable's value to change it.



**Figure 13-7:** Editing Variables Used in Custom Code

After changing the value and clicking **OK**, you can build the project and execute the installer. At run time, the message is the changed value:



**Figure 13-8:** Run-time Message Box

## Custom Variables and Response Files

InstallAnywhere variables used in many standard actions are automatically written to a response file (as described in Chapter 10) or installer log. Beginning with InstallAnywhere 2008 Value Pack 1, you can use the `ReplayVariableService` to control if and how variables used in your custom code classes are included in a response file. The `ReplayVariableService` class contains a `register` method, which accepts a variable name (or array of names) and title, where the title is a section name listed in the response file.

For example, including the following code (in the `install` method of a custom code action, for example) registers a custom variable called `$MYVAR$` to be written the response file when the user runs `install -r`:

```
ReplayVariableService rvs =  
    (ReplayVariableService)ip.getService(ReplayVariableService.class);  
rvs.register("$MYVAR$", "Custom variables");
```

When the user generates a response file with `install -r`, the corresponding section of the response file appears as follows, with the text "Custom variables" being taken from the second argument to `register`:

```
#Custom variables  
#-----  
MYVAR=User changed this value...
```

In addition, the `ReplayVariableService` contains methods for indicating a variable's value should be excluded or encrypted in the response file. For more information, refer to the javadoc documentation.

## Variables and Proxy Classes

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 methods called `substitute` and `getVariable`. Both of these methods can be used to access InstallAnywhere variables from within your custom code.

### substitute

This method fully resolves a String that may contain embedded InstallAnywhere variables.

```
public String substitute(String var)
```

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

This method returns the literal Object represented by an InstallAnywhere variable.

```
public java.lang.Object getVariable(String var)
```

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.



**Note:** Starting with InstallAnywhere 2008 Value Pack 1, you can obtain a Java Enumeration of all InstallAnywhere variable names using the `getVariables` method.

## Example: Displaying the Current Time

Suppose you want to create a custom code action that displays the current time in a simple message dialog. For this example, you will create a custom code action that uses the Java class `java.util.Date` to obtain the current date and time.

Code for the action might appear as follows, in a source file called `DisplayTime.java`:

```
import com.zerog.ia.api.pub.*;
import java.util.*;

public class DisplayTime extends CustomCodeAction
{
    public void install(InstallerProxy ip)
    {
        javax.swing.JOptionPane.showMessageDialog(null,
            "The current time is:\n\n" +
            new Date().toString());
    }

    public String getInstallStatusMessage()
    {
        return "Getting current time...";
    }

    public void uninstall(UninstallerProxy up) { }
    public String getUninstallStatusMessage() { return "..."; }
}
```

In preparation for packaging the code as an InstallAnywhere plug-in, create a `customcode.properties` file and place the following lines in it:

```
plugin.main.class=DisplayTime
plugin.name=Display Current Time
plugin.type=action
plugin.available=preinstall
```

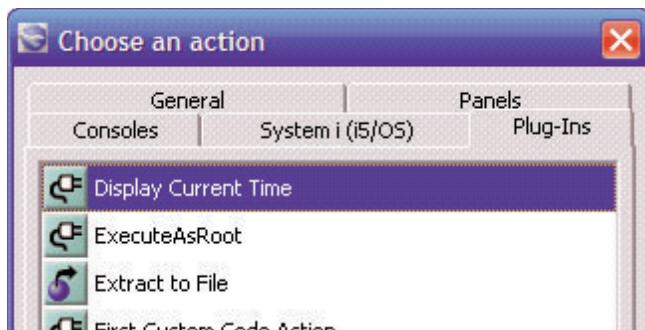
To compile your action, run a command similar to the following:

```
javac -classpath "...IA\IAClasses.zip" DisplayTime.java
```

To package the class into a .jar file, run the following command:

```
jar cvf DisplayTime.jar DisplayTime.class customcode.properties
```

The presence of `customcode.properties` in the .jar file makes the custom action a plug-in, so you can copy `DisplayTime.jar` into the `plugins` directory of the InstallAnywhere distribution, and it will be available in the **Plug-Ins** tab of the **Choose an action** dialog the next time you start InstallAnywhere.



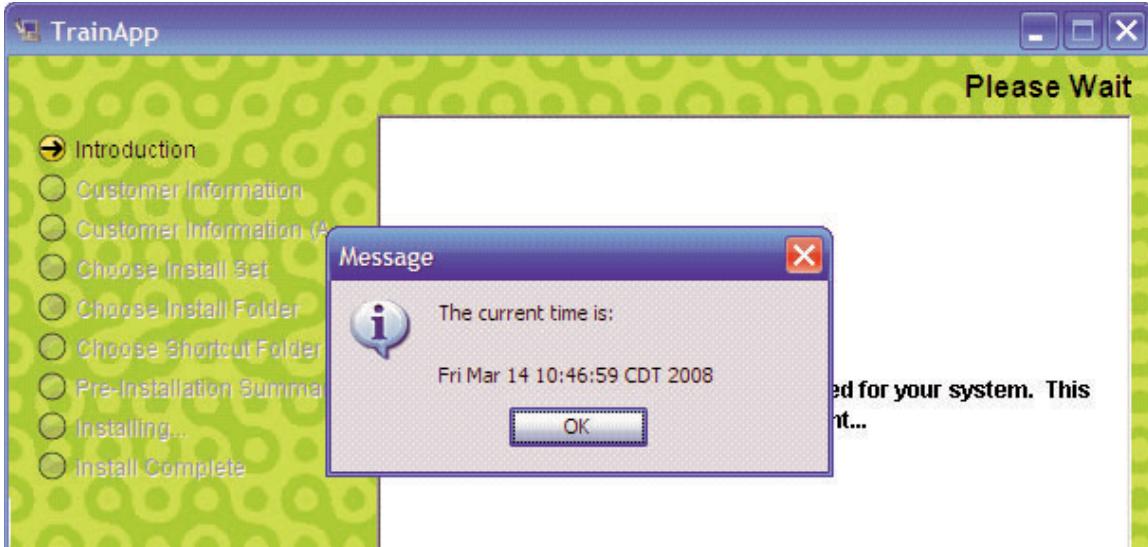
**Figure 13-9:** Adding the DisplayTime Plug-in

When you click **Add**, the Display Current Time action appears in the **Pre-Install** task as follows.



**Figure 13-10:** DisplayTime as a Plug-in

At run time, the action appears similar to the following:



**Figure 13-11:** DisplayTime at Run-time

## Custom Code Rules

The process of creating a custom rule is similar to creating a custom action. To create a rule, you create a custom class that extends `com.zerog.ia.api.pub.CustomCodeRule`, and this class must implement an `evaluateRule` method that returns true if the rule succeeds and false if the rule fails.

For example, the implementation of a rule that always succeeds would appear similar to the following:

```
import com.zerog.ia.api.pub.*;

public class AlwaysSucceedsRule extends CustomCodeRule
{
    public boolean evaluateRule()
    {
        return true; // always succeed
    }
}
```

You compile the rule class the same way you compile other custom code (by including `IAClasses.zip` in the compiler classpath), and package the `.class` file in a `.jar` file or `.zip` file as before.

To use the rule, you select the **Rules** tab in the customizer for the action to which you want to attach the rule, click the **Add Rule** button, and then select **Evaluate Custom Rule**.

In the customizer for the custom rule, you select the `.jar` or `.zip` file containing the custom class, and then specify the name of the Java class that implements the rule.

At run time, if the rule succeeds, the action associated with it will be installed or performed, and if the rule fails the action will be skipped.

## 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 does not explain how to make your code work with InstallAnywhere's Advanced Designer.



**Note:** This exercise assumes you know the goals of your rule but have not yet written any code to implement it. After each relevant step, an example shows 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. The `public abstract boolean evaluateRule()` 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;  
}  
}
```

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`. Refer to the Javadoc for `CustomCodeRule` for more information.

## InstallShield MultiPlatform Services

In addition to being able to call standard Java code in your custom actions, you can call methods defined inside ISMP (InstallShield MultiPlatform) services. The ISMP services are collections of functions for performing common installation-related tasks, implemented as a combination of Java code and native code. Installer services are unrelated to Windows services, which are a special type of executable on Windows platforms.

New in InstallAnywhere 8, the following ISMP services are available:

- **FileService**: manipulates files and directories and their privileges.
- **SecurityService**: manipulates system users and groups.
- **SystemUtilService**: works with environment variables, reboots, and startup commands.
- **Win32RegistryService**: manipulates the Windows registry.
- **Win32Service**: manipulates Windows services.

To work with installer services, you obtain a service handle in code by calling the `getService` method, similar to the following:

```
// ip is the InstallerProxy object passed to the install method  
// of a custom code action  
XxxService xxxService = (XxxService)ip.getService(XxxService.class);
```

To obtain a service handle in a custom code rule class, you must use the static `ruleProxy` member of the `CustomCodeRule` class:

```
XxxService xxxService = ruleProxy.getService(XxxService.class);
```

The ISMP classes are contained in various packages in `com.installshield.wizard.service` and `com.installshield.wizard.platform`; specific packages are listed in the Javadoc API documentation. Javadoc documentation for the ISMP services is included within the InstallAnywhere API documentation.

## Compiling and Packaging Code That Uses ISMP Services

To compile code that uses the ISMP services, you must include `resource\services\services.jar` on the compiler class path:

```
javac -classpath "...IA\IAClasses.zip";"IA\resource\services\services.jar"  
      ActionName.java
```

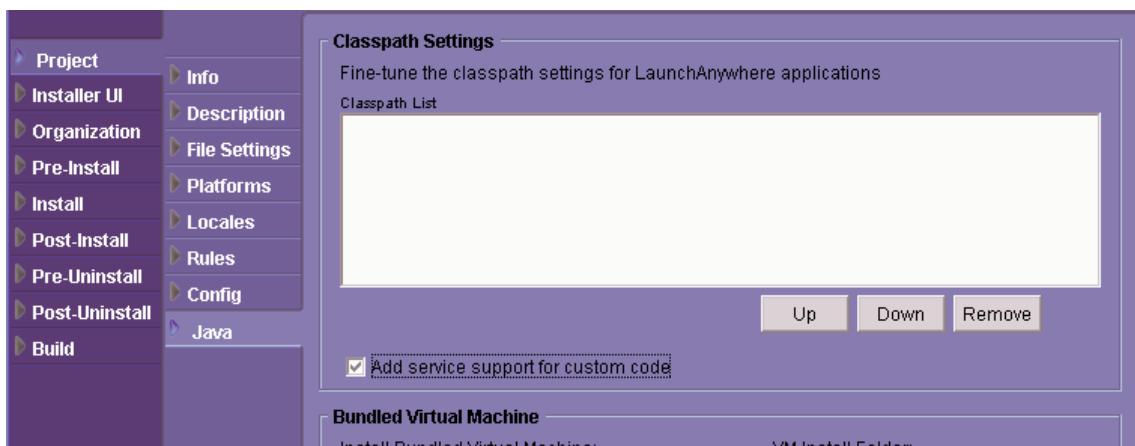
As with other actions, you must package your custom code that uses ISMP services into a `.jar` or `.zip` file:

```
jar cvf ActionName.jar ActionName.class
```

## Adding ISMP Code to Your Project

Adding the code to your project is similar to previous examples: you can execute the class using an Execute Custom Code action, or package the custom code as a plug-in.

To ensure the ISMP service classes are available to your installer at run time, you must indicate to include the classes in your project. In the **Java** task of your project, select the **Add Service Support for Custom Code** check box beneath the **Classpath List** entries.



**Figure 13-12:** Adding Service Support for Custom Code

## File Service

The file service enables you to manipulate and query files, directories, and partitions at run time. An advantage to using file-service methods instead of `java.io` methods is that the ISMP file service handles platform-specific behavior, such as setting file and directory owner and group information; and handles additional installation-related behavior, such as renaming a locked executable file after a reboot on Windows target systems.

Important methods of the file service include the following:

- `createDirectory`, `deleteDirectory`, `copyDirectory`, `isDirectory`, `isDirectoryWritable`
- `createAsciiFile`, `createBinaryFile`, `deleteFile`, `copyFile`, `moveFile`, `fileExists`
- `setFileCreated`, `setFileModified`
- `setFileAttributes`, `getFileAttributes`
- `getPartitionNames`, `getPartitionType`, `getPartitionFormat`, `getPartitionFreeSpace`,  
`supportsLongFileNames`

For example, the following class obtains the partition type of the `C:` drive on a Windows system:

```
import com.zerog.ia.api.pub.*;
import com.installshield.wizard.service.*;
import com.installshield.wizard.service.file.*;
import javax.swing.*;

public class FirstServiceAction extends CustomCodeAction
{
    public void install(InstallerProxy ip)
    {
        try {
            FileService fs =
                (FileService)ip.getService(FileService.class);

            JOptionPane.showMessageDialog(null,
                "C: format is: " + fs.getPartitionFormat("C:\\\"));

        } catch (ServiceException se) { System.out.println(se.getMessage()); }

        public void uninstall(UninstallerProxy up) { /* do nothing... */ }

        public String getInstallStatusMessage() { return "Installing FirstServiceAction..."; }

        public String getUninstallStatusMessage() { return "Uninstalling FirstServiceAction..."; }
    }
}
```

If an operation performed by a service fails, it throws a `ServiceException`. A `ServiceException` object provides `getMessage`, `getErrorCode`, and `getSeverity` methods for providing more information about the exception; the Javadoc documentation describes the various error codes.

## Security Service

The security service provides the following APIs for working with users and groups on the target system:

- **isCurrentUserAdmin**
- **createUser, deleteUser**
- **createGroup, deleteGroup**

For example, a custom rule that determines if the user running the setup program has administrative (root) privileges might appear as follows. Note that a custom rule must use the `ruleProxy` member of the class in order to use services, similar to the technique for working with variables in a custom rule.

```
import com.zerog.ia.api.pub.*;
import com.installshield.wizard.service.*;
import com.installshield.wizard.service.security.*;

public class AdminRule extends CustomCodeRule
{
    public boolean evaluateRule()
    {
        boolean isAdmin = false;

        try
        {
            SecurityService secService =
                (SecurityService)ruleProxy.getService(SecurityService.class);

            isAdmin = secService.isCurrentUserAdmin();
        }
        catch(ServiceException e) { e.printStackTrace(); }

        return isAdmin;
    }
}
```

## System Utility Service

The system utility service provides APIs for working with environment variables, reboots, startup commands, and operating system properties.

- **getEnvironmentVariable, setEnvironmentVariable, appendEnvironmentVariable, prependEnvironmentVariable, deleteEnvironmentVariable**
- **isRebootRequired, setRebootRequired, getRebootOnExit, setRebootOnExit**
- **deleteFileOnExit, deleteDirectoryOnExit**
- **addSystemStartupCommand, removeSystemStartupCommand**
- **getOSProperties, getOSServiceLevel**

For example, you can get the value of an environment variable using the system utility service as follows:

```
import com.zerog.ia.api.pub.*;
import com.installshield.wizard.service.*;
import com.installshield.wizard.service.system.*;
import javax.swing.*;

public class GetEnvVar extends CustomCodeAction
{
    public void install(InstallerProxy ip)
    {
        try {
            SystemUtilService ss =
                (SystemUtilService)ip.getService(SystemUtilService.class);

            JOptionPane.showMessageDialog(null,
                "PATH is: " + ss.getEnvironmentVariable("PATH"));

        } catch (ServiceException se) { se.printStackTrace(); }
    }

    // ...omitting uninstall and status-message methods...
}
```

After compiling and packaging the action class, you can execute it with an **Execute Custom Code** action or package it as a plug-in and insert it into your project.

## Advanced Topic: Differences Between Services in ISMP and IA

The ISMP services are provided to assist with migration of code from an InstallShield MultiPlatform project to an InstallAnywhere project. Naturally, there are some differences in service behavior and requirements.

- Not all of the ISMP services are available in InstallAnywhere: the JVM service, log service, and exit code service are unavailable. In some cases, InstallAnywhere provides equivalent functionality to unsupported services. For example, the `CustomError` class enables you to write additional information to the installation log, where ISMP used the log service to achieve the same effect.
- The `getService` method, which obtains a handle to a service, takes the class object (such as `FileService.class`), and not the string class name (`FileService.NAME`).
- It is not necessary to implement the `build` method and call `putRequiredService`; checking the **Add Service Support for Custom Code** check box ensures classes are included with the installer.

In addition, InstallAnywhere supplies additional services that provide access to specialized functionality, and which have no direct equivalent in ISMP. Examples are the `GUIAccess` class, which provides access to the appearance and behavior of navigation buttons on custom code panels (as described in Chapter 14); the `InstallerResources` class, which provides access to some resources and data inside an installer; the `ReplayVariableService`, which enables you to control if and how variables used in custom code are written to log and response files; and `CustomError`, which provides access to InstallAnywhere's logging functionality.

## Advanced Topic: Querying the ISMP Registry

Similar to InstallAnywhere's product registry is the InstallShield MultiPlatform product registry, which keeps track of the ISMP products, features, and components installed on a target system. To enable you to query the ISMP information—to determine if a product installed with ISMP is present or where one of its components is installed, for example—you can use the `InstallShieldUniversalRegistry` or `InstallShieldUniversal10AndOlderRegistry` service.

You can also query the ISMP registry using the built-in **Query InstallShield Universal Software Information** action.

You obtain a handle to the service using the same `getService` method of the appropriate proxy class. A sample class that reports the locations of any instances of an ISMP software object—a product, feature, or component—might appear as follows.

```
import com.zerog.ia.api.pub.*;  
  
public class QueryIsmpRegistry extends CustomCodeAction  
{  
    public void install(InstallerProxy ip)  
    {  
        InstallShieldUniversalRegistry ir =  
  
(InstallShieldUniversalRegistry)ip.getService(InstallShieldUniversalRegistry.class);  
  
        InstallShieldUniversalSoftwareObject sos[] =  
            ir.getSoftwareObjects("20a4c1921222b29b2ef50a88ce1d0623");  
  
        for (int i = 0; i < sos.length; i++)  
        {  
            javax.swing.JOptionPane.showMessageDialog(null,  
                "An instance of this product is installed at: " +  
                sos[i].getInstallLocation());  
        }  
    }  
  
    public void uninstall(UninstallerProxy up) { /* nothing to do for uninstall */ }  
  
    public String getInstallStatusMessage() { return "Querying registry..."; }  
    public String getUninstallStatusMessage() { return ""; }  
}
```

If you use the `InstallShieldUniversalRegistry` service in your code, you must add `hsqldb.jar` as a dependency. The file `hsqldb.jar` can be found in the `resource\dbclients` subdirectory of your `InstallAnywhere` distribution directory.



Figure 13-13: Adding `hsqldb.jar` as a Dependency



**Note:** To remove existing software that was installed with ISMP, you can use the built-in `Uninstall InstallShield Universal Software` action.

For additional information about the ISMP services, refer to the Javadoc documentation.

## 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.

Refer to Chapter 8 for more information on working with debug mode.



**Note:** 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.



**Note:** When developing and testing your custom code, it is recommended you place frequent `System.out` and `System.err` statements within your code. Doing this enables you to easily identify possible problems within the code.

# Advanced Action Methods

Additional advanced methods available to an action are provided by the `InstallerResources` class. To call these methods from an action, use code similar to the following:

```
InstallerResources ir = (InstallerResources)proxy.getService(InstallerResources.class);  
ir.methodName(...);
```

The following are the methods provided by the `InstallerResources` class. For more information, refer to the Javadoc documentation.

```
public long getAvailableDiskSpace()
```

A convenience method to return 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 bundles (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`.

# Writing 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, the following code enables you to write information to the installation log.

```
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 logs the following text in the installation log.

```
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
```

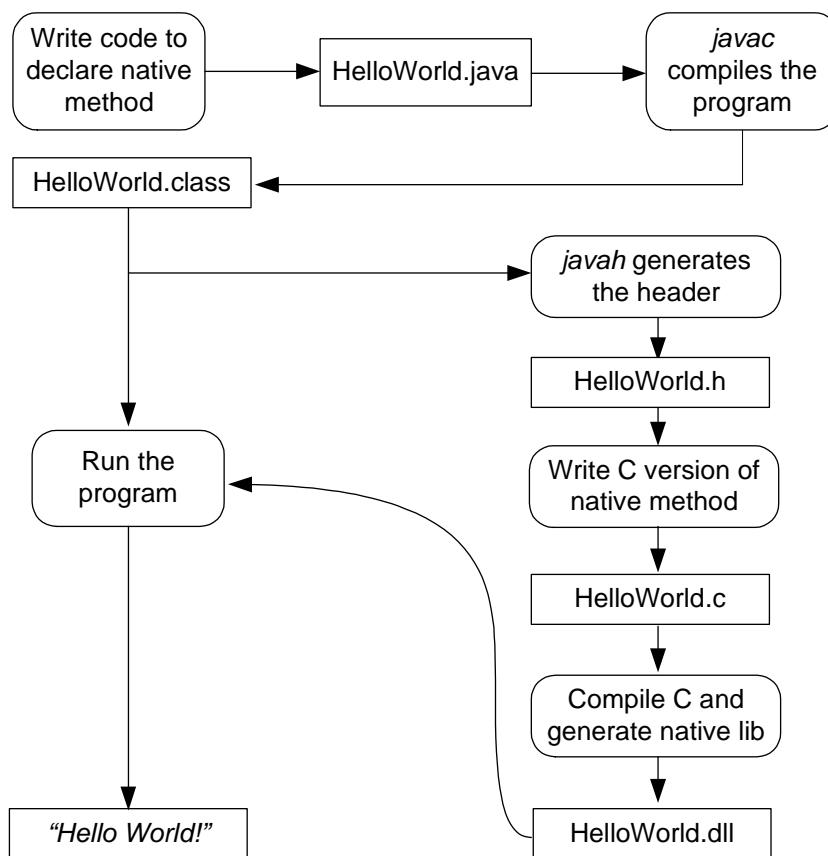
# Working with Java Native Interface (JNI)

From time to time, you will encounter legacy or existing code that is not written in Java.

This code may have been extensively tested and integrated with existing applications, or it may be serving a very specialized purpose (such as a statistical package) that would involve too much overhead to rewrite. Using the Java Native Interface, you can access these routines and avoid writing, testing, and implementing completely new packages.

For platform-specific behavior not covered in the product or Java APIs, you can write actions that call native libraries inside an installation using the Java Native Interface (JNI). Calling JNI code involves the following steps:

1. Write and compile the Java class.
2. Use `javah` to generate a native-code header file. This is the C header and stub generator used to write native methods.
3. Copy the function signature from the header file into the C/C++ source file for the native library.
4. Compile and build the native library.



**Figure 13-14:** JNI Process Diagram

The following examples guide you through some of the basics of using the Java Native Interface. Additional examples of JNI can be found in the `\CustomCode\Samples\RefreshEnvironment\native\` directory.

## JNI Example 1: Basic Text Display

This first example provides standalone code to introduce JNI. In the following example, you will see how JNI is called from an installer.

### Write and Compile the Java Class

The first step is to write your Java code. Inside the code, you will write a native method declaration for each method used. Specifically, this involves calling the `native` keyword.

```
public native void printText();
```

Additionally, load the native code library in a class static block using `System.loadLibrary("iaexample");`

Next, create the file `IAExample.java` with the code below:

```
class IAExample {  
    public native void printText();  
    static  
    {  
        System.loadLibrary("iaexample");  
    }  
    public static void main(String[] args)  
    {  
        IAExample iaexample = new IAExample();  
        iaexample.printText();  
    }  
}
```

Compile `IAExample.java` with `javac IAExample.java` at the command line.

### Use `javah` to Create a Header

The Java compiler uses `javah` to generate declarations from the `IAExample` class. From the command line, enter:

```
javah -jni IAExample
```

## Creating the Native Method

The next step is to copy the function signature from the header file into the C or C++ source file for the native library.

```
#include <jni.h>
#include <stdio.h>
#include "IAExample.h"

JNIEXPORT void JNICALL
Java_IAExample_iaexample(JNIEnv *env, jobject obj)
{
    printf("InstallAnywhere Training Example.\n");
    return;
}
```

In the bold above, notice the `ClassName_MethodName` that you created in the previous steps.

Your C or C++ code is then compiled and the native library is built; this creates the .dll file you will call. Instructions on how to compile and build the native library will be specific for your compiler and operating system. For additional information, refer to your compiler's documentation.

## Running the Program

Now you should be able to run the program and have your class file call a native method. To run the program from a command line, enter:

```
java IAExample
```

## JNI Example 2: Calling a Windows DLL from a Custom Code Action

The following example is a Java class called `NativeMessageBox` that calls a native function in a Windows dynamic-link library (DLL).

Below is partial source for the `NativeMessageBox` class. As with other custom actions, `NativeMessageBox` extends `CustomCodeAction` and implements the `install` and other required methods.

```
import com.zerog.ia.api.pub.*;
public class NativeMessageBox extends CustomCodeAction
{
    // method to be called from native library
    public native void displayMessageBox( );

    public void install(InstallerProxy ip)
    {
        /* stub to be filled in... */
    }

    public String getInstallStatusMessage( )
    {
        return "Calling NativeMessageBox...";
    }

    // unused during uninstallation
    public void uninstall(UninstallerProxy up) { }
    public String getUninstallStatusMessage( ) { return "(unused)"; }
}
```

After you have built the `NativeMessageBox` class, you generate a C header file that connects the Java class with your native C code using the Java SDK tool `javadoc`. From the command line, enter:

```
javadoc -classpath .;"...IA\IAClasses.zip" NativeMessageBox
```

to generate the C header file `NativeMessageBox.h`:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class NativeMessageBox */

#ifndef _Included_NativeMessageBox
#define _Included_NativeMessageBox
#ifdef __cplusplus
extern "C" {
#endif
```

```

/*
 * Class:      NativeMessageBox
 * Method:    displayMessageBox
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_NativeMessageBox_displayMessageBox(
    JNIEnv *, jobject);

#ifndef __cplusplus
}
#endif
#endif

```

In your C DLL code, you should use the function signature from the header file generated with *javah*, printed above in bold.

```

#include <windows.h>
#include <jni.h>
#include "NativeMessageBox.h"
// signature copied from NativeMessageBox.h
JNIEXPORT void JNICALL Java_NativeMessageBox_displayMessageBox(
    JNIEnv *env, jobject obj)
{
    MessageBox(GetForegroundWindow(),
               TEXT("Hi, I'm a native MessageBox!"),
               TEXT("Windows Native Code"),
               MB_OK | MB_ICONINFORMATION);
}

```

On Windows, the `TEXT` macro used in the code above treats strings as Unicode if the symbol `_UNICODE` is defined.



**Note:** Depending on your Windows development environment, it might also be necessary to suppress C++ name decoration in the DLL project, in order to ensure the function name exported from the DLL is the expected one. With Microsoft Visual C++, for example, you can add a text file called `NativeMessageBox.def` to your project, and give it the following contents:

```

LIBRARY NativeMessageBox
EXPORTS
    Java_NativeMessageBox_displayMessageBox

```

## Building the Distribution Archive

Naturally, the DLL must be available at run time on a target system in order for the installer to call the native method. When you package the custom class into a .jar file, you must include the DLL in the .jar file as well, using the appropriate software or a command-line tool similar to the following:

```
jar cvf NativeMessageBox.jar NativeMessageBox.class NativeMessageBox.dll
```

## Accessing the Library at Run Time

To extract the DLL to a temporary location at run time, you can modify the `install` method of the action to use the `getResource` method of the `InstallerProxy` object to get the location of the DLL inside the installer archive, and use the `saveURLContentToFile` method to save the DLL to a temporary location. You can then pass this temporary location to the `System.load` method, followed by invoking the native method.

The `install` method of the action now appears as follows:

```
public void install(InstallerProxy ip)
{
    try {

        File dll =
            ip.saveURLContentToFile(
                ip.getResource("NativeMessageBox.dll"));

        System.load(dll.getAbsolutePath());
        displayMessageBox();

    } catch (IOException ioe) {
        javax.swing.JOptionPane.showMessageDialog(
            null, "Couldn't find the DLL!");
    }
}
```

After recompiling the code and rebuilding the .jar file, you can add the action to one of your installation tasks. At run time, the native message box from the DLL appears as follows.

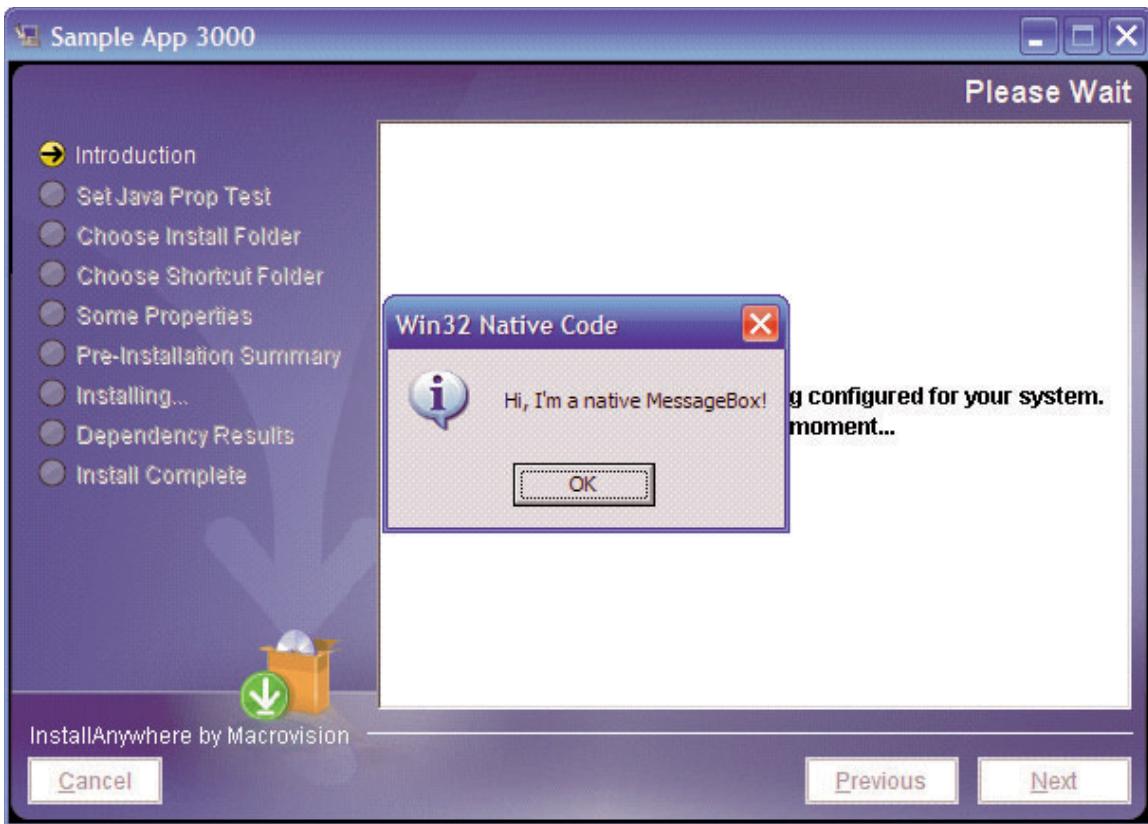


Figure 13-15: Executing Native Code at Run Time

## Potential Problems Using JNI

Using the native interface gives you access to your existing codebase, which can be a significant benefit when development and testing-time are factored in to a potential release date. However, using JNI is not without drawbacks that should be taken into consideration. Specifically, portability and security.

One of the factors that makes Java an attractive language to work with is the concept of “write once, run anywhere” cross-platform portability. This means that development can be done on any device, on any operating system, and compiled into a common bytecode that runs anywhere a Java Virtual Machine (JVM) is available.

However, when native code is used, it is usually contained in a DLL, similar to `refresh.dll` included in the InstallAnywhere installation path `\CustomCode\Samples\RefreshEnvironment\native\`. This introduces the issue of having non-portable, platform-specific code that must exist on the machine using the application.

Native code functions outside of the Java-based execution environment, so you must be aware of any potential security threats. Since a DLL can access a larger part of the system with machine-code, there is a greater possibility for viruses and security-holes to enter your solution. Be aware of the DLLs you are calling with JNI, and compare the advantages to the disadvantage prior to implementing it on a large scale—does the convenience of calling existing code outweigh security risks and lack of portability?

## Additional Resources

For more information on JNI, refer to:

- The JNI Programmer's Guide and Specification: <http://java.sun.com/docs/books/jni/>
- Sun's JNI Resource Page: <http://java.sun.com/javase/6/docs/technotes/guides/jni/index.html>

## 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

# 14

## Custom Panels and Consoles

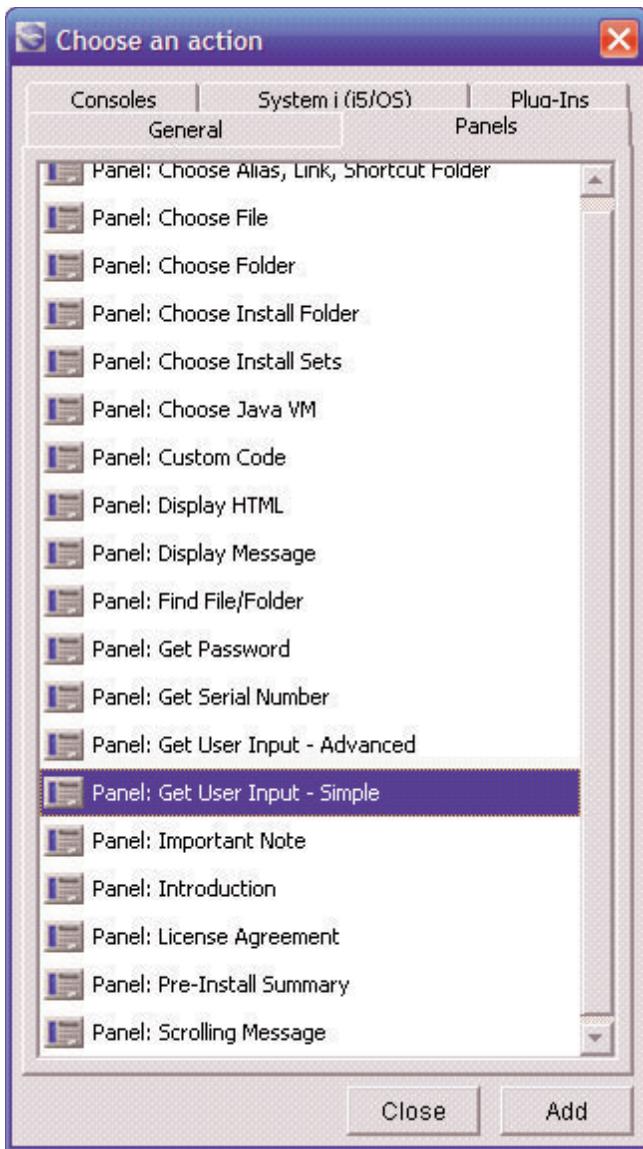
---

As described earlier, InstallAnywhere provides many standard panels and consoles that can be displayed to the end user, and you can customize many aspects of their appearance, such as bitmap, titles, display-information, and user input.

However, there may be times when you find that InstallAnywhere's standard panels and consoles do not meet your needs for appearance or behavior. This chapter describes how to use InstallAnywhere's **Get User Input** panels to design your own customized panels, and how to create custom panels and consoles using custom code.

### User Input Panels

To save you the effort involved in creating a custom code panel, InstallAnywhere provides Get User Input panels. Get User Input panels enable you to construct a panel by specifying the types of control to display to the end user, along with the variables in which to store the user input. At run time, the panel prompts the user for input, storing the user input in the variables you defined in the project. To add a **Get User Input** panel to your project, click **Add Action** in the desired task and select one of the Get User Input panels from the **Choose an Action** dialog box.



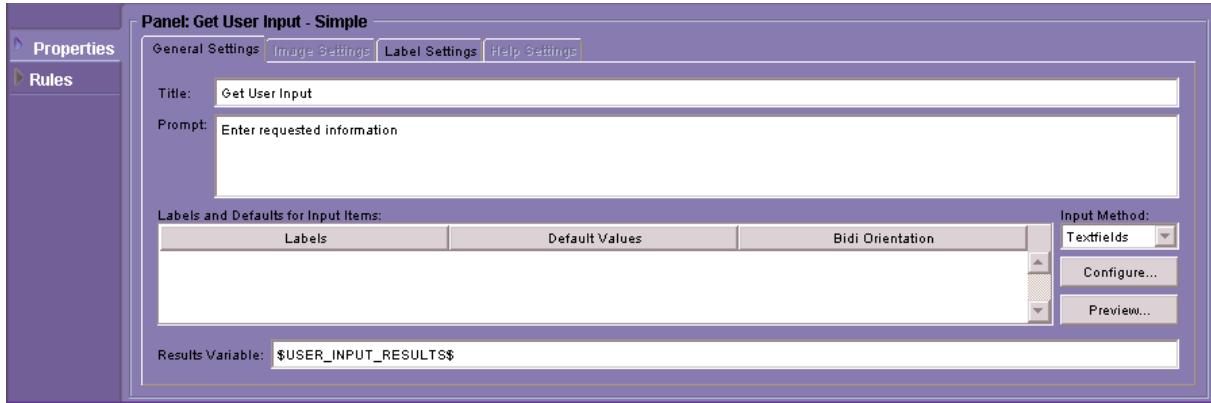
**Figure 14-1:** Adding a Get User Input Panel

The two types of **Get User Input** panel are the following:

- **Simple:** The panel can contain only user-input controls of a single type (edit fields, check boxes, radio button).
- **Advanced:** The panel can contain multiple control types, along with optional captions and static labels.

## Using the Simple Get User Input Panel

The default customizer for the Get User Input—Simple panel appears similar to the following.



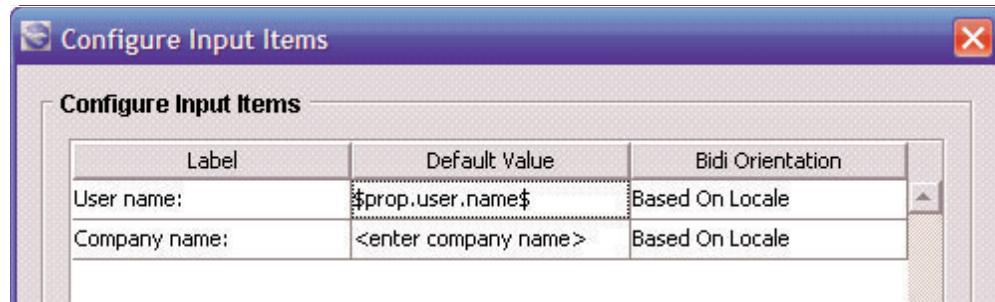
**Figure 14-2:** Customizer for Simple Get User Input Panel

First, you can specify the title and prompt displayed to the user. As with other panel types, you can include InstallAnywhere variables in the displayed text using the `$variable$` syntax.

In the **Labels and Defaults for Input Items** section of the customizer, you begin by selecting the desired control type from the **Input Method** list. The default control type is “Text fields”, and you can alternatively select “Checkboxes”, “Radio buttons”, “Popup menu”, or “List”.

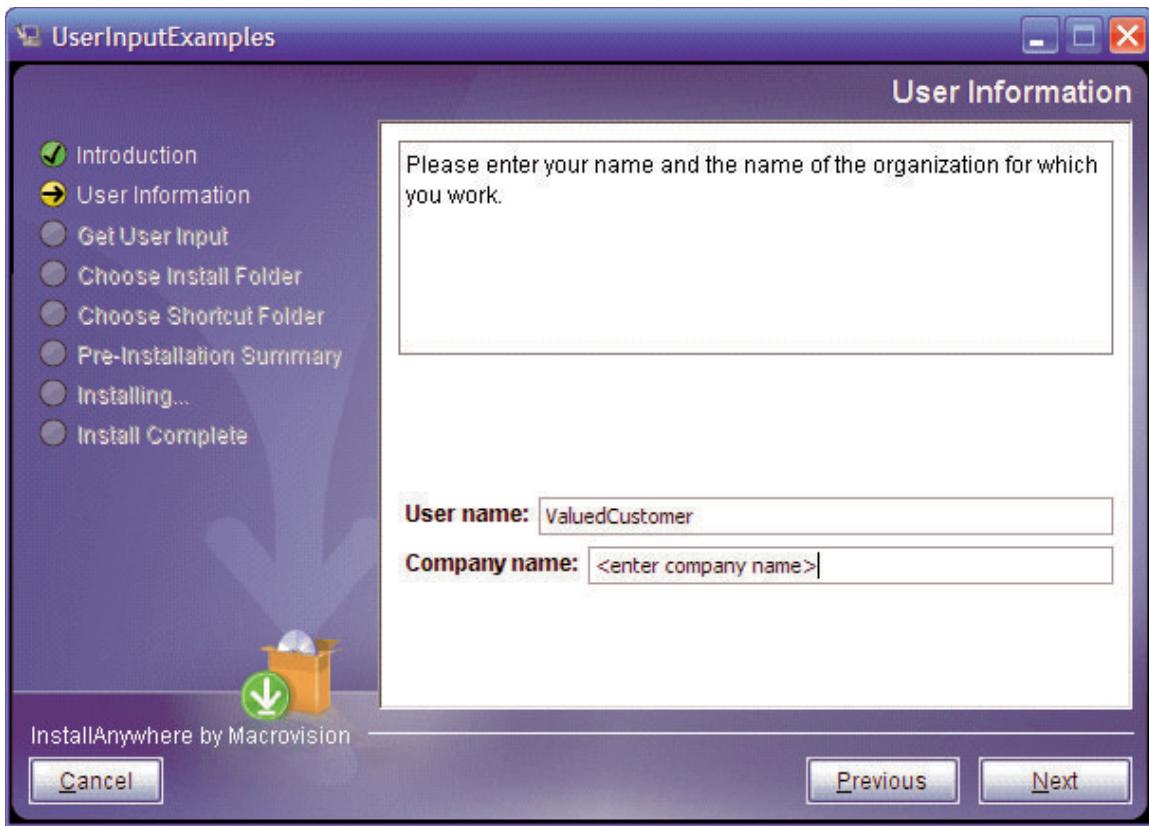
For this example, suppose you want the user to enter a user name and company name in two text fields you provide. In the **Input Method** list, make sure “Text fields” is the current selection, and click **Configure** to begin adding controls.

In the **Configure Input Items** dialog box, click **Add** to add a user-input field (a text field, in this example) label and the default value to display. (You can also specify the control’s orientation when running in a bidirectional locale; localization issues are discussed in Chapter 15.) For this example, two input items are used, with labels “User name:” and “Company name:”; with default values `$prop.user.name$`—which expands to the value of the Java system property `user.name`—and “<enter company name>”.



**Figure 14-3:** Configuring Text Fields

After closing the **Configure Input Items** dialog box, you can click **Preview** to see what the panel looks like at run time. At run time, the panel appears similar to the following:



**Figure 14-4:** Simple User Input Panel and Run Time

## Variables and the Get User Input Panel

The final item in the customizer for a Get User Input panel is the **Results Variable** field, with default value `$USER_INPUT_RESULTS$`, which can be changed to any valid InstallAnywhere variable, such as `$CUSTOMER_INFO_RESULTS$`.

After the panel is displayed, the specified variable (`$CUSTOMER_INFO_RESULTS$`) contains a comma-separated sequence of all of the user input, while the contents of individual fields are stored in numbered variations of the variable. For this example, the contents of the first (User name) field are stored in `$CUSTOMER_INFO_RESULTS_1$`, and the contents of the second (Company name) field are stored in `$CUSTOMER_INFO_RESULTS_2$`. You can then use the user input result variables in future actions, such as writing the results to a file.



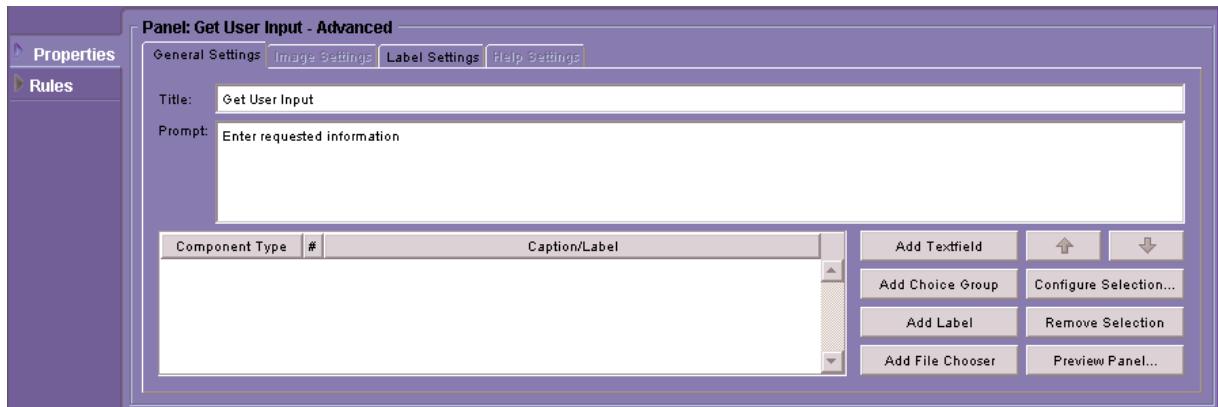
**Note:** The InstallAnywhere help topic “Get User Input Panels” describes the contents of the base and numbered variables when used with different input types.



**Tip:** To see how your Get User Input variables are populated, create a response file (using the installer command-line argument `-r`) for an installer containing your Get User Input panel.

## Using the Advanced Get User Input Panel

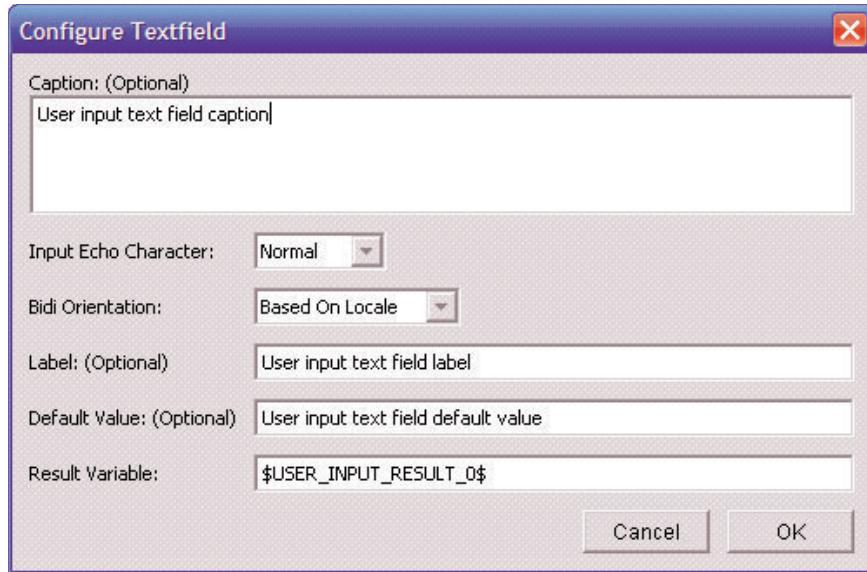
The **Get User Input—Advanced** panel gives you more flexibility with respect to multiple control types, labels, and so forth. The customizer for the panel appears similar to the following.



**Figure 14-5:** Customizer for Advanced Get User Input Panel

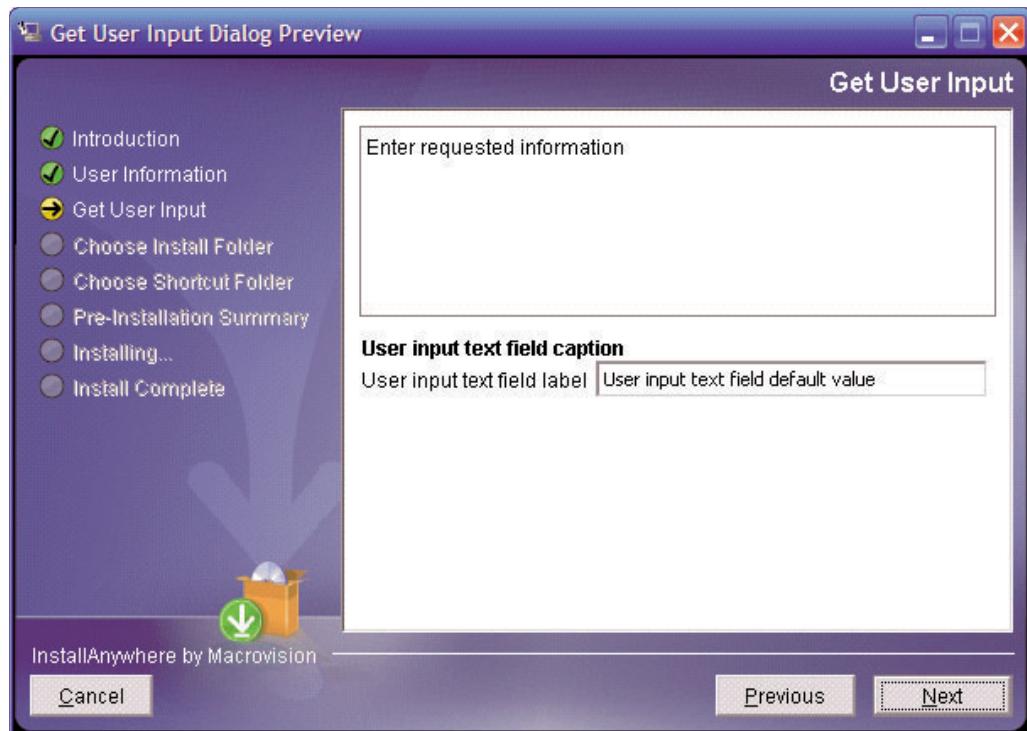
As with the simple panel, you click one of the **Add** buttons to add an input field, such as clicking **Add Text Field** to add a text field. Clicking **Add Choice Group** enables you to add a group of check boxes or radio buttons, or a popup menu or list control; and clicking **Add Label** or **Add File Chooser** respectively adds a static label or a file browser.

If you add a text field and click **Configure Selection**, the following dialog box enables you to enter an optional caption, label, and default value for the text field, along with the (required) name of the variable that will store the results.



**Figure 14-6:** Configuring a Text Field

At run time, the panel containing the text field appears as follows.



**Figure 14-7:** Advanced User Input Panel at Run Time

If you add multiple input components to the same panel, you can use the up-arrow and down-arrow buttons in the customizer to rearrange them.



**Tip:** For the sake of console installations, as described in Chapter 10, InstallAnywhere provides a **Get User Input** console action. With the **Get User Input** console action, you can prompt for text input or provide single-selection or multi-selection lists of responses during a console-mode installation.

## Custom Code Panels

For more sophisticated panel behavior than what is available in the Get User Input Panel, InstallAnywhere provides Custom Code Panels, which are the graphical equivalent to 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.

## Default Custom Code Panel

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;
}
```

## Creating a Custom Code Panel

To create a custom code panel with the same overall appearance (size, standard buttons, etc.) as the default panels, you create a Java class that extends `com.zerog.ia.api.pub.CustomCodePanel`. As with other types of custom code, the `CustomCodePanel` class defines methods that you override to control the behavior of the custom code panel. The methods to override are:

- `public boolean setupUI(CustomCodePanelProxy ccpp)`: called before the panel is displayed, this is where you add controls to the content area of the panel; if `setupUI` returns false, the panel is skipped.
- `public void panelIsDisplayed()`: called after the panel is on the screen; enables you to perform additional processing before the user clicks Next or Previous.
- `public boolean okToContinue()`: enables you to perform actions such as user-input validation when the user clicks the **Next** button; if you return false, the user will be unable to continue to the next panel.
- `public boolean okToGoPrevious()`: enables you to control whether the user can click the **Previous** button to return to the previous panel.
- `public String getTitle()`: returns the string to display as the panel's title at run time.

The framework of a custom code panel called `BlankCustomCodePanel` might appear as follows:

```
import com.zerog.ia.api.pub.*;
public class BlankCustomCodePanel extends CustomCodePanel
{
    public boolean setupUI(CustomCodePanelProxy ccpp)
    {
        return true;
    }
    public void panelIsDisplayed() { }
    public boolean okToContinue() { return true; }
    public boolean okToGoPrevious() { return true; }
    public String getTitle() { return "Blank Custom Code Panel"; }
}
```



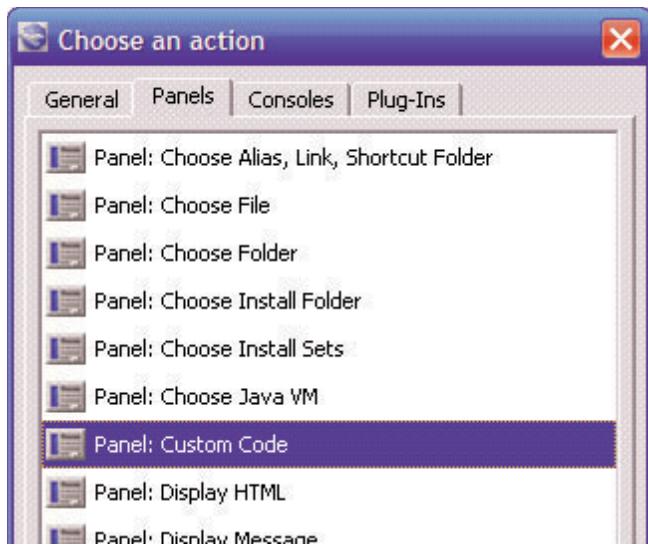
**Note: Advanced:** Defining the `okToContinue` or `okToGoPrevious` method to return false does not disable the Next or Previous button. To disable or enable the Next, Previous, or Cancel button, you can call the corresponding method `setNextButtonEnabled`, `setPreviousButtonEnabled`, or `setExitButtonEnabled`, as in the following.

```
// ccpp is the CustomCodePanelProxy argument passed to setupUI
// or the static customCodePanelProxy variable
GUIAccess gui = (GUIAccess)ccpp.getService(GUIAccess.class);
gui.setPreviousButtonEnabled(false);
```

Starting with InstallAnywhere 2008 Value Pack 1, you can also programmatically hide one of the buttons using the `setNextButtonVisible`, `setPreviousButtonVisible`, or `setCancelButtonVisible` method of the `GUIAccess` class. The visibility state is reset when the user exits a panel, which means that hiding the Previous button on one custom panel does not affect any other panel.

## Packaging, Adding, and Testing the Custom Code Panel

You package the `CustomCodePanel` class the same way you package other custom code, in a .jar or .zip file. To add the panel to your project, use the **Add Action** button in the desired task. In the **Choose an action** panel, select the **Panels** tab, and then select **Panel: Custom Code**.



**Figure 14-8:** Adding a Custom Code Panel

## Chapter 14: Custom Panels and Consoles

### Custom Code Panels

In the customizer for the custom code panel—as with other types of custom code—browse for the .jar or .zip file containing the compiled class, and enter the fully qualified, case-sensitive name of the custom class (`BlankCustomCodePanel`).



**Figure 14-9:** Specifying the Custom Code Panel Path

At run time, the custom code panel appears similar to standard panels. For example, it contains the usual **Cancel**, **Previous**, and **Next** buttons, and uses the project's background image.

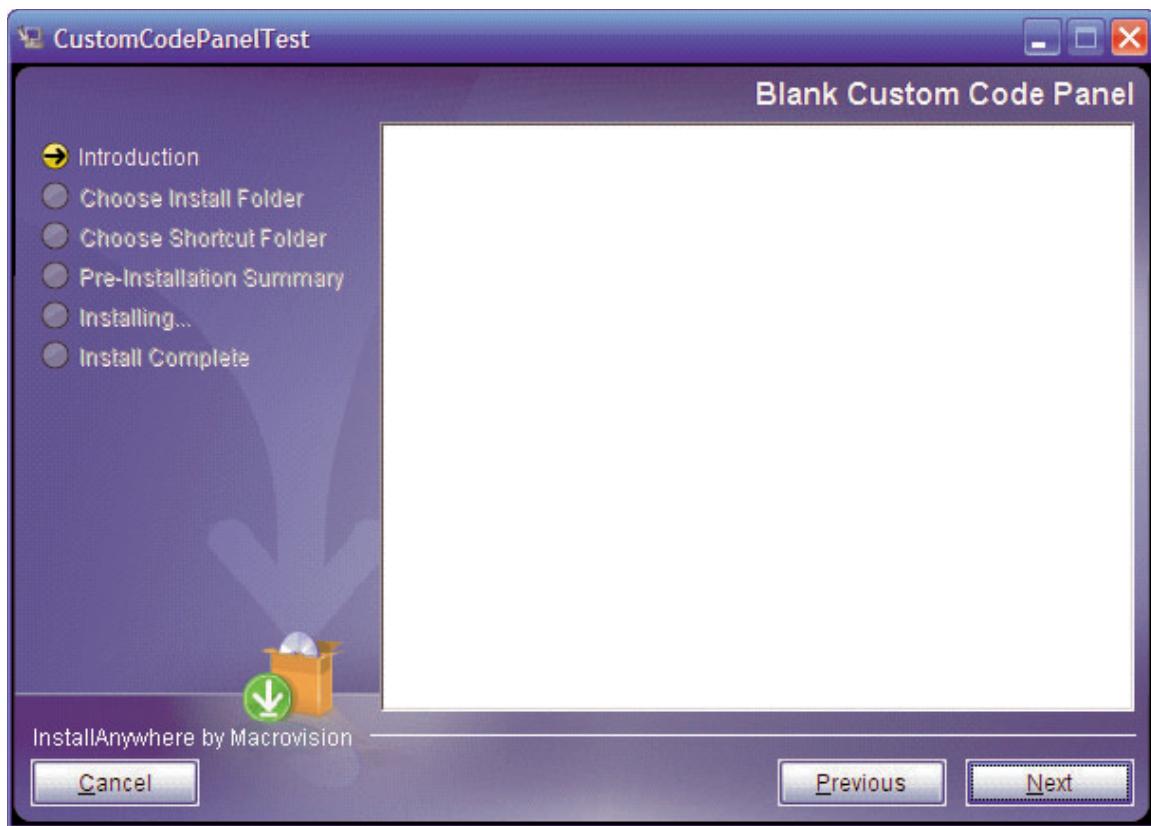


Figure 14-10: Custom Code Panel at Run-time

## Adding Controls to the Custom Code Panel

The content area of a custom code panel can use any Swing or AWT controls available to Java. You control the appearance and placement of controls in a custom code panel using the same classes you would use in any Java application.

## Laying Out the Custom Code Panel

You can specify any Swing or AWT layout manager for your custom code panel by calling the `setLayout` method of `CustomCodePanel` (inherited from `java.awt.Container`). For example, you can use a `GridLayout` layout manager—which adds equally sized components to an invisible grid on your panel—using code similar to the following in your `setupUI` method:

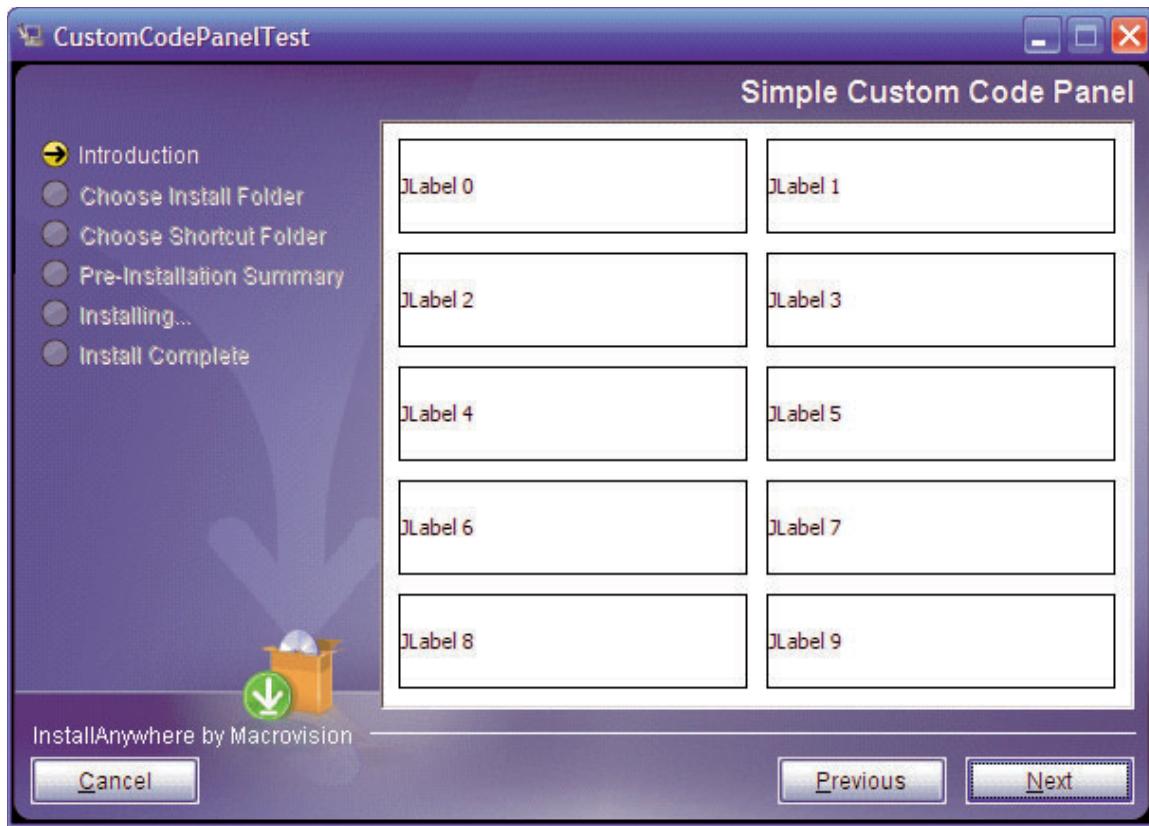
```
setLayout(new GridLayout(0, 2, 10, 10));
```

The following custom code panel adds several `JLabel` controls to the panel, arranged using the AWT `GridLayout` layout manager.

```
import java.awt.*;
import javax.swing.*;
import com.zerog.ia.api.pub.*;
public class SimpleCustomCodePanel extends CustomCodePanel
{
    public boolean setupUI(CustomCodePanelProxy ccpp)
    {
        setLayout(new GridLayout(0, 2, 10, 10));
        JLabel[] jlabels = new JLabel[10];

        for (int i = 0; i < 10; i++)
        {
            jlabels[i] = new JLabel("JLabel " + i);
            jlabels[i].setBorder(BorderFactory.createLineBorder(Color.black));
            add(jlabels[i]);
        }
        return true;
    }
    // ...omitting setTitle, panelIsDisplayed, okToContinue, and okToGoPrevious...
}
```

After you package the class into a `.jar` file and add it to your project as a **Panel: Custom Code** action, the panel appears at run time as follows.



**Figure 14-11:** Custom Code Panel at Run-time

A similar class using the `FlowLayout` layout manager might implement the `setupUI` method as follows:

```
public boolean setupUI(CustomCodePanelProxy ccpp)
{
    setLayout(new FlowLayout(FlowLayout.LEFT));
    JLabel[] jlabels = new JLabel[10];

    for (int i = 0; i < 10; i++)
    {
        jlabels[i] = new JLabel("JLabel " + i);
        jlabels[i].setBorder(BorderFactory.createLineBorder(Color.black));
        add(jlabels[i]);
    }

    return true;
}
```

At run time, the panel appears as follows.

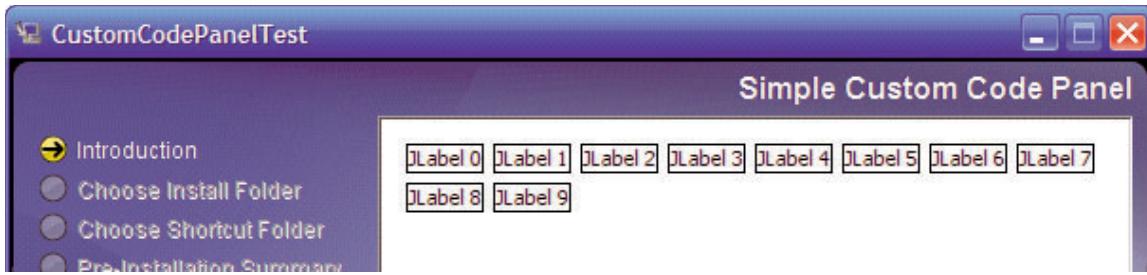


Figure 14-12: Custom Code Panel at Run-time

## Working with Variables

The purpose of most custom panels is to collect information from the user. In your custom code panels, you can use Java's input components from the `java.awt.*` and `javax.swing.*` packages.

The following example illustrates a custom code panel that displays two `TextField` components. The default text-field contents (as well as the panel layout) are set up during the `setupUI` method. When the user exits the dialog box by clicking **Next**, the `okToContinue` method verifies that both text fields contain data, and then stores the user input in `InstallAnywhere` variables `$USERNAME$` and `$COMPANYNAME$`. In a class that extends `CustomCodePanel1`, the `customCodePanel1Proxy` static variable and the `CustomCodePanel1Proxy` object passed to `setupUI` give access to variables, services, and so forth.

```
import com.zerog.ia.api.pub.*;
import javax.swing.*;
import java.awt.*;

public class CustomerInfoPanel extends CustomCodePanel1
{
    private boolean initialized = false;

    private JTextField userNameField;
    private JTextField companyField;

    public boolean setupUI(CustomCodePanelProxy ccpp)
    {
        // no need to re-initialize the panel
        if (initialized) { return true; }

        // create TextField objects with default contents
        userNameField = new JTextField(System.getProperty("user.name"));
        companyField = new JTextField("Your company name");

        // set panel layout to a vertical column, add controls
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        add(new Label("Enter user name:"));
        add(userNameField);
        add(new Label("Enter company name:"));
        add(companyField);
        add(Box.createRigidArea(new Dimension(10, 200)));

        initialized = true;
        return true;
    }
}
```

```
public boolean okToContinue( )
{
    // prevent user from continuing if either field is empty
    String username = userNameField.getText( ).trim( );
    String companyname = companyField.getText( ).trim( );

    if ((username.length( )) == 0 || (companyname.length( )) == 0)
    {
        JOptionPane.showMessageDialog(null,
            "Please enter a user name and company name.");
        return false;
    }

    // populate InstallAnywhere variables based on contents
    customCodePanelProxy.setVariable("$USERNAME$", username);
    customCodePanelProxy.setVariable("$COMPANYNAME$", companyname);

    return true;
}

public boolean okToGoPrevious( ) { return true; }
public void panelIsDisplayed( ) { /* do nothing */ }
public String getTitle( ) { return "Customer Information"; }
}
```

As with other examples, you compile the class and package it in a .jar file, and then add it to a task using the **Panel: Custom Code** action or package it as a plug-in and add the plug-in. At run time, the panel appears similar to the following:

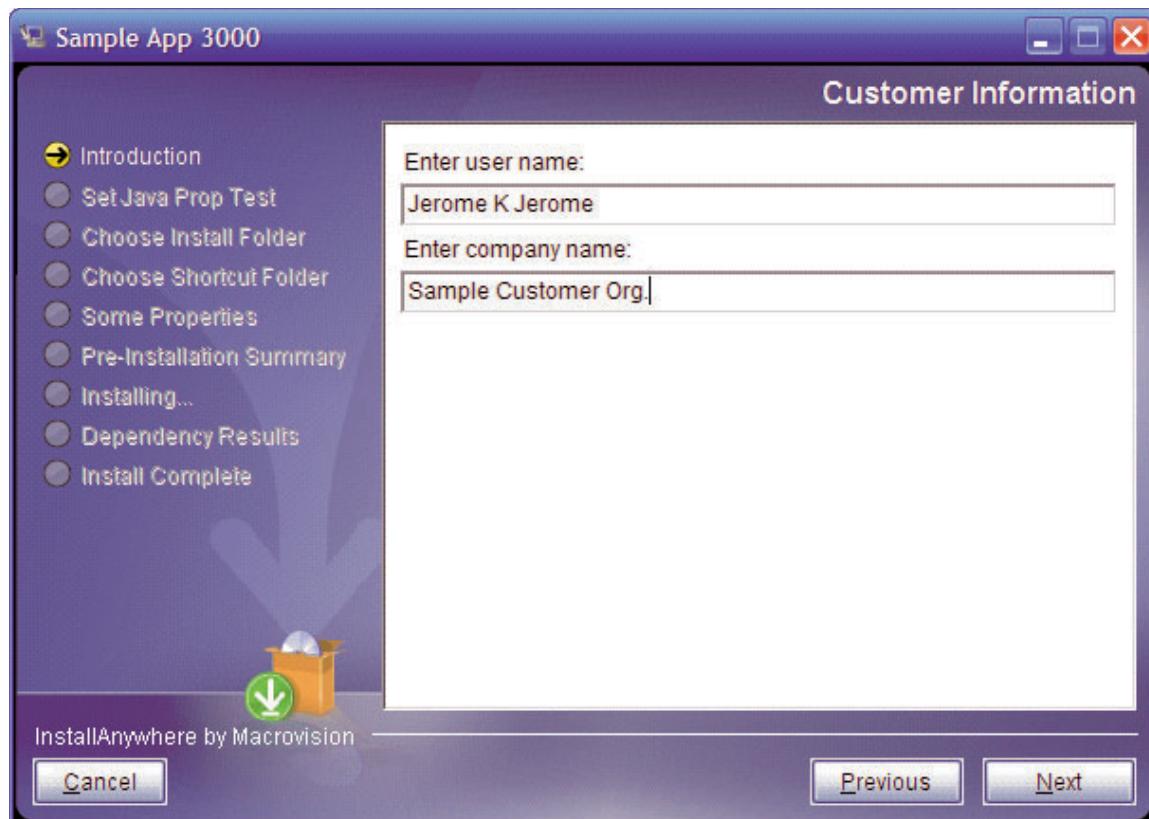


Figure 14-13: Custom Code Panel at Run Time

Because the user input is stored in the variables `$USERNAME$` and `$COMPANYNAME$`, any subsequent action can read the variable values and write them to the target system, compare them to specific values using a rule, and so forth.

As with other types of custom code, with InstallAnywhere 2008 Value Pack 1 you can use methods in the `ReplayVariableService` interface to control what variables are written—and how they are written—to response files and log files created by the user.

## Buttons and Action Listeners

While custom panels containing user-input fields are generally handled more easily with a Get User Input panel instead of custom code, custom code can be necessary for more sophisticated installer behavior. For example, the Get User Input panels do not support adding a button to a panel (other than the **Choose** and **Restore Default** buttons involved in a file-chooser control).

In order to react to button clicks and other events, a Java class must implement a listener. The following example adds a button to the custom panel, and the button contains an action listener that displays a simple message dialog.

```
import com.zerog.ia.api.pub.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class CustomPanelWithButton extends CustomCodePanel
{
    private boolean initialized = false;
    private Button button;

    public boolean setupUI(CustomCodePanelProxy ccpp)
    {
        if (initialized) { return true; }

        Panel p = new Panel();
        p.setLayout(new FlowLayout(FlowLayout.LEFT));
        p.add(new Label("Click to display additional information:"));

        // create button, add action listener
        button = new Button("Show Info");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                JOptionPane.showMessageDialog(null,
                    "(Place information here.)");
            }
        });
    }
}
```

```
p.add(button);

add(p);

initialized = true;
return true;
}

public void panelIsDisplayed() { /* do nothing */ }

public boolean okToContinue() { return true; }
public boolean okToGoPrevious() { return true; }

public String getTitle() { return "Custom Panel with Button"; }
}
```

As with other examples, you compile the class and package it into a .jar file. In this case, the anonymous inner action listener class causes an additional class file (`CustomPanelWithButton$1.class`) to be created, and both class files must be included in the .jar file. The following command ensures both classes are included in the .jar file:

```
jar cvf CustomPanelWithButton.jar CustomPanelWithButton*.class
```

After adding the custom panel to a task and building the project, the custom panel appears similar to the following at run time:



**Figure 14-14:** Custom Panel Containing a Button

This following 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.

## Example: Custom Code Panel

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 (refer to the 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. Call the `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.cs.ejalbert.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, text field 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 initied = false;
    private TextField tf;
    private Button b;

    public boolean setupUI(CustomCodePanelProxy customCodePanelProxy)
    {
```

```
// Use a boolean flag here to prevent duplicate GUI elements.  
if (initied == true)  
    return true;  
initied = 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() {}**

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. Refer to the Javadoc for `CustomCodePanel` for more information.

# 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 would be similar to:

```
=====
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:
=====
```

## Custom Code Console Actions

If you specified support for console mode in your project (in the **Installer UI > Look & Feel** task), you should create a custom code console action corresponding to each graphical custom code panel you create.

Write a class that extends `CustomCodeConsoleAction`, implementing the following methods:

- `public boolean setup()`: similar to the `setupUI` method for a custom panel, this method performs initialization before displaying any data on the console; return false to skip the console action at run time.
- `public void executeConsoleAction() throws PreviousRequestException`: this method should display data on the console or prompt the user for information.
- `public String getTitle()`: returns the string title for the action at run time.

The `ConsoleUtils` class provides the various information displays and prompts for user input during a console-mode installation. As illustrated in the example below, you obtain a handle to the `ConsoleUtils` class with the `getService` method of the console action's proxy class.

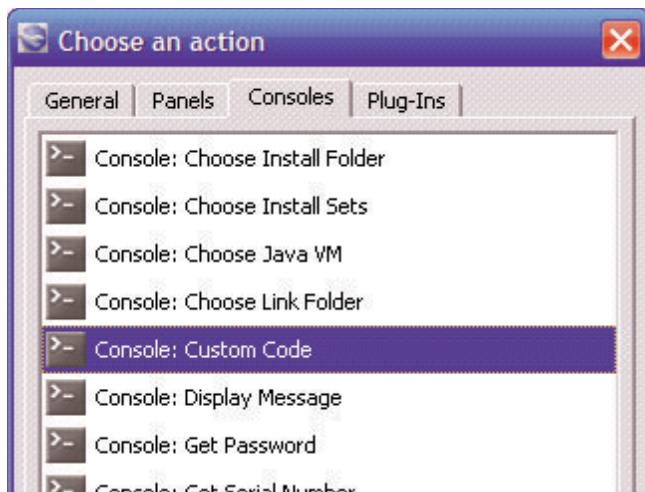
- `wprint` and `wprintln`: display a message to the user, wrapping lines as appropriate.
- `enterToContinue`: displays a message and waits for the user to press Enter.
- `promptAndYesNoChoice`: prompts the user to select yes or no based on the specified prompt.
- `promptAndGetValue`: prompts the user to enter a string.
- `createChoiceListAndGetValue`: prompts the user to select an item from a list presented at the console.

`CustomCodeConsoleProxy` gives the usual access to variables, installer services, and so forth. The `CustomCodeConsoleAction` class provides a static `consoleProxy` variable you can use to access variables, services, and so forth.

The following sample is a custom class called `BlankCustomCodeConsole`, which displays only a title and a prompt to continue.

```
import com.zerog.ia.api.pub.*;  
  
public class BlankCustomCodeConsole extends CustomCodeConsoleAction  
{  
    // true = display the console  
    public boolean setup( ) { return true; }  
  
    // display a user prompt to press Enter  
    public void executeConsoleAction( ) throws PreviousRequestException  
    {  
        ConsoleUtils cu = (ConsoleUtils)consoleProxy.getService(ConsoleUtils.class);  
        cu.enterToContinue( );  
    }  
  
    public String getTitle( ) { return "Blank Custom Code Console"; }  
}
```

After building and packaging the action, you add the action to a project using the **Console: Custom Code** action.



**Figure 14-15:** Console Actions

After you build and deploy the installer, at run time (when the user runs the installer with the **-i console** switch) the `BlankCustomCodeConsole` action appears at the console as follows:

```
=====  
Blank Custom Code Console  
-----  
  
PRESS <ENTER> TO CONTINUE:
```

For more information about console mode, refer to Chapter 10.

## Additional Console Controls

As listed above, the `ConsoleUtils` class provides other types of prompts for user input. The following class demonstrates more of the console prompts and display methods.

```
import com.zerog.ia.api.pub.*;
public class SimpleCustomCodeConsole extends CustomCodeConsoleAction
{
    public boolean setup( ) { return true; }
    public void executeConsoleAction( ) throws PreviousRequestException
    {
        // obtain handle to ConsoleUtils class
        ConsoleUtils cu = (ConsoleUtils)consoleProxy.getService(ConsoleUtils.class);

        // prompt for a yes-no answer
        boolean yesno = cu.promptAndYesNoChoice("Yes or no?");

        // display a blank line
        cu.wprintln(" ");

        // prompt for a string
        String username = cu.promptAndGetValue("Please enter your user name");

        cu.wprintln(" ");

        // prompt for a selection from a list
        String[] daysOfWeek =
            new String[] {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};

        int day =
            cu.createChoiceListAndGetValue("Pick a day", daysOfWeek, 2);

        // display results
        cu.wprint("\nYou selected ");
        if (yesno) cu.wprint("Yes"); else cu.wprint("No");
        cu.wprint(", your name is " + username + ", ");
        cu.wprintln("and your day is " + daysOfWeek[day] + ".\n");

        cu.enterToContinue( );
    }

    public String getTitle( ) { return "Simple Custom Code Console"; }
}
```

At run time, the console appears as follows:

```
=====
Simple Custom Code Console
-----
Yes or no? (Y/N): y
Please enter your user name: Jerome K. Jerome
1- Monday
2- Tuesday
->3- Wednesday
4- Thursday
5- Friday
Pick a day: 4
You selected Yes, your name is Jerome K. Jerome, and your day is Thursday
PRESS <ENTER> TO CONTINUE:
```

Several variations of each console component are provided; the Javadoc documentation has further information and examples.

## 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 does not 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 (refer to the 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. Refer to the Javadoc for `CustomCodeConsoleAction` for more information.

## 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



# 15

## Localizing and Internationalizing Installers

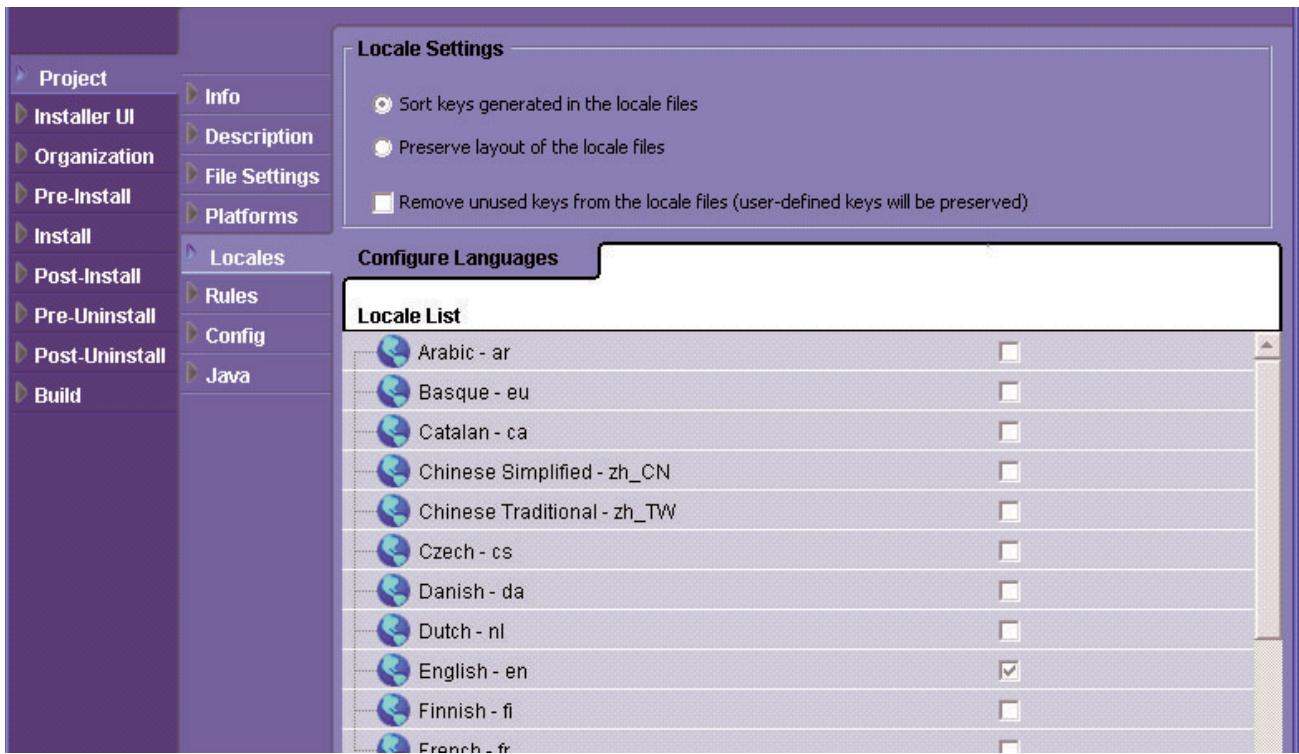
---

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. InstallAnywhere Enterprise Edition supports 31 different locales, and the Standard Edition supports 9.

This chapter contains information on:

- Dynamic and Static Text
- Localization and the Internationalized Designer
- Specific Localization Concerns
- Localizable Elements

To generate multi-language installers, click the **Project > Locales** task in the Advanced Designer, and use the checkboxes to select the appropriate languages.



**Figure 15-1:** Locale List



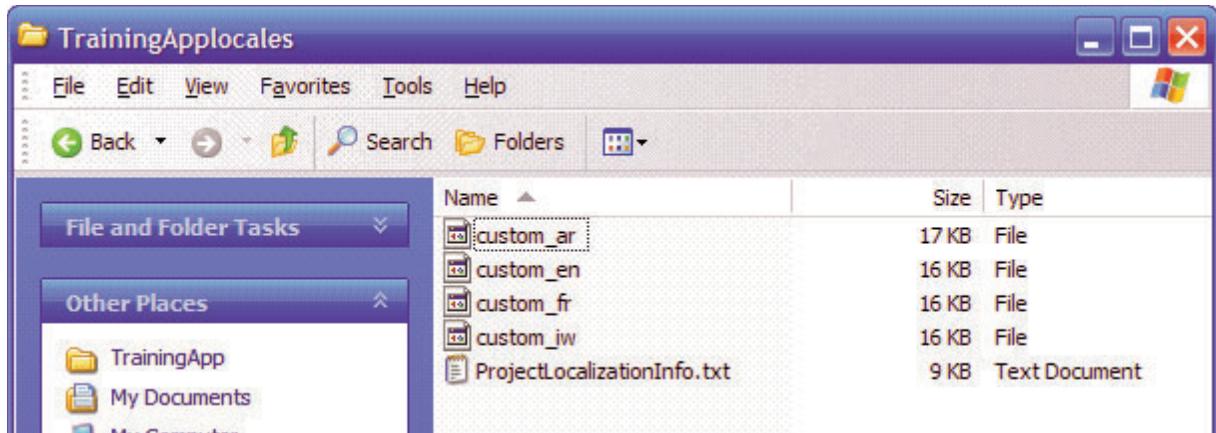
**Tip:** You can specify the locale in which the installer should run by specifying the `-l` (lowercase L) switch followed by the desired locale abbreviation. For example, to run the installer in French (assuming the project is built to support French), you can run the command `install -l fr`.

If you want to further modify text strings by locale, the string files are output each time an installer is built, in a directory 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. You may alter the text strings and, upon the next build of the installer, the new localized text will be displayed with the action.

When an installer project is first built, a directory 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`.



**Figure 15-2:** Locales Folder

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.a2ff402calcd.installSetName=Typical
InstallSet.a2ff402calcd.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 (refer to Figure 15-1) 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, refer to the 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 figures show the Arabic and Hebrew translations of the **Introduction** panel.

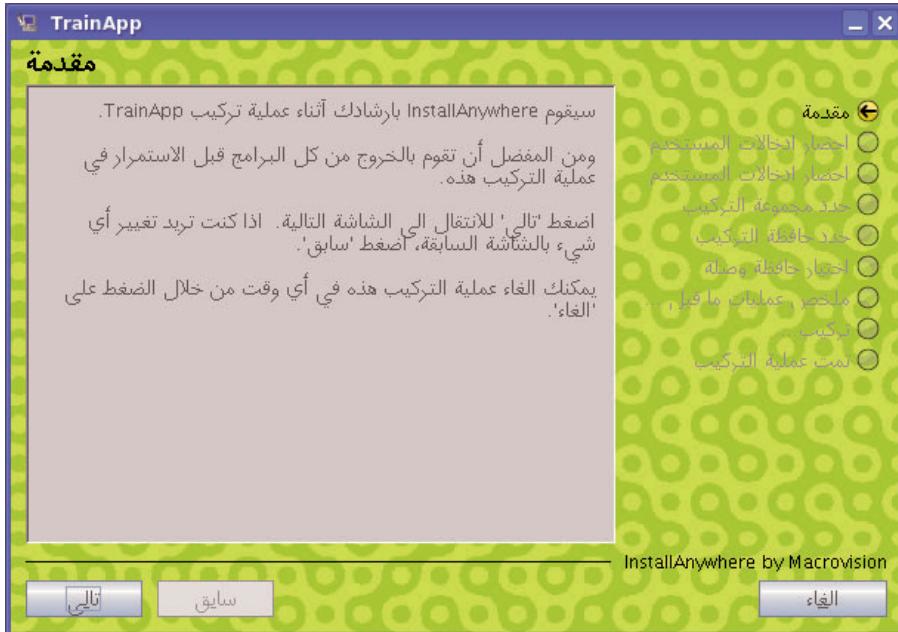


Figure 15-3: Arabic Translation of the Introduction Panel

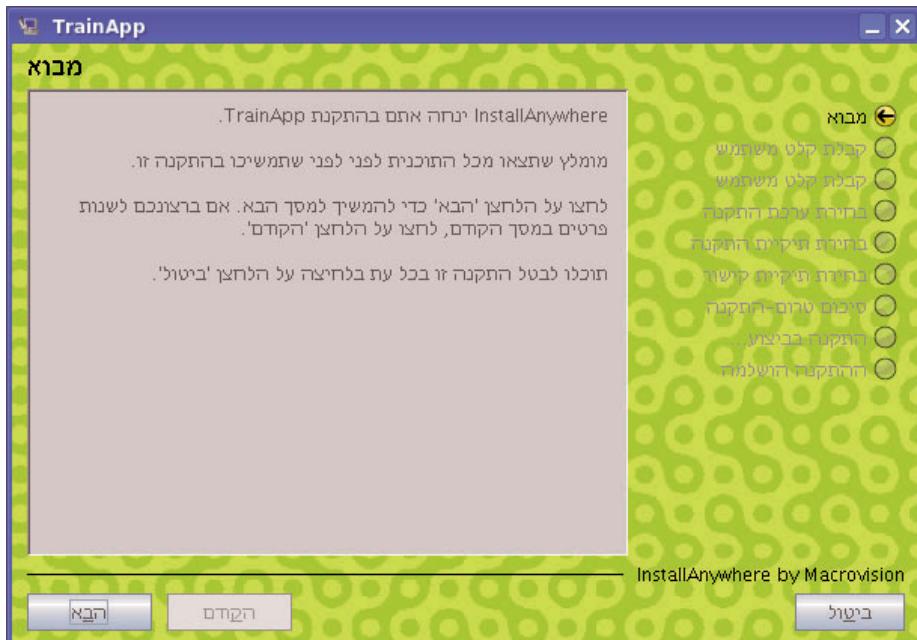


Figure 15-4: Hebrew Translation of the Introduction Panel



**Note:** 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.



**Figure 15-5:** Installer Background Image Dialog

## Dynamic and Static Text

Text that can be entered into the InstallAnywhere Advanced Designer is dynamic text. Dynamic text means that you 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 you are 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 it is unlikely that you will need to change the static text, InstallAnywhere provides you 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. As a developer, it is your job to localize these strings as well.

## Localization and the Internationalized Designer

You can localize, not only the installers created 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

You 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. You can then change these paths and the filenames. For example, to localize the License Agreement:

1. Make sure to include every resource in the **Install** task. Installers will not have access to resources not specified in this task.
2. 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.
3. Find the line that contains the text `LicenseAgr.#.Path`.
4. Specify the pathname 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. Below is 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. You can create any name for the key that you like, as long as it does not conflict with previously defined keys. You 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 you save, so update the files each 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 have 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 directory—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 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.

For a complete list of locale abbreviations, refer to the online help.



**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 directory inside of your InstallAnywhere directory. 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

Refer to 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.

**Table 15-1:** Language Resource Properties

Property	Definition
<b>Installer.#.ProductName</b>	Name of product displayed on installer title bar.
<b>Installer.#.RulesFailedMessage</b>	Message displayed if specified rules prevent the installer from running.
<b>Installer.#.ShortcutDestinationPath MacOS</b>	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.
<b>Installer.#.ShortcutDestinationPath Win32</b>	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.
<b>Installer.#.ShortcutDestinationPath Solaris</b>	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.
<b>InstallSet.#.Description</b>	Description of one of the installer's features.
<b>InstallSet.#.InstallSetName</b>	Name of one of the installer's features.
<b>IntroAction.#.message</b>	Text to display on the installer's Introduction step.
<b>Intro.#.stepTitle</b>	Title to display on the installer's Introduction step. Note that this is a filename only, and not a fully qualified pathname.
<b>LicenseAgr.#.Path</b>	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.)
<b>LicenseAgr.#.Title</b>	Title of License Agreement step in the installer.
<b>MakeExecutable.#.destinationName</b>	Name of the LaunchAnywhere Executable to be created on the destination computer.
<b>MakeRegEntry.#.Value</b>	Value to be written to the Windows registry.
<b>ShortcutLoc.#.macTitle</b>	Title of Mac OS X Choose Alias Location step in the installer.
<b>ShortcutLoc.#.SolarisTitle</b>	Title of Unix Choose Link Folder step in the installer.
<b>ShortcutLoc.#.Win32Title</b>	Title of Windows Choose Shortcut Folder step in the installer.

**Table 15-1:** Language Resource Properties

Property	Definition
<b>Billboard.#.ImageName</b>	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.)
<b>Billboard.#.ImagePath</b>	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.)
<b>ChooseInstallSet.#.Title</b>	Title of Choose Install Set step in the installer.
<b>ChooseJavaVM.#.Title</b>	Title of Choose Java Virtual Machine step in the installer.
<b>CreateShortcut.#.DestinationName</b>	Name of the shortcut/alias/link to be created on the destination computer.
<b>human.readable.language.name</b>	The name of the language represented by the data in this resource file (as in English, Español, etc.).
<b>ImportantNote.#.FileName</b>	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.
<b>ImportantNote.#.Path</b>	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.
<b>ImportantNote.#.Title</b>	Title of Important Note step in installer.
<b>InstallBundle.#.BundleName</b>	Name of component.
<b>InstallBundle.#.Description</b>	Description text describing component.
<b>InstallComplete.#.DisplayText</b>	Text to display on the Install Complete step of the installer.
<b>InstallComplete.#.Title</b>	Title of the Install Complete step in the installer.
<b>InstallDir.#.Title</b>	Title of the Choose Installation Directory step of the installer.
<b>Installer.#.InstallerName</b>	Name of the installer.

# A

## Installation Planning Worksheet

---

Use the following worksheet to assist in installation planning. Fill in general information for supported platforms, deployment media, application type, and make note of any installation or configuration information necessary.

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

---

---

---

---

# B

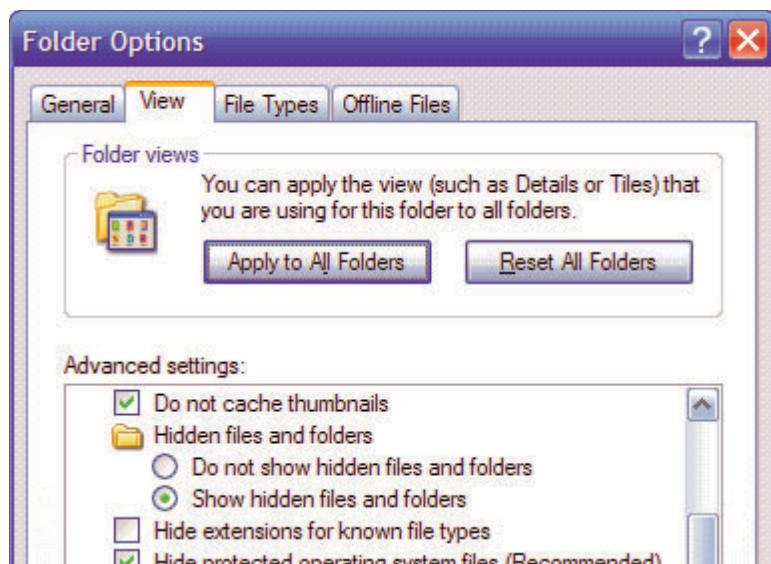
## Exercises

The exercises in this section are designed to reinforce the concepts covered in the training and test your knowledge of InstallAnywhere's features.

Exercises are organized by the chapter that introduces and explains the material. Some of these exercises use the OfficeSuite source files available in the InstallAnywhere directory; others use resources from the TrainApp directory provided with your course files.

Before beginning, it is recommended that you verify the following Windows settings by pulling down the operating system's **Tools** menu, selecting **Folder Options**, and selecting the **View** tab:

- Select **Show hidden files and folders**.
- Clear the check box labeled **Hide extensions for known file types**.



**Figure B-1:** Setting Windows View Options

# Chapter 2 Exercises

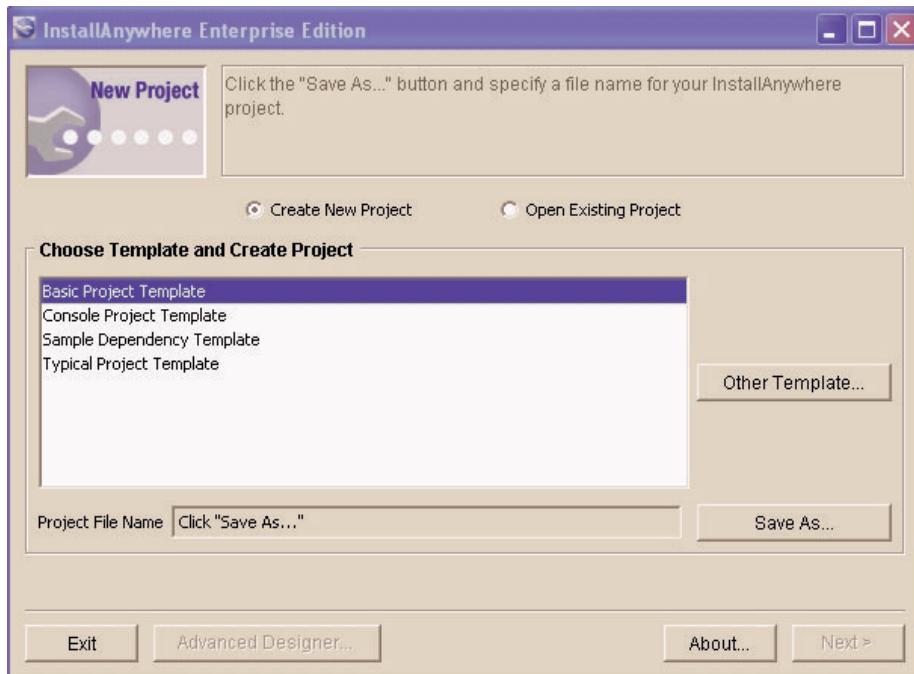
This section guides you through the process of creating the **OfficeSuite** and **TrainApp** projects. Both projects are used throughout the exercises.

## Creating the OfficeSuite Project

In this exercise, you use the Project Wizard to create a project for the “OfficeSuite for Java” application, which is included in the `InstallAnywhere` directory.

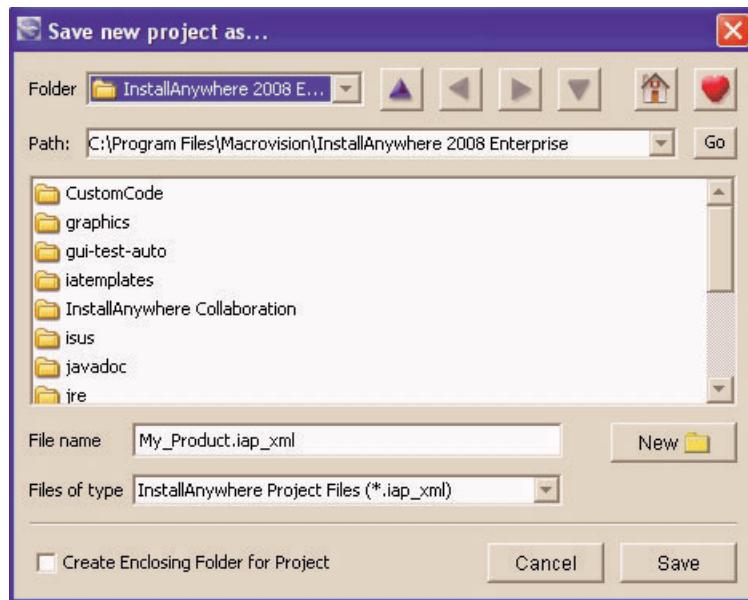
To create a new project:

1. Launch `InstallAnywhere`.
2. On the initial screen, the **Create New Project** option should already be selected.



**Figure B-2:** Create New Project Window

3. 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 this can be changed.

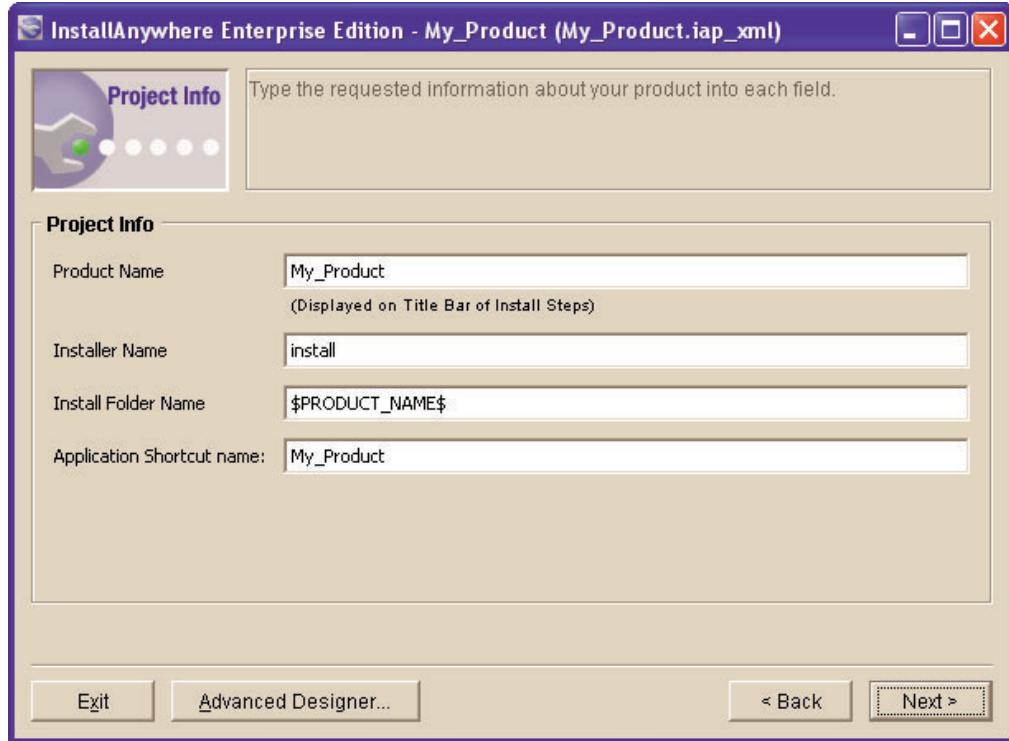


**Figure B-3:** Save New Project Window

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

## Setting Project Information

Setting the project information 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.



**Figure B-4:** Setting Project Info Using the Wizard

1. Enter the information in the appropriate text boxes. For this tutorial, refer to the following table:

Heading	Value
Product Name	OfficeSuite
Installer 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.

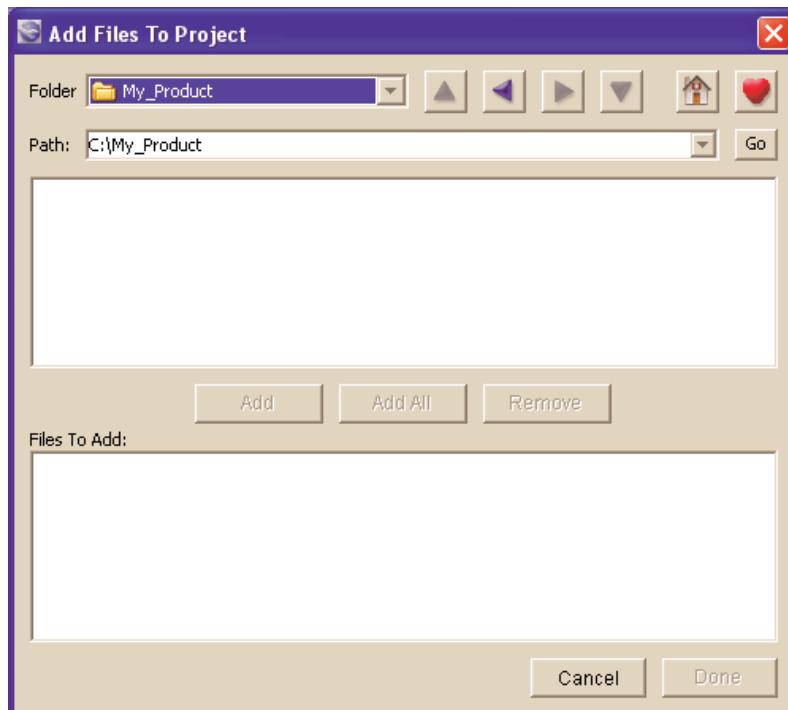
2. Click **Next** to move forward in the Project Wizard.

## Installing Tasks

This section consists of several steps. First add files to the project, choose the main Java class for starting the project, and set the classpath that the project uses.

### Add Files

1. Click **Add Files**. The **Add Files to Project** dialog box appears.



**Figure B-5:** Adding Files to the Project Using the Wizard

2. Browse through the list to find the `OfficeSuiteSourceFiles` folder, located within the `InstallAnywhere` installation directory.
3. Click **Add All** to add the `ImagesAndDocs` and `OfficeSuite2000` directories, which are inside the `OfficeSuiteSourceFiles` folder. These files appear in the **Files to Add** list.

This type of file linking adds a static list of files to your project: any files you add later 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.

4. 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.

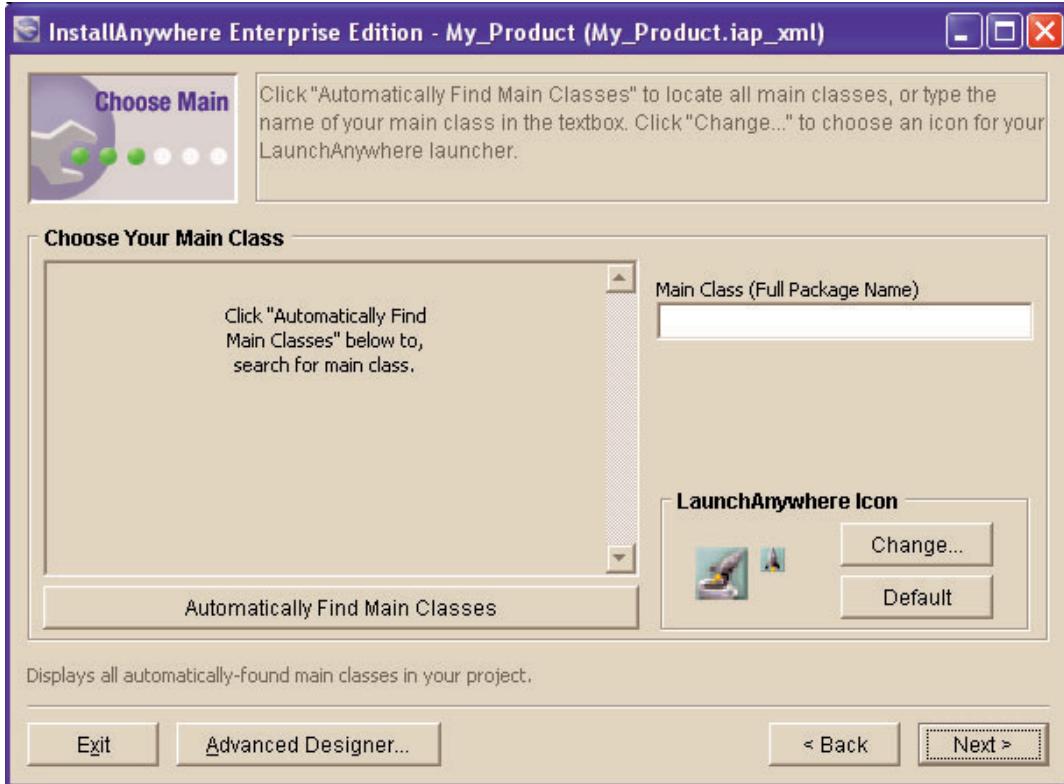
5. Click **Next** to move to the next step in the Project Wizard.

## 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.



**Figure B-6:** Choosing a Starting Class

1. Click **Automatically Find Main Classes** at the bottom of the screen.
2. Select the main class.
3. Specify a custom icon for the LaunchAnywhere executable by clicking **Change** and choosing a 32-by-32 or a 16-by-16 pixel .gif for the application icon. Navigate to the `Image and Docs` directory and choose `OfficeIcon.gif`.

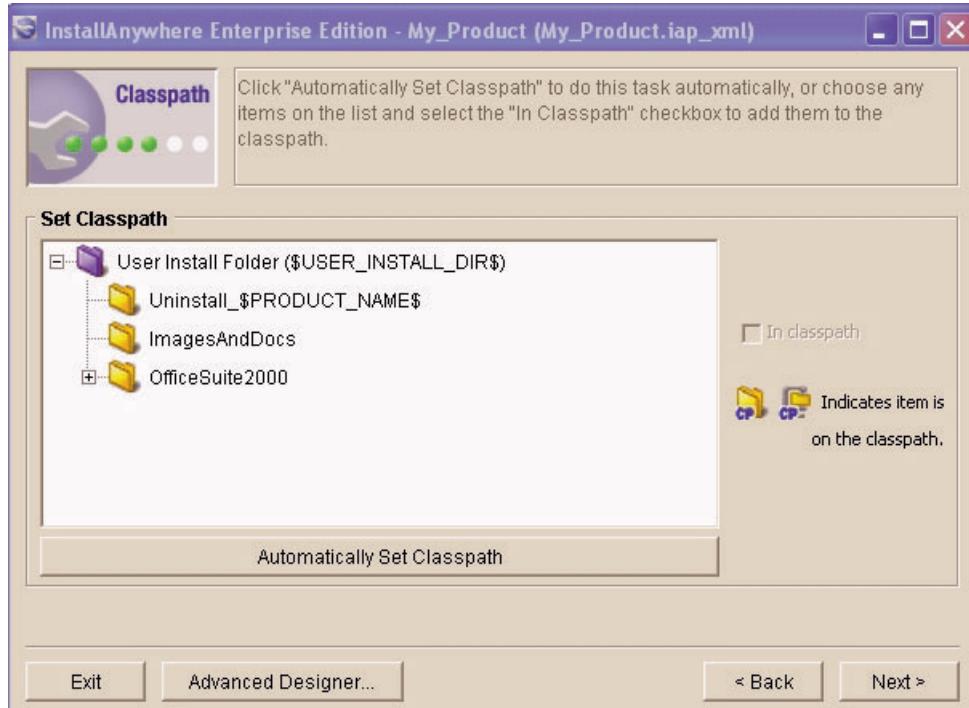


**Note:** Windows-only .ico files are not supported.

4. Click **OK** to confirm and close the dialog box. The icon appears on the main screen.
5. Click **Next** to move to the next step in the Project Wizard.

## Setting the Classpath

1. **Automatically Set Classpath** configures a Java application's classpath, which is a list of directories and .jar files containing classes used by the application.



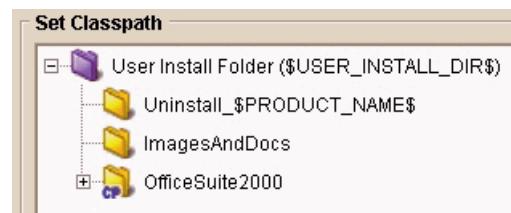
**Figure B-7:** Automatically Set Classpath Window

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.

2. 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.

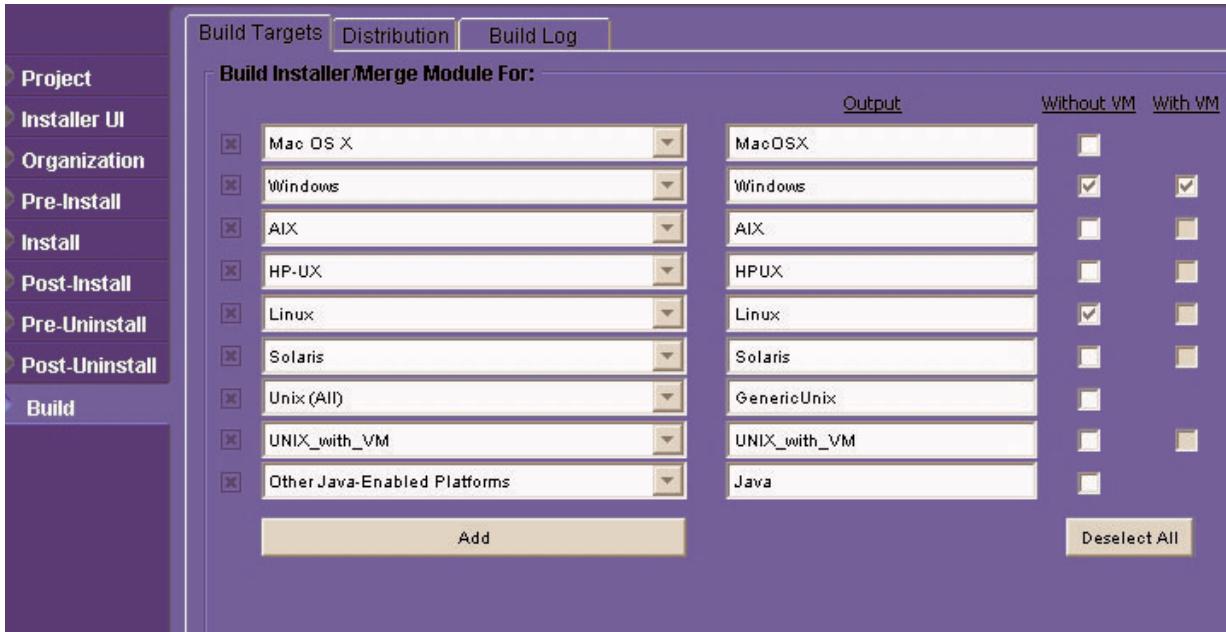


**Figure B-8:** File List After Automatically Setting Classpath

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

## Building the Installer

The first several items on the **Build Installer** screen, from **Mac OS X** through **Unix (All)**, represent installers that can be double-clicked on their respective platforms.



**Figure B-9:** Build Installer Options

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. You 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 allows end users to choose the appropriate installer for their system.

1. Choose the desired destination platforms.
2. Click **Build**.

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

## Testing the Project

Now that an installer is built, it is important to test the project to verify that it functions as desired. To test the project:

1. Click **Try It**.
2. After deploying the sample installer:
  - On Windows, go to the OfficeSuite program group and choose OfficeSuite.
  - On Unix, change directories where the program was installed and enter “OfficeSuite”.
  - On Mac OS X, double-click the OfficeSuite icon on the desktop.
3. After launching OfficeSuite for Java, quit by selecting **Exit** from the **File** menu.

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



**Tip:** On Windows, hold down the Control (Ctrl) key while the installer launches to see the debug output.

4. 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 are located in the `Build_Output` directories found in the same folder as the `.iap.xml` project file.

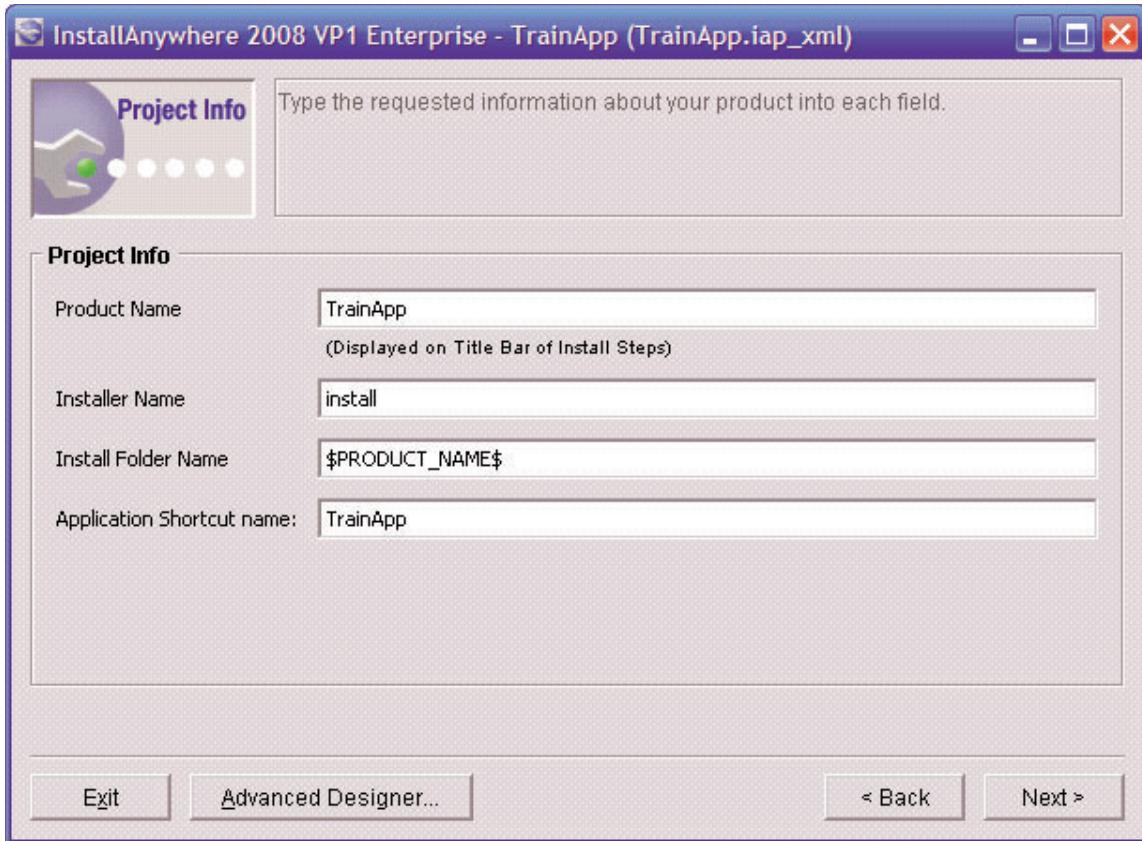
The `Build_Output` folder, also contains the `Web_Installers`, and `CDROM_Installers`. From within each of these sub-directories, choose the platform to test. For the CDROM installer, transfer the entire contents of the `CDROM_Installers` sub-directory.

## Creating the TrainApp Project

In this exercise, you use the Project Wizard to create a project for the **TrainApp** application. After you create the project, subsequent exercises will build on it, adding more installation-related tasks.

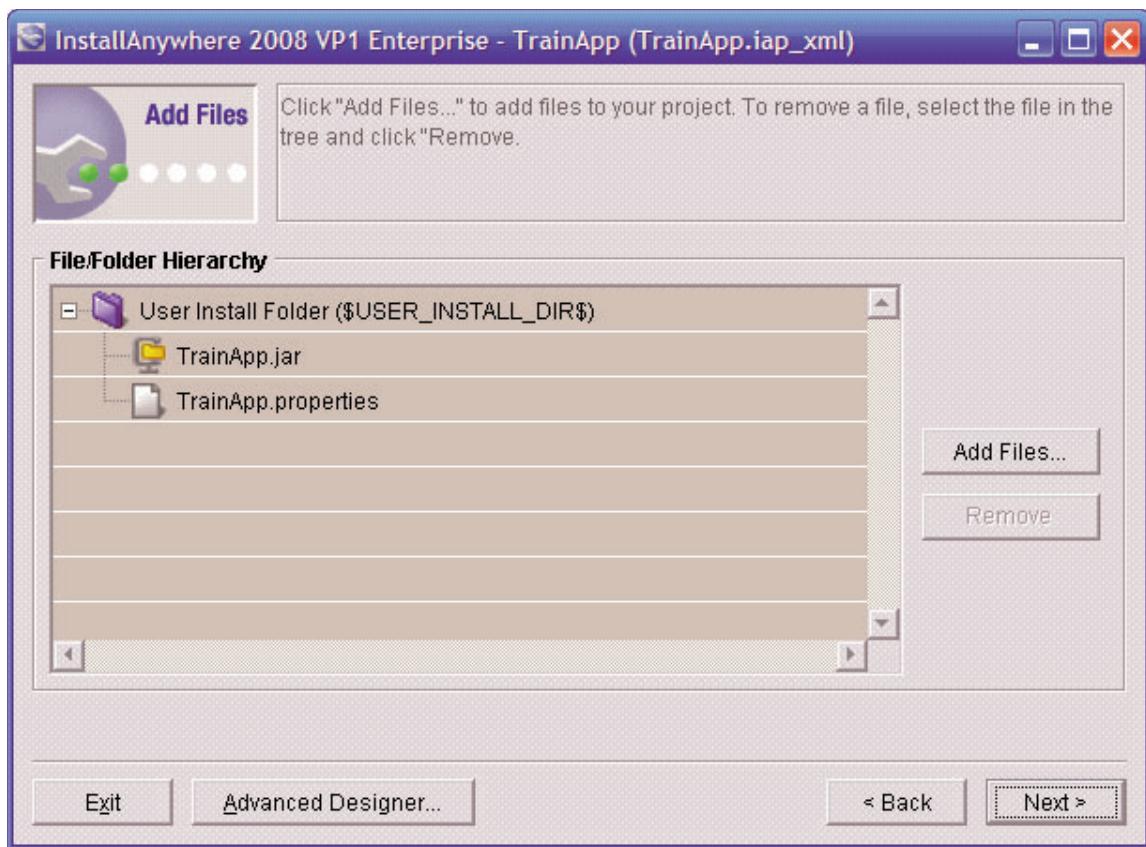
After launching InstallAnywhere, select **Basic Project Template** and then click **Save As**, saving a new project file called `TrainApp.iap_xml` to a directory of your choice. After saving the project, click **Next** to begin the Project Wizard.

In the **Project Info** panel, verify the default settings.



**Figure B-10:** Project Information in the Wizard

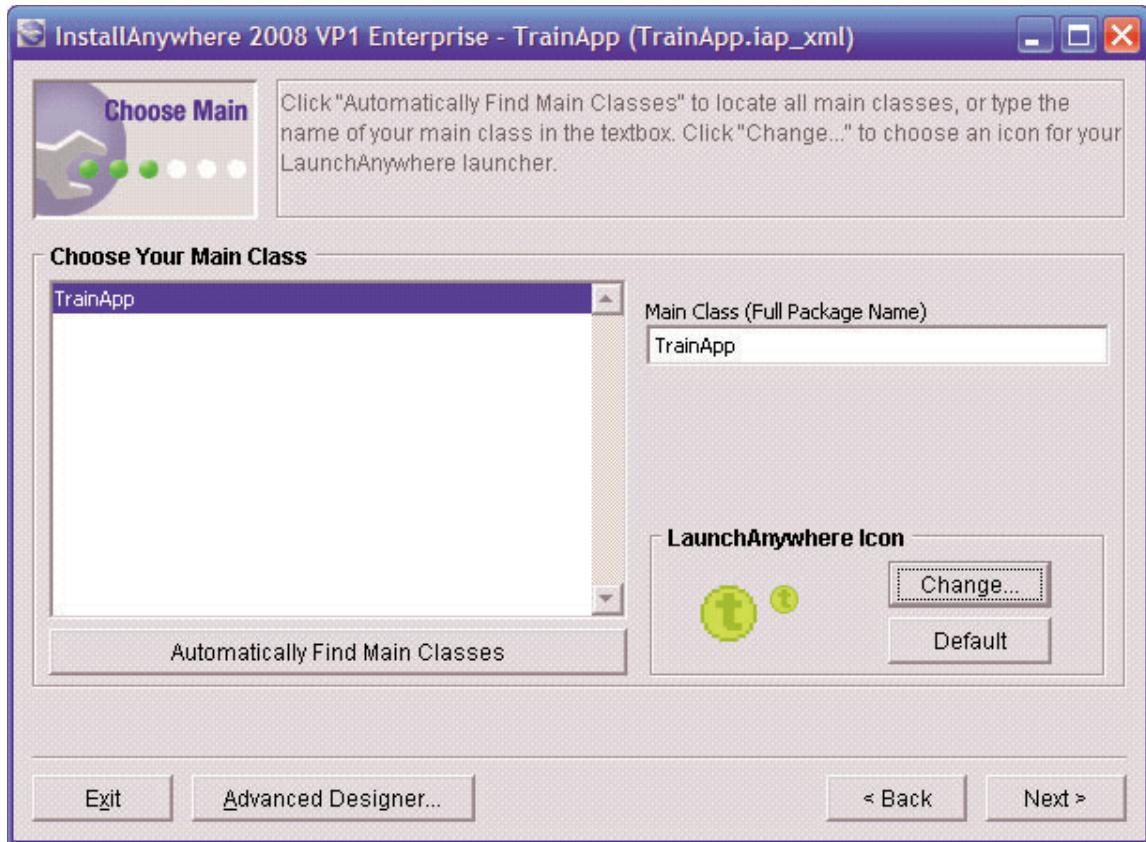
Click **Next** to open the **Add Files** panel. Click **Add Files** and browse for `TrainApp.jar` and `TrainApp.properties`, installing both to the default installation location `$USER_INSTALL_DIR$`.



**Figure B-11:** Adding Files to Your Project

Click **Next** to continue to the **Choose Main** panel.

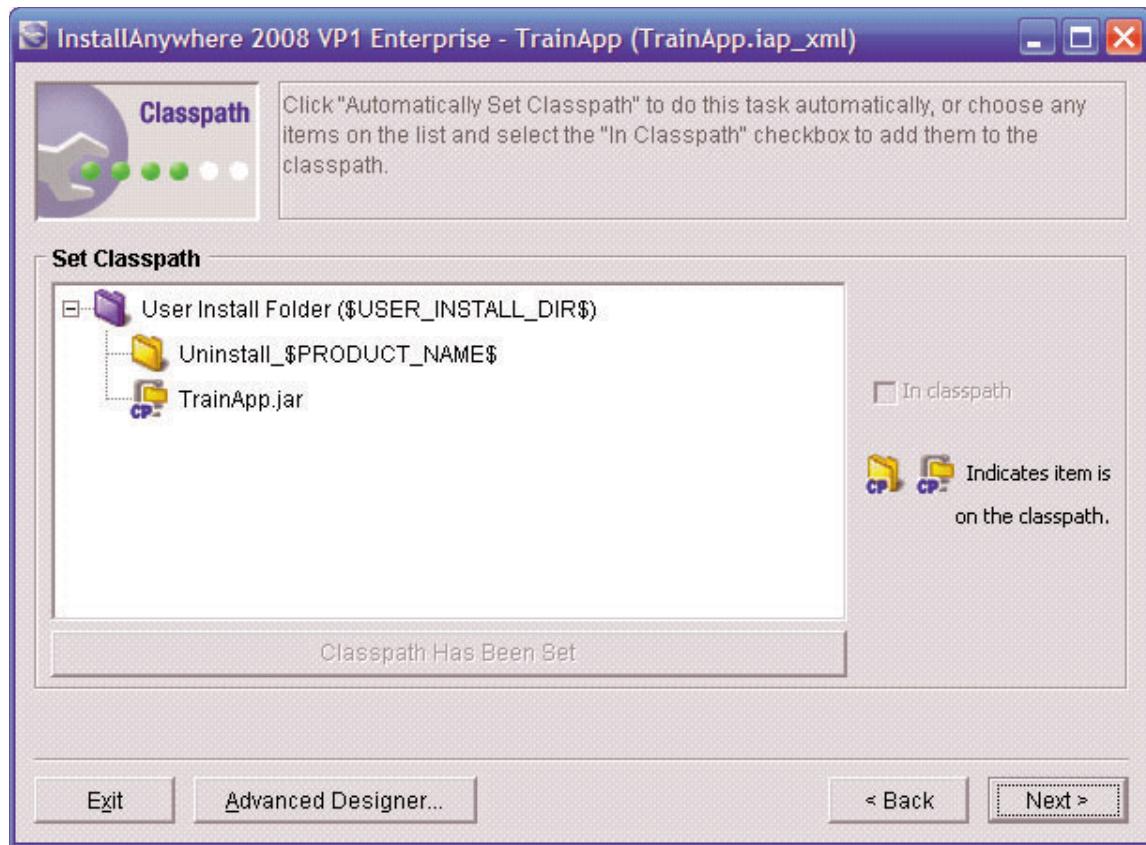
Because **TrainApp** is a Java application, creating a LaunchAnywhere launcher for it will make it much easier for an end user to launch the application. In this panel, click **Automatically Find Main Classes**, and InstallAnywhere will detect the class name TrainApp. In the **LaunchAnywhere Icon** area of the panel, browse for wizicon.gif.



**Figure B-12:** Choose Your Main Class for TrainApp

Click **Next** to continue to the **Classpath** panel.

Again, because **TrainApp** is a Java application, having its .jar file on the class path can be useful; click **Automatically Set Classpath** to add **TrainApp.jar** to the class path. Afterward, the **TrainApp.jar** icon has a **CP** overlay that indicates its being on the class path.



**Figure B-13:** Setting the Project Classpath

Click **Next** to continue to the **Build Installer** panel.

If desired, click **Build** to build an installation image for testing.



**Figure B-14:** Selecting the Operating Systems to Build For

When the build is complete, you can click the **Try It** button on the **Try Installer** panel (not pictured) to run the installation on your development system.

Stepping through the installer panels as an end user, note the default installation location `$USER_INSTALL_DIR$` being resolved to `C:\Program Files\TrainApp` on a Windows system.

If you opted to create an application shortcut at run time, you should see a shortcut in the **All Programs** menu under the Windows **Start** menu. Selecting the shortcut launches the LaunchAnywhere installer, which in turn launches the **TrainApp** Java application without your having to specify any special command-line arguments.

There should also be an uninstallation entry for **TrainApp** in the **Add or Remove Programs** panel on a Windows system. If you completed the **TrainApp** installation, run the uninstaller to remove the product files, shortcuts, and other data from your system.

Finally, click the **Advanced Designer** button at the bottom of the Project Wizard to prepare for other exercises.

# Chapter 3 Exercises

## Building an Installer with the Advanced Designer

In this exercise you will rebuild the 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.

This exercise guides you through:

- Creating a new project
- Pre-install Actions
- Defining the Installation Tasks
- Adding a LaunchAnywhere Executable to the Install Task
- Post-install Actions

### Creating a New Project

1. Launch InstallAnywhere. On the first screen, the **Create New Project** option should already be selected.
2. Select the **Basic Project Template**. This template should already be selected
3. 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.
4. Click 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.
5. Complete the **Installer Title** and **Product Name** fields. For now, you will skip the **Installer UI**, **Organization**, and **Install** tasks.

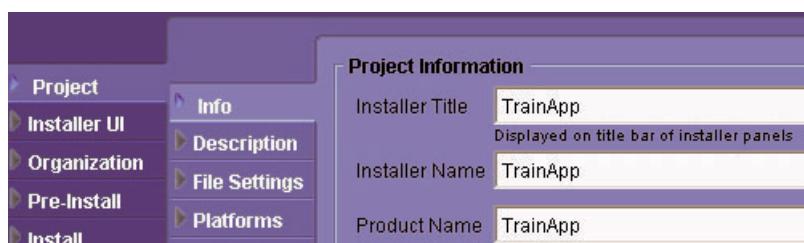


Figure B-15: Product Information

## Pre-Install Actions

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 you 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.

## Defining the Installation Tasks

1. 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.



**Note:** 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.

**2.** 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` directory 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 sub-directories 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.

## Adding a LaunchAnywhere Executable to the Install Task

### 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 you to add as many launchers as needed.

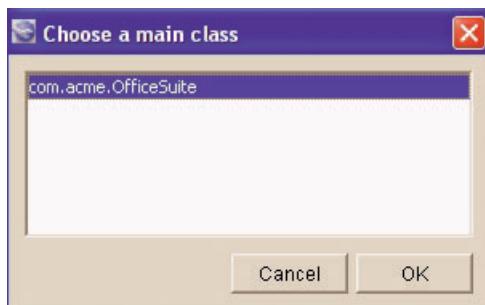
There are two ways to add a LaunchAnywhere Launcher to an InstallAnywhere project file. The **Create LaunchAnywhere for Java Application** option may be selected from the **Add Action** palette, or can be added by clicking the **Add Launcher** button on the middle control bar in the Advanced Designer.

1. Highlight the **User Install Folder** in the Advanced Designer, and click the **Add Launcher** button.
2. 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.

3. Choose the `com.acme.OfficeSuite` as main class for the application.

Since OfficeSuite is a simple project, you are presented with only the `com.acme.OfficeSuite` class.



**Figure B-16:** OfficeSuite Main Class from the Add Launcher

4. 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.

## 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.

The customizer (below the **arguments** field) has 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.

1. Click **Change** to alter the icon.
2. In the **Choose Icon** dialog, click **Choose GIF File**.
3. 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.



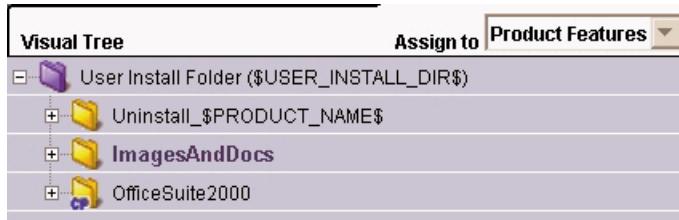
**Note:** Interlaced GIF files cannot be used with InstallAnywhere. The conversion process does not support these files and their use can result in blank icons. For Mac OS X, provide an ICNS file (created with iconbuilder—part of the Mac OS X Developer Tools).

## Set the InstallAnywhere Classpath

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

1. 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.



**Figure B-17:** OfficeSuite2000 Directory with Added Classpath

2. 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, you have only the main **OfficeSuite2000** folder (which contains loose class files). If this 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.

## 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.

1. Click the **Add Action** button to open the **Action** palette.
2. The **Action Palette** is divided by tabs that vary based on the task that is active at the time the palette is called.
3. 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. **Execute Target File** is not available in **Pre-Install** because files cannot be executed that are not installed yet.

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

The **Add Action** palette remains open so additional actions may be added.

5. 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 dependent on the execution. There is also a subtask that allows you 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 you 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 use the up and down arrows or drag the action to the bottom of the task list.

## 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.



**Note:** 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.

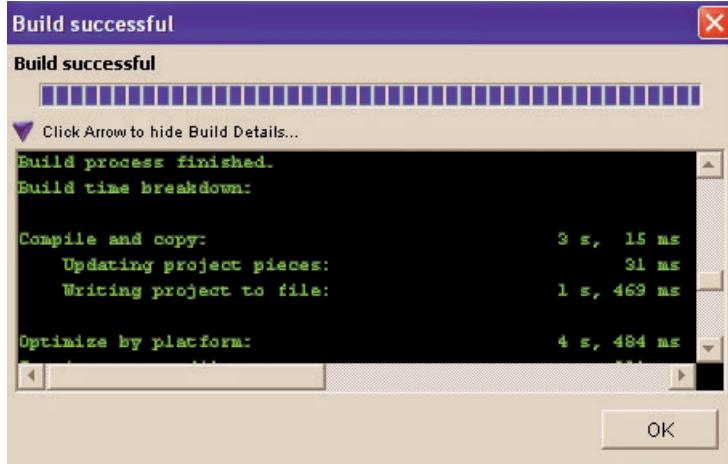
On the **Build Targets** tab in the **Build** task, select the platform(s). Selecting **With VM**, will bundle the installer with a VM. **With VM** is only selectable for platforms which have a VM pack. VM packs should be placed in the <InstallAnywhere>/resource/installer\_vms directory, and InstallAnywhere 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 you 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.

1. Click **Build Project** to build the OfficeSuite installer. The **Build** dialog will appear.
2. Click the blue arrow on the lower left of that dialog to see the build details console.



**Figure B-18:** Successful Build with Details

When the build is complete, a notification is displayed. In this case, the build should take a minute or less.

## 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 a browser and the InstallAnywhere Web Install Page generated by the build process.

1. Click **Try Web Install**. The **Web Install Page** will load, and should request a security access.
2. 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.
3. Click the **Start Installer for Windows** button below the image. The applet checks for sufficient disk space, download the installer, and execute the installer.
4. 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.

## Examining the TrainApp Project in the Advanced Designer

Reopen the **TrainApp** project in the Advanced Designer, and examine the details of the actions created by the Project Wizard. For example, in the **Install** task you can view the files and shortcuts created on the target system.



Figure B-19: TrainApp in Advanced Designer

Similarly, in the **Project > Description** task, you can see the project properties filled in by the Project Wizard. If you want, you can add product and company URLs and a product description; depending on the target platform, this information may be written to a system's native product registry, such as the Add or Remove Programs support details on a Windows system. Some of the information is also written to the platform-independent InstallAnywhere product registry.

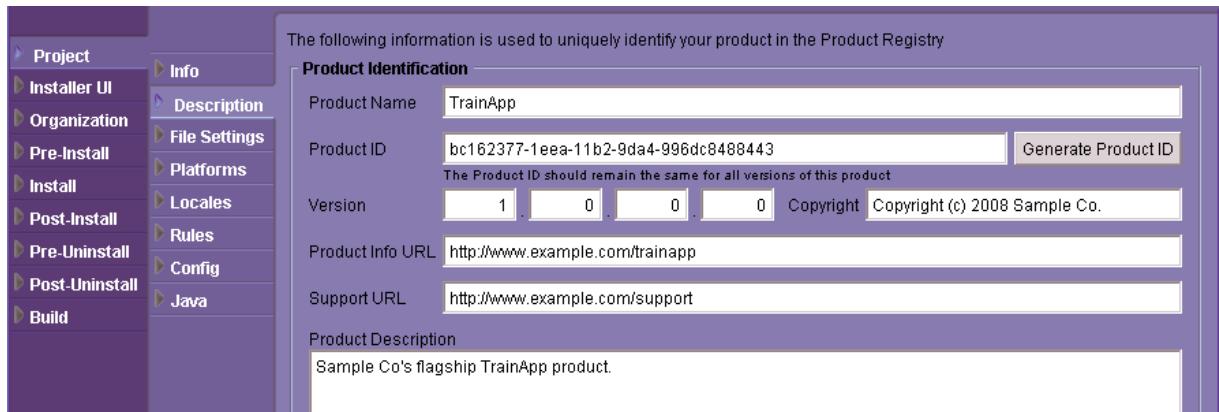


Figure B-20: Project Properties

In the remaining chapter exercises, you will use different tasks in the Advanced Designer to modify your projects.

# Chapter 4 Exercises

## Building the TrainApp Project

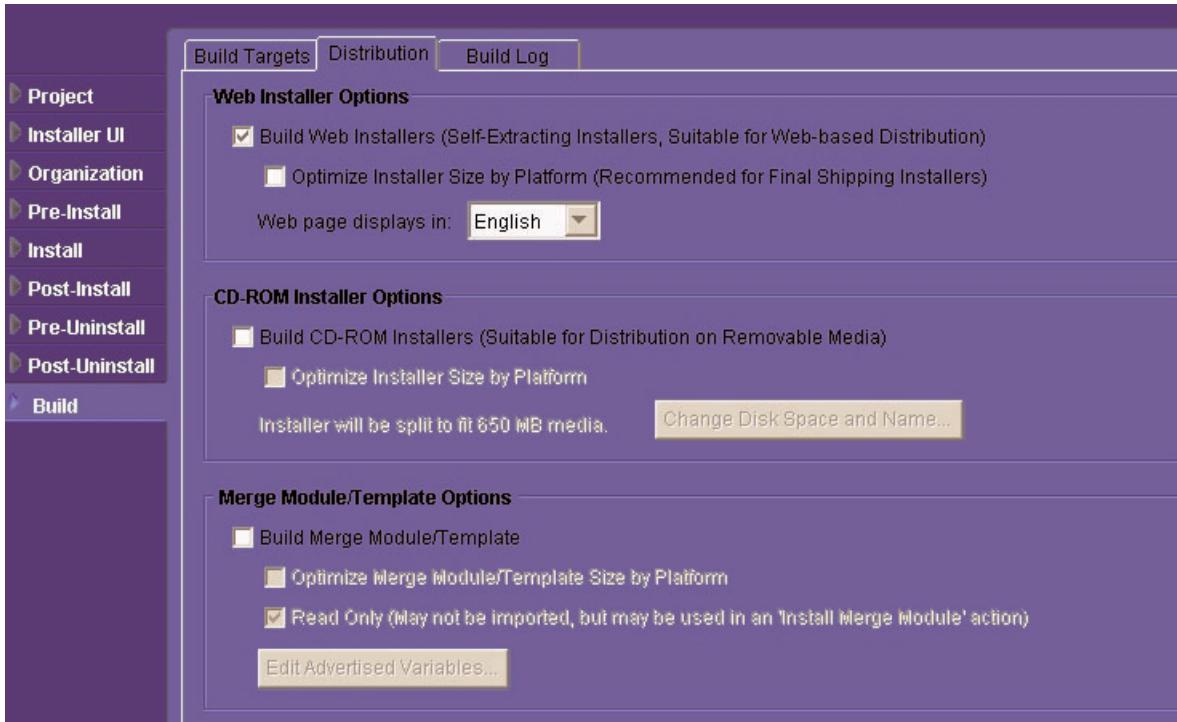
In the **Build** task, you define the types of installation media to build for testing and eventual duplication and distribution.

In the **Build Targets** tab of the **Build** task, verify that at least the Windows (Without VM) setting is selected. Select some platforms you will not use in the course, such as **Unix (All)**, and click the **X** button to delete the build target.



**Figure B-21:** Selecting Build Targets

In the **Distribution** tab, you must select at least one of Web Installer, CD-ROM Installer, or Merge Module/Template. For this exercise, select only the CD-ROM installer option.

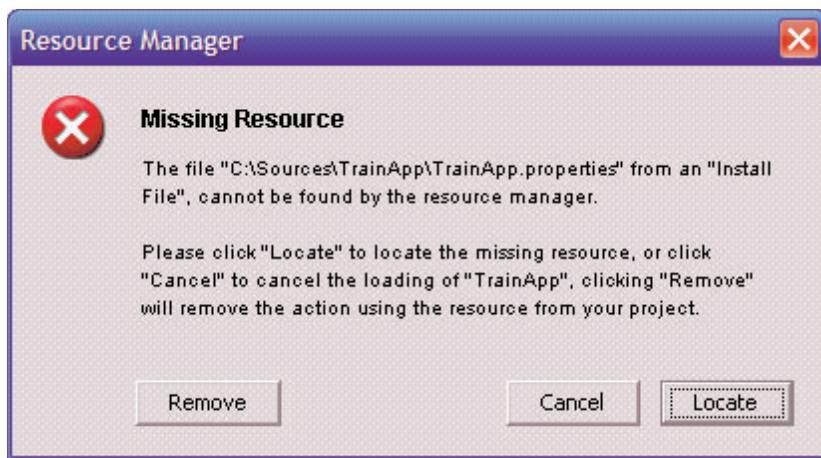


**Figure B-22:** Selecting Distribution Options

To build the project, click **Build Project** at the bottom of the task. When the build is complete, dismiss the small progress window and click **Open in Explorer**. You can then explore the output directory to view the structure of the files to be burned to a CD-ROM.

If you have time, add the **Web Installer** option to your distribution settings, and examine the web page used to install your project. If you run the entire installation, be sure to uninstall the product afterward.

If you have more time, force a build error by temporarily moving or renaming one of the source files. When you perform the build, you will see an error similar to the following.



**Figure B-23:** Resource Manager

For this example, click **Cancel** and then restore the source file to its original location and file name, and then rebuild the project to verify that the build finishes correctly.

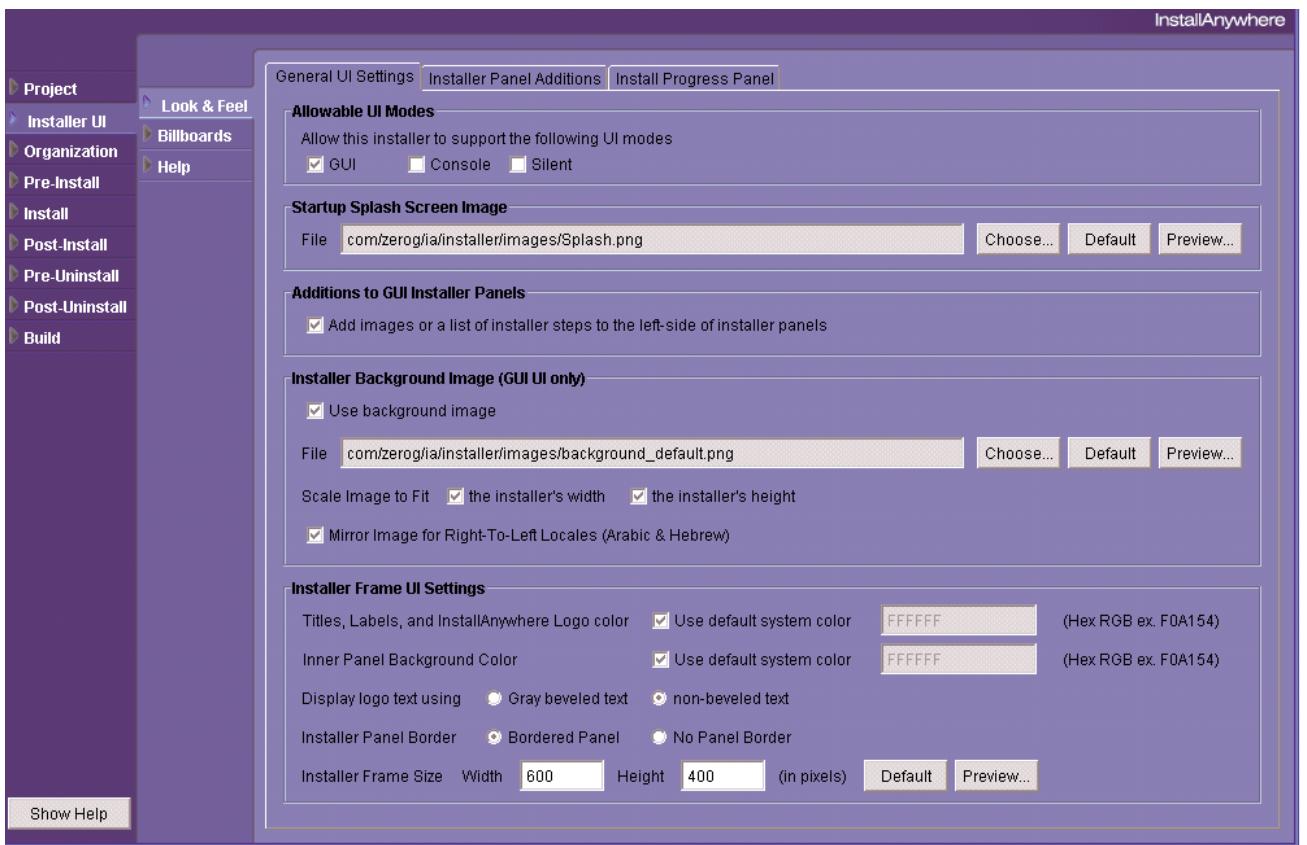
Other exercises cover command-line build tools.

# Chapter 5 Exercises

## 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.



**Figure B-24:** Installer User Interface

## Exploring the Installer UI Tasks

The **Look & Feel** task includes several sub-tasks 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 task:

Project > Look & Feel > General UI Settings

The **Installer UI > Look & Feel** subtask contains three tabs which enable you to configure the look and feel of the installer. These tabs allow you 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. You can 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**.
3. In the **Allowable UI Modes** section, select **GUI**.

If a silent or console installer is desired, it can be enabled here.

4. Explore the available splash screens.



Figure B-25: Previewing the 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 .jpg file of any size, although the preferred size is 470 by 265 pixels.

**5. In the Installer Background Image section:**

- a. Select Preview.**
- b. Click Choose.**
- c. Select a background graphic.**
- d. 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.

## Making Additions to the GUI Installer Panels.

**1. 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.

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

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

**3. In the Type 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. You 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.

**4. Click Preview.**

5. In **Type 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 you 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 you to view and alter the installation steps labels. The buttons to the right enable you to add or remove labels, or change the order of the labels. You 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.

6. Alter Order of Steps—Using the arrows allows you to change the order of the steps.
7. Edit Label—Change a label, **Pre-Installation Summary** can be changed to **Summary**.
8. Change Icons—**Choose Icons** enables you 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.

## Selecting 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.

1. Select **Installer UI > Billboards**.
2. In the **Billboard** section near the bottom of the window, click **Preview**.
3. Near the center of the screen, click the **Add Billboard** button.

4. In the `InstallAnywhere 2008 Enterprise\OfficeSuiteSourceFiles\ImagesAndDocs` directory, select `billboard1.gif`.

There are several billboard graphics available in the `ImagesAndDocs` directory within the `OfficeSuiteSourceFiles` folder. In this case, it is advised that you add 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 .jpg 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.

5. Click **Choose**.
6. Select `billboard2.gif`.
7. 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**.

## Exploring the Help Options

1. Browse to **Installer UI > Help**.
2. In the top section, select **Enable installer help**.
3. In **Help Text Format**, select **HTML**.
4. In **Help Context**, select **Use the same help text for all panels**.
5. In the **Title** text field, enter **Sample Help Text**.
6. In the **Help Text** text field, enter:  

```
<b>OfficeSuite</b> Help <br> This is an <i>example</i> of HTML help.
```
7. Click **Preview**.
8. Click **Close**.

## Rebuild the Project

1. Click **Build** and **Build Project** to build the OfficeSuite installer.
2. Click the blue arrow on the lower left of that dialog to view the **Build Details** console.
3. Once the project is rebuilt, test web-applet page customization as well as the installer customizations.

## 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 the 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.

To add the rule to the project:

1. Open the **Project > Rules** task
2. Click **Add Rule**.
3. Select **Check Platform**.
4. Click **Add**.

To set the condition for the rule:

1. 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.
2. 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 you've specified, the installer should run normally.

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.

## 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, you will 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.

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.

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.

You've created a set of install panels that will appear in the pre-install section of the installer. Now add a rule to one of the panels.

1. Select **Display Message**.
2. In the **Title** text bar, name the **Display Message** "Running as root" so it will be easily recognized if other display messages are used.
3. In the text box **Enter message to be displayed during installation**, enter display only if running as root.
4. In the **Customizer**, click **Rules**.
5. Click **Add Rule**.
6. Select Compare InstallAnywhere Variables, then click **Add**.
7. Create the following rule:

**Install Only If:**

**Operand1**

`$prop.user.name$` equals

**Operand2**

`root`



**Note:** 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.

- #### **8. Create another similar rule:**

## **Install Only If:**

## Operand1

\$prop.user.name\$ does not equal

## Operand2

-  
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.

- ## **9.** 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 will not see that panel. Try removing that rule and re-running. The panel should now appear.

The example you 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, you will be building a more complex installer, integrating end-user input with the concepts covered here.

# 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, you will return to an action added to the OfficeSuite Installer—the execute action added in the **Post-Install** task.

Generally, it is a good practice to ask the end user if they would like to launch the application when the installation is complete. In order to add this functionality, you will need both a method to ask the end user if they would like to launch the application, and a method to control that action.

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

In this next exercise, you will put the two together in a useful manner.

## Step One: Retrieving End-User Input

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

Later you will create some new projects, but in this section you will continue to work with the 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 it is a Yes or No question, use the radio buttons option as the input method.

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

Next, configure the panel to present your 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, 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 and 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 added earlier, add a rule to that action. You want the action to occur only if the end user has selected the “Yes” option. Since that information is stored in the variable selected in the **Get User Input - Simple** panel, you will 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. Add the following rule:

**Install Only If:**

**Operand 1:**

\$LAUNCH\_APPLICATION\$

contains

**Operand 2**

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.



**Tip:** Although it is strictly necessary only when retrieving variables, it is 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.

## Changing the Splash Screen

In the **Installer UI > Look & Feel** task, browse for `TrainAppSplash.png` in the **Startup Splash Screen Image** section. After building and running your project, the splash screen should appear as follows.

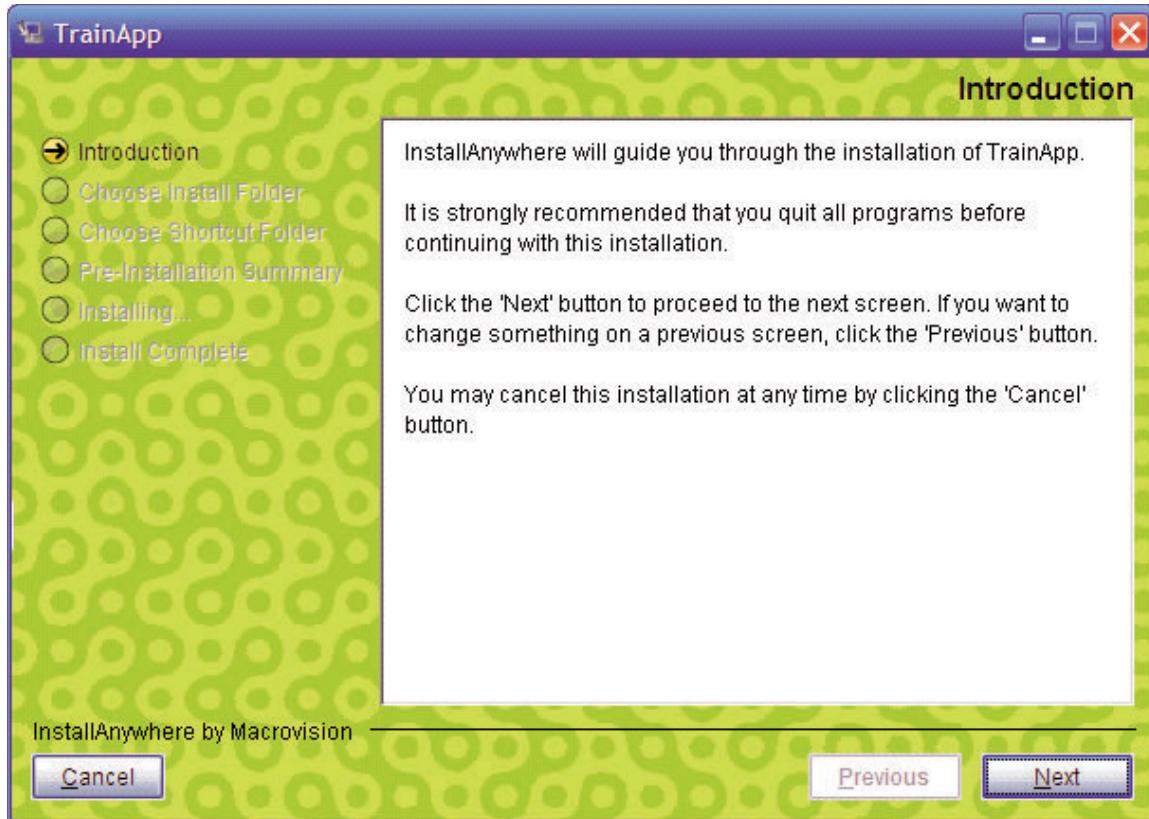


**Figure B-26:** Splash Screen Using TrainAppSplash.png

## Changing the Background Image

Also in the **Installer UI > Look & Feel** task, browse for TrainAppBackground.png as the installer background image. To ensure the panel labels are legible on the light-colored background, it may be desirable to set the **Titles, Labels, and InstallAnywhere Logo color** setting to 000000 (black).

After building the project, the panels should appear similar to the following at run time.



**Figure B-27:** Installer Using TrainAppBackground.png

# Chapter 6 Exercises

## Using the Basic Installer Organization

While component architecture is covered in other parts of the course, you have advanced enough to delve into the basics of installer organization.

In this next exercise, you will extend the OfficeSuite installer to employ InstallAnywhere Features and Install Sets.

1. Open your OfficeSuite Project in the InstallAnywhere Advanced Designer.
2. Click **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, you are going to add some auxiliary files so you have a little more to work with in the installer.
  - Click the **Add Files** button.
  - 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, you will 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 `OfficeSuiteSourceFiles` directory should now be empty.  
You should now have three separate directories under the root of your installation: `OfficeSuite2000`, `ImagesandDocs`, and `UninstallerData`.
6. Rename the `OfficeSuite2000` and `ImagesAndDocs` directories. 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.

## Magic Folders

In this exercise, you will 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.

For this example, use a project other than OfficeSuite.

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, you will be choosing a magic folder that will result in the file being installed to a location that matches the file name. This will enable you 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. Browse to the **Organization > Install Sets** task.

A default InstallAnywhere project contains two Install Sets—Typical and Minimal. You will now 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.

7. 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 you are ready to assign features to the **Install Sets**. In this simple example, you have only two features: Application and Documentation.

8. 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 will meet your needs, you will not have to add any additional features; however the process by which they are added is identical to that used for **Install Sets**.

9. Add a description for the features you will use in this example.

Now you will assign the features to the **Install Sets**. You have defined three distinct sets; two have only one feature, and one has both of the features.

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



**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.

10. 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 you have all the files and actions assigned properly and the organization is complete. However, you have not presented a method to the end user by which they can alter the choice of **Install Sets**. In its current configuration, your installer will simply use the default install set.

11. In order to offer your user a choice, you need to add a panel to the **Pre-Install** that will allow them to select.

- a. Browse back to the **Pre-Install** task, where you will 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 select **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 the current example, it is suggested that you engage this option so that you may see the results.

12. You are now ready to build your first Organized Installer. To build the 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.

## Examining the TrainApp Project Organization

In the **Organization** task of your **TrainApp** project, you can modify properties of the install sets, features, and components used in your project. For example, in the **Organization > Install Sets** task, you can select an install set and change the description displayed to the end user. For example, you can select the **Typical** install set and change the beginning of the description from “The most common product features...” to “The most common \$PRODUCT\_NAME\$ features...”, and the product name will be spliced into the description at run time.

At development time, the customization appears as follows.

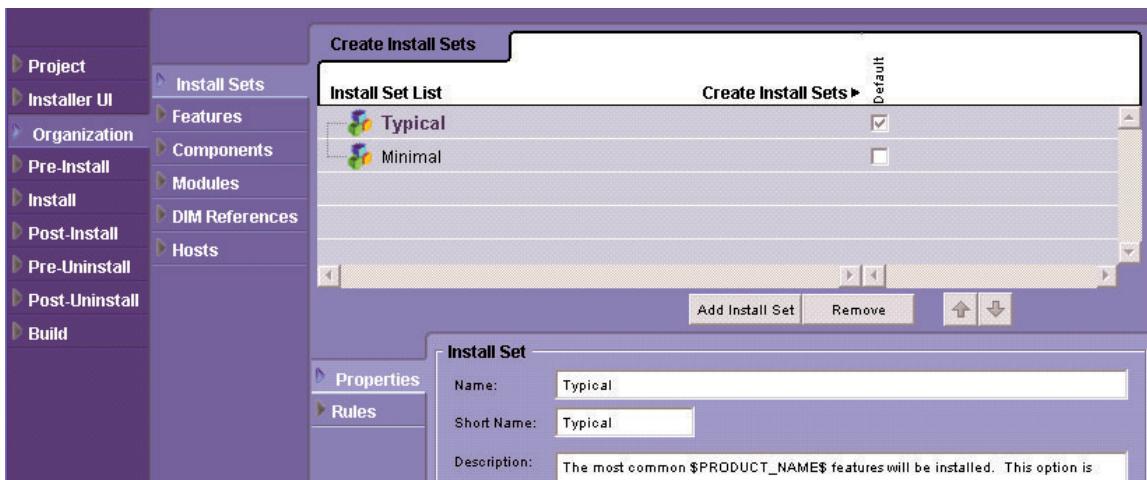


Figure B-28: Modifying Install Set Description

At run time, the changed description appears as follows. To ensure that the **Choose Install Set** panel is displayed at run time, verify that the **Choose Install Sets** panel is present in the **Pre-Install** task, adding it if necessary.

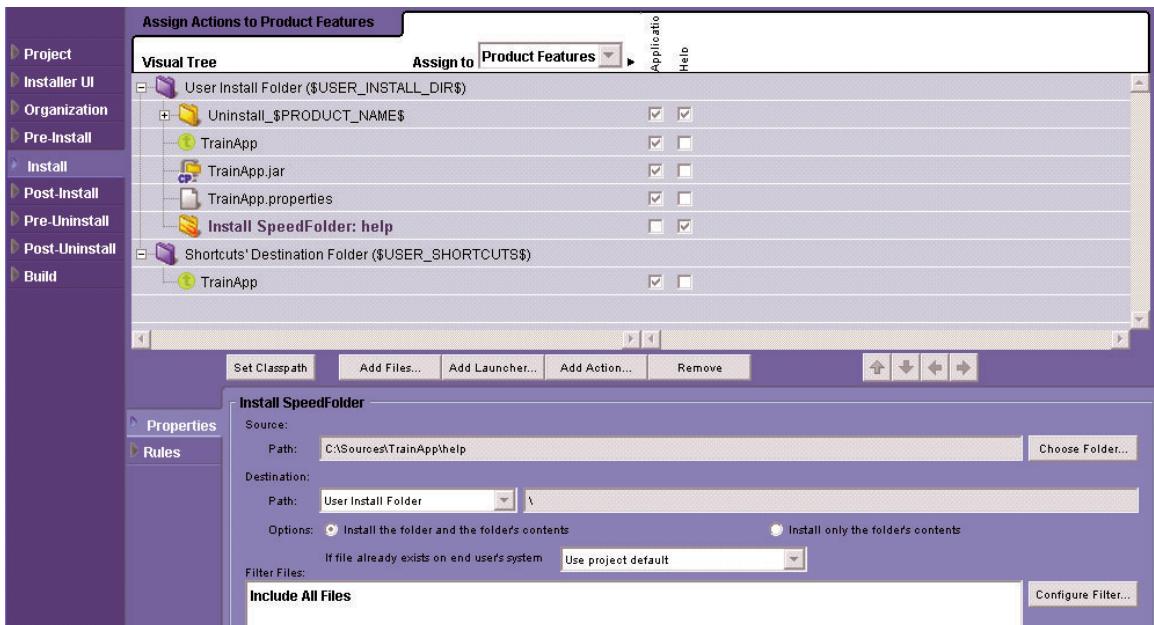


Figure B-29: “Typical” Description at Run-time

## Working with SpeedFolders

To duplicate a directory structure of files from the source machine to a target system, you can use a SpeedFolder. For example, **TrainApp** includes a directory of help files, and instead of statically adding the individual files, you can define a SpeedFolder that builds the current contents of the directory structure into your installer at build time, and installs those contents at run time.

To create a SpeedFolder, navigate to the **Install** task, click the **Add Action** button. Select **Install SpeedFolder**. In the Visual Tree, the blank **Install SpeedFolder: help** action should appear; in its customizer, select the **TrainApp** help directory as the source.



**Figure B-30:** Installing SpeedFolders

As described in the manual, you can click **Configure Filter** to modify which files and folders are included and excluded, or preview the files that will be included.

In this view you can also control which files, shortcuts, and SpeedFolders are included with each feature. For this example, you can specify that **TrainApp.jar** and **TrainApp.properties**, along with the **LaunchAnywhere** executable and shortcut, are associated with the **Application** feature; you can then associate the help SpeedFolder with the **Help** feature. If you do so, at run time an end user who selects the Custom install set—assuming you have enabled the **Custom** install set in the **Choose Install Sets** panel in the **Pre-Install** task—can select which features to install.

# Chapter 7 Exercises

## 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 you 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.

## 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 and select **Install SpeedFolder**.
  - c. Click **Add**.
3. Add a **Set System Environment Variable** action with the following values.

<b>Variable Name:</b>	\$PRODUCT_NAME\$_DIR
<b>Set Value to:</b>	\$USER_INSTALL_DIR\$
<b>When setting this Variable:</b>	Append to existing value
<b>Set this Variable for:</b>	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:

<b>Comment:</b>	Set Windows Registry Test
<b>Registry Key:</b>	-HKEY_LOCAL_MACHINE\SOFTWARE\\$PRODUCT_NAME\$
<b>Value Name:</b>	InstallDirectory
<b>Data Type:</b>	String
<b>Data:</b>	\$USER_INSTALL_DIR\$
<b>Uninstall Options:</b>	Remove if value has not changed
5. Add **Show Message Dialog** action. Display the following text.

<b>Variable Name is</b>	\$PRODUCT_NAME\$_DIR
<b>Value is</b>	\$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.

**10.** Define the User Magic Folders that you've used. As the folders must be defined before they're installed, you will define them in the pre-install section of the installer.

**a.** Browse to the **Pre-Install** task.

As you are only exploring the magic folders in this installer, you can remove all the panels from the pre-install. This decision will enable the installer to run more quickly and without interaction.

Now, to define the User Magic Folders, you 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.txt` file.

**c.** Select a location where you have write permissions, and enter that path as the value for the file.

**11.** Add a *Set InstallAnywhere* variable action which you will use to define `$USER_MAGIC_FOLDER_2$`.

For this variable, try using one of the Magic folder variables to help define the path. Use something similar to `$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 it should be used in place of the system-specific file separators.

**12.** 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 where, 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 potentially problematic element 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, refer to the InstallAnywhere help library.

## Creating Installer Logic Using Jump Labels and Actions

In this exercise, you will use the advanced features 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.

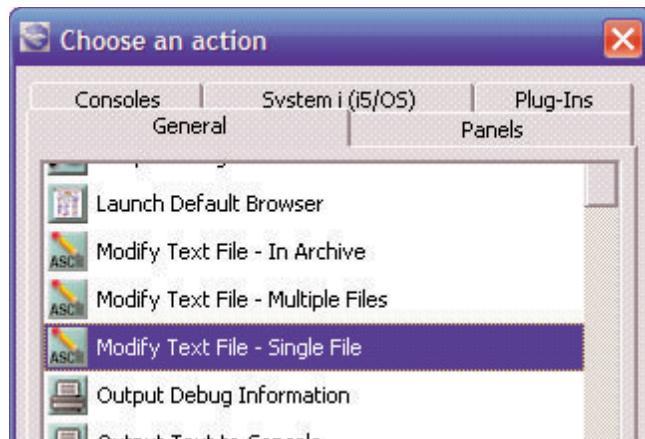
## Modifying TrainApp.properties at Run-time

One of the files being installed with **TrainApp** is **TrainApp.properties**, which is a text file containing the following line:

```
username=%USERNAME%
```

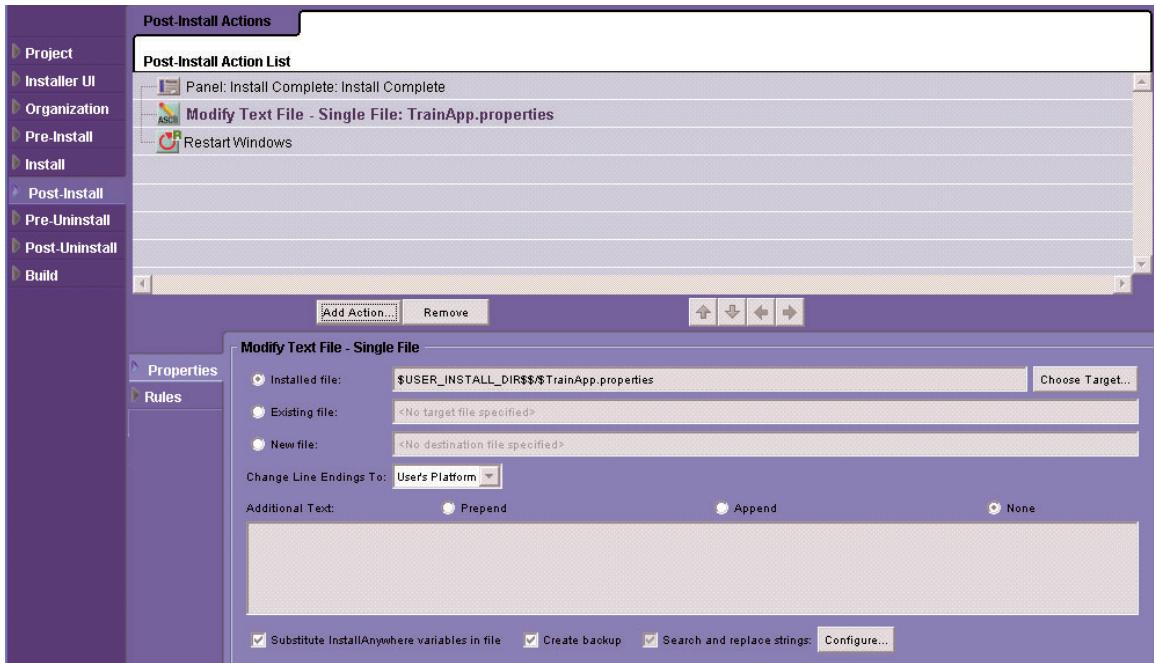
At run time, it may be desirable to replace `%USERNAME%` with the name of the user running the installer, which can be obtained with the Java system property `user.name`.

To modify the file at run time, begin by opening the **Post-Install** task, which is appropriate for a file being installed. Click the **Add Action** button and add a **Modify Text File—Single File** action.



**Figure B-31:** Adding the Modify Text File Action

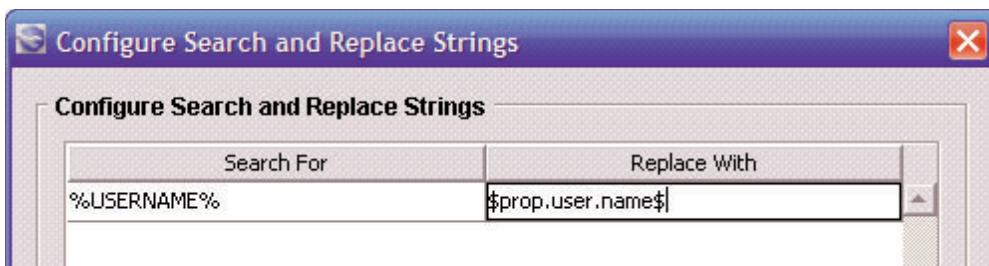
In the customizer for the **Modify Text File** action, select the **Installed File** radio button and then browse for *TrainApp.properties*.



**Figure B-32:** Selecting the Installed File

To search and replace contents of the file, select the **Search and replace strings** option at the bottom of the customizer and click **Configure**; in the configuration, enter **%USERNAME%** as the search string and enter **\$prop.user.name\$** as the replacement text.

As you have seen, InstallAnywhere expands **\$prop.user.name\$** to the value of the Java system property **user.name**.



**Figure B-33:** Configuring the Search and Replace Option

In the main action customizer, you can also clear the **Create backup** option to prevent a copy of the original file called *TrainApp.properties.backup* from being created.

After you build and run the installer, browse for the installation directory and open *TrainApp.properties*. The file contents should now be similar to the following:

```
username=TrainingUser
```

As always, uninstall **TrainApp** after you have verified that the installer worked as expected.

Naturally, a simpler technique is to place `$prop.user.name$` directly in the properties file, and ensure the **Substitute InstallAnywhere** variables in file option is selected. If you have time, modify the text file to use the property directly, and delete the search and replace instructions.

# Chapter 9 Exercises

## 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.

# Chapter 10 Exercises

## Building a Console-Enabled Installer Using OfficeSuite

In this exercise you will build a console-enabled version of the OfficeSuite installer.

1. Open InstallAnywhere and create a new project named **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 (preferably three) installation options.
  - c. Add the necessary panels in pre-install to enable the user's choice of installation options. Choose **Install Sets**.
  - d. Add launchers and setup the classpath for OfficeSuite.
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 is 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.

## Building a Silent-Mode Installer

1. Open InstallAnywhere, and create a new project.
2. Set up the project with the following.
  - 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 the 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.

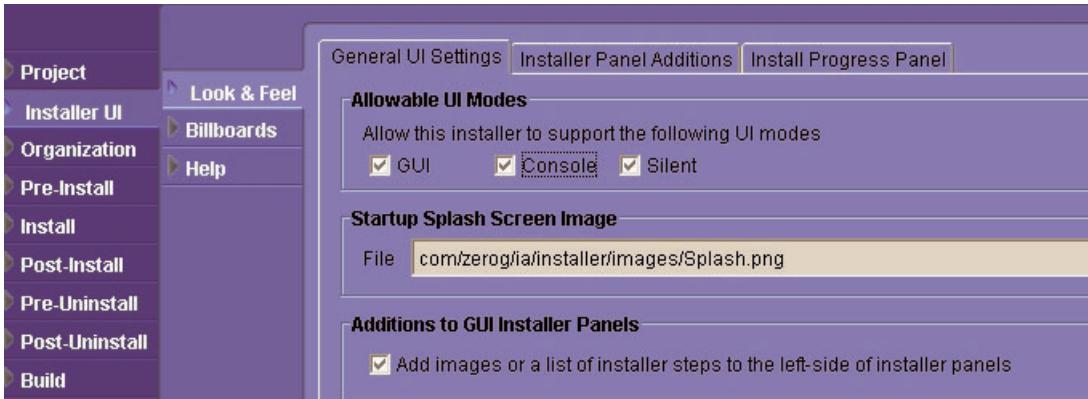
Did the installer install where you expected?



**Note:** For more information on silent installers, refer to *Silent and Console Installers* in the *InstallAnywhere Help Library*.

## Installing TrainApp in Silent Mode

To be able to install **TrainApp** in silent mode, open the **Installer UI > Look & Feel** task and ensure that **Silent** is selected in the **Allowable UI Modes** setting.



**Figure B-34:** Enabling Silent Mode

If necessary, rebuild the project using the **Build** task, and then open the build location by clicking **Open in Explorer**.

First, create a response file by running the command:

```
install.exe -r
```

This runs your installer normally, but saves input such as the installation location and shortcut location into a file called `installer.properties`. (For this example, install **TrainApp** to a non-default location such as `C:\Program Files\DifferentLocation`.) When the installation has finished, open `installer.properties`. Its contents should be similar to the following:

```
# Thu Apr 31 25:00:00 CDT 2008
# Replay feature output
#
# -----
# This file was built by the Replay feature of InstallAnywhere.
# It contains variables that were set by Panels, Consoles or Custom Code.

#Choose Install Folder
#
USER_INSTALL_DIR=C:\\\\Program Files\\\\DifferentLocation

#Choose Shortcut Folder
#
USER_SHORTCUTS=C:\\\\Documents and Settings\\\\All Users\\\\Start Menu\\\\Programs\\\\TrainApp
```

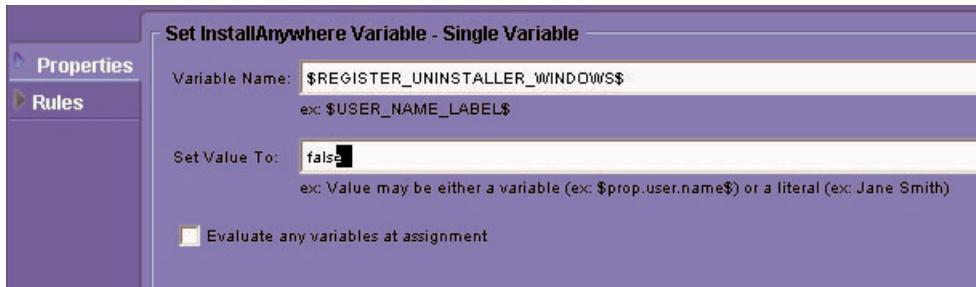
Next, uninstall **TrainApp**, and then perform the silent installation with the following command:

```
install -i silent -r installer.properties
```

When the silent installation has finished, browse to the installation directory specified in the response file to verify that the product has been installed to the correct location.

## Hiding TrainApp from Add or Remove Programs

InstallAnywhere gives you control over the level of integration between the installed product and the target operating system. For example, to prevent **TrainApp** from being listed in the **Add or Remove Programs** panel on Windows target systems, you can insert a **Set InstallAnywhere Variable** action into your **Pre-Install** task, setting `$REGISTER_UNINSTALLER_WINDOWS$` to false.



**Figure B-35:** Suppressing the Add or Remove Programs Listing

After you rebuild the project and run the installer, **TrainApp** will not be listed in **Add or Remove Programs**. Note that the uninstaller is still being created; to uninstall **TrainApp**, browse for the installation directory, open its `Uninstall_TrainApp` directory, and launch the uninstaller executable.

To re-integrate **TrainApp** with **Add or Remove Programs** for future installations, delete the **Set InstallAnywhere Variable** action or set the property's value to `true`.

If you have other platforms available, you can specify integration with native registries on Linux and AIX using the **Project > Platforms > UNIX** task.

# Chapter 11 Exercises

## 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 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.

# Chapter 12 Exercises

## Building TrainApp from the Command Line

To build the **TrainApp** project from the command line, you can open a command prompt and run a command similar to the following (closing the InstallAnywhere interface if it is still open):

```
"C:\Program Files\Macrovision\InstallAnywhere 2008\build.exe"  
"C:\Projects\TrainApp\TrainApp.iap_xml"
```



**Note:** The InstallAnywhere path and the project path will need to be adjusted for your development system.

If the build runs properly, you should see system settings and progress messages scroll by, ending with a breakdown of build tasks by time.

If you have time, examine the arguments you can pass to the builder by running:

```
build.exe -?
```

Next, try adding and removing platforms or changing the output media type using the command-line arguments.

# Chapter 13 Exercises

## Adding FirstCustomCodeAction to TrainApp

The source code to the **FirstCustomCodeAction** example is provided with your course files. Following the description in the manual, compile the action, package it into a .jar file, and insert it into your project with an **Execute Custom Code** action. Verify that the message box appears at run time.

If you have time, package `FirstCustomCodeAction` as a plug-in by adding the `customcode.properties` file to the .jar file. Delete the original action, add a copy of the action as a plug-in, and verify that it works the same as before.

## Setting Variables from Custom Code

Modify the `FirstCustomCodeAction` class to create a custom variable called `$NOW$`, whose value should be set to the current time. (You can use `java.util.Date` to compute the current time, as done in the `DisplayTime` class.) To create and set the variable, add the following to the install method (where `ip` is the name of the `InstallerProxy` argument):

```
ip.setVariable("NOW", new java.util.Date().toString());
```

Recompile the class and repackage it into a .jar file; verify that the action works by (for example) using the expression `$NOW$` in a panel that follows the updated action.

## Using the InstallerResources Class

The `InstallerResources` class provides methods for obtaining information about install sets, required and available disk space, special directory locations, and Java virtual machines available at run time.

For this exercise, create a custom code action class called `ShowVMs` that calls the `getJavaVMList` method of the `InstallerResources` class, which returns a vector of available VMs on the target system. The custom code action should display the JVMs on the console or in a message dialog.

As described in Chapter 13, including the following code in the install method of a `CustomCodeAction` class will give you a handle to the `InstallerResources` class:

```
InstallerResources ir =
(InstallerResources)ip.getService(InstallerResources.class);
```

One approach might be the following:

```
import com.zerog.ia.api.pub.*;
import java.util.*;

public class ShowVMs extends CustomCodeAction
{
    public void install(InstallerProxy ip)
    {
        // get the InstallerResources handle
        InstallerResources ir =
            (InstallerResources)ip.getService(InstallerResources.class);

        Vector vecJVM = ir.getJavaVMList();

        // convert the vector into a newline-separated string
        // that can be displayed at run time
        String allJVMs = "JVMs on this system:\n\n";
        for (int i = 0; i < vecJVM.size(); i++)
        {
            allJVMs += (String)vecJVM.elementAt(i) + "\n";
        }
        javax.swing.JOptionPane.showMessageDialog(
            null, allJVMs);
    }

    public String getInstallStatusMessage()
    {
        return "Checking VMs...";
    }

    public void uninstall(UninstallerProxy up) { }
    public String getUninstallStatusMessage() { return "..."; }
}
```

After compiling and packaging the class and including it in your project, you should see a message similar to the following at run time:

```
JVMs on this system:
C:\Program Files\Java\jdk1.5.0_06\bin\java.exe
C:\Program Files\Java\jdk1.5.0_06\jre\bin\java.exe
C:\Program Files\Java\jre1.5.0_11\bin\java.exe
C:\Program Files\Java\jre1.6.0_03\bin\java.exe
```

If you have time, modify your custom code action class to call other methods of the `InstallerResources` class, such as `getRequiredDiskSpace` and `getAvailableDiskSpace`.

## Logging Errors from Custom Code

The `CustomError` class provides methods for writing information to the installer log file. For this exercise, modify `FirstCustomCodeAction` to use the `appendMessage` and `log` methods to write a message to the log file.

As described in Chapter 13, you obtain a `CustomError` object using code similar to the following:

```
CustomError myErr = (CustomError)ip.getService(CustomError.class);
```

To write a message to the log file, add code similar to the following in the `install` method of `FirstCustomCodeAction`:

```
CustomError myErr = (CustomError)ip.getService(CustomError.class);
myErr.appendMessage("Calling FirstCustomCodeAction...");
// call log method last to write data to log file
myErr.log( );
```

To ensure the installer creates the installation log, open the **Project > Info** task and select the check box labeled **Generate** install log during installation.

After rebuilding and repackaging the class, run the installer and then browse for the installation log. It should contain the message specified in the `appendMessage` method, "Calling FirstCustomCodeAction..."

## Creating a Custom Rule

You can detect an installer's user-interface mode at run time by reading the value of the `$INSTALLER_UI$` variable, which takes one of the values SWING, CONSOLE, or SILENT. For this exercise, create a custom code rule called `SwingModeRule` that succeeds only if the installer is running in GUI (Swing) mode.

One such implementation is the following.

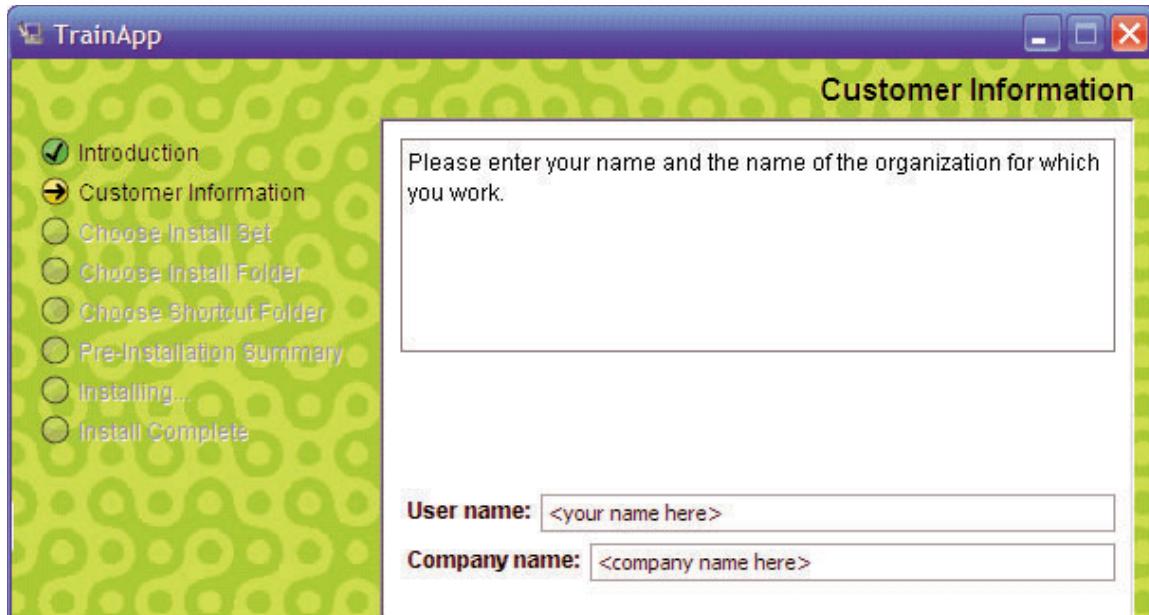
```
import com.zerog.ia.api.pub.*;
public class SwingModeRule extends CustomCodeRule
{
    public boolean evaluateRule( )
    {
        // ruleProxy is static variable to assist with getting variable values
        String uimode = ruleProxy.substitute("$INSTALLER_UI$");
        // succeed if $INSTALLER_UI$ = "SWING"
        return uimode.equalsIgnoreCase("SWING");
    }
}
```

Compile the class and package it into a .jar file, and then attach the rule to an action using the **Evaluate Custom Rule** option in the action's Rules customizer. Verify that the custom code rule succeeds by running the installer in GUI mode and console mode, and note whether the action runs in GUI mode but not console mode.

# Chapter 14 Exercises

## Adding a Get User Input Panel to TrainApp

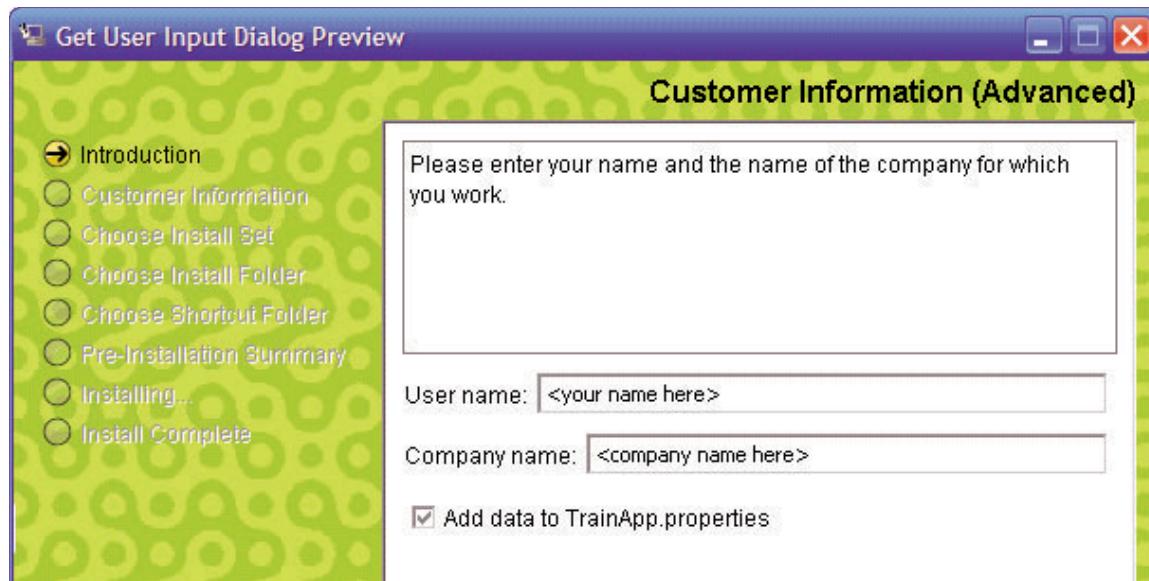
As described in the manual, create a simple **Get User Input** panel that prompts the user for a user name and company name (similar to the following figure), storing the values in InstallAnywhere variables. The panel should be displayed immediately after the **Introduction** panel.



**Figure B-36:** Get User Input Panel in the Installer

If you have time, change the **Modify Text File** action used earlier to write the user name and company name entered by the user to the installed file `TrainApp.properties`.

If you have extra time, use an advanced **Get User Input** panel to design a panel similar to the following, where the panel contains an additional check box asking whether to save the user input to `TrainApp.properties`.



**Figure B-37:** An Advanced Get User Input Panel

You should then modify the **Modify Text File** action to use a rule based on the checkbox variable to determine whether to write the data to `TrainApp.properties`.

## Interacting with a Custom Panel's Navigation Buttons

As described in Chapter 14, you can control the enabled state and visibility state of a custom panel's navigation buttons (the **Next**, **Previous**, and **Cancel** buttons) using methods in the `GUIAccess` class.

For this example, modify `BlankCustomCodePanel` so that it uses the `GUIAccess` class to disable the panel's **Previous** button and hide the panel's **Cancel** button. (Recall that the methods for hiding a button are new to InstallAnywhere 2008 Value Pack 1.)

To manipulate the navigation buttons, enter the following code in the `panelIsDisplayed` method of your custom code panel class:

```
// obtain GUIAccess handle...
GUIAccess guiacc = (GUIAccess)ccpp.getService(GUIAccess.class);
// ...disable Previous button...
guiacc.setPreviousButtonEnabled(false);
// ...and hide Cancel button
guiacc.setExitButtonVisible(false);
```

After recompiling and repackaging the custom code panel class, the panel should appear at run time with the specified buttons disabled and hidden.

# Chapter 15 Exercises

## Localizing the TrainApp Installer

In the **TrainApp** project, add support for one or two non-English locales using the **Project > Locales** task. For example, the following figure shows the French locale being added to the project. (Note the locale abbreviations displayed after the locale names—*fr* for French, for example. These abbreviations will be needed when creating localized resource bundles.)



Figure B-38: Locale List

When you rebuild and run the installer as an end user, you will be prompted for the locale to use for installation.

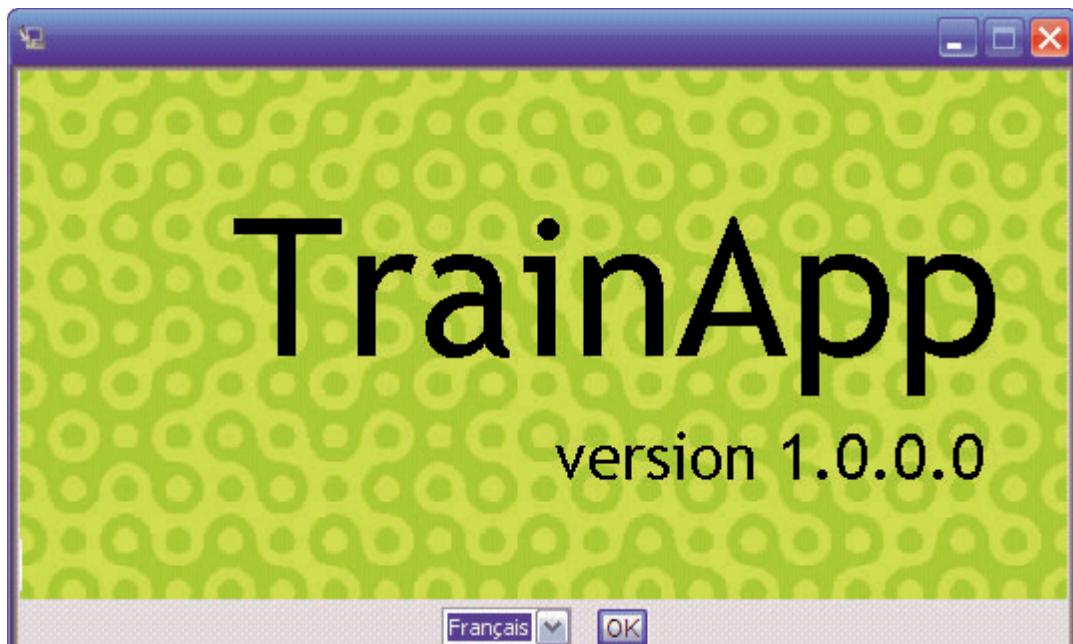
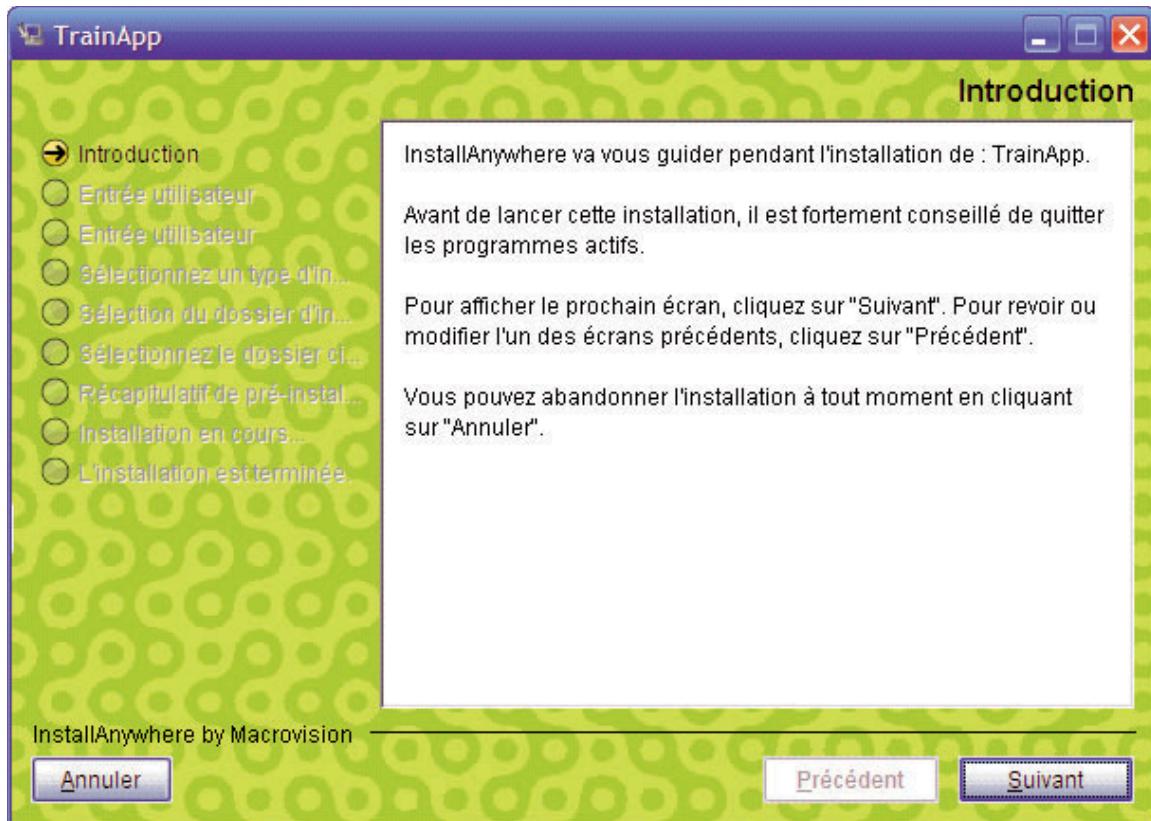


Figure B-39: Installer with Language Prompt

At run time, the text on predefined panels will be displayed using the selected locale.



**Figure B-40:** Installer with French Localization



# Index

## A

Action Groups 81, 86  
 Add Action 296  
 Add Files 19, 255  
 Add or Remove Programs 136  
 Adding Directories with SpeedFolders 75  
 Adding Merge Modules 150  
 Advanced Action Methods 197  
 Advanced Designer 3, 265–267  
 Advertised Variables 144, 150  
 AIX Software Vital Product Database 34  
 Ant 165, 168  
 Ant Build Integration 168  
 Apache Ant 165  
 Application Aervers 97  
 Authoring Environment 3  
 Authoring Environment Requirements 3  
 Auto Populate 279  
 available features 6

## B

Background Images 50, 53  
 Best Practices for Components 69  
 Best Practices for Features 70  
 Best Practices for Install Sets 70  
 bidirectional text 240  
 Billboards 13, 50, 54, 279–280  
 Build 113, 270–271  
 Build Log 47  
 Build Options 149  
 Build Targets 39  
 Building Releases 39  
 Build-properties File 167  
 Build-time Merge Modules 148  
 Bundled Virtual Machine 41

## C

CD-ROM Installers 45  
 Changing  
     Localized Text 244  
 Choose Alias, Link Shortcut Folder 266  
 Choose Icons 279  
 Choose Install Folder 266  
 Classpath 15  
 Clean Components 66  
 Collaboration 12, 153, 158  
 command-line  
     build options 166  
 command-line builds 165  
 Component 27  
 Component Dependencies 68  
 Components 61, 69  
 Console actions 81  
 Console Installation 6  
 Contents view 155  
 Creating a DIM 154  
 Creating Debug Output 35  
 Custom Code 171, 243  
 Custom Errors 199  
 Customizer 269–270  
     Common Properties 100

## D

Database Server 97  
 Database Servers 97  
 Debug 35, 113–114  
 Debugging 35, 109–112, 114  
 Debugging Custom Code 196  
 Default Translations 244  
 Dependencies 158  
 Dependency 67  
 Dependency Failed Message 68  
 Deploy WAR/EAR Archive 99  
 Deployment 2  
 Design Time Merge Module 151  
 Design-time Merge Modules 148  
 digital certificates 167  
 digitally signed installers 167  
 DIM 156, 159, 162  
 DIM References 160

DIMs 153  
Distribution 43, 270  
dynamic build targets 39  
Dynamic Merge Modules 149  
Dynamic Text 241

## E

Eclipse 12, 153, 155, 159  
Editions 6  
Enterprise Edition 6, 114, 237  
Execute Target File 269  
Exit Codes 166

## F

Features 65  
features available 6  
File Service 192  
Find Component in Registry 68  
FLEXnet Connect 163

## G

general actions 81  
Graphics 13

## H

headless systems 165  
Help 103  
hsqldb.jar 196  
  
I  
i5/OS 34  
Image Settings 101  
Images 278  
Install Actions 15  
Install Merge Module 144  
Install Progress Panel 278  
Install Set 64  
Install Sets 59, 62  
Install Anywhere 2008 Value Pack 1 features 34, 138, 166, 185–186, 216, 224, 309  
InstallAnywhere Collaboration 6, 12, 153  
InstallAnywhere installer 7

InstallAnywhere Variables 74, 114  
InstallAnywhere Web Installer Applet 7  
Installation 2  
Installation Log 199  
Installer Panel Additions 101–102, 278–279  
Installer UI 101  
Installers  
    Internationalizing 237  
    Localizing 237  
InstallShield 156  
InstallShield Collaboration 153  
Install-time Merge Modules 148  
Internationalizing Installers 237  
ISMP 191, 194–195  
ISMP file service 192  
ISMP Manifests 145

## J

Jakarta 168  
Java Virtual Machine  
    International 245  
Java VM 36  
JDBC drivers 98  
JUnit tests 35

## L

Label Settings 101–102  
Language Resource Properties 245  
LaunchAnywhere 20, 36, 104, 109, 112, 256, 267  
LaunchAnywhere executables 112  
LAX  
    LaunchAnywhere executable 112, 267  
List of Installer Steps 101, 279  
Locales 35  
locales 35, 113, 237–239, 242  
Locales subtask 35  
Localizable Elements 245  
Localization 241–242  
localization 35  
Localized Text 244  
Localizing  
    Best Practices for, 243  
    Custom Code 243  
    Custom Installer Labels 242

Resources 242  
 Localizing Installers 237  
 logging 199

## M

Mac OS X 104  
   debugging 111  
   troubleshooting 111  
 Mac OS X Installer 111  
 Mac OS X Support 33  
 Magic Folders 74  
 Manifest File Format 78  
 Manifest Files 78  
 Merge Module Customizer 153  
 Merge Module Size 149  
 Merge Modules 143, 148  
 Meta Information 161  
 Microsoft Visual Studio 153  
 Modifying  
   Localized Text 244  
 multiplatform deployment 2  
 Multiplatform Installation 2

## N

native2ascii 239

## O

Optional Installer Arguments 36  
 Organization 61  
 Organizing Features and Components 69

## P

Panel Actions 81  
   Actions 101  
 platform abbreviations 166  
 Platform Support 32  
 Platforms 32  
 plug-in actions 81  
 Plug-ins 84  
 Post-Install 86, 269–270  
 Pre-Install 86, 266

pre-installation 12  
 Project 16, 27–28  
 Project Wizard 3, 15  
 Pure Java Installer 111

## Q

Querying the ISMP Registry 195

## R

RAIR 34  
 Red Hat Package Management Support 34  
 Registered Application Information Repository 34  
 requirements 3  
 response files 6  
 RPM Support 34  
 Rules 35, 64, 66, 83, 269

## S

security 167  
 Security Service 193  
 Setting the Classpath 21, 257  
 Shared Components 67–68  
 silent installation 6  
 Source Path 117–118  
 Source Paths 116–117  
 SpeedFolders 75  
 Splash Screen 50  
 SQL 97  
 Standard Component 67  
 Standard Edition 6, 237  
 Static Text 241  
 stderr 35, 110, 112  
 stdout 35, 110, 112  
 SWPD 34  
 System i (i5/OS) Support 34  
 System Utility Service 193

## T

Target Environments 3–4  
 task  
   Java 36

Rules 35  
Templates 143, 145  
Timestamp 30  
Translations 244  
Types of Components 67

## U

UI Modes 51  
UI Settings 54  
Uninstaller 101, 278  
Unit Tests 159  
Unix  
    debugging 111  
    LaunchAnywhere 104  
    with VM 40  
Unix installation 12  
unsigned code 167  
User Account Control 32

## V

Virtual Machine 36  
    Bundled 41  
Visual Studio 153  
Visual Tree 78  
VM Packs 41  
VM Selection 42

## W

WAR/EAR deployment 97  
Web Installers 44  
Win32 installer 110  
Windows 104, 269  
Windows NT 110  
    troubleshooting 110  
Windows Vista  
    command line builder 165  
Writing Custom Errors 199