Creating Installers Using InstallAnywhere

Legal Information

Book Name:	Creating Installers Using InstallAnywhere
Part Number:	IA-MANUAL-ENG-2025-R1
Revision Date:	June 2025

Copyright Notice

Intellectual Property

Restricted Rights Legend

Contents

1 **Introduction** 13

Planning Your Installation 15

Multiplatform Installation 16

Introduction to InstallAnywhere 16

Requirements 17

Authoring Environment Requirements 17

Target Environment 19

Supported Java Virtual Machines 22

Editions 22

Premier Edition 22

Professional Edition 23

Premier with Virtualization and Cloud Edition 23

The InstallAnywhere End-User Experience 23

2 **Starting a Project Using the InstallAnywhere Project Wizard** 31

Creating a New Project 32

Setting Project Information 34

Installing Tasks 35

Add Files 35

Choose Main Class 37

Setting the Classpath 39

Building the Installer 40

Testing the Project 41

3 **Introduction to the Advanced Designer** 43

Installation Planning 43

Installation Goals 44

Working with Advanced Designer 44

Defining Installer Projects and the Product Registry 45

Product Registry 45

Installer Identification and Version ... 46

File Settings: Timestamps and Overwrite Behavior ... 46

Installed File Timestamps ... 47

Default Overwrite Behavior ... 47

Platforms. ... 49

Locales ... 52

Rules Before the Pre-Install Task ... 52

Creating Debug Output ... 53

Virtual Machines ... 54

Optional Installer Arguments ... 54

Java ... 55

Quick Quiz ... 56

4 Building Releases ... 59

Build Targets ... 59

VM Packs ... 61

VM Selection ... 62

Distribution ... 64

Web Installers ... 65

CD-ROM/DVD Installers ... 66

Burning CD-ROM Installers ... 66

Merge Modules and Templates ... 67

Build Log ... 68

5 Basic Installer Customization ... 69

Customizing the Installer Look and Feel ... 70

Installer UI Modes ... 70

Splash Screens ... 71

GUI Panel Additions ... 72

Installer Panel Additions ... 73

Panel Labels ... 73

Using Jump Actions and Logic ... 74

Background Images ... 74

Panel Action Settings ... 75

Image Settings ... 75

Label Settings ... 76

Help ... 77

Frame UI Settings ... 78

Billboards ... 78

Help ... 78

Conditional Logic ... 78

Quick Quiz ... 81

6 Installer Organization ... 83

Install Sets, Features, and Components ... 83

Install Sets ... 83

Features ... 84

Components ... 85

Using the Organization Task **85**
Install Sets 86
Features 89
Components 90
Types of Components 91
Shared Components 91
Component Dependencies 92
Organizing Your Features and Components **92**
Best Practices for Components 93
Best Practices for Features 93
Best Practices for Install Sets 93
Adding Files to Your Project **94**
Adding Individual Files 94
Magic Folders 96
Adding Directories with SpeedFolders 97
Specifying Source Files with Manifest Files 100
Quick Quiz **101**

7 Introduction to Advanced Actions and Panel Actions 103
Adding an Action **104**
Action Availability by Task 105
General Actions 107
Action Groups 107
Pre-Install/Uninstall and Post-Install/Uninstall Actions 108
Examples of Common Actions **109**
Set InstallAnywhere Variable Action 109
Display Message Panel 111
Set System Environment Variable 112
Display HTML Panel **113**
Application Servers and Database Servers **115**
Common Properties **117**
LaunchAnywhere Executable 118
Quick Quiz **119**

8 Customizing the Uninstaller 121
About Uninstaller Customization **121**
Benefits of Customizing the Uninstaller 122

Overview of the Uninstall Task 122
Adding Uninstall Categories and Actions to the Uninstall Task **123**
About Adding Uninstall Categories and Actions 123
Adding Categories and Actions 124
Available Uninstall Actions 125
Assigning Actions to Product Features and Components 125
Preventing an Uninstall Category From Being Uninstalled **126**
Removing the Uninstall Action from an Uninstall Category 126
Assigning a Rule 126

9 Implementing Maintenance Mode 127
Enabling and Configuring Maintenance Mode **128**
Enabling Maintenance Mode 129

About Maintenance Mode Action Groups 131
Pre-Install Task 132
Pre-Installation Action Group *132*
Add Features Action Group *133*
Repair Installation Action Group *133*
Pre-Uninstall Task 134
Pre-Uninstallation Action Group *134*
Remove Features Action Group *135*
About Check Running Mode Rules 135
Specifying Instance Management Options 135
About the Uninstaller / Maintenance Mode Launcher 139
Maintenance Mode End User Experience 141
Initial Experience 141
Adding a Feature User Experience 142
Removing a Feature User Experience 144
Repairing a Product User Experience 145
Uninstalling a Product User Experience 146
Maintenance Mode User Experience on a Machine With Multiple Installed Instances 147

10 Applying Basic and Intermediate Development Concepts 149
Building the Installer 150
Debugging InstallAnywhere Installers 151
During Installer Development 151
Debugging a Windows Installer 152
Debugging a Unix/Linux Installer 153
Debugging a Mac OS X Installer 153
Debugging a Pure Java Installer 153
Debugging LaunchAnywhere Executables 153
Reviewing Debug Information 154
Using Output Debug Information Actions 155
Debugging Using the Display Message Panel 155

11 Source and Resource Management 157
How Source Paths Work 157
Adding Source Paths 158
InstallAnywhere Preferences 159
System Environment Variables 160
Updating the Location of Files and Resources 160
Managing Source Files 161
The Resource Manager 161
Adding Source Path Management Capability to Your Installer Project 161
Enabling/Disabling Source Paths 162
Default Source Paths 162
Adding Source Paths 162
InstallAnywhere Preferences 162
Set System Environment Variables 163
Using Source Paths in Your Project 163
Switching Access Path Locations 164
Quick Quiz 164

12 Advanced Installer Concepts 167
Console Installers 168

Silent Installers **171**
Using a Response File 171
Configuring Variables Used in Response Files 172
Uninstallation **174**
Feature-Level Uninstallation 175
Uninstaller Integration with the Target System 176
Setting Installation Rollback Options **178**

13 Creating and Editing Build Configurations 181

About Build Configurations **181**
Benefits of Using Build Configurations 182
Default Build Configurations for New and Migrated Projects 182
Build Configuration Options in the InstallAnywhere User Interface 183
Using Build Configuration Tags 184
Locale Settings 185
Creating a New Build Configuration **186**
Creating Migrated Build Configurations **186**
Adding a Build Configuration to the Project Build. **187**
Using Tags to Customize Build Configurations **188**
Creating New Tags 190
Assigning Tags to Project Elements 191
Associating Tags to Build Configurations 192
Searching for Tags 192

14 Advanced Organizational Concepts 193

Using the Find Component in Registry Action **193**
Merge Modules and Templates **194**
Merge Modules 194
Templates 196
Merge Module Types 197
Design-Time Merge Modules 197
Build-Time Merge Modules 197
Install-Time Merge Modules 197
Dynamic Merge Modules 198
Creating Merge Modules and Templates 198
Build Options 198
Merge Module Size 198
Creating Merge Modules as Read Only 198
Advertised Variables 199
Adding Merge Modules 200
Importing a Design-Time Merge Module **201**
Advanced Topic: Importing ISMP Manifests **203**
InstallAnywhere Collaboration and DIMs **204**
Creating a DIM 205
Unit Tests 210
Consuming a DIM in an InstallAnywhere Project 211
FLEXnet Connect **214**
Quick Quiz **215**

15 Integrating InstallAnywhere with Automated Build Environments 217

InstallAnywhere Command-Line Build Facility **217**

Builder Arguments. 218
Advanced Topic: Project Automation 219
Digitally Signed Installers 220
Ant Build Integration 220
Task Definition 221
Task Settings 221

16 Custom Code 223

Writing Custom Code 224
Custom Code Actions 224
Custom Code Action Example 225
Writing the Action Code 225
Compiling and Packaging the Action Code 226
Adding the Custom Action to Your Project. 227
Packaging Custom Code as a Plug-in 228
Adding Plug-in Help 230
Using Custom Code Actions in Uninstall 230

Accessing Properties and Variables 231
Custom Variables and Response Files 232
Variables and Proxy Classes. 233
substitute 233
getVariable 233
Example: Displaying the Current Time 233
Custom Code Rules 235
Reading the InstallAnywhere Product Registry 237
InstallShield MultiPlatform Services 239
Compiling and Packaging Code That Uses ISMP Services 240
Adding ISMP Code to Your Project 240
File Service 240
Security Service 242
System Utility Service 243
Advanced Topic: Service Differences Between ISMP and IA. 244
Advanced Topic: Querying the ISMP Registry 244
Debugging Custom Code 245
Advanced Action Methods 245
Writing Custom Errors in the Installation Log 247
Working with Java Native Interface (JNI) 248
JNI Example 1: Basic Text Display 250
Write and Compile the Java Class 250
Use javah to Create a Header 250
Creating the Native Method 250
Running the Program 251
JNI Example 2: Calling a Windows DLL from a Custom Code Action 251
Building the Distribution Archive 253
Accessing the Library at Run Time. 253
Potential Problems Using JNI 254
Additional Resources 255
Quick Quiz 255

17 Custom Panels and Consoles 257

User Input Panels 257

Using the Simple Get User Input Panel 259
Variables and the Get User Input Panel 261
Using the Advanced Get User Input Panel 261
Custom Code Panels 264
Creating a Custom Code Panel 265
Packaging, Adding, and Testing the Custom Code Panel 266
Adding Controls to the Custom Code Panel 267
Laying Out the Custom Code Panel 268
Working with Variables 270
Buttons and Action Listeners 272
Custom Code Consoles 273
Custom Code Console Actions 274
Additional Console Controls 276
Quick Quiz 277

18 Localizing and Internationalizing Installers 279
Bidirectional Text Support 282
Dynamic and Static Text 283
Localization and the Internationalized Designer 283
Specific Localization Concerns 284
Localizing Resources 284
Localizing the Splash Screen 284
Localizing Custom Installer Labels 286
Using External Resource Bundles 286
Localizing Custom Code 287
Best Practices for Localizing 288
Changing Localized Text 288
Modifying Localized Text 288
Changing Default Translations Provided in Language Packs 289
Localizable Elements 289
Localizing Items in the Installer 289

A Installation Planning Worksheet 293
Target Platforms 293
Installer Target 294
Deployment Media 294
Application Type 294
Java-Specific Options 295
Installation Needs 295
List Configuration That Must Be Done to the Target Platform 296
What Information Must Be Collected from the End User? 296
Team Development Options 297
Source Paths 297
Uninstall Options 298
Pre-Uninstall 298
Post-Uninstall 299

B Exercises 301
Chapter 2 Exercises: Starting a Project Using the InstallAnywhere Project Wizard 303

Creating the OfficeSuite Project 303
Setting Project Information 305
Installing Tasks 306
Adding Files to the Project 306
Choosing the Main Java Class for Starting the Application 308
Setting the Classpath 310
Building the Installer 311
Testing the Project 312
Creating the TrainApp Project 315
Chapter 3 Exercises: Introduction to the Advanced Designer 319
Building an Installer with the Advanced Designer 319
Reviewing the Pre-Install Actions 319
Defining the Installation Tasks 322
Adding a LaunchAnywhere Executable to the Install Task 323
Select the Add Launcher Button 323
Customize the Launcher 325
Set the InstallAnywhere Classpath 326
Post-Install Actions 327
Building the Installer 332
Testing the Installer 334
Examining the TrainApp Project in the Advanced Designer 335
Chapter 4 Exercises: Building Releases 337
Building the TrainApp Project 337
Specifying VM Requirements 339
Chapter 5 Exercises: Basic Installer Customization 340
Exploring Look and Feel 341
Exploring the Installer UI Tasks 341
Making Additions to the GUI Installer Panels 344
Selecting Billboards 346
Exploring the Help Options 348
Rebuilding the Project 350
Using Installer Rules 350
Using Rules to Control Visual Elements 353
Managing Installer Flow Based on End-User Input 359
Step One: Retrieving End-User Input 359
Step Two: Applying the End User’s Choice 363
Changing the Splash Screen 363
Changing the Background Image 364
Creating Installer Rules 366
Chapter 6 Exercises: Installer Organization 367
Using the Basic Installer Organization 367
Magic Folders 371
Examining the TrainApp Project Organization 374
Working with SpeedFolders 376
Modifying the Default Uninstaller 377
Chapter 7 Exercises: Introduction to Advanced Actions and Panel Actions 380
Using Panels in Pre-Install 380
Using Install Task Actions 384
Creating Installer Logic Using Jump Labels and Actions 387
Modifying TrainApp.properties File at Runtime 387

Chapter 8 Exercises: Customizing the Uninstaller 390

Chapter 9 Exercises: Implementing Maintenance Mode 392
- Enabling Maintenance Mode 392
- Setting Instance Management Options 401

Chapter 11 Exercises: Source and Resource Management 402
- Creating Source Paths 403
- Managing Resources in the InstallAnywhere Project File 404

Chapter 12 Exercises: Advanced Installer Concepts 405
- Building a Console-Enabled Installer Using OfficeSuite 405
- Building a Silent-Mode Installer 407
- Installing TrainApp in Silent Mode 407
- Hiding TrainApp from Add or Remove Programs 408
- Setting Installation Rollback Options 409

Chapter 13 Exercises: Creating and Editing Build Configurations 412
- Creating a New Build Configuration 412
- Using Build Tags to Customize Build Configurations 414

Chapter 14 Exercises: Advanced Organizational Concepts 419
- Creating Merge Modules 419

Chapter 15 Exercises: Integrating InstallAnywhere with Automated Build Environments 419
- Building TrainApp from the Command Line 419

Chapter 16 Exercises: Custom Code 420
- Adding FirstCustomCodeAction to TrainApp 420
- Setting Variables from Custom Code 421
- Using the InstallerResources Class 421
- Logging Errors from Custom Code 423
- Creating a Custom Rule 423
- Using Other ISMP Services 424
- Additional Exercises 424

Chapter 17 Exercises: Custom Panels and Consoles 425
- Adding a Get User Input Panel to TrainApp 425
- Interacting with a Custom Panel's Navigation Buttons 427

Chapter 18 Exercises: Localizing and Internationalizing Installers 427
- Localizing the TrainApp Installer 427
- Translating Custom Strings 429

Index 431

1 Introduction

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.

Information in this Training Guide is organized in the following chapters:

Table 1-1 • InstallAnywhere Training Guide

#	Chapter	Description
1	**Introduction**	Provides an overview of InstallAnywhere, lists the system requirements, and provides a demonstration of the InstallAnywhere end-user experience.
2	**Starting a Project Using the InstallAnywhere Project Wizard**	Explains how to use the Project Wizard authoring mode to create a simple installer project.
3	**Introduction to the Advanced Designer**	Explains how to use the Advanced Designer authoring mode to create a more complex installer project.
4	**Building Releases**	Explains how to build releases from the graphical InstallAnywhere environment.
5	**Basic Installer Customization**	Describes how to customize an installer project, including the look and feel, splash screens, panels, background images, billboards, help, and conditional logic.
6	**Installer Organization**	Describes the different levels of design that make up your installation project, and demonstrates the different ways to add files to your installation.

Table 1-1 • InstallAnywhere Training Guide (cont.)

#	Chapter	Description
7	**Introduction to Advanced Actions and Panel Actions**	Covers the use of some of InstallAnywhere's more advanced and more useful installer actions.
8	**Customizing the Uninstaller**	Explains how to use the **Uninstall** task to customize the Uninstaller by adding, removing or changing some of the uninstall actions.
9	**Implementing Maintenance Mode**	Explains how to implement Maintenance Mode in an installer so that end users are able to add or remove features to previously installed products as well as repair broken installations.
10	**Applying Basic and Intermediate Development Concepts**	Contains a single, unstructured exercise where you build an installer with specific features, actions, and panels, and use the debugging features at various stages of development.
11	**Source and Resource Management**	Covers how to effectively manage the availability of source files and resources, focusing on how source paths work, managing source files, and creating source paths.
12	**Advanced Installer Concepts**	Describes additional options that you can use when building an installer including: console-mode, silent installations, and customized uninstallers.
13	**Creating and Editing Build Configurations**	Each InstallAnywhere project can have multiple Build Configurations, each representing how the installer will be built for particular set of platforms, files, build distributions, JVMs, locales, and other settings. This chapter explains how to create and modify Build Configurations.
14	**Advanced Organizational Concepts**	Describes advanced organizational topics including how to use the Find Component in Registry action, using merge modules and templates, importing ISMP manifests, and using Collaboration and DIMs.
15	**Integrating InstallAnywhere with Automated Build Environments**	Explains how to use the InstallAnywhere command-line build facility, how to digitally sign installers, and how to perform Ant build integration.
16	**Custom Code**	Explains how to create your own custom components using the Custom Code API.
17	**Custom Panels and Consoles**	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.
18	**Localizing and Internationalizing Installers**	Explains how to localize an InstallAnywhere project.

Table 1-1 • InstallAnywhere Training Guide (cont.)

#	Chapter	Description
A	**Installation Planning Worksheet**	Provides a worksheet to use to assist in gathering information during installation planning.
B	**Exercises**	Provides exercises that reinforce the concepts covered in the training and test your knowledge of InstallAnywhere's features.

Planning Your Installation

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 EMC 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

InstallAnywhere has the following system requirements for systems running the InstallAnywhere authoring environment:

Table 1-2 • Authoring Environment System Requirements

Item	Description
RAM	256 MB; 512 MB preferred
Color	Minimum of 8-bit color depth (256 colors)
Resolution	Minimum 1024 x 768 resolution

Table 1-2 • Authoring Environment System Requirements (cont.)

Item	Description		
Operating System	InstallAnywhere runs on the latest versions of these operating systems, fully updated with the most recent patches and service packs:		
	Windows	●	Windows 10 (x86 and x64)
		●	Windows 8.1 (x86 and x64)
		●	Windows Server 2012 R2 (x64)
		●	Windows 8 (x86 and x64)
		●	Windows Server 2012 (x64)
		●	Windows 7 (x86 and x64)
		●	Windows Server 2008 R2 (x64)
		●	Windows Vista
		●	Windows Server 2008 (x86 and x64)
	Macintosh	●	OS X El Capitan (10.11) with Oracle Java 7 or 8
		●	OS X Yosemite (10.11) with Oracle Java 7 or 8
		●	OS X Mavericks (10.9.2) with Oracle Java 7 or 8
		●	OS X Mountain Lion (10.8) with Oracle Java 7 or 8
		●	OS X Lion (10.7.3) with Oracle Java 7
	Linux	●	Red Hat Enterprise Linux 7/7.1
		●	Red Hat Enterprise Linux 6.x (desktop and server editions: x86 and x64)
		●	Red Had Enterprise Linux 5.x (x86 and x64)
		●	OpenSUSE Linux 11.x, 12.x, and 13.1 (x86 and x86)
		●	SUSE Linux Enterprise 11 (SP2 and SP3; x64) and 12 (x64)
		●	Linux PPC 64-bit (build time only) with Java 6
		●	Ubuntu 15.04 (x64)
		●	Ubuntu 14.x (x64)
		●	Ubuntu 13.x (desktop and serer editions; x86 and x64)
		●	Ubuntu 10.x, 11.x, and 12.x (x86 and x64)
		●	Fedora 18, 19, and 20 (desktop editions)

Table 1-2 • Authoring Environment System Requirements (cont.)

Item	Description	
Operating System (Copntinued)	**Solaris**	Solaris 9 and 10 (SPARC)
	HP-UX	HP-UX 11i (PA-RISC)
	AIX	AIX 5.2, 5.3, 6.1, and 7.1 (Power/PowerPC)
	***Note** • A Japanese-localized version of InstallAnywhere Premier Edition is 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 Premier Edition. Localizations for 9 languages are included with Professional Edition.*	

***Note** • A language-localized version of InstallAnywhere Premier Edition for Japanese developers is 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 Premier Edition. Localizations for 9 languages are included with Professional Edition.*

Target Environment

The following are the system requirements for systems running InstallAnywhere installers:

Table 1-3 • Target Environment System Requirements

Item	Description
RAM	64 MB free
Color	High color (16-bit color depth)
Resolution	Minimum 640 x 480 screen resolution

Table 1-3 • Target Environment System Requirements (cont.)

Item	Description		
Operating System	Installers run on any version of these operating systems, as long as the operating system supports Java 1.6 or newer: ***Note*** • *InstallAnywhere installers are not supported on beta or on early access releases unless explicitly mentioned.*		
	Windows	●	Windows 10 (x86 and x64)
		●	Windows 8.1 (x86 and x64)
		●	Windows Server 2012 R2 (x64)
		●	Windows 8 (x86 and x64)
		●	Windows Server 2012 (x64)
		●	Windows 7 (x86 and x64)
		●	Windows Server 2008 R2 (x64)
		●	Windows Vista
		●	Windows Server 2008 (x86 and x64)
		●	Windows XP (x86, x64, Itanium 2, and AMD-64)
		●	Windows Server 2003 (x86, x64, Itanium 2, and AMD-64)
	Macintosh	●	OS X El Capitan (10.11) with Oracle Java 7 or 8
		●	OS X Yosemite (10.11) with Oracle Java 7 or 8
		●	OS X Mavericks (10.9.2) with Oracle Java 7 or 8
		●	OS X Mountain Lion (10.8) with Oracle Java 7 or 8
		●	OS X Lion (10.7.3) with Oracle Java 7

Table 1-3 • Target Environment System Requirements (cont.)

Item	Description	
Operating System (Continued)	**Linux**	● Red Hat Enterprise Linux 7/7.1 ● Red Hat Enterprise Linux 6.x (desktop and server editions: x86 and x64) ● Red Had Enterprise Linux 5.x (x86 and x64) ● OpenSUSE Linux 11.x, 12.x, and 13.1 (x86 and x86) ● SUSE Linux Enterprise 11 (SP2 and SP3; x64) and 12 (x64) ● Linux PPC 64-bit (build time only) with Java 6 ● Ubuntu 15.04 (x64) ● Ubuntu 14.x (x64) ● Ubuntu 13.x (desktop and serer editions; x86 and x64) ● Ubuntu 10.x, 11.x, and 12.x (x86 and x64) ● Fedora 18, 19, and 20 (desktop editions)
	Solaris	● Solaris 11 (x86 and SPARC) ● Solaris 9, 10 (x86, SPARC, and AMD-64)
	HP-UX	HP-UX 11i (Itanium 2 and PA-RISC)
	AIX	AIX 5.2, 5.3, 6.1 and 7.1 (Power/PowerPC)
	IBM	● i5/OS (OS/400) on System i - V5R3 and V5R4 (Enterprise Edition only), IBM i 6.1, and IBM i 7.1 ● z/OS
	Other	● FreeBSD ● Other Linux and Unix operating systems (POSIX-compliant shell required)

Supported Java Virtual Machines

InstallAnywhere supports the following Java virtual machines:

Table 1-4 • Supported Java Virtual Machines

Manufacturer	Versions
Sun	1.6.x, 1.7.x
IBM	1.6.x, 1.7.x, 1.8.x
HP	1.6.x, 1.7.x
Oracle	1.7.x, 1.8.x

The InstallAnywhere installer installs Java 1.7_60 VM packs. Any Java virtual machine can be bundled with an installer ensuring that the target system meets the minimum requirements for both the installers and your applications. To download additional VM packs, go to http://www.flexerasoftware.com/installanywhere/utilities and open the **VM Packs** tab.

Note • *InstallAnywhere installers are not supported on beta or on early access releases of Java.*

Editions

There are three editions of InstallAnywhere: Professional, Premier, and Premier with Virtualization and Cloud.

Each edition is designed to meet product deployment needs for the different types of customers. This manual describes the features available in the Premier Edition.

Premier Edition

Premier Edition provides the ultimate in configuration options, user interaction, and client-server features. It simplifies complex installations and provides maximum developer customization. The Premier Edition is available in English 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 Premier Edition are:

- A localized development environment (English or Japanese)
- 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

Professional Edition

Professional 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.

Premier with Virtualization and Cloud Edition

Premier with Virtualization and Cloud Edition offers all of the features of Premier Edition and adds the ability to build Virtual Appliances for popular Cloud Platforms, as well as the ability to build Docker Containers directly from InstallAnywhere.

For a detailed list of features available in each edition, refer to:

http://www.flexerasoftware.com/products/installanywhere/features.htm

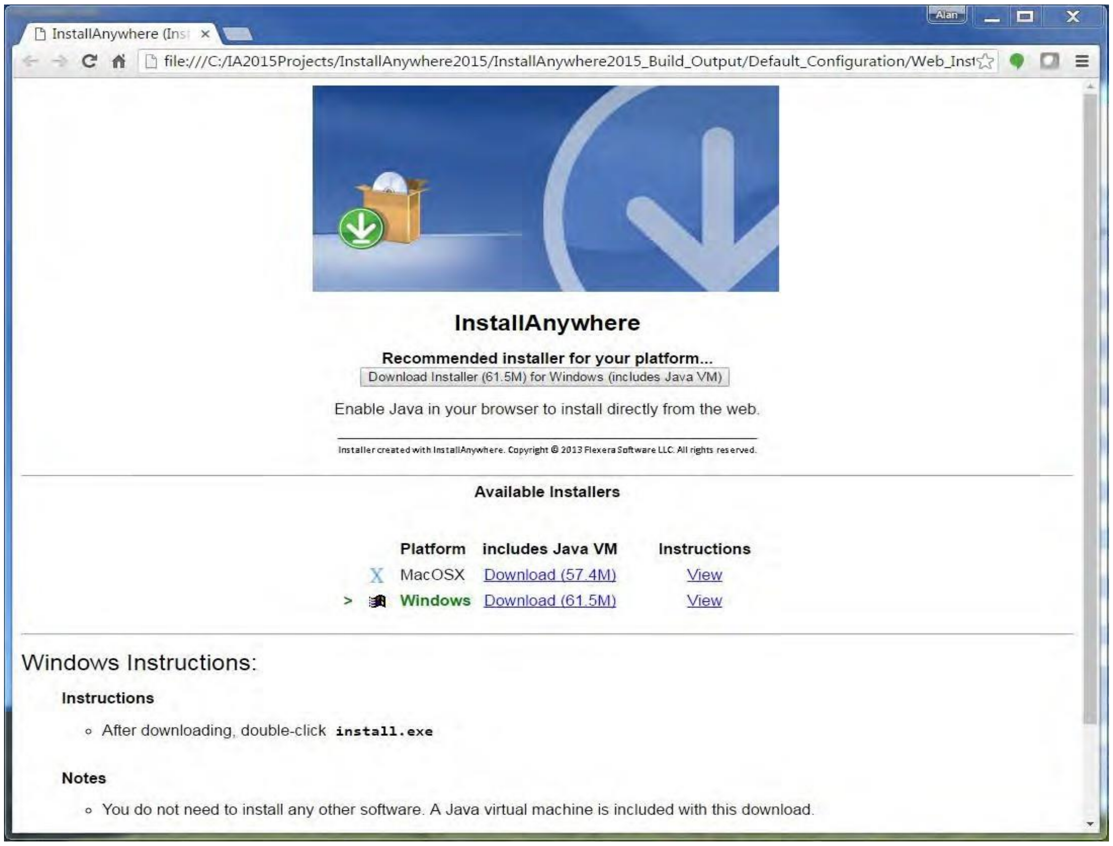
The InstallAnywhere End-User Experience

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.


InstallAnywhere (Inst
file:///C:/IA2015Projects/InstallAnywhere2015/InstallAnywhere2015_Build_Output/Default_Configuration/Web_Inst

InstallAnywhere

Recommended installer for your platform...

Download Installer (61.5M) for Windows (includes Java VM)

Enable Java in your browser to install directly from the web.

Installer created with InstallAnywhere. Copyright © 2013 Flexera Software LLC. All rights reserved.

Available Installers

	Platform	includes Java VM	Instructions
	MacOSX	Download (57.4M)	View
>	Windows	Download (61.5M)	View

Windows Instructions:

Instructions

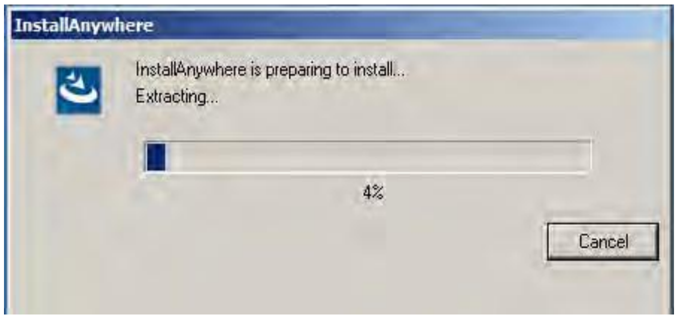
- After downloading, double-click `install.exe`

Notes

- You do not need to install any other software. A Java virtual machine is included with this download.

Figure 1-1: InstallAnywhere Web Install Applet

The progress bar displays the percent of the initialization action complete.



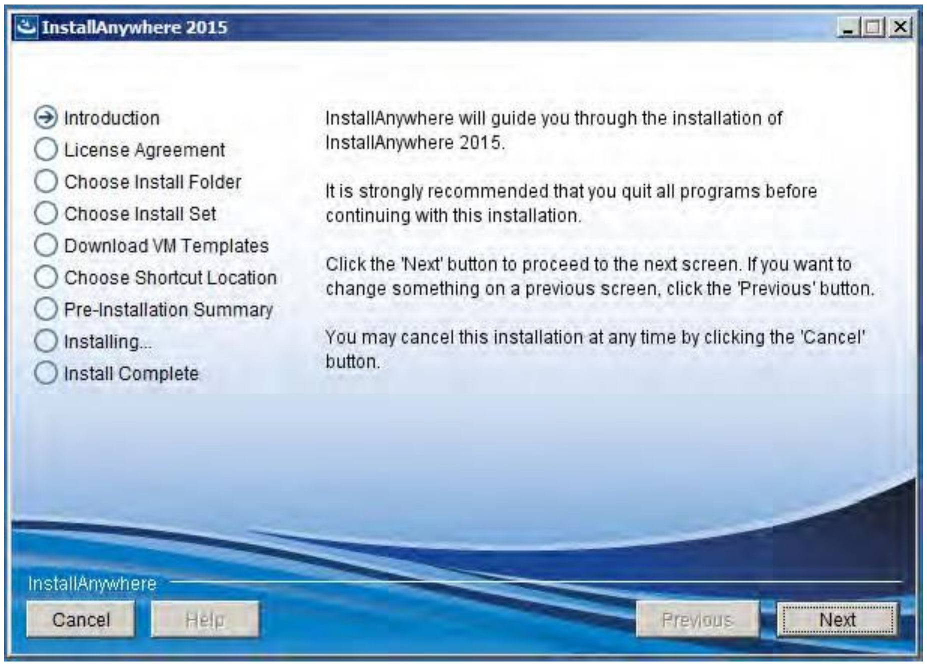
InstallAnywhere is preparing to install...
Extracting...

4%

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.



InstallAnywhere will guide you through the installation of InstallAnywhere 2015.

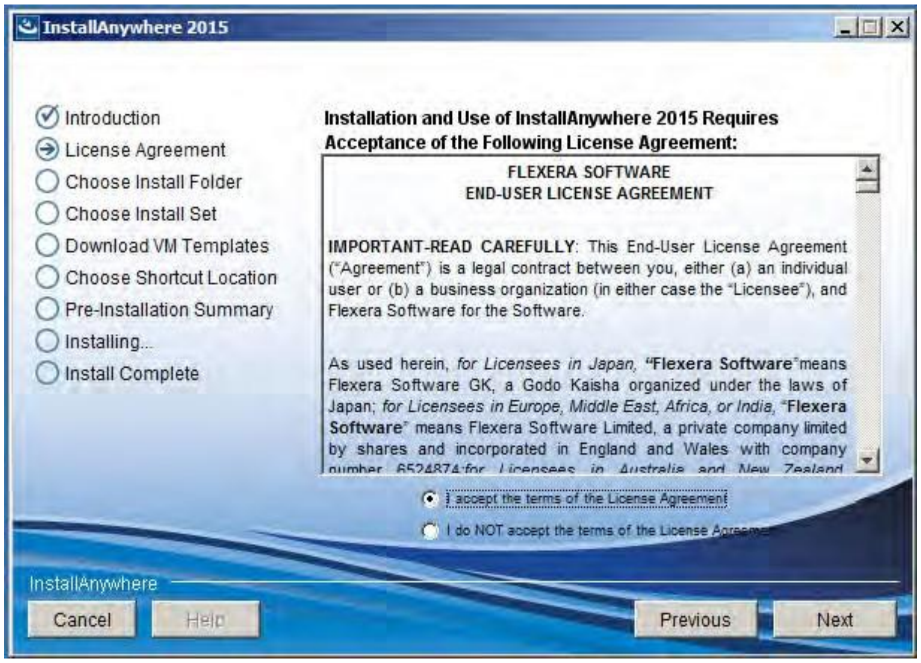
It is strongly recommended that you quit all programs before continuing with this installation.

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

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

Figure 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.



Installation and Use of InstallAnywhere 2015 Requires Acceptance of the Following License Agreement:

FLEXERA SOFTWARE
END-USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This End-User License Agreement ("Agreement") is a legal contract between you, either (a) an individual user or (b) a business organization (in either case the "Licensee"), and Flexera Software for the Software.

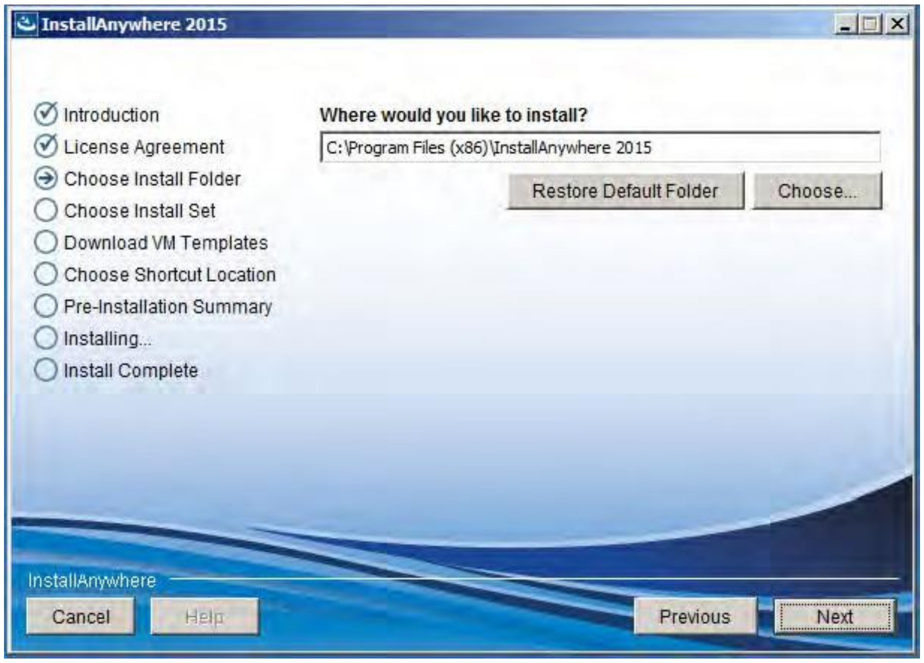
As used herein, *for Licensees in Japan*, **"Flexera Software"** means Flexera Software GK, a Godo Kaisha organized under the laws of Japan; *for Licensees in Europe, Middle East, Africa, or India*, "**Flexera Software**" means Flexera Software Limited, a private company limited by shares and incorporated in England and Wales with company number 6524874;for Licensees in Australia and New Zealand

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

Figure 1-4: License Agreement

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

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

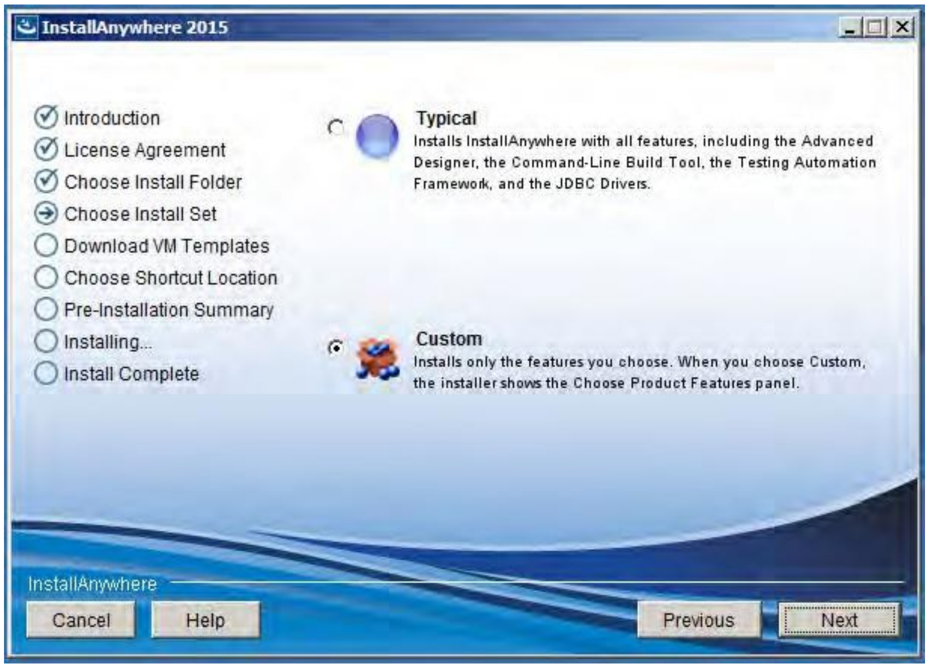


Where would you like to install?

C:\Program Files (x86)\InstallAnywhere 2015

Figure 1-5: Choosing an Installation Path

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

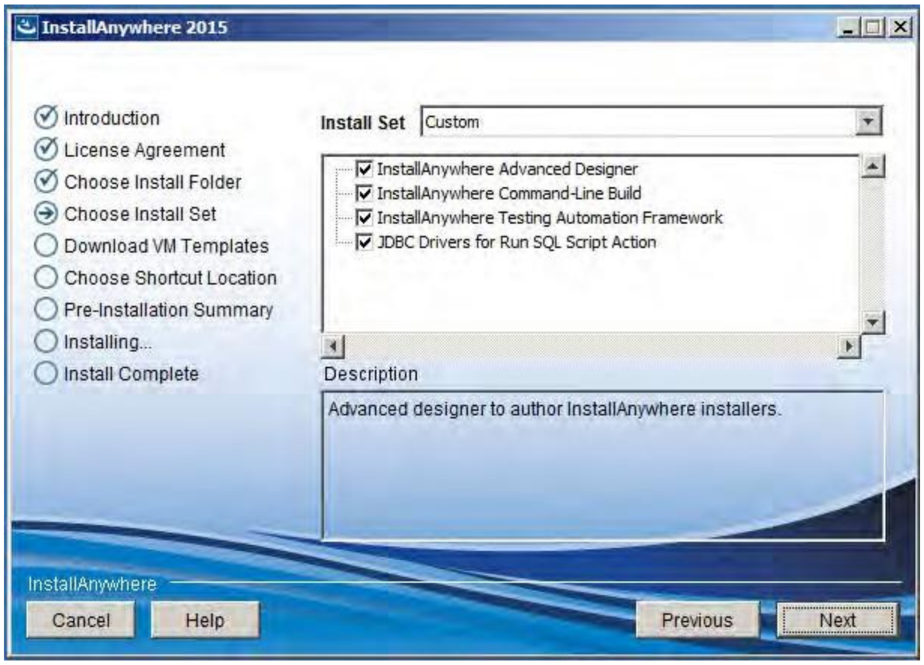


Typical
Installs InstallAnywhere with all features, including the Advanced Designer, the Command-Line Build Tool, the Testing Automation Framework, and the JDBC Drivers.

Custom
Installs only the features you choose. When you choose Custom, the installer shows the Choose Product Features panel.

Figure 1-6: 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.



Install Set: Custom

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

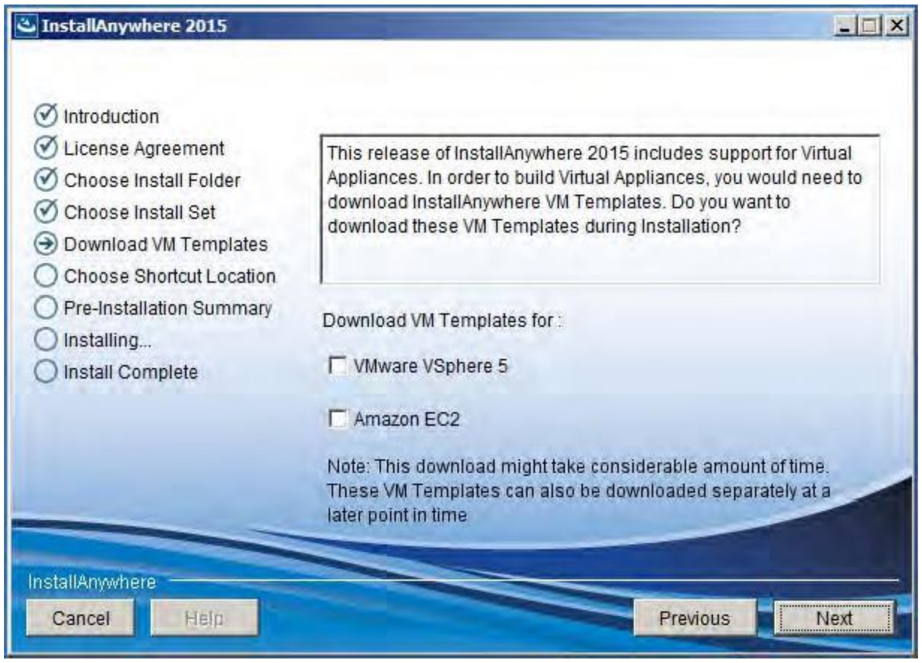
Description

Advanced designer to author InstallAnywhere installers.

Figure 1-7: 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 allows users to download VM Templates for use when building Virtual Appliances (Premier with Virtualization and Cloud Edition only).



This release of InstallAnywhere 2015 includes support for Virtual Appliances. In order to build Virtual Appliances, you would need to download InstallAnywhere VM Templates. Do you want to download these VM Templates during Installation?

Download VM Templates for :

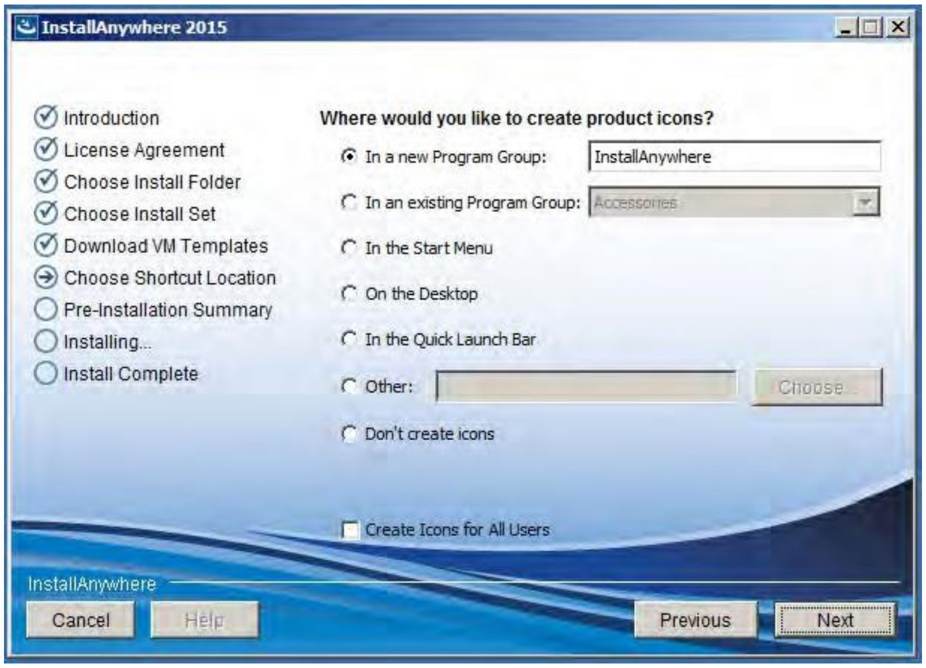
- [ ] VMware VSphere 5
- [ ] Amazon EC2

Note: This download might take considerable amount of time. These VM Templates can also be downloaded separately at a later point in time

Figure 1-8: Downloading Virtual Templates

The InstallAnywhere installer provides the option to download Virtual Templates. If not downloaded at install time, these can be downloaded later.

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



Where would you like to create product icons?

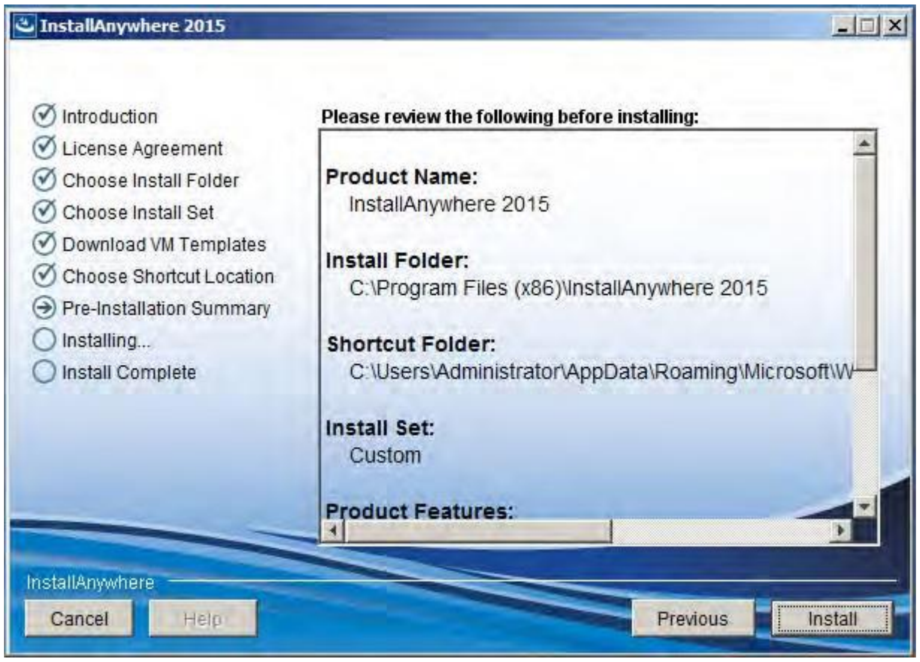
- [✓ In a new Program Group:] InstallAnywhere
- In an existing Program Group: Accessories
- In the Start Menu
- On the Desktop
- In the Quick Launch Bar
- Other:
- Don't create icons

☐ Create Icons for All Users

Figure 1-9: 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.



Please review the following before installing:

Product Name:
InstallAnywhere 2015

Install Folder:
C:\Program Files (x86)\InstallAnywhere 2015

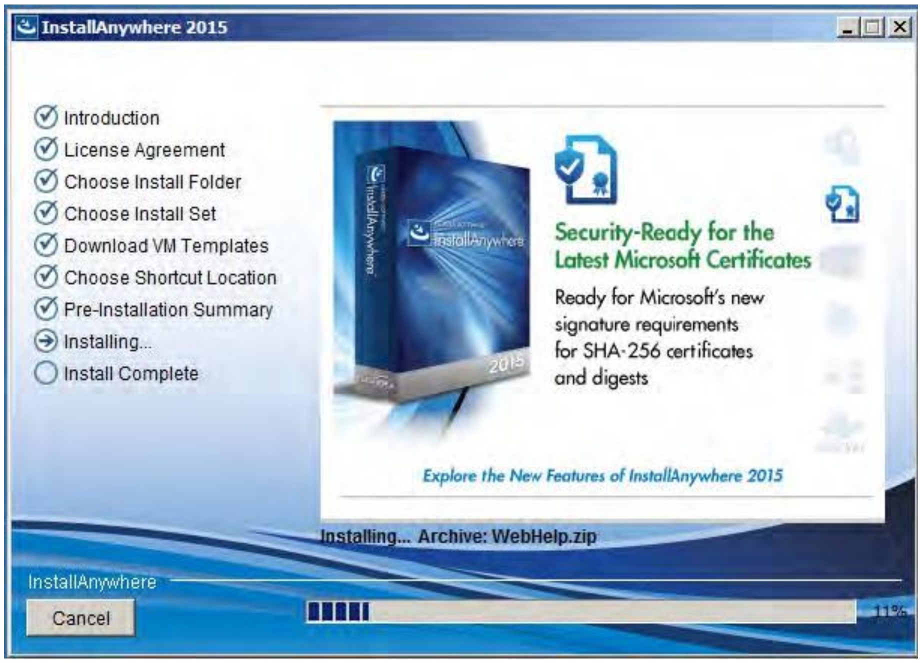
Shortcut Folder:
C:\Users\Administrator\AppData\Roaming\Microsoft\W

Install Set:
Custom

Product Features:

Figure 1-10: Pre-installation Summary Now

the installation of files begins.

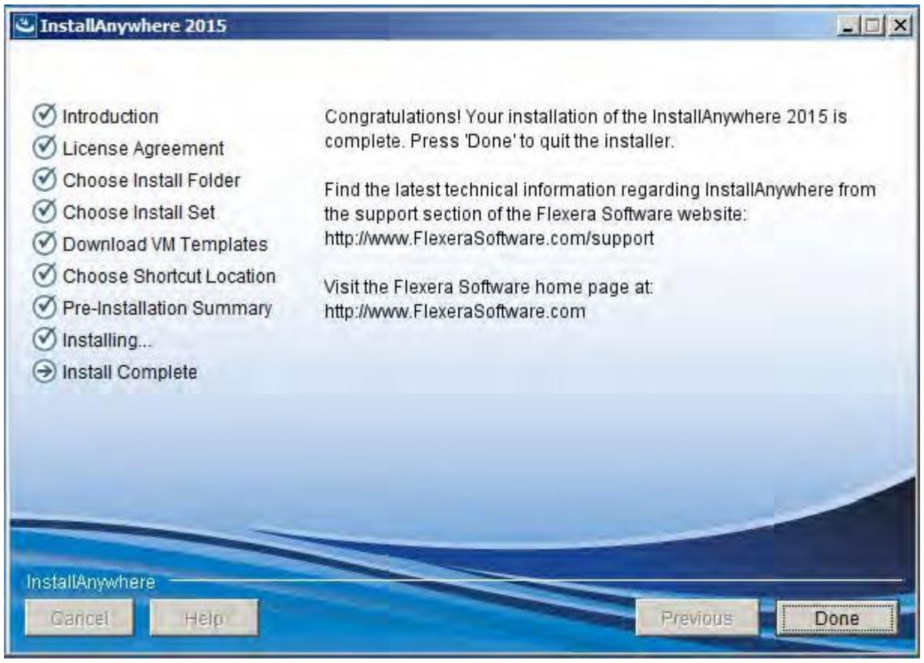


Installing... Archive: WebHelp.zip

Figure 1-11: 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, InstallAnywhere 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.



Congratulations! Your installation of the InstallAnywhere 2015 is complete. Press 'Done' to quit the installer.

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

Visit the Flexera Software home page at:
http://www.FlexeraSoftware.com

Figure 1-12: 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.

***Note** • 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.*

The general process for developing an InstallAnywhere project is:

Table 2-1 • Steps in Creating an InstallAnywhere Project

Step	Description
Step 1	Create a new project
Step 2	Set project information
Step 3	Add Pre-Install actions
Step 4	Define Install tasks
Step 5	Add Post-Install actions
Step 6	Customize the Uninstall task
Step 7	Configure the project uninstaller
Step 8	Build the project

Table 2-1 • Steps in Creating an InstallAnywhere Project

Step	Description
Step 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.

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:

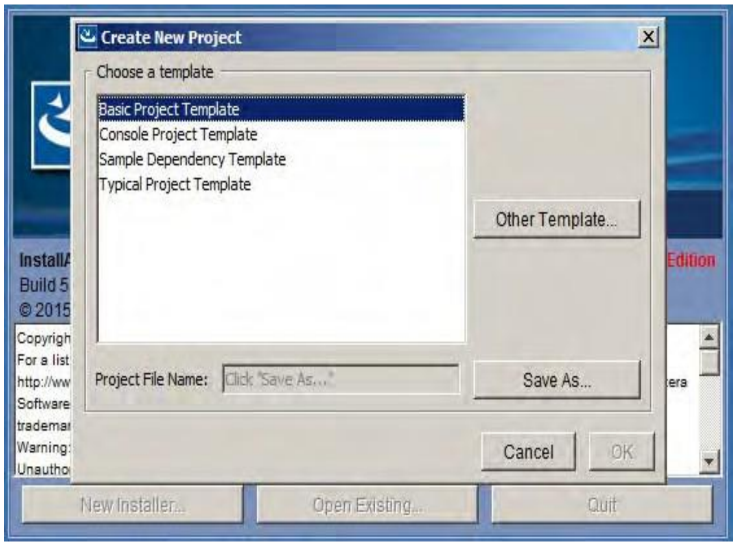
- Creating a New Project
- Setting Project Information
- Installing Tasks
- Building the Installer
- Testing the Project

Creating a New Project

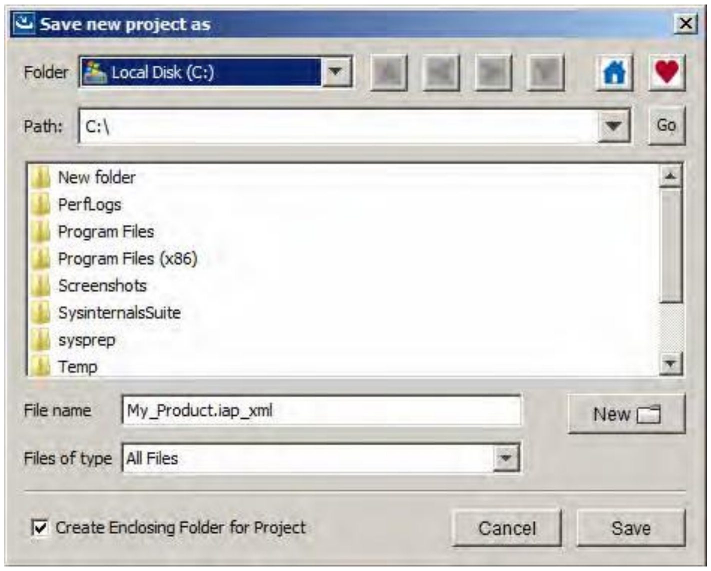
InstallAnywhere stores every project in its own XML file. These XML-based project files can be checked in and out of source control systems, and can be modified with text and XML editors. For added flexibility, project files may also be modified using XSL transformations, providing the ability to modify 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.

Task ***To create a new project:***

1. Launch InstallAnywhere.
2. On the initial screen select **New Installer...** and the **Create NewProject** dialog will open:



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.



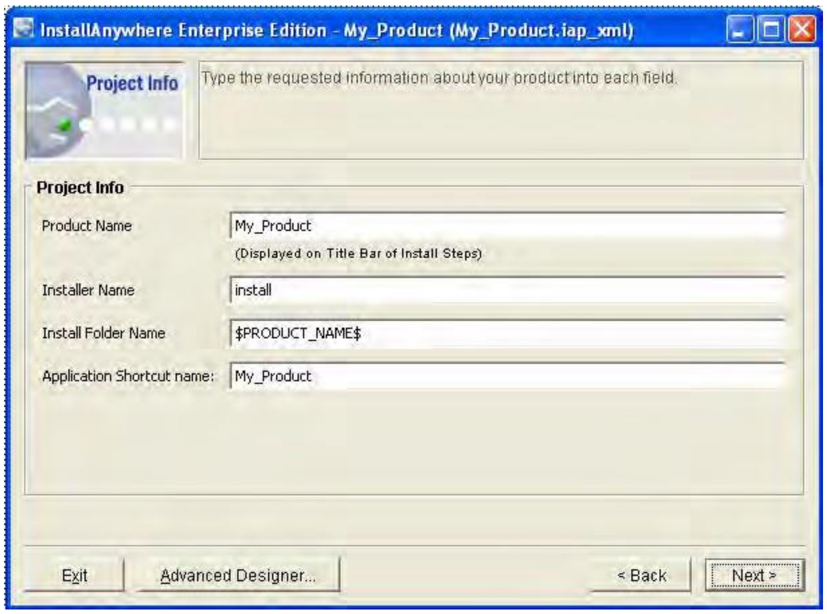
4. Click **Save** to confirm the name and close this dialog box.
5. Click **Next**. The **Project Info** panel opens.
6. Continue with the steps in Setting Project Information.

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.

Task ***To set project information:***

1. Perform the steps in Creating a New Project. The **Project Info** panel opens.


InstallAnywhere Enterprise Edition - My_Product (My_Product.iap_xml)
Project Info
Exit
Advanced Designer...
< Back
Next >

Type the requested information about your product into each field.

Project Info

Product Name: My_Product
(Displayed on Title Bar of Install Steps)

Installer Name: install

Install Folder Name: $PRODUCT_NAME$

Application Shortcut name: My_Product

2. 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.

3. Click **Next**. The **Add Files** panel opens.

4. Continue with the steps in Installing Tasks.

Installing Tasks

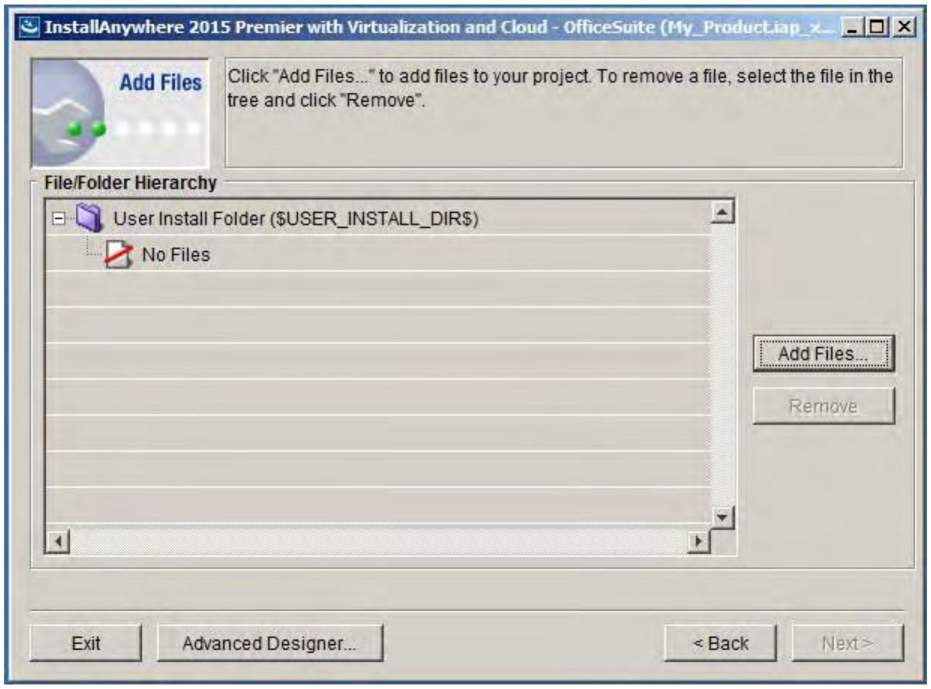
This section consists of several steps. First add files to the project, choose the main Java class for starting the project, and set the classpath that the project uses.

Add Files

To add files to the project, perform the following steps.

Task ***To add files to the project:***

1. Perform the steps in Setting Project Information. The **Add Files** panel opens.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_x...

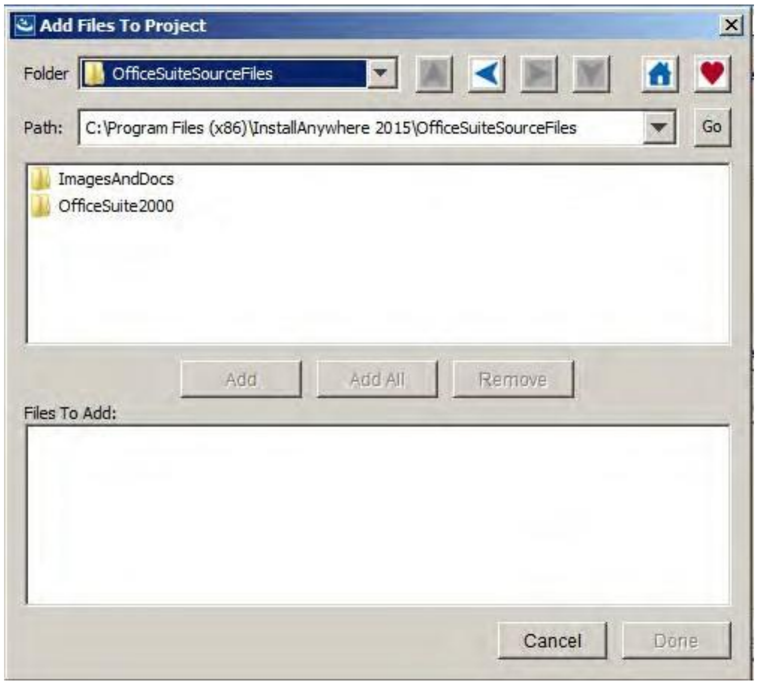
Add Files

Click "Add Files..." to add files to your project. To remove a file, select the file in the tree and click "Remove".

File/Folder Hierarchy

User Install Folder ($USER_INSTALL_DIR$)
No Files

2. Click **Add Files**. The **Add Files to Project** dialog box opens.


Add Files To Project

Folder: OfficeSuiteSourceFiles
Path: C:\Program Files (x86)\InstallAnywhere 2015\OfficeSuiteSourceFiles

ImagesAndDocs
OfficeSuite2000

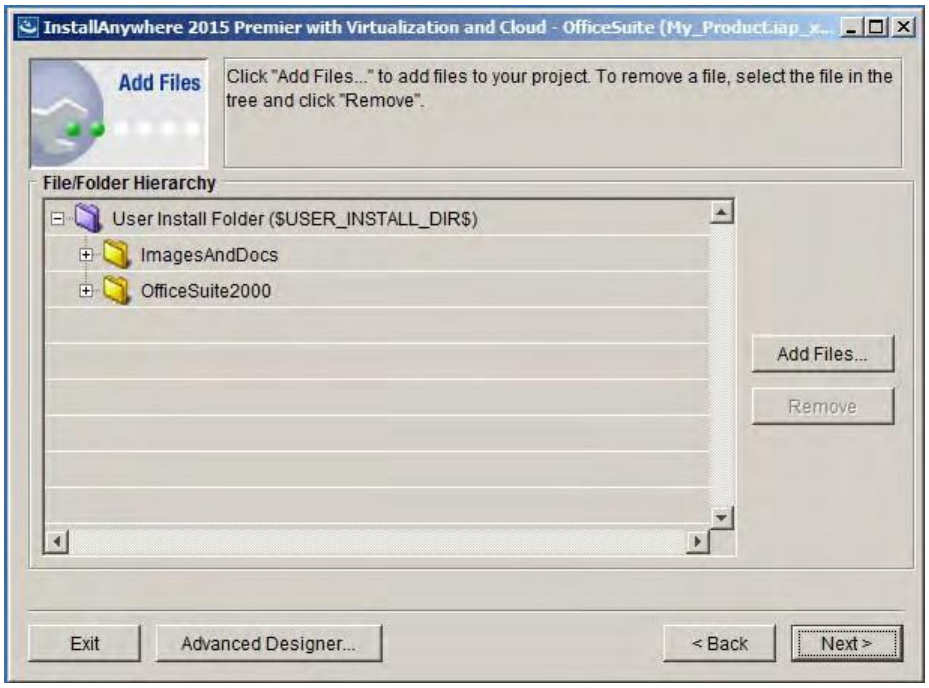
Files To Add:

3. Browse through the list to find the `OfficeSuiteSourceFiles` folder, located within the InstallAnywhere installation directory.

4. Click Add All to add the `ImagesAndDocs` and `OfficeSuite2000` directories, which are inside the `OfficeSuiteSourceFiles` folder. These files appears in the **Files to Add** list.

   Note • *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.*

5. Click **Done**. The selected files should appear in the **File/Folder Hierarchy**.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_x...
Add Files
File/Folder Hierarchy
User Install Folder ($USER_INSTALL_DIR$)
ImagesAndDocs
OfficeSuite2000
Add Files...
Remove
Exit
Advanced Designer...
< Back
Next >

Click "Add Files..." to add files to your project. To remove a file, select the file in the tree and click "Remove".

   Note • *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.*

6. Click **Next**. The **Choose Main** panel opens.

7. Continue with the steps in Choose Main Class.

Choose Main Class

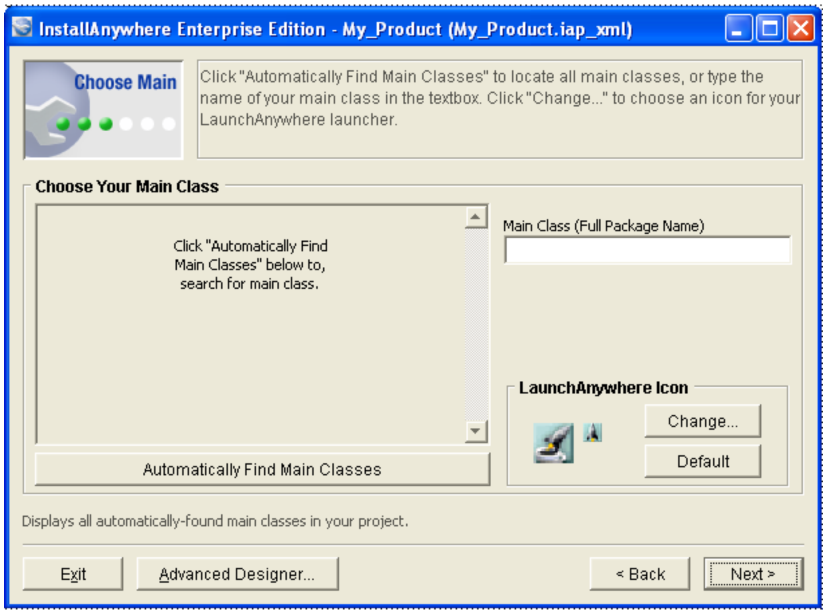
The **Choose Main Class** selects the starting class for the application. A Java application contains one or more classes that implement a method called `main`. This wizard panel is where you specify the class whose `main` method you want to execute when a user launches your LaunchAnywhere executable.

Note • *If you are not installing a Java application, you would click **Next** without specifying a main class.*

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

Task ***To choose a main class:***

1. Perform the steps in Add Files. The **Choose Main** panel opens.


InstallAnywhere Enterprise Edition - My_Product (My_Product.iap_xml)
Choose Main
Choose Your Main Class
Main Class (Full Package Name)
LaunchAnywhere Icon
Change...
Default
Automatically Find Main Classes
Exit
Advanced Designer...
< Back
Next >

Click "Automatically Find Main Classes" to locate all main classes, or type the name of your main class in the textbox. Click "Change..." to choose an icon for your LaunchAnywhere launcher.

Click "Automatically Find Main Classes" below to, search for main class.

Displays all automatically-found main classes in your project.

2. Click **Automatically Find Main Classes** at the bottom of the screen. A class is automatically added to the class list.
3. Select the main class.
4. Specify a custom icon for the LaunchAnywhere executable by clicking **Change** and choosing the `OfficeIcon.gif` file located in the `ImagesAndDocs` directory.

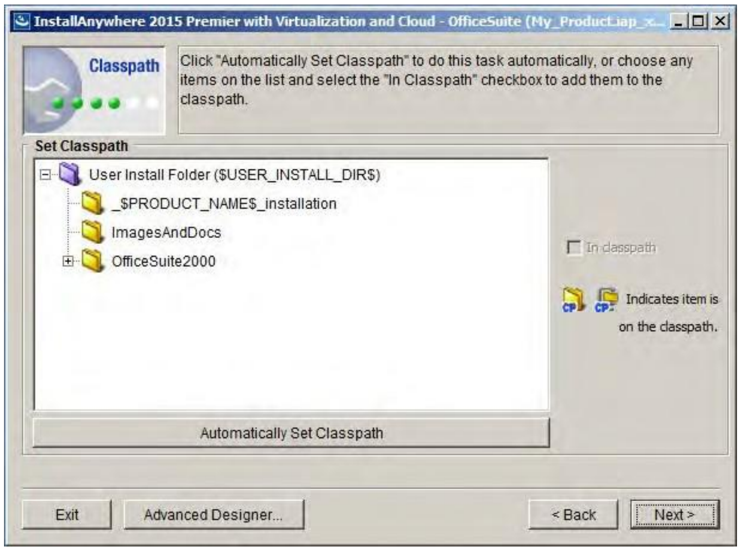
   Note • *You can select a 32-by-32 or a 16-by-16 pixel GIF for the application icon. Windows-only .ico files are not supported.*
5. Click **OK** to confirm and close the dialog box. The icon appears on the main screen.
6. Click **Next**. The **Classpath** panel opens.
7. Continue with the steps in Setting the Classpath.

Setting the Classpath

To set the classpath, perform the following steps:

Task ***To set the classpath:***

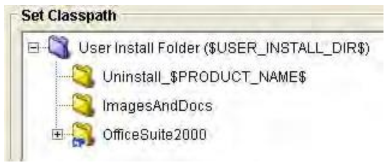
1. Perform the steps in Choose Main Class. The **Classpath** panel opens.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_x...
Classpath
Set Classpath
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
ImagesAndDocs
OfficeSuite2000
In classpath
Automatically Set Classpath
Exit
Advanced Designer...
< Back
Next >

Click "Automatically Set Classpath" to do this task automatically, or choose any items on the list and select the "In Classpath" checkbox to add them to the classpath.

Indicates item 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.



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

```
java -cp TrainApp.jar TrainingAppMainClass
```

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

3. Click **Next**. The **Build Installer** panel opens.

4. Continue with the steps in Building the Installer.

Building the Installer

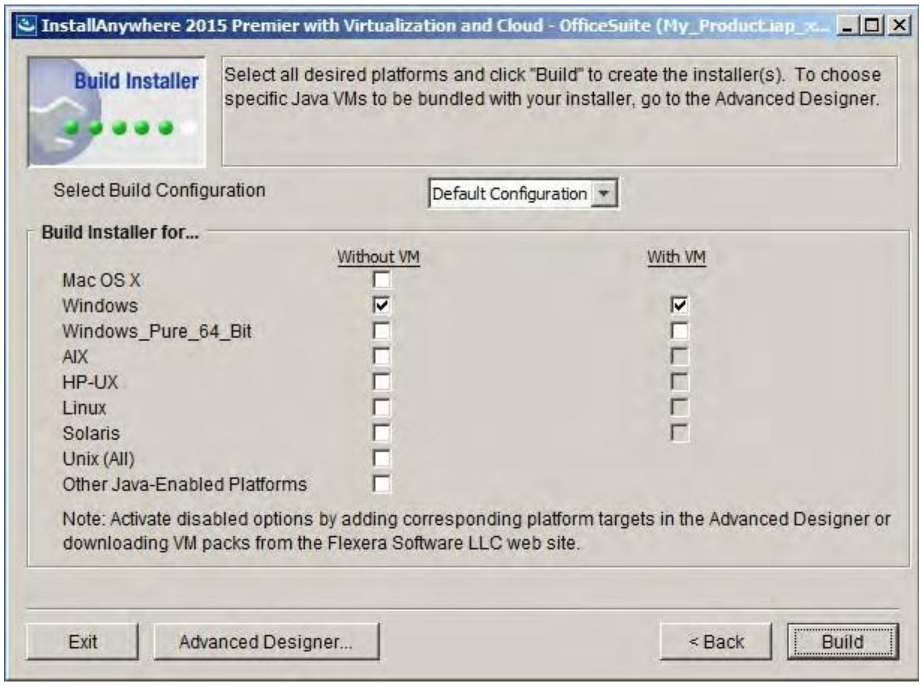
The first several items on the **Build Installer** screen, from **Mac OS X** through **Unix (All)**, represent installers that can be double-clicked on their respective platforms.

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.

***Note** • 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.*

Task ***To build the installer:***

1. Perform the steps in Setting the Classpath. The **Build Installer** panel opens.



InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_x...

Build Installer

Select all desired platforms and click "Build" to create the installer(s). To choose specific Java VMs to be bundled with your installer, go to the Advanced Designer.

Select Build Configuration: Default Configuration

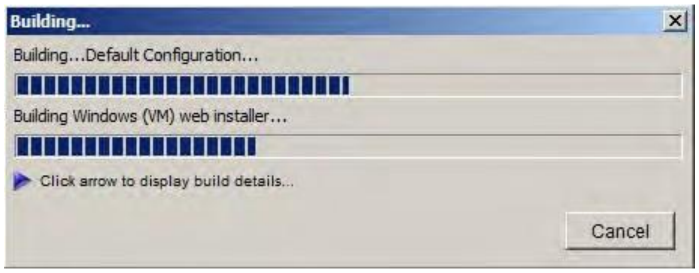
Build Installer for...

	Without VM	With VM
Mac OS X	☐	
Windows	✓	✓
Windows_Pure_64_Bit	☐	☐
AIX	☐	☐
HP-UX	☐	☐
Linux	☐	☐
Solaris	☐	☐
Unix (All)	☐	
Other Java-Enabled Platforms	☐	

Note: Activate disabled options by adding corresponding platform targets in the Advanced Designer or downloading VM packs from the Flexera Software LLC web site.

2. Choose the desired destination platforms.

3. Click **Build**. The **Building** dialog box opens:



Building...

Building...Default Configuration...

Building Windows (VM) web installer...

Click arrow to display build details...

When build is complete, the **Try Installer** panel opens.

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

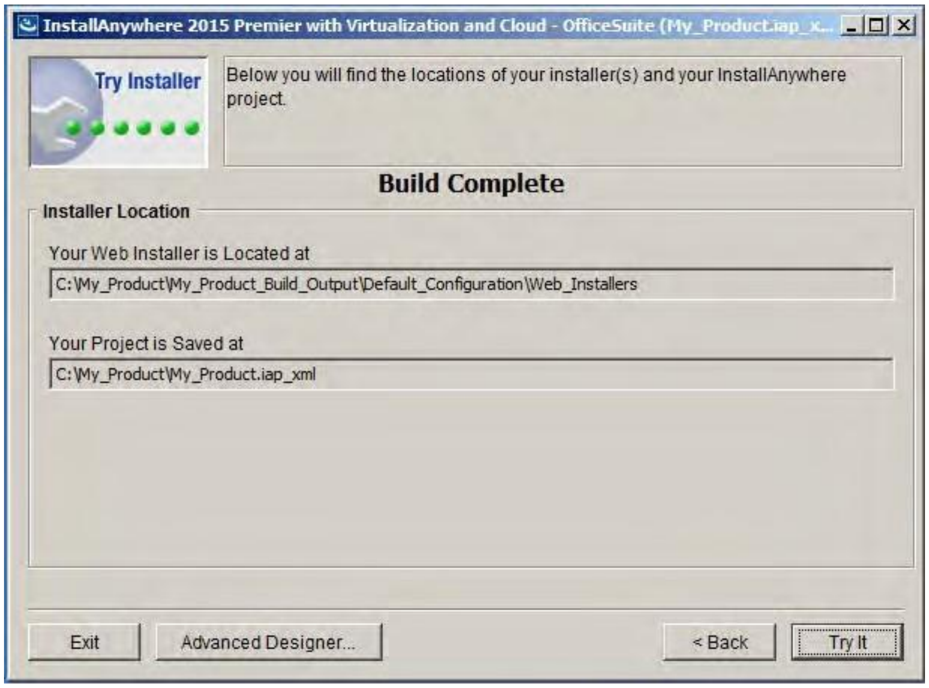
4. Continue with the steps in Testing the Project.

Testing the Project

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

Task ***To test the project:***

1. Perform the steps in Building the Installer. The **Try Installer** panel opens.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_x...

Try Installer

Below you will find the locations of your installer(s) and your InstallAnywhere project.

Build Complete

Installer Location

Your Web Installer is Located at

C:\My_Product\My_Product_Build_Output\Default_Configuration\Web_Installers

Your Project is Saved at

C:\My_Product\My_Product.iap_xml

2. Click **Try Installer** and follow the prompts to install the product.

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

3. After installation is complete, do the following to launch the application:
   - 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.
4. After launching OfficeSuite for Java, quit by selecting **Exit** from the **File** menu.

Installing the Application Onto Another Platform

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

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 CD-ROM installer, transfer the entire contents of the `CDROM_Installers` subdirectory.

3

Introduction to the Advanced Designer

This chapter introduces you to using the InstallAnywhere Advanced Designer interface and contains information on the following topics:

- Installation Planning
- Installation Goals
- Working with 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:

- **Degree of flexibility**—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?
- **Platforms and architectures**—What platforms and architectures will your deployment project target?

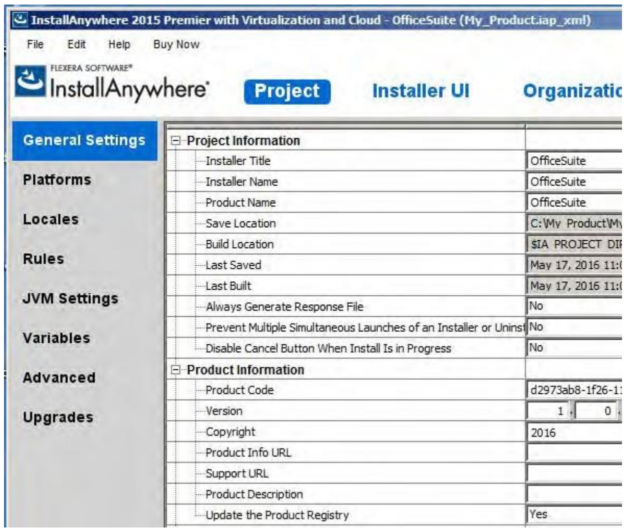
To help you plan your installation process, an Installation Planning Worksheet can be used to structure and manage your installation development project. A worksheet template can be found in Appendix A, Installation Planning Worksheet.

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 top of the window. Each tab represents tasks and settings specific to each installation project.



Project Information	
Installer Title	OfficeSuite
Installer Name	OfficeSuite
Product Name	OfficeSuite
Save Location	C:\My Product\My
Build Location	$IA PROJECT DIR
Last Saved	May 17, 2016 11:(
Last Built	May 17, 2016 11:(
Always Generate Response File	No
Prevent Multiple Simultaneous Launches of an Installer or Uninst	No
Disable Cancel Button When Install Is in Progress	No
Product Information	
Product Code	d2973ab8-1f26-1:
Version	1 . 0 .
Copyright	2016
Product Info URL	
Support URL	
Product Description	
Update the Product Registry	Yes

Figure 3-1: Advanced Designer

The following table provides Advanced Designer task descriptions.

Table 3-1 • Advanced Designer Task Descriptions

Name	Description
Project	Settings related to your specific project. These include general settings, file settings, and localization settings.
Installer UI	Set the look and feel for the installer by adding background images, billboards, and other graphical elements.
Organization	Manage Install Sets, Features, Components, and Merge Modules.
Sequence	An ordered sequence of panels and actions that occur at specific times during the install or uninstall. For Install, these sequences include Pre-Install, Install, and Post-Install. Similarly, for Uninstall, these sequences include Pre-Uninstall, Install, and Post-Uninstall.
Build	Manage build settings, including bundling of a Java Virtual Machine.

Each Advanced Designer task contains sub-tabs that offer greater fine-tuning of InstallAnywhere's features. For an example, refer to the *Building an Installer Using the Advanced Designer* tutorial in the InstallAnywhere Help Library.

Defining Installer Projects and the Product Registry

This section introduces you to the product registry and explains how installers are identified.

- Product Registry
- Installer Identification and Version

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.*

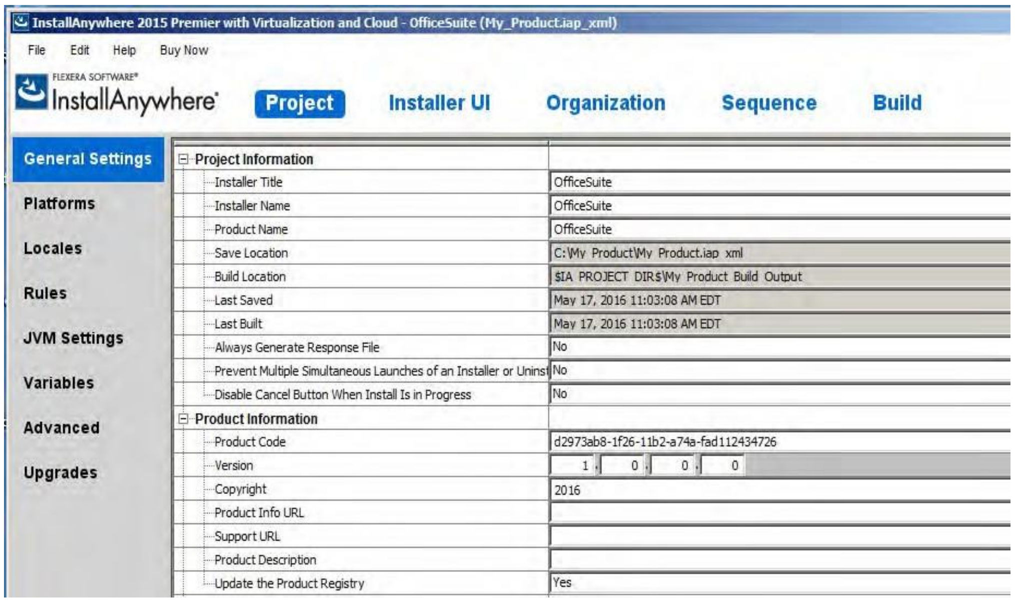
Product ID and vendor information is entered in the **Project > Description** subtask.

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. These settings are available in the Log Settings section under Project -> General Settings sub task

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.



Project Information	
Installer Title	OfficeSuite
Installer Name	OfficeSuite
Product Name	OfficeSuite
Save Location	C:\My Product\My Product.iap xml
Build Location	$IA PROJECT DIR$\My Product Build Output
Last Saved	May 17, 2016 11:03:08 AM EDT
Last Built	May 17, 2016 11:03:08 AM EDT
Always Generate Response File	No
Prevent Multiple Simultaneous Launches of an Installer or Uninst	No
Disable Cancel Button When Install Is in Progress	No
Product Information	
Product Code	d2973ab8-1f26-11b2-a74a-fad112434726
Version	1 . 0 . 0 . 0
Copyright	2016
Product Info URL	
Support URL	
Product Description	
Update the Product Registry	Yes

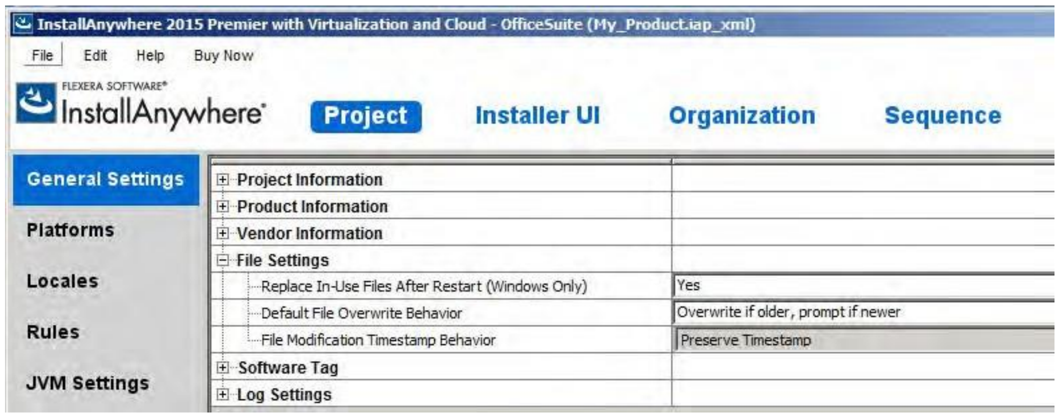
Figure 3-2: Project Information

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.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_xml)

Project Information	
Product Information	
Vendor Information	
File Settings	
Replace In-Use Files After Restart (Windows Only)	Yes
Default File Overwrite Behavior	Overwrite if older, prompt if newer
File Modification Timestamp Behavior	Preserve Timestamp
Software Tag	
Log Settings	

Figure 3-3: File Settings

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.

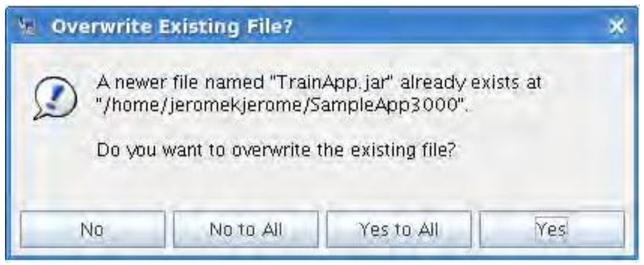
***Note** • For Windows systems, InstallAnywhere also includes an overwrite-after-restart option for files that are in use during installation. To enable this option, click **Replace in-use files after restart**.*

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

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


Overwrite Existing File?

A newer file named "TrainApp.jar" already exists at "/home/jeromekjerome/SampleApp3000".

Do you want to overwrite the existing file?

Figure 3-4: Sample Overwrite Prompt

Platforms

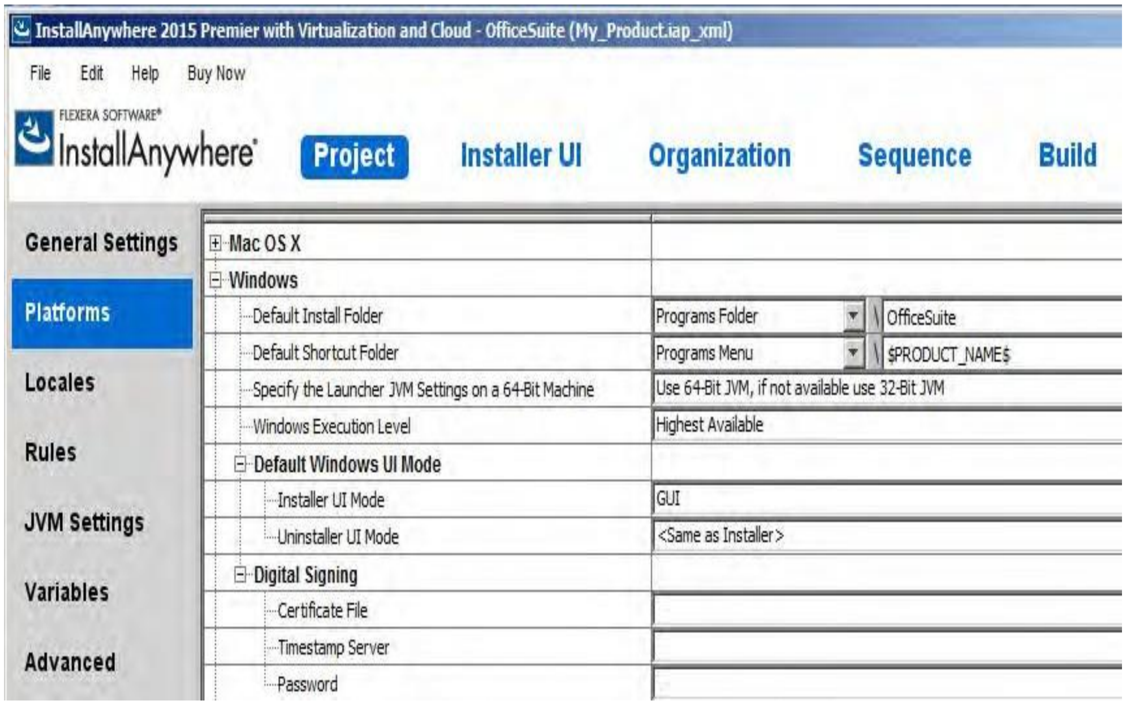
While InstallAnywhere runs on any Java-enabled platform, there are features such as default install folders and default link folders (Unix), default shortcut folders (Windows), and default alias folders (Mac OS X) that should be defined separately for each target operating system.

The **Platforms** task is separated into different platforms.

- Windows
- Mac OS X
- UNIX
- System i (i5/OS)
- Pure Java

Windows

For Windows, you specify the default install and shortcut folders, and choose between using a graphical launcher or a console launcher.



Mac OS X	
Windows	
Default Install Folder	Programs Folder \ OfficeSuite
Default Shortcut Folder	Programs Menu \ $PRODUCT_NAME$
Specify the Launcher JVM Settings on a 64-Bit Machine	Use 64-Bit JVM, if not available use 32-Bit JVM
Windows Execution Level	Highest Available
Default Windows UI Mode	
Installer UI Mode	GUI
Uninstaller UI Mode	<Same as Installer>
Digital Signing	
Certificate File	
Timestamp Server	
Password	

Figure 3-5: Platform Support

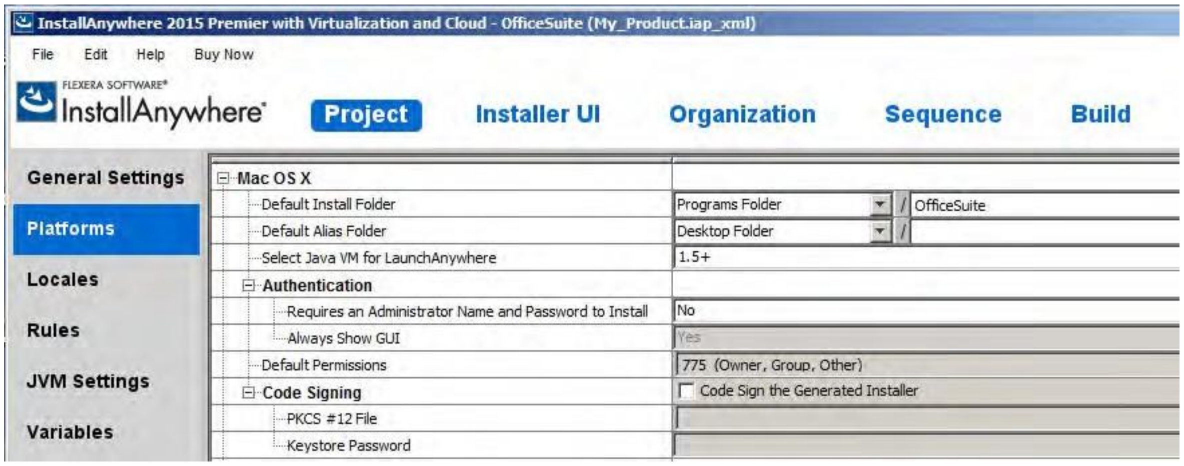
To support User Account Control (UAC) on Windows Vista target systems, you can also specify the execution level:

- **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.

You can use the **Windows JVM Search Paths** settings in this task to control how the InstallAnywhere run-time locates compatible Java virtual machines. For more information about specifying compatible Java virtual machines, see the Virtual Machines section later in this chapter, as well as Chapter 4, Building Releases.

Mac OS X

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.



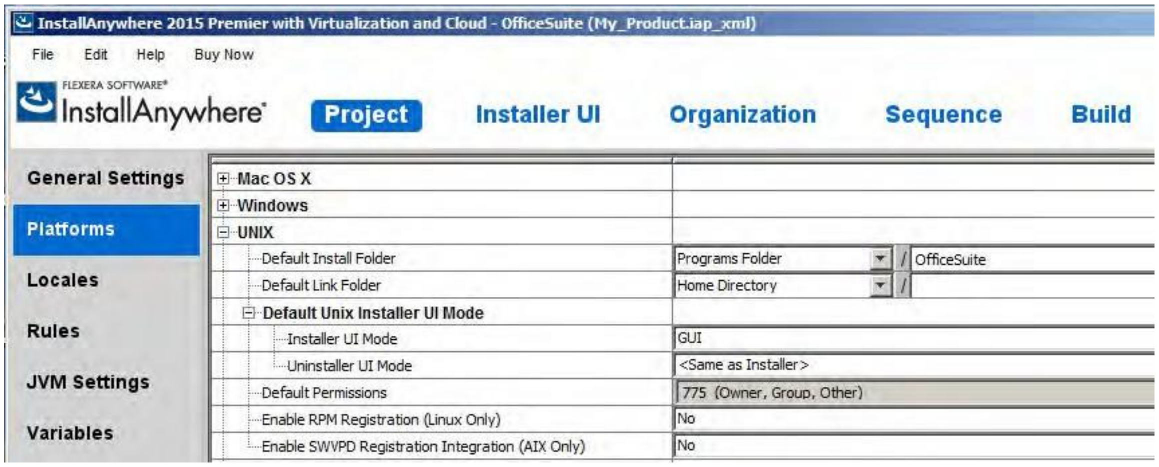
Mac OS X	
Default Install Folder	Programs Folder / OfficeSuite
Default Alias Folder	Desktop Folder /
Select Java VM for LaunchAnywhere	1.5+
Authentication	
Requires an Administrator Name and Password to Install	No
Always Show GUI	Yes
Default Permissions	775 (Owner, Group, Other)
Code Signing	☐ Code Sign the Generated Installer
PKCS #12 File	
Keystore Password	

Figure 3-6: Mac OS X Support

UNIX

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.

You can additionally specify to integrate with the AIX Software Vital Product Database (SWVPD).



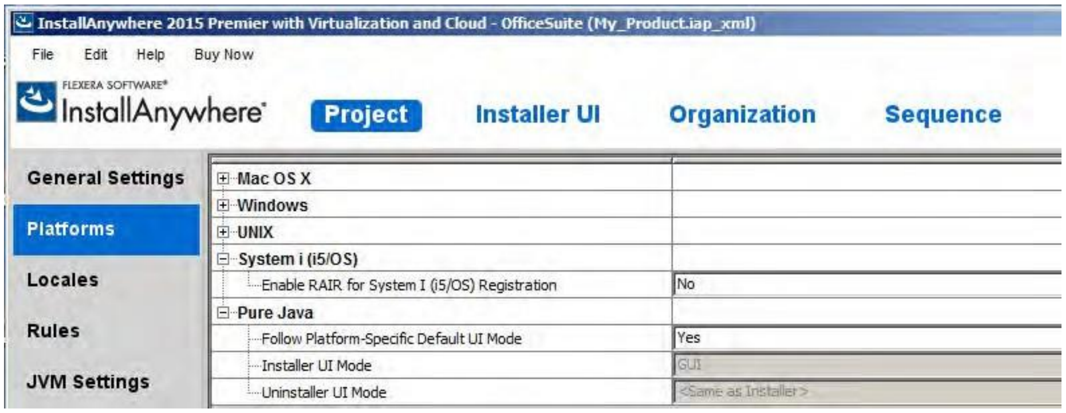
Default Install Folder	Programs Folder / OfficeSuite
Default Link Folder	Home Directory /
Default Unix Installer UI Mode	
Installer UI Mode	GUI
Uninstaller UI Mode	<Same as Installer>
Default Permissions	775 (Owner, Group, Other)
Enable RPM Registration (Linux Only)	No
Enable SWVPD Registration Integration (AIX Only)	No

Figure 3-7: Unix Support

Starting with InstallAnywhere 2009, you can use the Unix JVM Search Paths settings (not pictured) to control InstallAnywhere's detection of Java virtual machines.

System i (i5/OS)

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

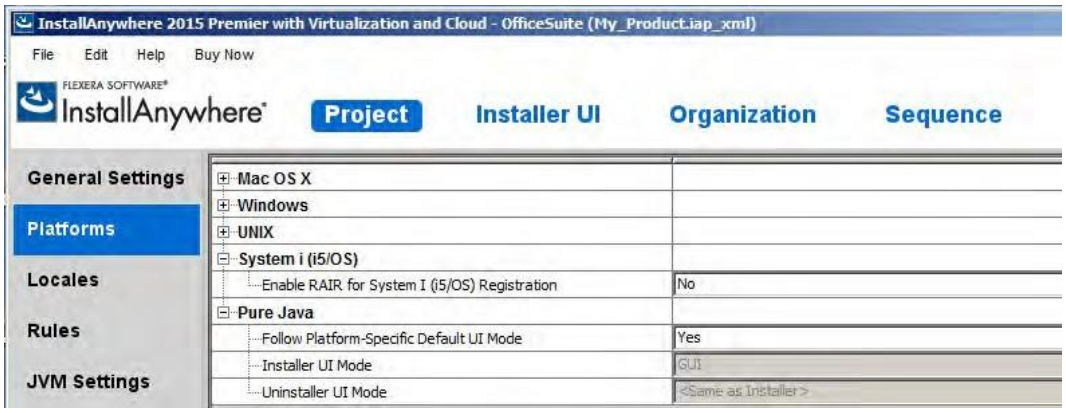


System i (i5/OS)	
Enable RAIR for System I (i5/OS) Registration	No
Pure Java	
Follow Platform-Specific Default UI Mode	Yes
Installer UI Mode	GUI
Uninstaller UI Mode	<Same as Installer>

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

Pure Java

You can use the Pure Java task to control the user-interface mode a pure Java installer and uninstaller should use.



Mac OS X	
Windows	
UNIX	
System i (i5/OS)	
Enable RAIR for System I (i5/OS) Registration	No
Pure Java	
Follow Platform-Specific Default UI Mode	Yes
Installer UI Mode	GUI
Uninstaller UI Mode	<Same as Installer>

Figure 3-9: Pure Java Settings

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 Chapter 18, Localizing and Internationalizing Installers.

Rules Before the Pre-Install Task

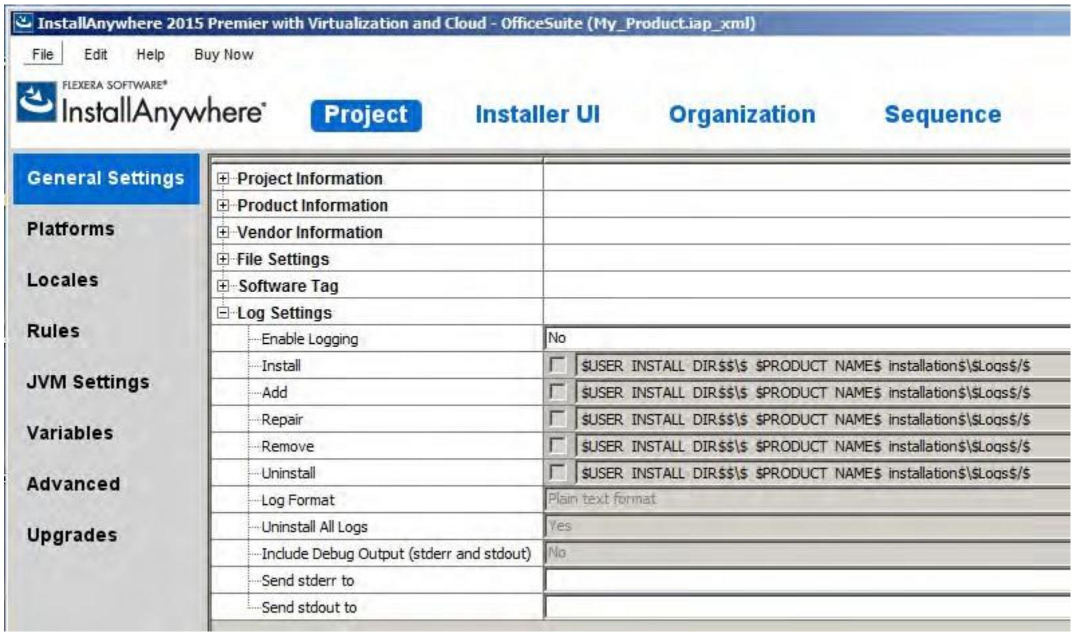
InstallAnywhere Rules can be applied to any action within the InstallAnywhere installer, as well as to organizational units such as Install Sets, Features, and Components.

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. The **Project > General Settings** subtask contains **Log Settings** settings.



Log Settings	
Enable Logging	No
Install	$USER_INSTALL_DIR$$\$_$PRODUCT_NAME$_installation$\$Logs$/$
Add	$USER_INSTALL_DIR$$\$_$PRODUCT_NAME$_installation$\$Logs$/$
Repair	$USER_INSTALL_DIR$$\$_$PRODUCT_NAME$_installation$\$Logs$/$
Remove	$USER_INSTALL_DIR$$\$_$PRODUCT_NAME$_installation$\$Logs$/$
Uninstall	$USER_INSTALL_DIR$$\$_$PRODUCT_NAME$_installation$\$Logs$/$
Log Format	Plain text format
Uninstall All Logs	Yes
Include Debug Output (stderr and stdout)	No
Send stderr to	
Send stdout to	

Figure 3-10: Project > General Settings Task

The settings made in the **Log Settings** section of the **Project > General Settings** subtask determine whether debug output is enabled, and whether any output is sent to a file or to a live console:

- **Disable**—If the **Send stderr to** or the **Send stdout to** field under **Installer Debug Output** is left blank, the corresponding output of the installer will be discarded.
- **Send to live console**—To send the output to a live console to monitor the output, enter `console` in the text field.
- **Send to file**—To send the information to a file, enter the file name.

As described in Chapter 10, Applying Basic and Intermediate Development Concepts, you can include environment variables and Java system properties in the redirection target instead of hard-coding file locations.

InstallAnywhere provides classes that enable automated testing of installers using JUnit tests. For information and examples, refer to the `gui-test-auto` subdirectory of your InstallAnywhere distribution.

For more information on debugging, refer to Chapter 10, Applying Basic and Intermediate Development Concepts and Chapter 16, Custom Code.

Virtual Machines

InstallAnywhere enables you to define a valid list of JVMs (Java virtual machines) your installer can use. This option can be used to select VMs that have been fully tested. LaunchAnywhere searches for VMs sequentially based on VM type. Valid VM types are listed in the LaunchAnywhere Executable property `lax.nl.valid.vm.list`. LaunchAnywhere uses the following default 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.

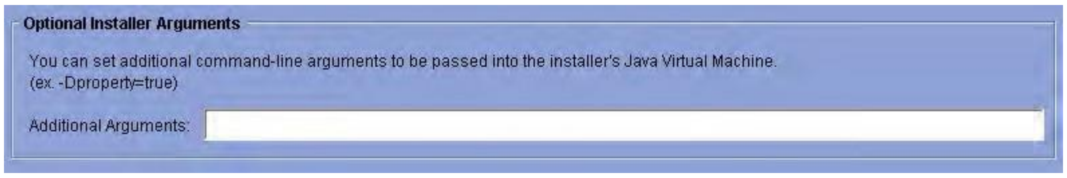
You can use settings in the **Platforms** task to override the default JVM search behavior for the installation on different platforms.

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 **Additional Arguments** field under **Optional Installer Arguments** on the **Installer Settings** tab of the **Project > JVM Settings** subtask.


Optional Installer Arguments

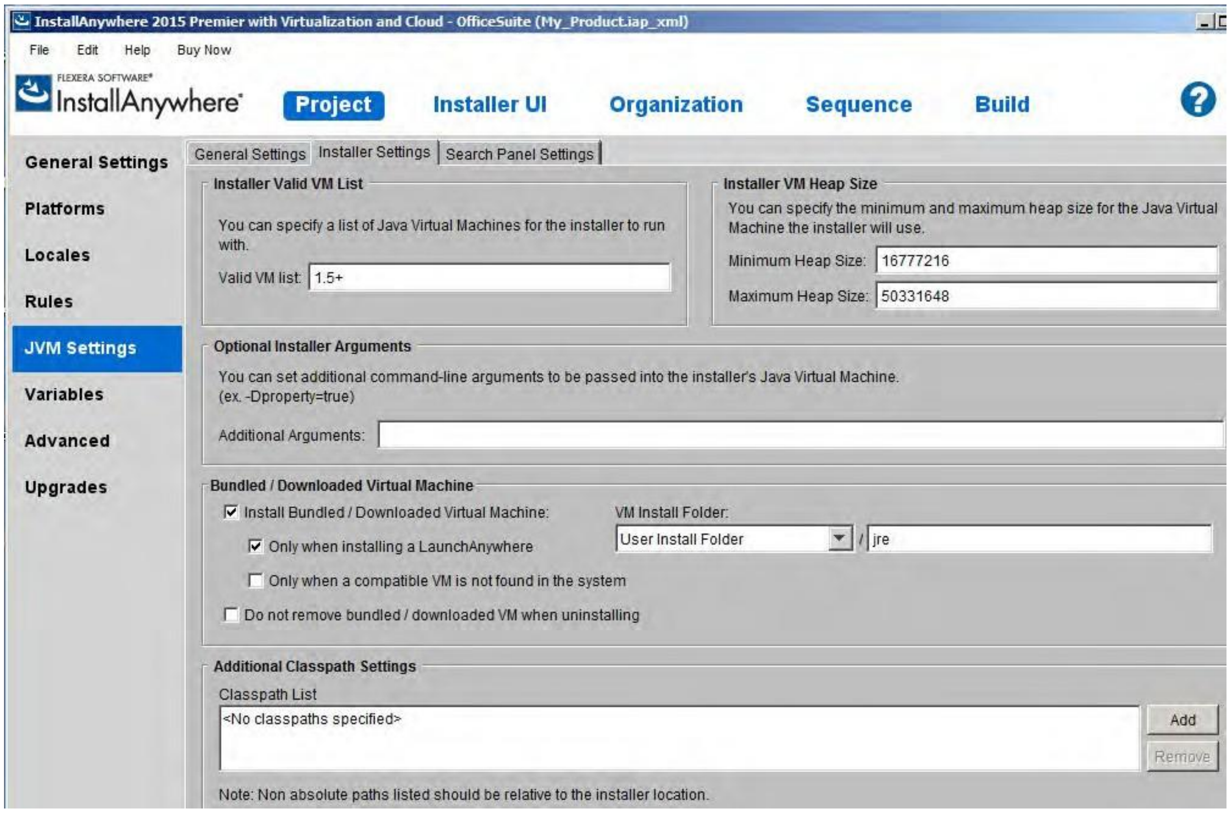
You can set additional command-line arguments to be passed into the installer's Java Virtual Machine.
(ex. -Dproperty=true)

Additional Arguments:

Figure 3-11: Optional Installer Arguments on Project > JVM Settings Subtask

Java

The **Project > JVM Settings** subtask enables you to fine tune the classpath settings and decide whether to install the bundled Java VM.



Installer Valid VM List

You can specify a list of Java Virtual Machines for the installer to run with.

Valid VM list: 1.5+

Installer VM Heap Size

You can specify the minimum and maximum heap size for the Java Virtual Machine the installer will use.

Minimum Heap Size: 16777216

Maximum Heap Size: 50331648

Optional Installer Arguments

You can set additional command-line arguments to be passed into the installer's Java Virtual Machine. (ex. -Dproperty=true)

Additional Arguments:

Bundled / Downloaded Virtual Machine

[✓] Install Bundled / Downloaded Virtual Machine:

[✓] Only when installing a LaunchAnywhere

[ ] Only when a compatible VM is not found in the system

[ ] Do not remove bundled / downloaded VM when uninstalling

VM Install Folder: User Install Folder / jre

Additional Classpath Settings

Classpath List

<No classpaths specified>

Note: Non absolute paths listed should be relative to the installer location.

Figure 3-12: Project > JVM Settings Subtask

You 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 you choose to install the VM, the **VM Install folder** drop-down list provides a variety of locations.

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

For additional information on custom code, refer to Chapter 16, Custom Code.

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

Table 3-3 • Answers to Quick Quiz

Question	Answer
1	b
2	a, b
3	a

4

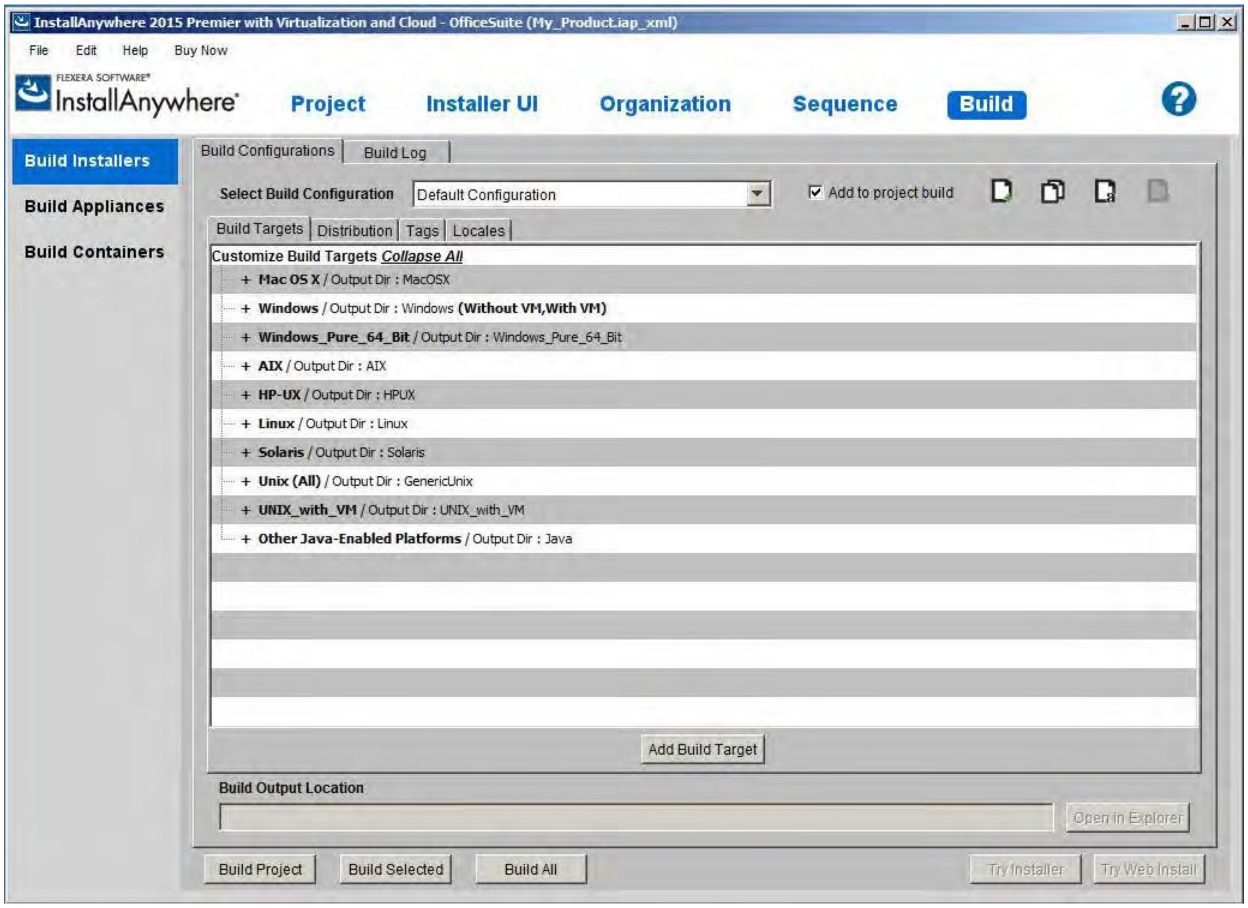
Building Releases

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.

Note • *For additional information on automated tools for building your InstallAnywhere projects, refer to Chapter 15, Integrating InstallAnywhere with Automated Build Environments.*

Build Targets

The **Build** task provides the options for building an installer to multiple targets in multiple Build Configurations. On the **Build Targets** subtab of the **Build Installer** tab of the **Build** task, you specify the desired target operating systems.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_xml)

Select Build Configuration: Default Configuration
☑ Add to project build

Customize Build Targets *Collapse All*
+ **Mac OS X** / Output Dir : MacOSX
+ **Windows** / Output Dir : Windows **(Without VM,With VM)**
+ **Windows_Pure_64_Bit** / Output Dir : Windows_Pure_64_Bit
+ **AIX** / Output Dir : AIX
+ **HP-UX** / Output Dir : HPUX
+ **Linux** / Output Dir : Linux
+ **Solaris** / Output Dir : Solaris
+ **Unix (All)** / Output Dir : GenericUnix
+ **UNIX_with_VM** / Output Dir : UNIX_with_VM
+ **Other Java-Enabled Platforms** / Output Dir : Java

Build Output Location

Figure 4-1: Build Targets Subtab of the Build ConfigurationsTab of the Build Installers Task

On the **Build Targets** tab, 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.

Using the **Build Targets** tab, you can define dynamic build targets: 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.

- **Creating a build target**—To create a build target, click **Add Build Target** at the bottom of the list of existing targets.
- **Deleting a build target**—To delete a build target, expand the target by clicking the **+** next to the name of the target and then click the **Delete** button on the right hand side of the target details.

When you click **Build Project**, all of the targets for each of the Build Configurations with the **Add to project build** check box selected will be built. While the build is taking place, a progress indicator similar to the following is displayed.

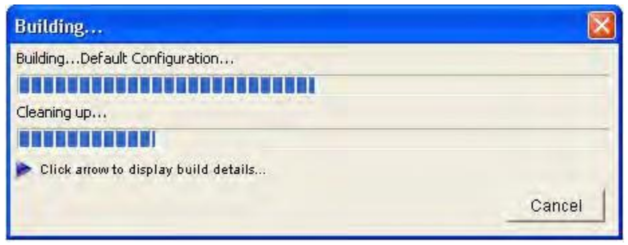

Building...
Building...Default Configuration...
Cleaning up...
Click arrow to display build details...
Cancel

Figure 4-2: Build-Progress Indicator

Note • *If you click **Build Selected**, only the selected Build Configuration will be built. If you click **Build All**, all Build Configurations will be built, even those that have not been added to the project build (by selecting the **Add to project build** option).*

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

Build Configurations

Each InstallAnywhere project can have multiple Build Configurations, each representing how the installer will be built for particular set of platforms, files, build distributions, JVMs, locales, and other settings. For information on using Build Configurations, see Chapter 13, "Creating and Editing Build Configurations,"

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 Flexera Software Web site provides additional VM packs for you to download. If you click the **Download Additional VM Packs** button under the **Help** menu for InstallAnywhere, the **InstallAnywhere Files & Utilities** page of the Flexera Software Web site opens, where you can download additional VM packs.

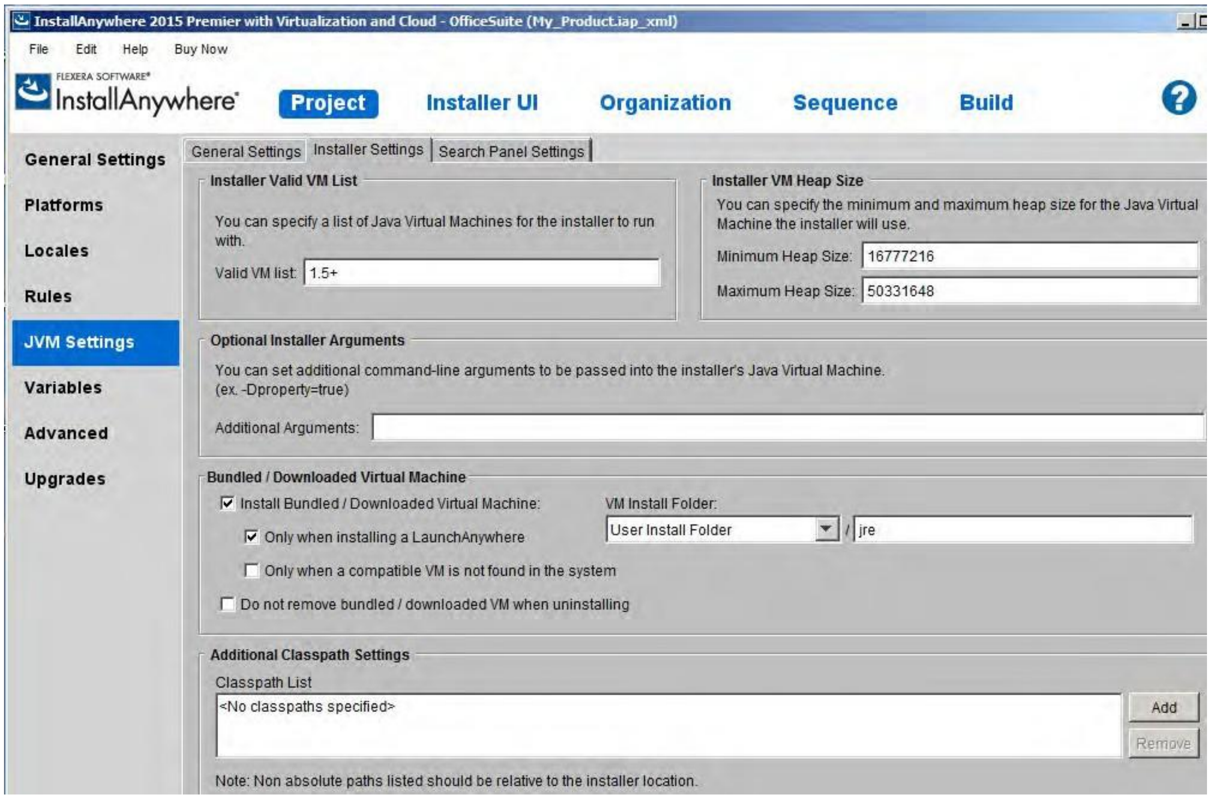
VM packs are stored as `.vm` files, which are `.zip` or `.jar` archives that contain the Java VM and a `vm.properties` file.

Note • *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 using the **VM Pack Utility** tool installed with InstallAnywhere. For information on creating a VM pack, see the *Creating VM Packs* help topic in the InstallAnywhere Help Library.

In the **Bundled Virtual Machine** area of the **Project > JVM Settings** 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).


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_xml)

Installer Valid VM List

You can specify a list of Java Virtual Machines for the installer to run with.

Valid VM list: 1.5+

Installer VM Heap Size

You can specify the minimum and maximum heap size for the Java Virtual Machine the installer will use.

Minimum Heap Size: 16777216

Maximum Heap Size: 50331648

Optional Installer Arguments

You can set additional command-line arguments to be passed into the installer's Java Virtual Machine. (ex. -Dproperty=true)

Additional Arguments:

Bundled / Downloaded Virtual Machine

☑ Install Bundled / Downloaded Virtual Machine:

☑ Only when installing a LaunchAnywhere

☐ Only when a compatible VM is not found in the system

☐ Do not remove bundled / downloaded VM when uninstalling

VM Install Folder: User Install Folder / jre

Additional Classpath Settings

Classpath List

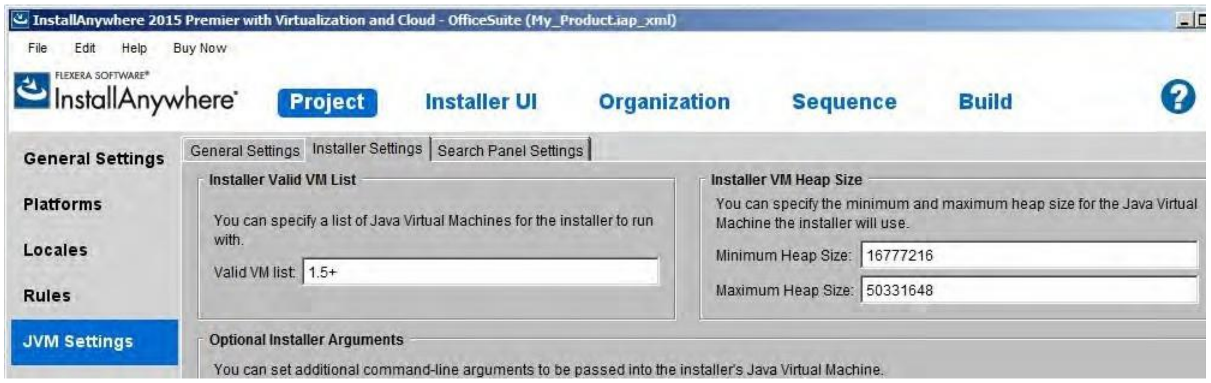
<No classpaths specified>

Note: Non absolute paths listed should be relative to the installer location.

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.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_xml)

Installer Valid VM List

You can specify a list of Java Virtual Machines for the installer to run with.

Valid VM list: 1.5+

Installer VM Heap Size

You can specify the minimum and maximum heap size for the Java Virtual Machine the installer will use.

Minimum Heap Size: 16777216

Maximum Heap Size: 50331648

Optional Installer Arguments

You can set additional command-line arguments to be passed into the installer's Java Virtual Machine.

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.

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 specify only one or the other.

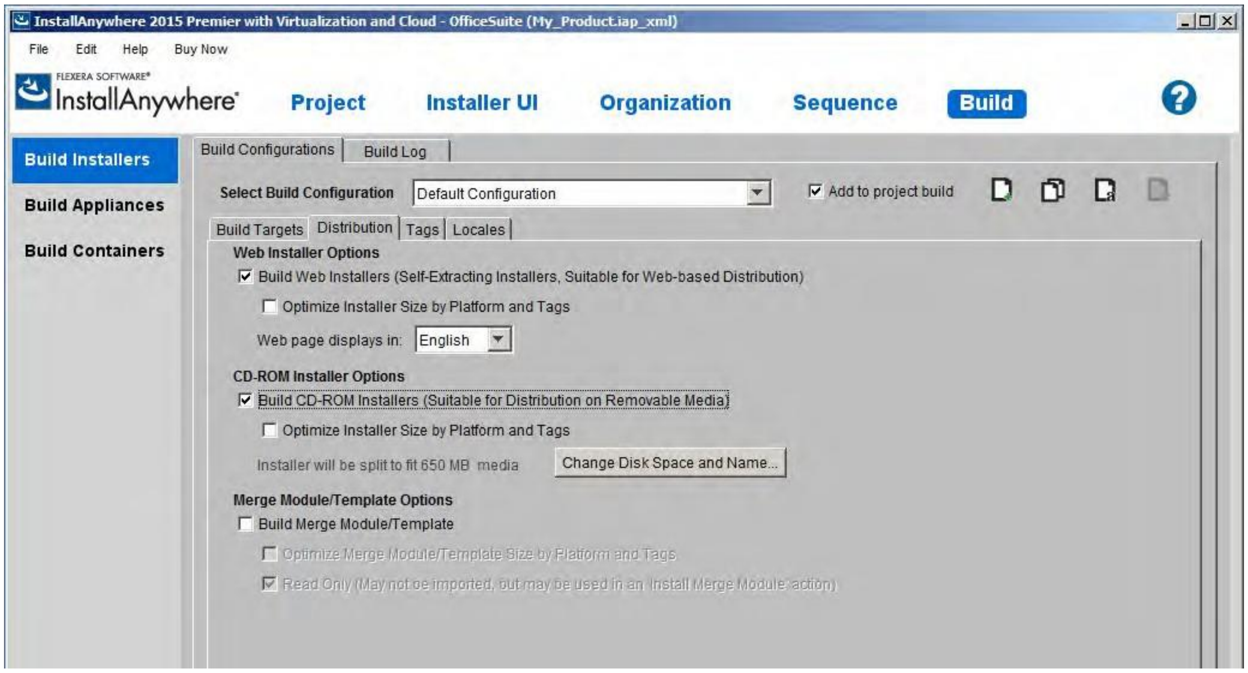
The version number can have varying degrees of precision; however, it is recommended that you have at least the major and minor version numbers specified.

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).

You can also indicate a particular vendor whose VMs you want to detect, where common vendor names are **`IBM`**, **`SUN`**, **`HP`**, and **`APPLE`**. If you specify a vendor, it must be separated from other specifiers with an underscore character. For example, to require an IBM JRE with version 1.5.0, the specifier would be **`IBM_JRE_1.5.0*`**; and to detect any IBM version 1.5.0 VM, use **`IBM_ALL_1.5.0*`**.

Distribution

In the **Distribution** subtab of the **Build Configurations** tab, 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.



Select Build Configuration: Default Configuration
☑ Add to project build

Web Installer Options
☑ Build Web Installers (Self-Extracting Installers, Suitable for Web-based Distribution)
☐ Optimize Installer Size by Platform and Tags
Web page displays in: English

CD-ROM Installer Options
☑ Build CD-ROM Installers (Suitable for Distribution on Removable Media)
☐ Optimize Installer Size by Platform and Tags
Installer will be split to fit 650 MB media

Merge Module/Template Options
☐ Build Merge Module/Template
☐ Optimize Merge Module/Template Size by Platform and Tags
☑ Read Only (May not be imported, but may be used in an 'Install Merge Module' action)

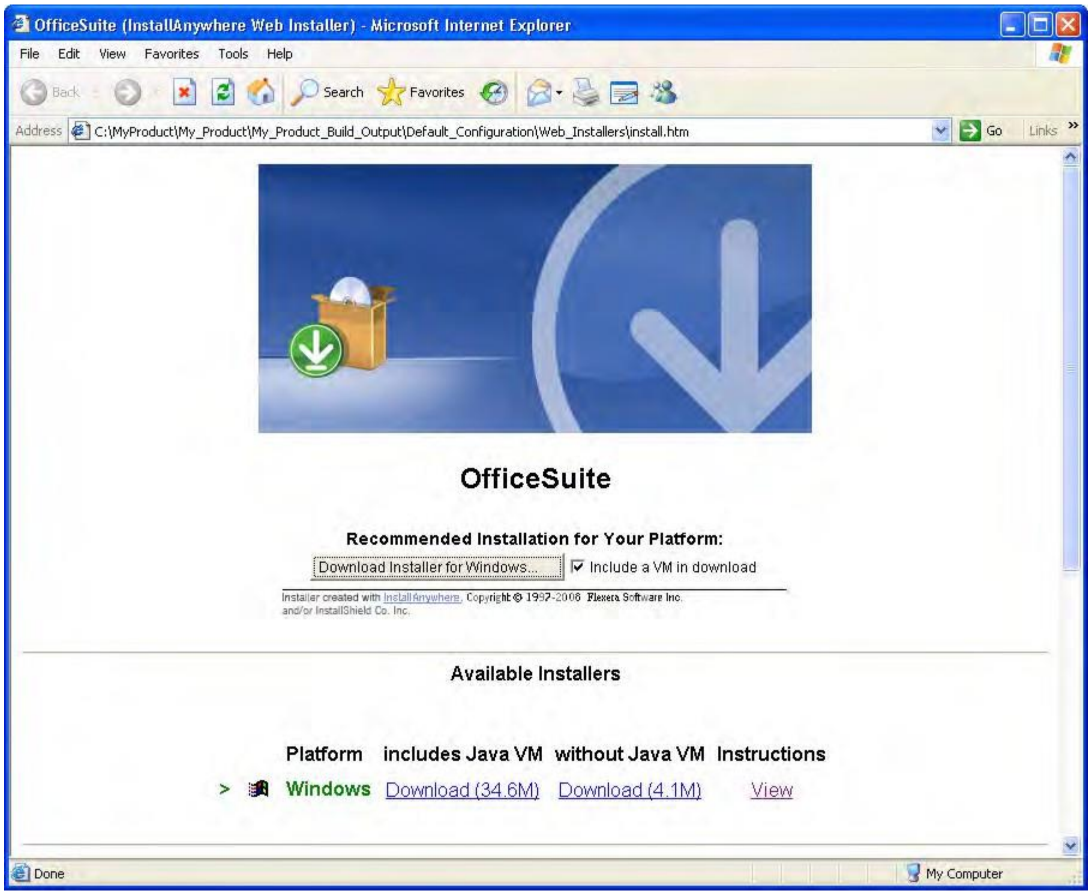
Figure 4-5: Distribution Tab

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 the **Optimize Installer Size by Platform and Tags** option 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.


OfficeSuite (InstallAnywhere Web Installer) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\MyProduct\My_Product\My_Product_Build_Output\Default_Configuration\Web_Installers\install.htm
Go Links
Done
My Computer

OfficeSuite

Recommended Installation for Your Platform:

Download Installer for Windows... [✓] Include a VM in download

Installer created with InstallAnywhere. Copyright © 1997-2008 Flexera Software Inc. and/or InstallShield Co. Inc.

Available Installers

	Platform	includes Java VM	without Java VM	Instructions
>	Windows	Download (34.6M)	Download (4.1M)	View

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 to access to large installers. The output of this build process can be directly burned onto disk.

InstallAnywhere CD/DVDs Installers have the ability to span multiple CDs/DVDs. By default, the installer will automatically segment the installer into a new disk if the size of the installer exceeds the media size (default: 650 MB). To control when InstallAnywhere will span to new disks, configure your disk names and size by clicking the **Change Disk Space and Name** button. This 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

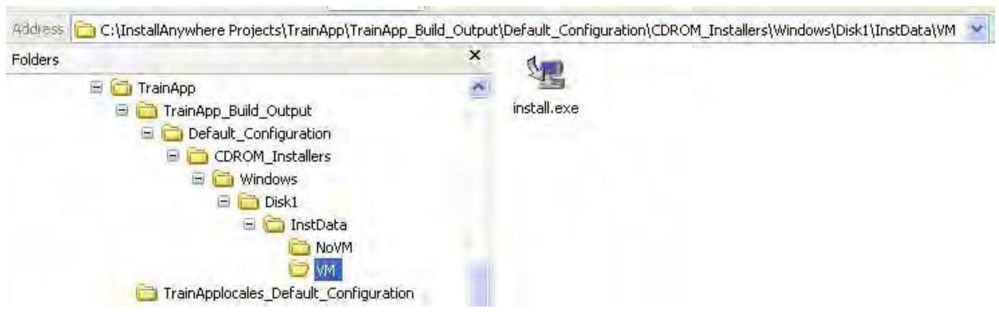
The directory structure for CD-ROM installers is:

```
Platform1/Disk1/InstData/
    |-…
    |-MediaId.properties
    |-Resource1.zip
Platform1/Disk2/InstData/
    |-MediaId.properties
    |-Resource2.zip
Platform1/Diskn/InstData/
    |-MediaId.properties
    |-ResourceN.zip
…
```

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

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

This build output might appear in Windows Explorer as follows:



Address C:\InstallAnywhere Projects\TrainApp\TrainApp_Build_Output\Default_Configuration\CDROM_Installers\Windows\Disk1\InstData\VM

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:

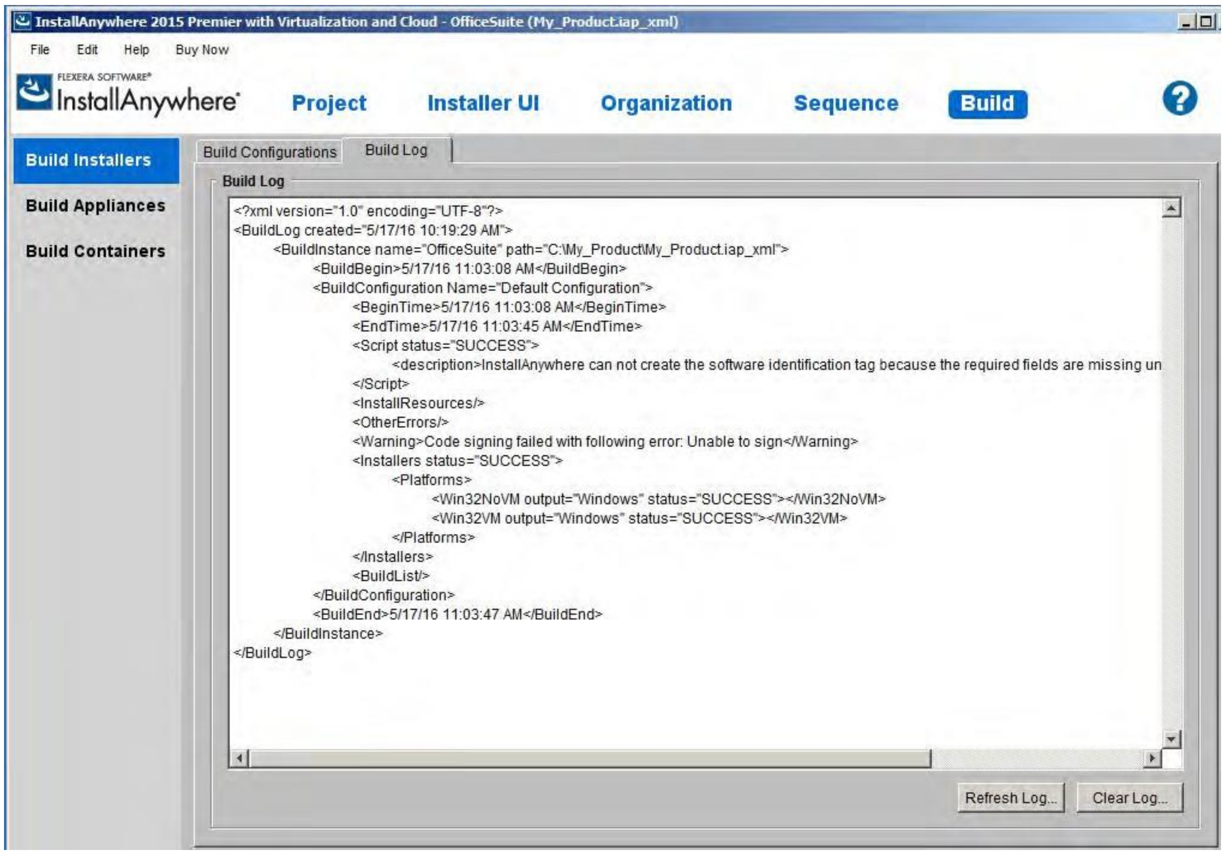
```
ISO CD NAME
    |-Disk1
    |-Disk2
```

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 14, Advanced Organizational Concepts.

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 **Clear Log** to remove the log.



```
<?xml version="1.0" encoding="UTF-8"?>
<BuildLog created="5/17/16 10:19:29 AM">
        <BuildInstance name="OfficeSuite" path="C:\My_Product\My_Product.iap_xml">
                <BuildBegin>5/17/16 11:03:08 AM</BuildBegin>
                <BuildConfiguration Name="Default Configuration">
                        <BeginTime>5/17/16 11:03:08 AM</BeginTime>
                        <EndTime>5/17/16 11:03:45 AM</EndTime>
                        <Script status="SUCCESS">
                                <description>InstallAnywhere can not create the software identification tag because the required fields are missing un
                        </Script>
                        <InstallResources/>
                        <OtherErrors/>
                        <Warning>Code signing failed with following error: Unable to sign</Warning>
                        <Installers status="SUCCESS">
                                <Platforms>
                                        <Win32NoVM output="Windows" status="SUCCESS"></Win32NoVM>
                                        <Win32VM output="Windows" status="SUCCESS"></Win32VM>
                                </Platforms>
                        </Installers>
                        <BuildList/>
                </BuildConfiguration>
                <BuildEnd>5/17/16 11:03:47 AM</BuildEnd>
        </BuildInstance>
</BuildLog>
```

Figure 4-8: 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
- Installer Panel Additions
- Background Images
- Panel Action Settings
- 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
GUI	The GUI mode uses Swing, an end-user interface toolkit provided by the Java Foundation Classes. InstallAnywhere's GUI installer interface provides a rich end-user experience including background graphics, rendered HTML, and alpha transparency for graphics used in the installer. The InstallAnywhere installer itself is an example of a GUI installer.
Console	Console mode provides a TTY or terminal-style interface that can 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.)

Table 5-1 • UI Run Modes

Type	Description
Silent	Silent installers are, as the name implies, a silent installer that requires and provides no end-user interaction. Silent installers can either run without any input, or can accept data from a properties file containing the values for specific InstallAnywhere variables used to control the installation.

***Note** • As of 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** section of the **General UI Settings** tab.

You can specify additional optional settings for the splash screen window such as a title and (for multi-language installations) short instructions and a label for the confirmation button.

The default splash screen appears similar to the following figure.

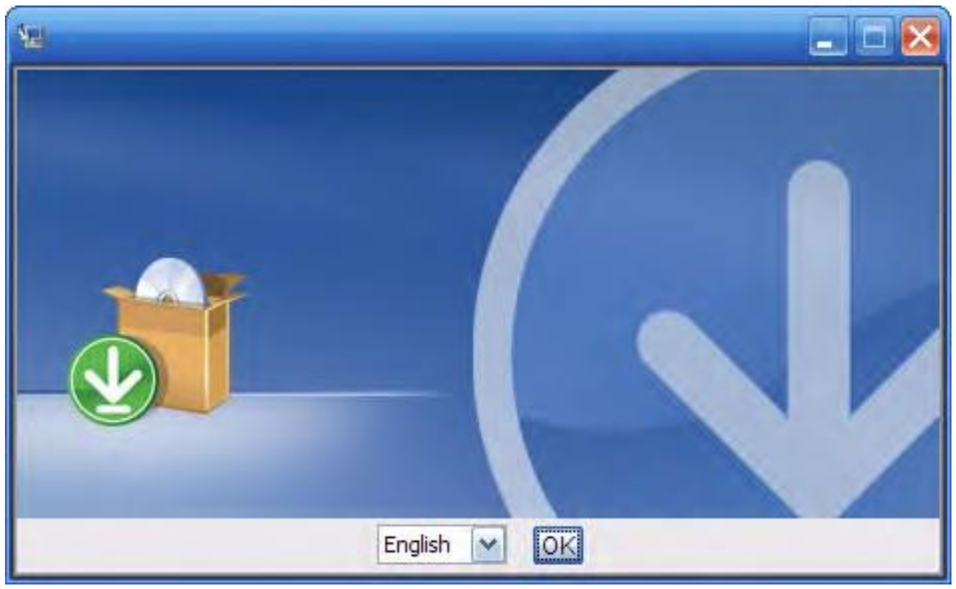


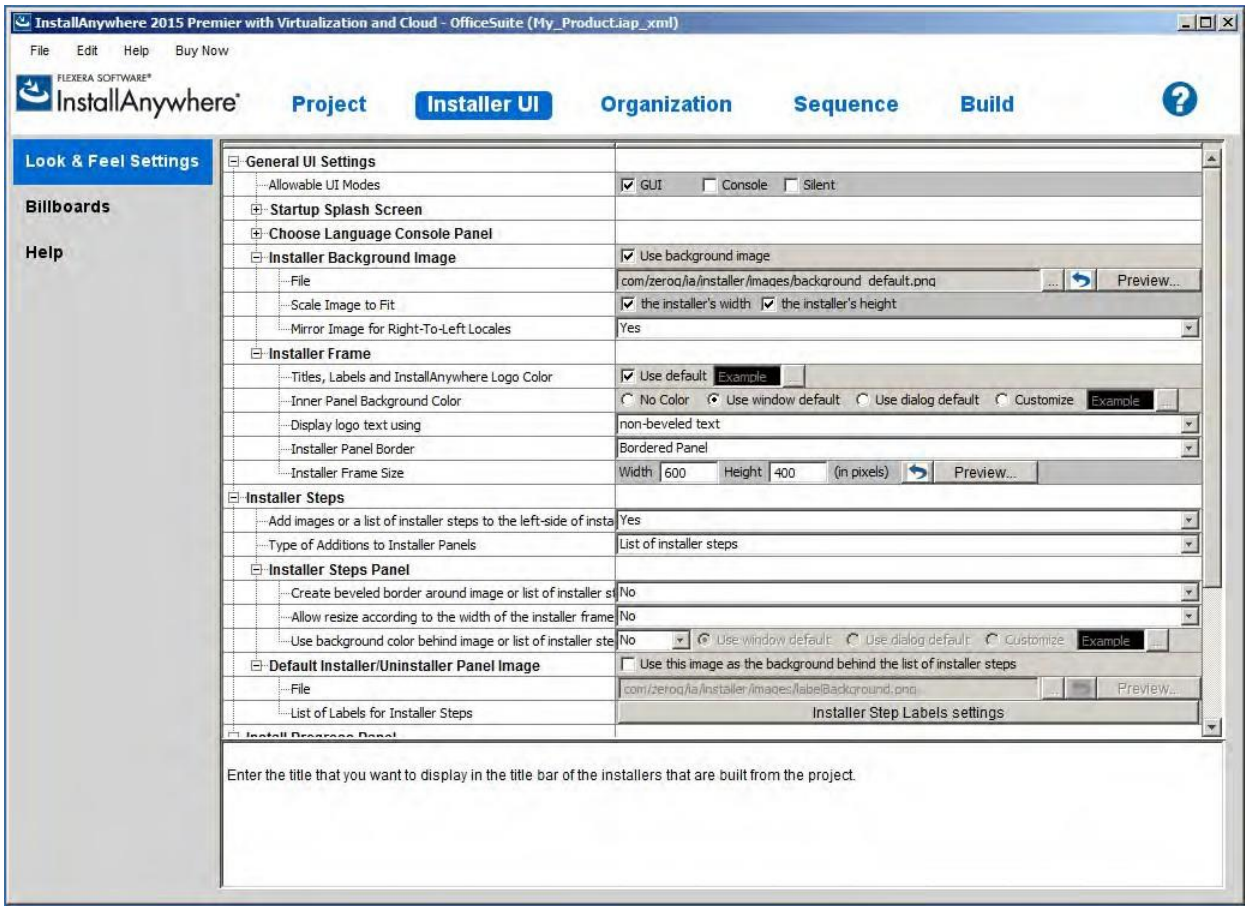
Figure 5-1: Default Splash Screen

***Note** • For information about localizing splash screens, see Chapter 18, Localizing and Internationalizing Installers.*

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 Steps** tab of the **Installer UI > Look & Feel Settings** task.



General UI Settings	
Allowable UI Modes	[✓ GUI] Console Silent
Startup Splash Screen	
Choose Language Console Panel	
Installer Background Image	[✓ Use background image]
File	com/zerog/ia/installer/images/background_default.png ... Preview...
Scale Image to Fit	[✓ the installer's width] [✓ the installer's height]
Mirror Image for Right-To-Left Locales	Yes
Installer Frame	
Titles, Labels and InstallAnywhere Logo Color	[✓ Use default] Example
Inner Panel Background Color	No Color [✓ Use window default] Use dialog default Customize Example
Display logo text using	non-beveled text
Installer Panel Border	Bordered Panel
Installer Frame Size	Width 600 Height 400 (in pixels) Preview...
Installer Steps	
Add images or a list of installer steps to the left-side of insta	Yes
Type of Additions to Installer Panels	List of installer steps
Installer Steps Panel	
Create beveled border around image or list of installer st	No
Allow resize according to the width of the installer frame	No
Use background color behind image or list of installer ste	No [✓ Use window default] Use dialog default Customize Example
Default Installer/Uninstaller Panel Image	Use this image as the background behind the list of installer steps
File	com/zerog/ia/installer/images/labelBackground.png ... Preview...
List of Labels for Installer Steps	Installer Step Labels settings

Enter the title that you want to display in the title bar of the installers that are built from the project.

Figure 5-2: Installer Look & Feel: Installer Steps

At run time, the installer steps might look similar to the following:

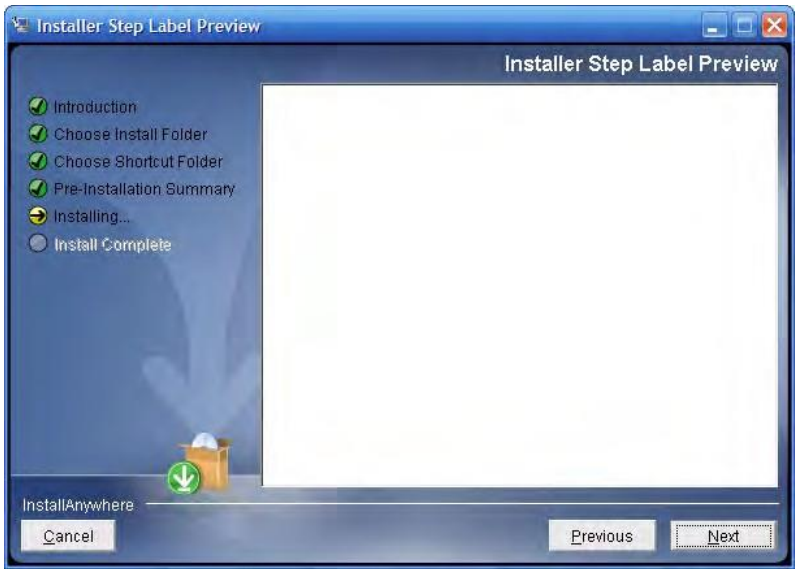


Figure 5-3: Installer Modifications

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 Settings > Installer Steps**, 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, and specify borders and background colors.

Panel Labels

On the **List of Labels for Installer Steps** area of the **Installer Steps Panel** tab, you can 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.

In order to get access these **Installer Steps** go to the **Installer UI** task, click the **Look & Feel Settings** task, then locate the **Installer Progress Panel** item and click the **InstallStep Label Settings** button.

To control the label order, or to edit the contents of a label, use the arrows and other control buttons found to the right of the list of panels.

***Note •** Using the **Installer UI > Look & Feel Settings > Installer Steps** 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 way as HTML targets 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.

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:

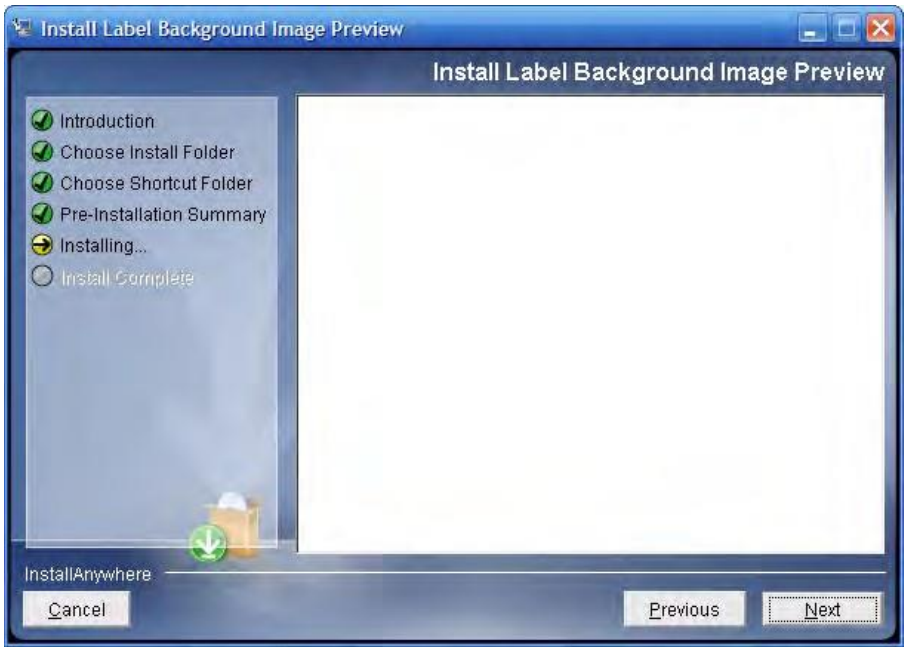


Figure 5-4: Default Background Image

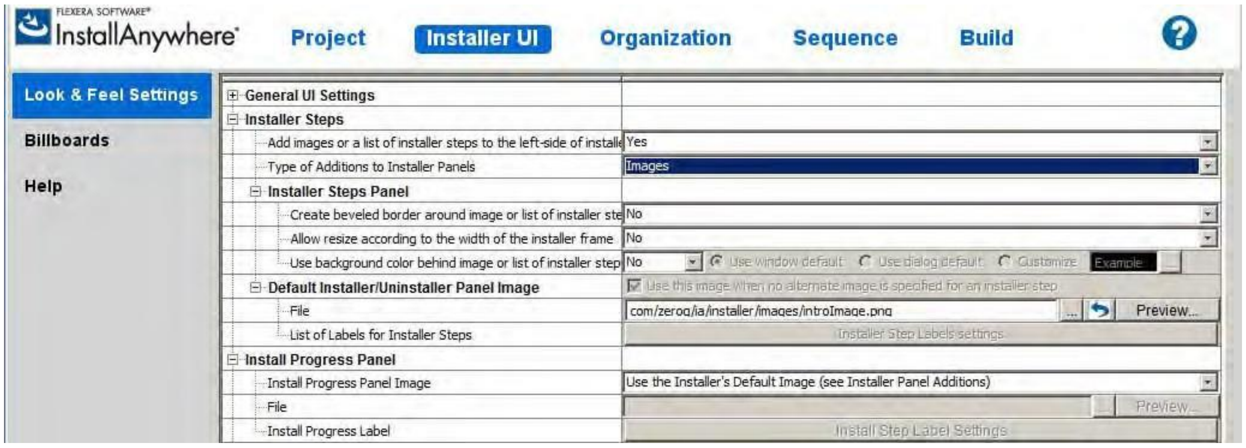
In addition, you can specify the reverse the image when the installer runs using a right-to-left locale (Arabic or Hebrew). For more information about right-to-left locales, refer to Chapter 18, Localizing and Internationalizing Installers.

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 Settings > Installer Steps > 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 Settings task.*



General UI Settings	
Installer Steps	
Add images or a list of installer steps to the left-side of installe	Yes
Type of Additions to Installer Panels	Images
Installer Steps Panel	
Create beveled border around image or list of installer ste	No
Allow resize according to the width of the installer frame	No
Use background color behind image or list of installer step	No [✓ Use window default] Use dialog default Customize Example
Default Installer/Uninstaller Panel Image	✓ Use this image when no alternate image is specified for an installer step
File	com/zerog/ia/installer/images/introImage.png ... Preview...
List of Labels for Installer Steps	Installer Step Labels settings
Install Progress Panel	
Install Progress Panel Image	Use the Installer's Default Image (see Installer Panel Additions)
File	Preview...
Install Progress Label	Install Step Label Settings

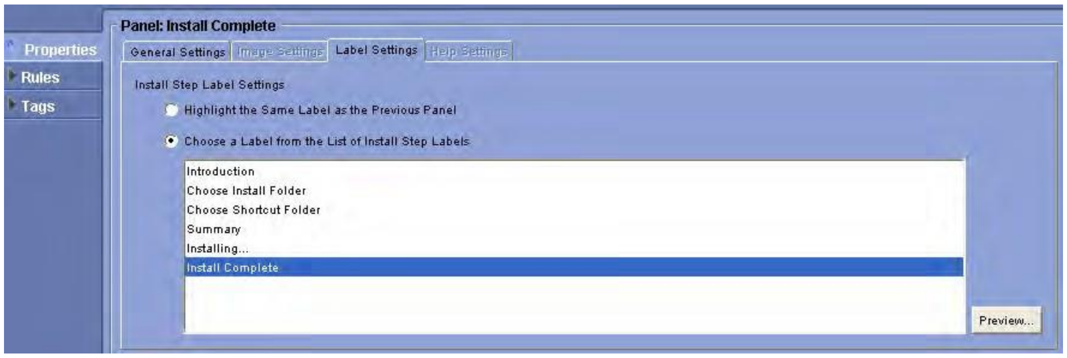
Figure 5-5: Selecting Type of Additions to Installer Panel

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.



Panel: Install Complete

Install Step Label Settings

( ) Highlight the Same Label as the Previous Panel

(●) Choose a Label from the List of Install Step Labels

Introduction
Choose Install Folder
Choose Shortcut Folder
Summary
Installing...
Install Complete

Figure 5-6: Label Settings Tab on the Post-Install Task

Note • *Using the Installer UI > Look & Feel Settingsl task's Installer Steps tab and the **Label 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 Steps** tab use the arrows and other control buttons found to the right of the list of panels.

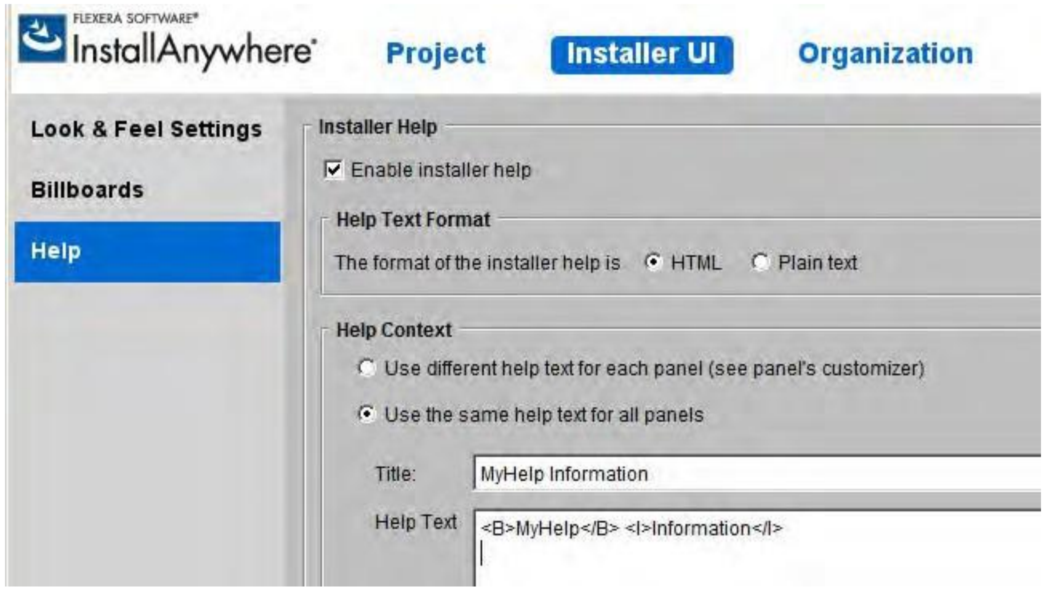
Help

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>
```

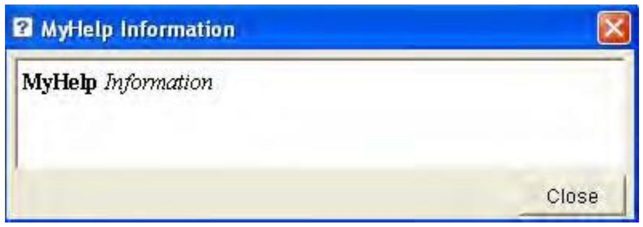
causes InstallAnywhere to display **MyHelp** Information.



Installer Help
[✓] Enable installer help
Help Text Format
The format of the installer help is [✓ HTML] Plain text
Help Context
Use different help text for each panel (see panel's customizer)
[✓ Use the same help text for all panels]
Title: MyHelp Information
Help Text <B>MyHelp</B> <I>Information</I>

Figure 5-7: HTML Tags in Installer Help

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



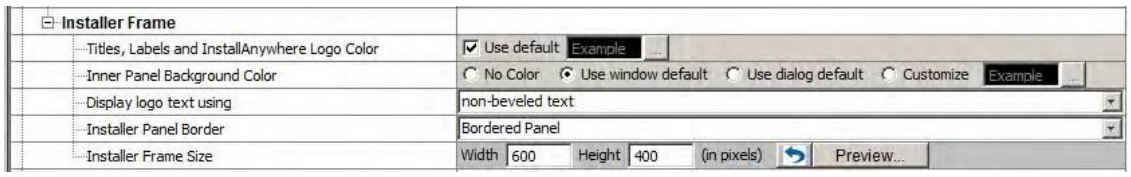
MyHelp *Information*

Figure 5-8: 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** and **Post-Install** tasks.

Frame UI Settings

In the **Installer Frame** area of the **General UI Settings** tab of the **Installer UI > Look & Feel** subtask, you can modify the color scheme of the installer text, and the installer panel's background color. This enables you to match the text and background color schemes of your installer to your organization's branding. You can clear either of the **Use default** check boxes and click **Choose** to browse for the color to use.



Installer Frame

Titles, Labels and InstallAnywhere Logo Color	☑ Use default Example ...
Inner Panel Background Color	No Color [✓ Use window default] Use dialog default Customize Example ...
Display logo text using	non-beveled text
Installer Panel Border	Bordered Panel
Installer Frame Size	Width 600 Height 400 (in pixels) Preview...

Figure 5-9: Installer Frame 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 an InstallAnywhere 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 section 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 later in this chapter).

The rules logic enables 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 • Predefined Rules

Name	Editions	Description
Check File/Folder Attributes	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 readable and/or writable.
Check If File/Folder Exists	E S	This rule is designed to be applied to individual file/folder install actions. It will check to see if the file or folder to which it is attached already exists in the specified install location. You can choose to install either if it does, or does not exist at that location.
Check Platform	E S	This rule enables you to specify actions or files to be run or installed only on specific platforms. The platform is determined by the Java virtual machine, and reported to the installer.
Check Running Mode		Enables you to customize the events that occur when an end user launches Maintenance Mode.
Check System Architecture	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.
Check User-Chosen Language	E S	You can use Check User-Chosen Language to make installation decision based on the locale chosen by the end user at installation time.
Compare File Modification Timestamp	E	This rule enables you to compare the timestamp of an existing target file in order to make an overwrite decision.
Compare InstallAnywhere Variables	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.
Evaluate Custom Rule	E	Custom Rules are rules built using the specifications outlined in the InstallAnywhere API and can be tailored to fit the needs of your installation. Information concerning custom and API development is covered in Chapter 13.

Table 5-2 • Predefined Rules

Name	Editions	Description
Match Regular Expression	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.
System i (i5/OS) Licensed Program Exists Condition Rule	E	This rule allows developers to determine if a licensed program, matched by the Licensed Program Identifier, Option, and Release Level, exists on the target i5/OS system. Any action to which this rule is assigned can be conditionally performed based on whether a matching licensed program is found or not found on the i5/OS system.
System i (i5/OS) Primary Language Install Condition Rule	E	This rule allows developers to compare one or more primary language settings with the primary language of a target i5/OS system. Any action to which this rule is assigned can be conditionally performed based on whether the primary language was matched on the target i5/OS system.
System i (i5/OS) Program Temporary Fix (PTF) Condition Rule	E	This rule allows developers to determine if a PTF, matched by the Licensed Program Identifier, PTF ID, Release Level, and PTF Status, exists on the target i5/OS system. Any action to which this rule is assigned can be conditionally performed based on whether a matching PTF is found or not found on the i5/OS system.

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 user's name is `Jim`, what effect will the following "install only if" rule have on a panel?

Operand 1	Operator	Operand 2
`$prop.user.name$`	equals	**`jim`**

    a. The panel will display

    b. The panel will not display

Table 5-3 • Answers to Quick Quiz

Question	Answer
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
- Using the Organization Task
- Organizing Your Features and Components
- Adding Files to Your Project
- Quick Quiz

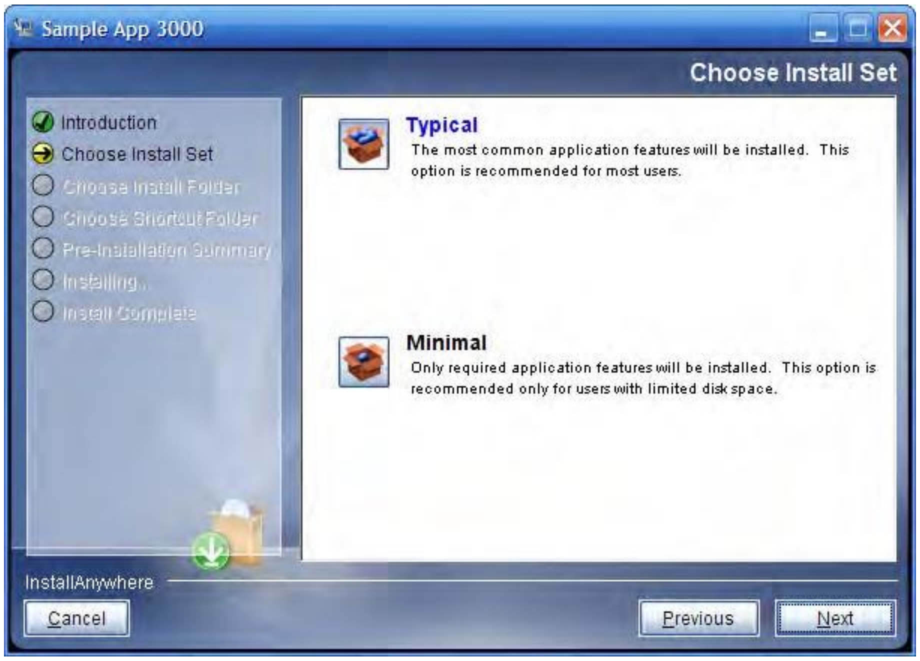
Install Sets, Features, and Components

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.



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

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

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**.

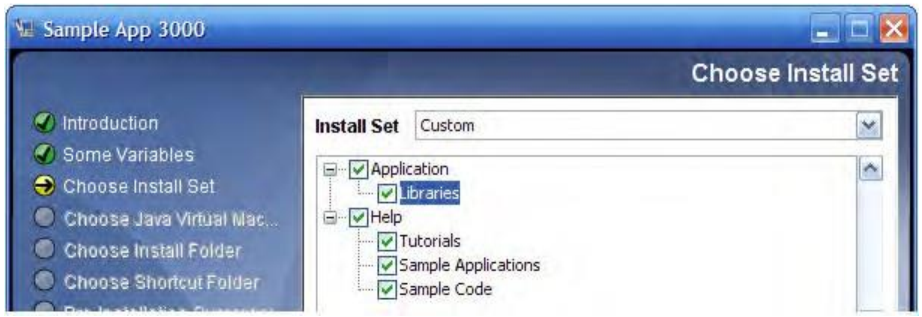


Figure 6-2: Adding Specific Features

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, and merge modules.

Note • *Merge Modules are discussed in Chapter 14, Advanced Organizational Concepts.*

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.

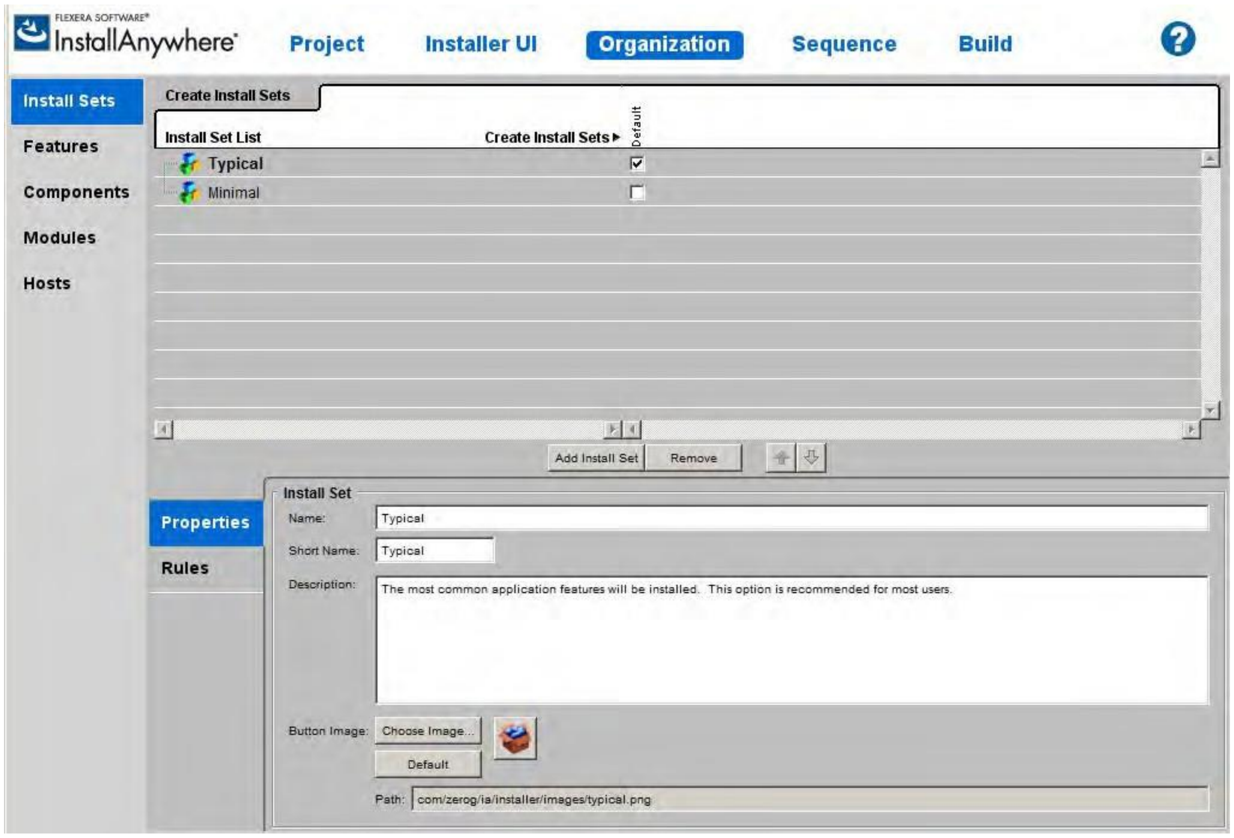
- If an install set is added in the **Organization > Install Sets** task, features can be assigned to that install set in **Organization > Features**.
- If a feature is added in **Organization > Features,** components can be assigned to that feature in **Organization > Components**.
- If you add a component 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.

This section includes information on the following topics:

- Install Sets
- Features
- Components
- Types of Components

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.



Install Set List	Default
Typical	☑
Minimal	☐

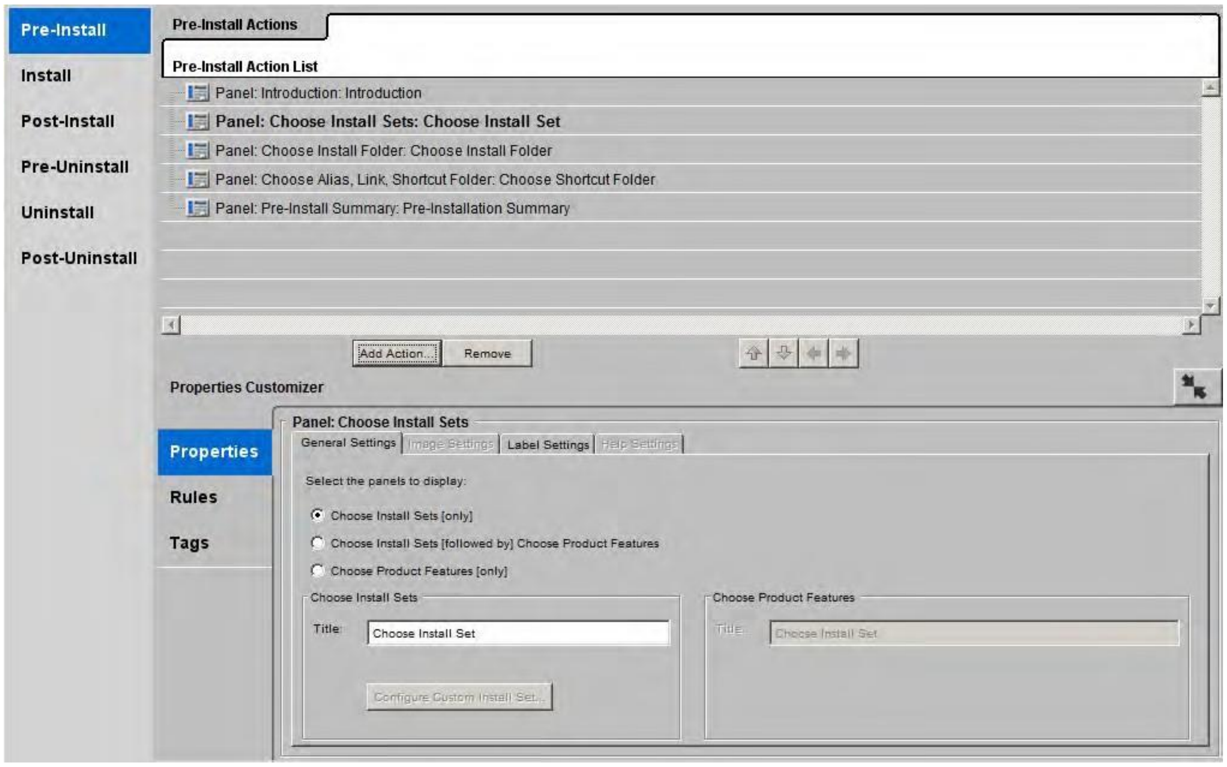
Name: Typical
Short Name: Typical
Description: The most common application features will be installed. This option is recommended for most users.
Button Image:
Path: com/zerog/ia/installer/images/typical.png

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 **General Settings** tab of the **Choose Install Sets** panel customizer.



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

Panel: Choose Install Sets

Select the panels to display:

[✓ Choose Install Sets [only]]
Choose Install Sets [followed by] Choose Product Features
Choose Product Features [only]

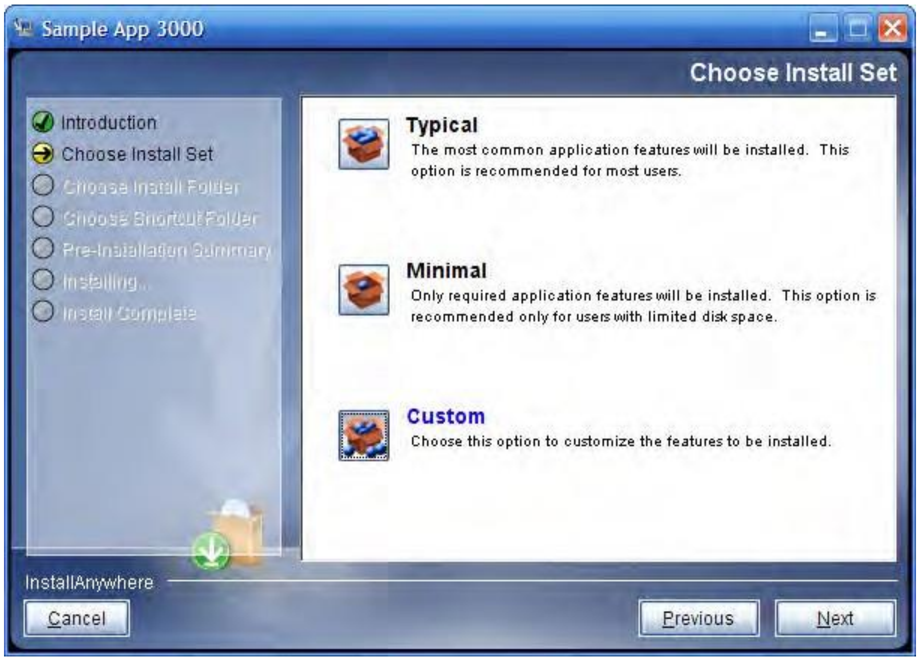
Choose Install Sets
Title: Choose Install Set

Choose Product Features
Title: Choose Install Set

Figure 6-4: Choose Install Sets Customizer

Using the customizer, you can also modify the custom install set’s name, description, and icon by selecting either **Choose Install Sets [followed by] Choose Product Features** or **Choose Product Features [only]** option and then clicking the **Configure Custom Install Set** button.

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 Set** panel containing the **Custom** install set appears as follows.



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

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

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

Figure 6-5: Custom Install Set

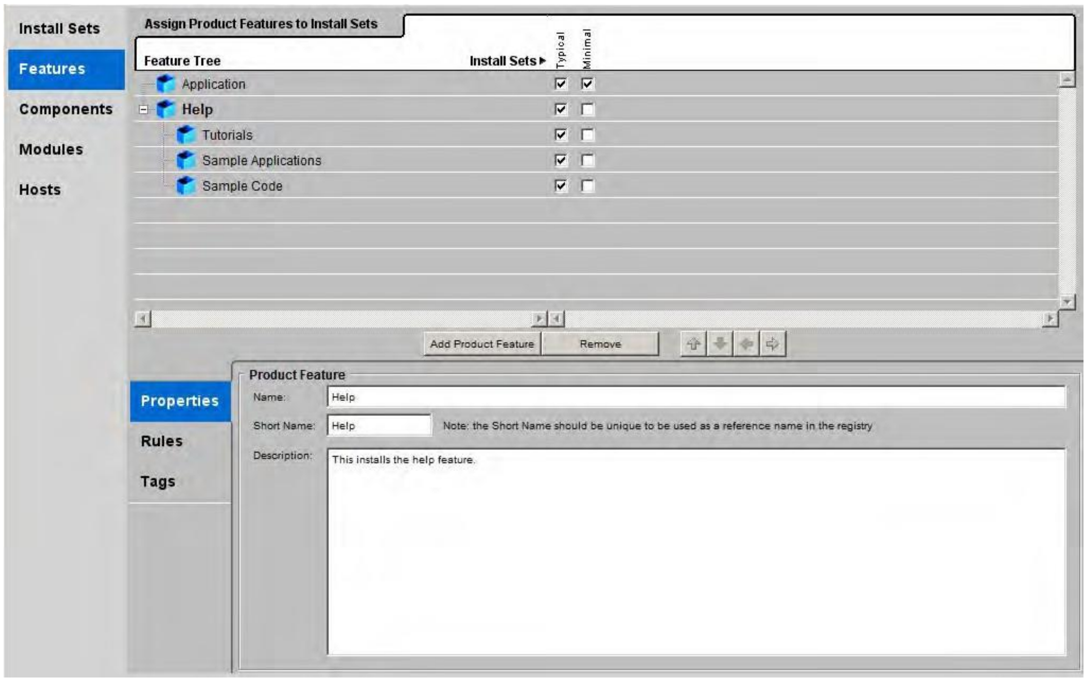
The feature-selection panel in which the user selects which individual features to install is pictured earlier in this chapter.

You can associate one or more rules with an install set, by selecting the **Rules** tab 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.



Feature Tree	Typical	Minimal
Application	✓	✓
Help	✓	
Tutorials	✓	
Sample Applications	✓	
Sample Code	✓	

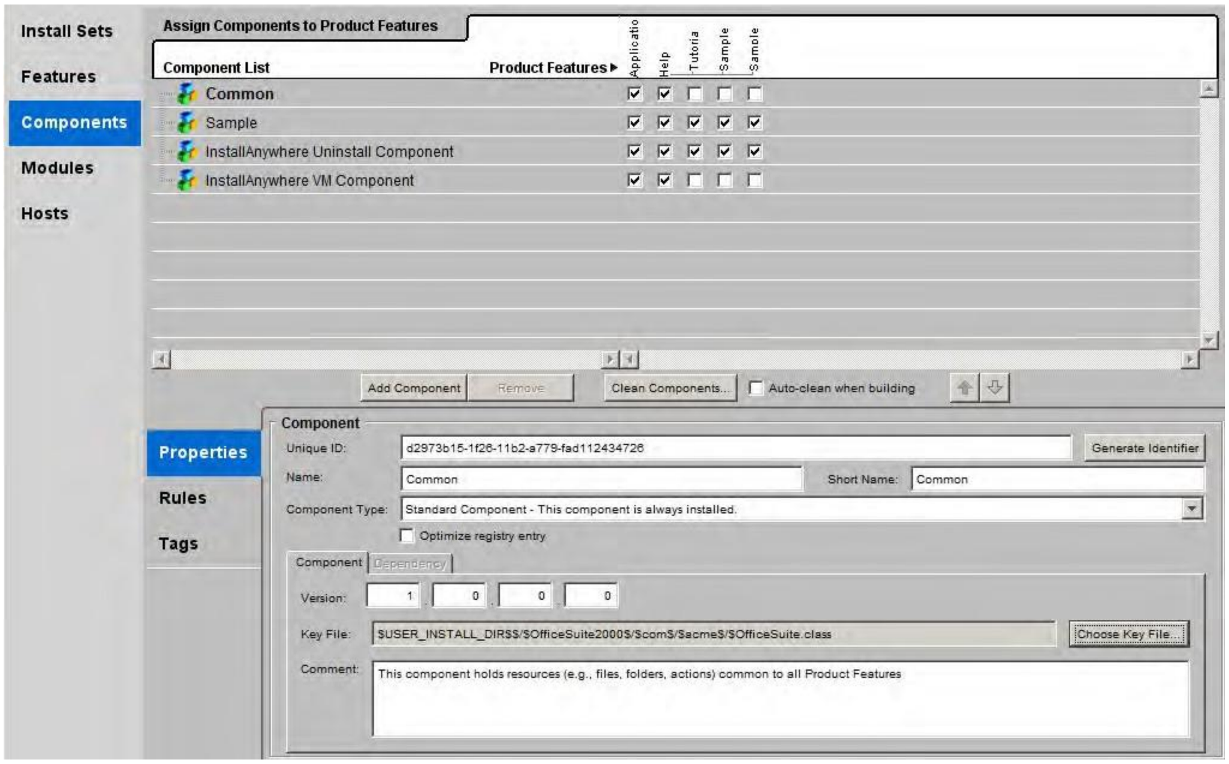
Name: Help
Short Name: Help Note: the Short Name should be unique to be used as a reference name in the registry
Description: This installs the help feature.

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.



Component List	Applicatio	Help	Tutoria	Sample	Samole
Common	✓	✓			
Sample	✓	✓	✓	✓	✓
InstallAnywhere Uninstall Component	✓	✓	✓	✓	✓
InstallAnywhere VM Component	✓	✓			

Component

Unique ID: d2973b15-1f26-11b2-a779-fad112434726

Name: Common   Short Name: Common

Component Type: Standard Component - This component is always installed.

Optimize registry entry

Component | Dependency

Version: 1 . 0 . 0 . 0

Key File: $USER_INSTALL_DIR$$/$OfficeSuite2000$/com/$acme$/$OfficeSuite.class

Comment: This component holds resources (e.g., files, folders, actions) common to all Product 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.

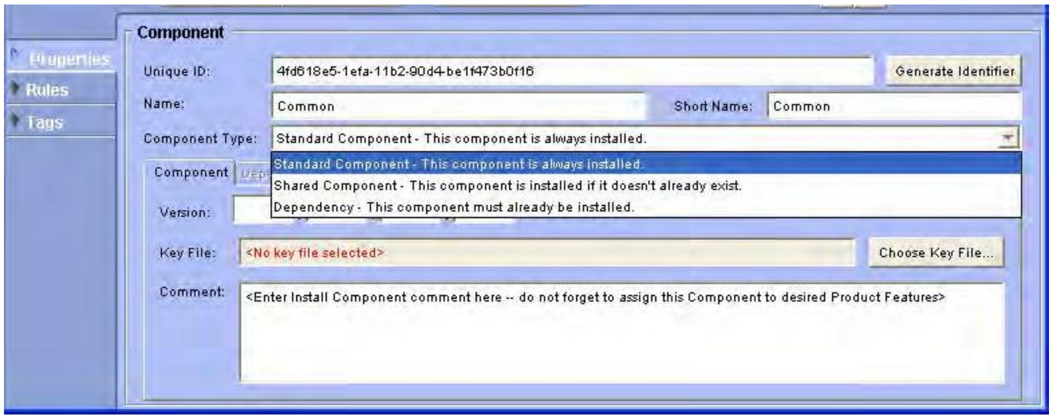
On occasion you may have components that are no longer needed or do not have any files assigned to them. To remove empty components from your project, click **Clean Components** in the **Components** task.

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.



Unique ID: 4fd618e5-1efa-11b2-90d4-be1f473b0f16
Name: Common
Short Name: Common
Component Type: Standard Component - This component is always installed.
Standard Component - This component is always installed.
Shared Component - This component is installed if it doesn't already exist.
Dependency - This component must already be installed.
Version:
Key File: <No key file selected>
Comment: <Enter Install Component comment here -- do not forget to assign this Component to desired Product Features>

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.

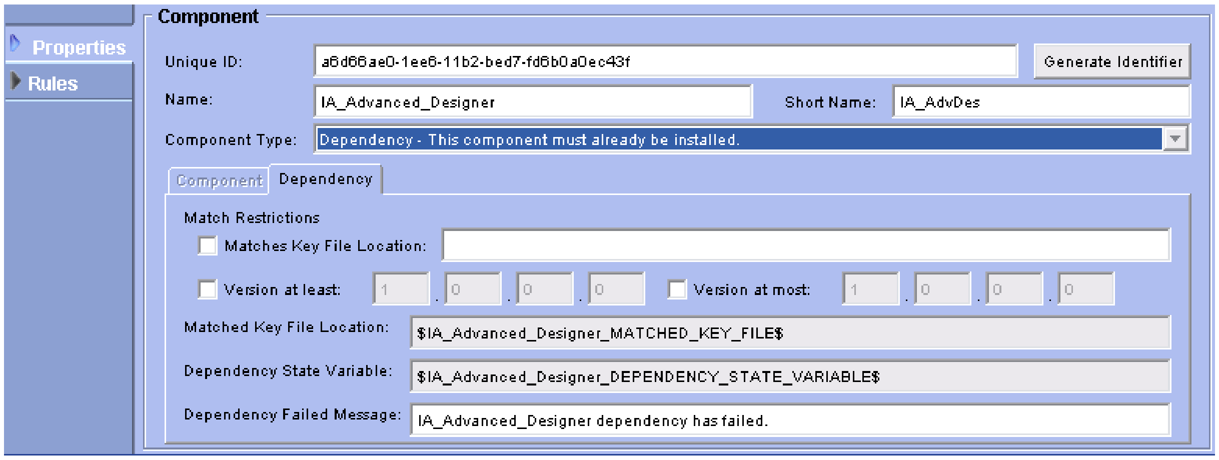
Shared Components

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.



Unique ID: a6d66ae0-1ee6-11b2-bed7-fd6b0a0ec43f

Name: IA_Advanced_Designer

Short Name: IA_AdvDes

Component Type: Dependency - This component must already be installed.

Matches Key File Location:

Version at least: 1 . 0 . 0 . 0

Version at most: 1 . 0 . 0 . 0

Matched Key File Location: $IA_Advanced_Designer_MATCHED_KEY_FILE$

Dependency State Variable: $IA_Advanced_Designer_DEPENDENCY_STATE_VARIABLE$

Dependency Failed Message: IA_Advanced_Designer dependency has failed.

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.

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.

Note • *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. But 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.

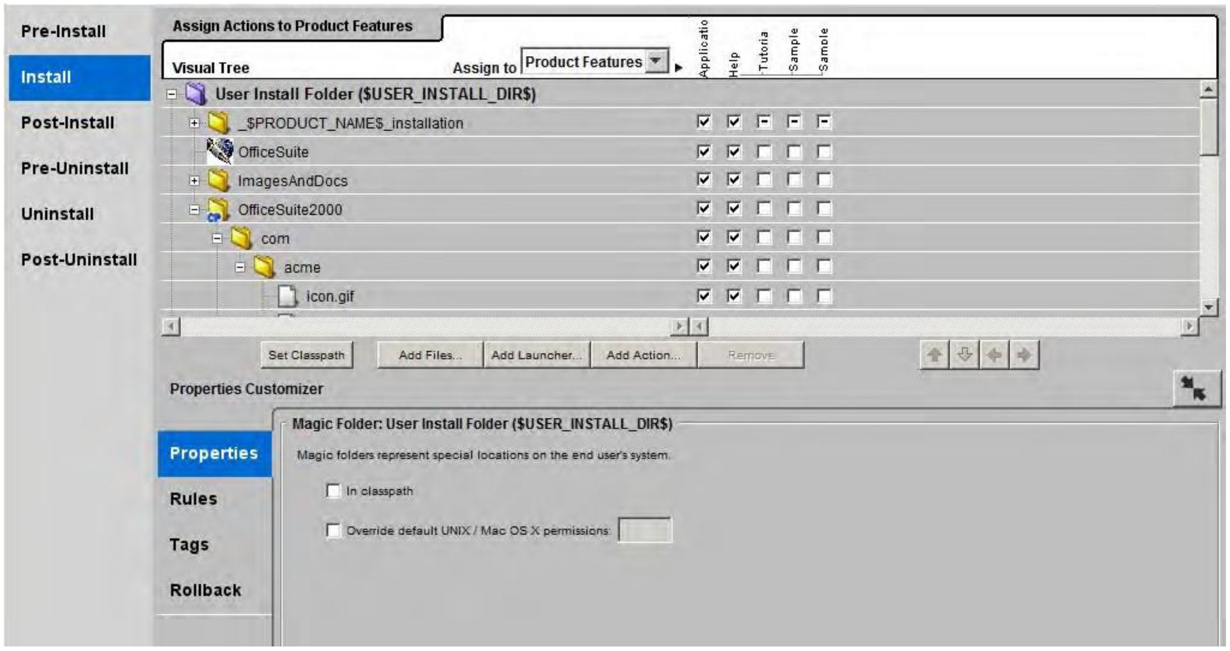
Adding Files to Your Project

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

- Adding Individual Files
- Magic Folders
- Adding Directories with SpeedFolders
- Specifying Source Files with Manifest Files

Adding Individual Files

You add individual files to your project using the **Install** task under the **Sequence** 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.



Magic Folder: User Install Folder ($USER_INSTALL_DIR$)

Magic folders represent special locations on the end user's system.

☐ In classpath

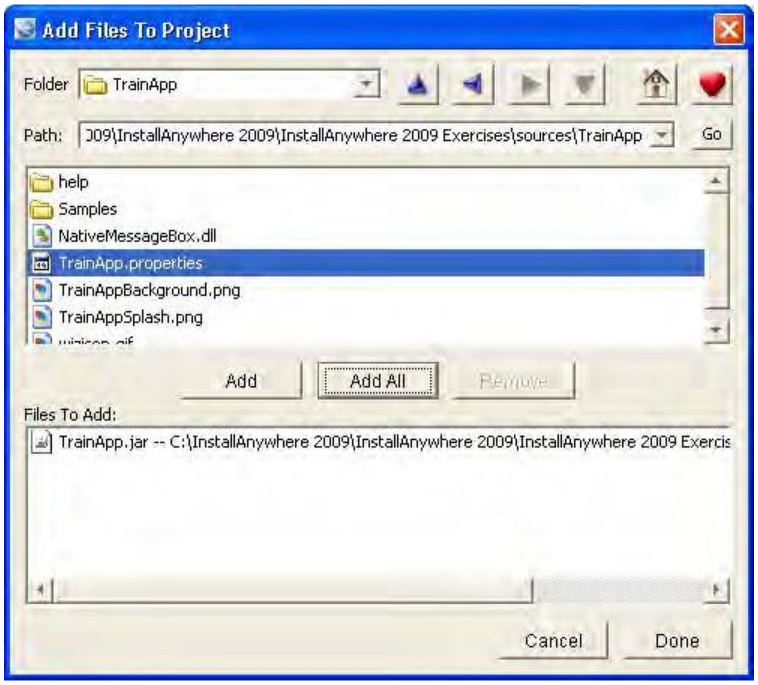
☐ Override default UNIX / Mac OS X permissions:

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.

Note • *You can control whether actions are by default associated with features or with components using the options in the **Default Install Task View** area on the **General Settings** tab of the **InstallAnywhere Preferences** dialog box.*

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.



Folder: TrainApp

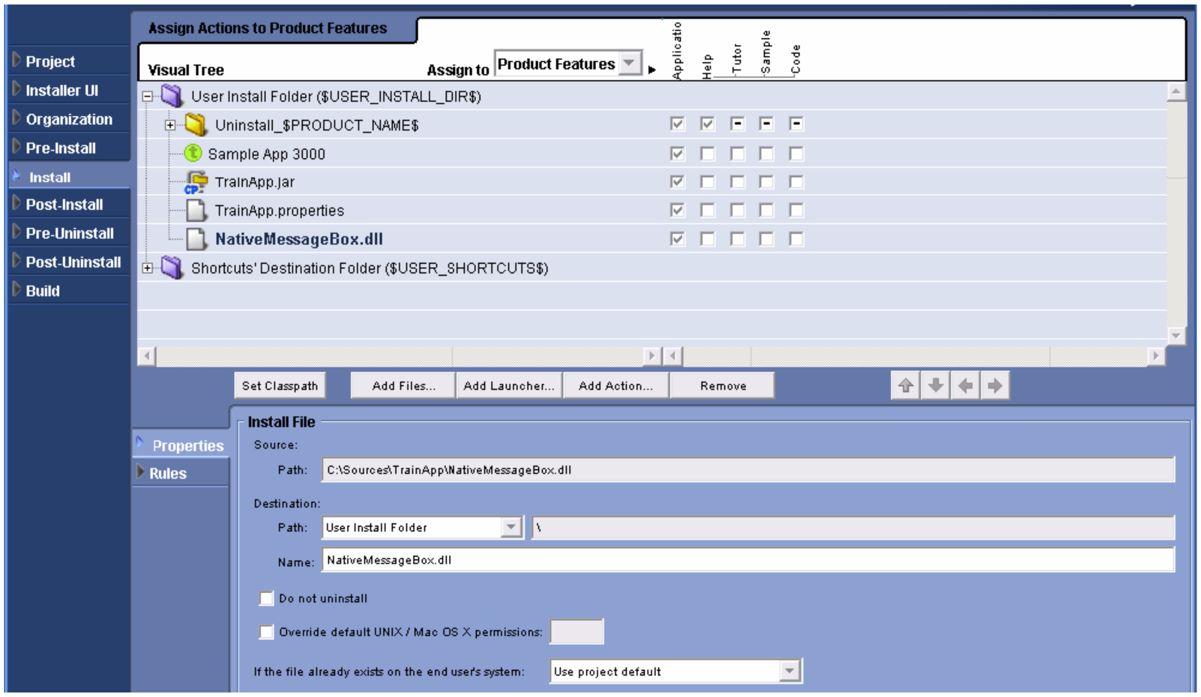
Path:)09\InstallAnywhere 2009\InstallAnywhere 2009 Exercises\sources\TrainApp

help
Samples
NativeMessageBox.dll
TrainApp.properties
TrainAppBackground.png
TrainAppSplash.png
wizicon.gif

Files To Add:
TrainApp.jar -- C:\InstallAnywhere 2009\InstallAnywhere 2009\InstallAnywhere 2009 Exercis

Figure 6-11: Adding Files to a Project

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



User Install Folder ($USER_INSTALL_DIR$)
Uninstall_$PRODUCT_NAME$
Sample App 3000
TrainApp.jar
TrainApp.properties
NativeMessageBox.dll
Shortcuts' Destination Folder ($USER_SHORTCUTS$)

Install File

Source:
Path: C:\Sources\TrainApp\NativeMessageBox.dll

Destination:
Path: User Install Folder \
Name: NativeMessageBox.dll

Do not uninstall

Override default UNIX / Mac OS X permissions:

If the file already exists on the end user's system: Use project default

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:

Table 6-1 • Variables Not Inititialized at Install Time

Variable	Description
`$USER_INSTALL_DIR$`	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**.

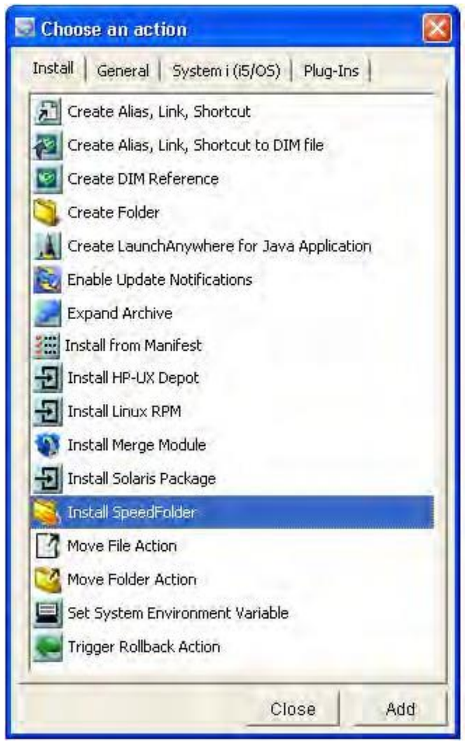
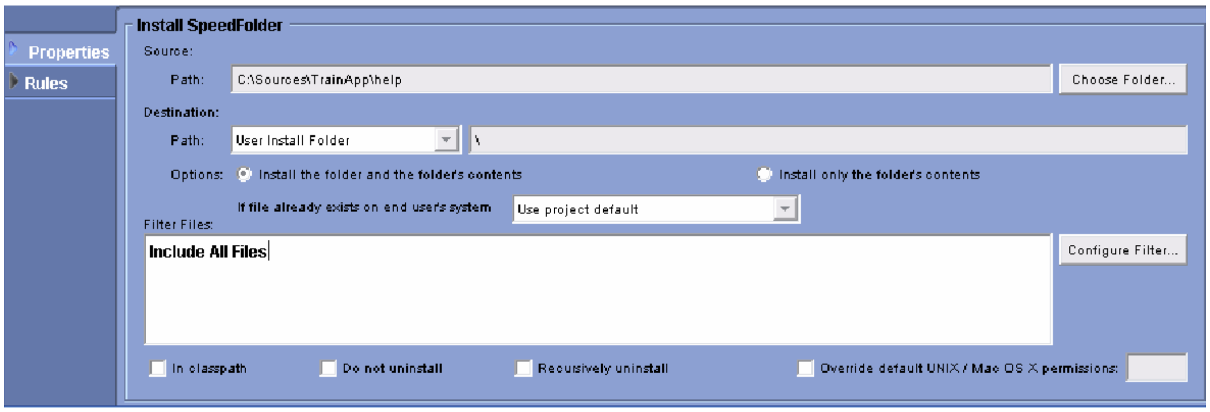


Figure 6-13: Installing a SpeedFolder

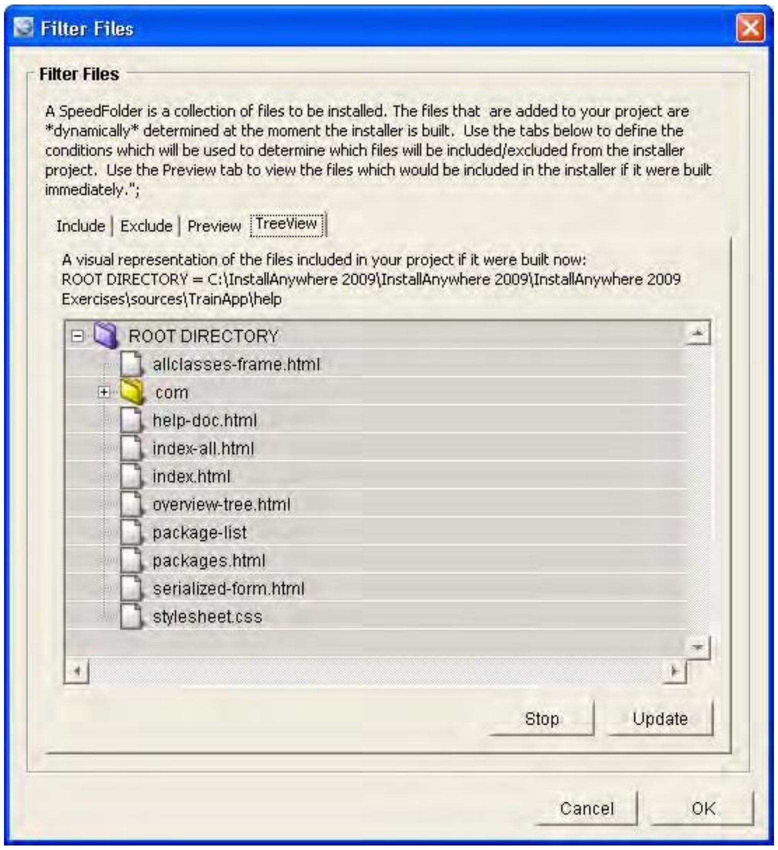
After you click **Add**, the customizer for the **Install SpeedFolder** action is where you specify the source and destination locations, along with overwrite and uninstallation behavior.



Source:
Path: C:\Sources\TrainApp\help
Destination:
Path: User Install Folder \
Options: Install the folder and the folder's contents / Install only the folder's contents
If file already exists on end user's system: Use project default
Filter Files:
Include All Files
In classpath / Do not uninstall / Recursively uninstall / Override default UNIX / Mac OS X permissions:

Figure 6-14: Install SpeedFolder Properties

By clicking **Configure Filter**, you can specify files or folders to include and exclude from the SpeedFolder. 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.


Filter Files

Filter Files

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

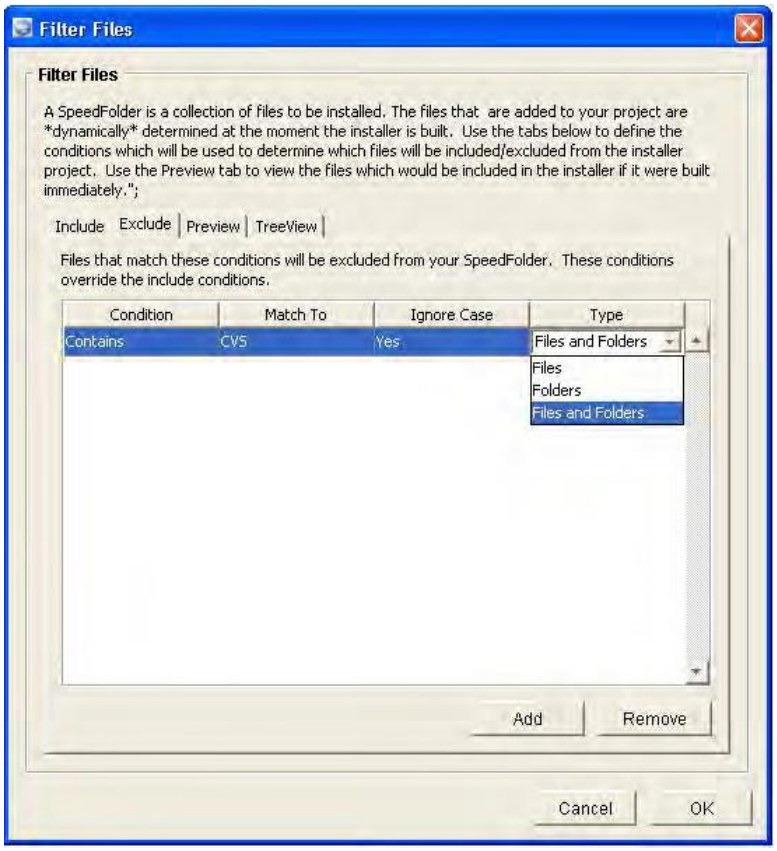
Include | Exclude | Preview | TreeView

A visual representation of the files included in your project if it were built now:
ROOT DIRECTORY = C:\InstallAnywhere 2009\InstallAnywhere 2009\InstallAnywhere 2009 Exercises\sources\TrainApp\help

- ROOT DIRECTORY
  - allclasses-frame.html
  - com
  - help-doc.html
  - index-all.html
  - index.html
  - overview-tree.html
  - package-list
  - packages.html
  - serialized-form.html
  - stylesheet.css

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.


Filter Files

Filter Files

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

Include | Exclude | Preview | TreeView

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

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

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 that specify a list of files and directories. A manifest file has a specific format, illustrated below. The format specifies each 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 whether 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, use the following format:

```
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, use the following format:

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

To set a file's permissions on Unix, use the following format:

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

For directories, use the following format:

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

The following are 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

Table 6-2 • Answers to Quick Quiz

Question	Answer
1	b
2	c

7

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.

- **General actions**—Most general actions make system changes, search the target system for data, or read data from the target system. Examples include creating or deleting folders, modifying a text file, or reading data from the Windows registry. These actions are generally transparent to the end user, and do not require any user input.
- **Panel actions**—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**—Console actions are displays of information and requests for user input used during a command-line installation.
- **Plug-in actions**—Custom code can also be integrated with the InstallAnywhere Designer and will appear as a plug-in action.
- **Action groups**—Action groups make InstallAnywhere projects more manageable and easier to understand by enabling you to logically group a set of actions or panels. Rules applied to an action group affect all of the actions or panels contained inside the group. Action groups are useful with complex installations that contain numerous actions and panels.

Information on advanced and panel actions is presented in the following sections:

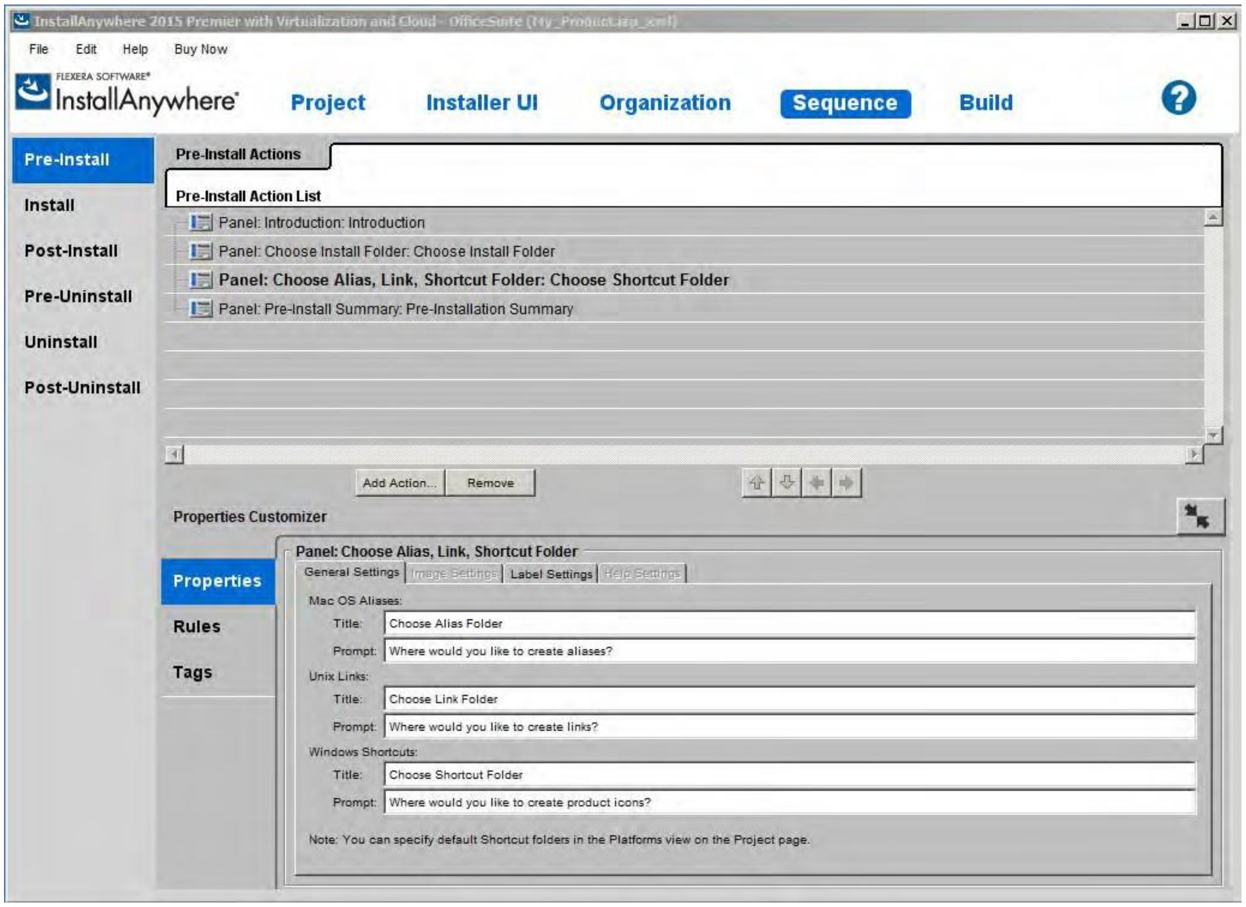
- Adding an Action
- Examples of Common Actions
- Display HTML Panel

- Application Servers and Database Servers
- Common Properties
- Quick Quiz

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**, **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.



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

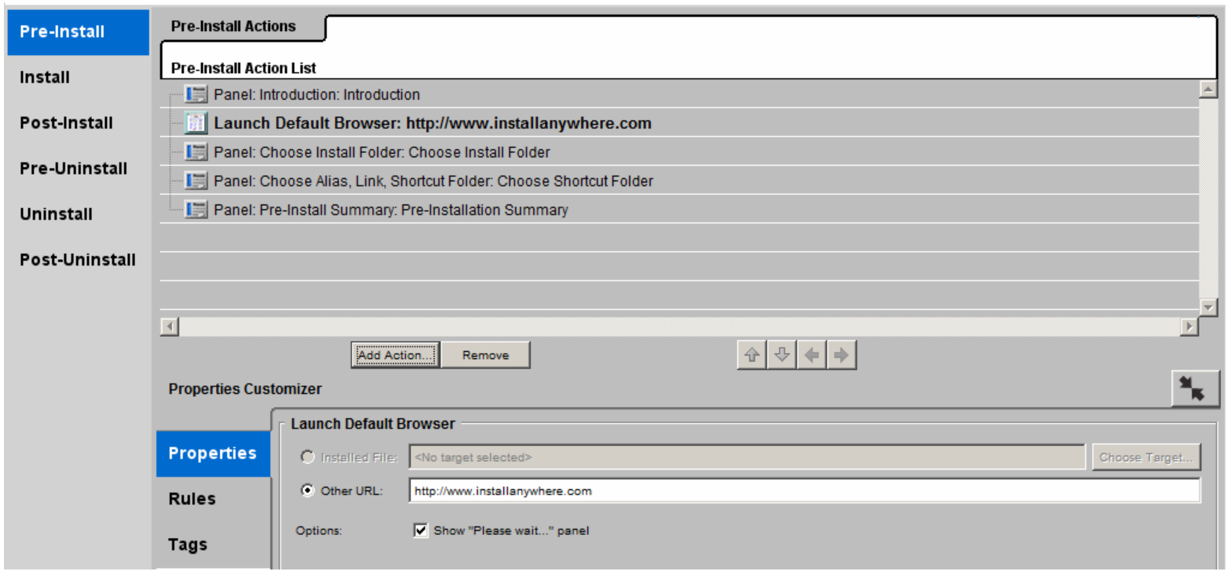
Panel: Choose Alias, Link, Shortcut Folder
Mac OS Aliases:
Title: Choose Alias Folder
Prompt: Where would you like to create aliases?
Unix Links:
Title: Choose Link Folder
Prompt: Where would you like to create links?
Windows Shortcuts:
Title: Choose Shortcut Folder
Prompt: Where would you like to create product icons?
Note: You can specify default Shortcut folders in the Platforms view on the Project page.

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.

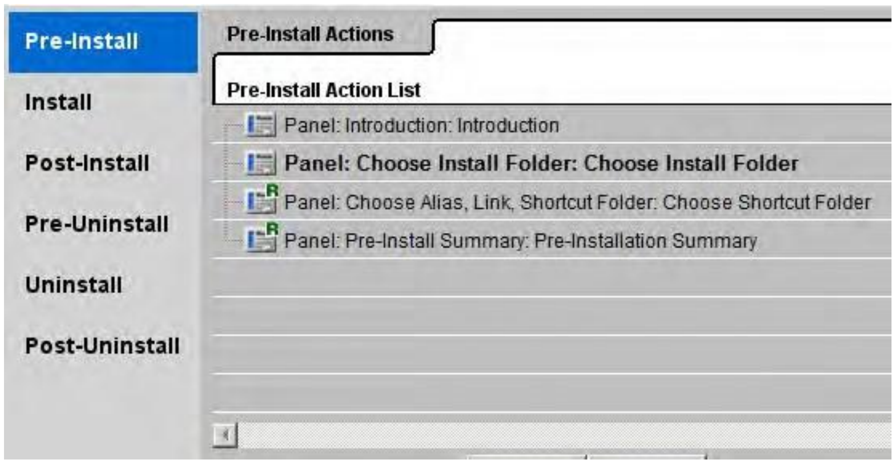


Panel: Introduction: Introduction
Launch Default Browser: http://www.installanywhere.com
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

Launch Default Browser
Installed File: <No target selected>
[✓ Other URL:] http://www.installanywhere.com
Options: [✓] Show "Please wait..." panel

Figure 7-2: Launch Default Browser Action

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, while the second two actions have rules added.



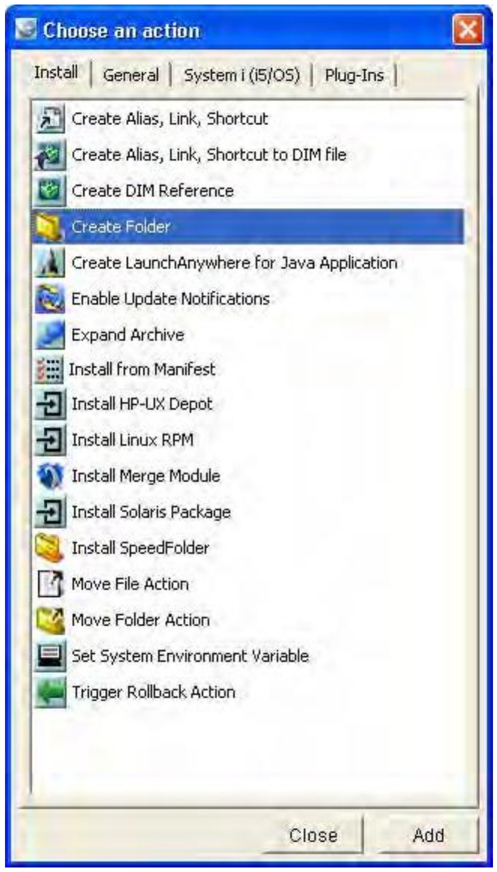
Panel: Introduction: Introduction
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

Figure 7-3: Actions with Associated Rules

Action Availability by Task

Some actions are available only to certain 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.



Create Alias, Link, Shortcut
Create Alias, Link, Shortcut to DIM file
Create DIM Reference
Create Folder
Create LaunchAnywhere for Java Application
Enable Update Notifications
Expand Archive
Install from Manifest
Install HP-UX Depot
Install Linux RPM
Install Merge Module
Install Solaris Package
Install SpeedFolder
Move File Action
Move Folder Action
Set System Environment Variable
Trigger Rollback Action

Figure 7-4: Choosing an Action

The **Pre-Install**, **Post-Install**, and **Uninstall** tasks have other actions listed under **Panels**, **Consoles**, **System i (i5/OS)**, 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**.

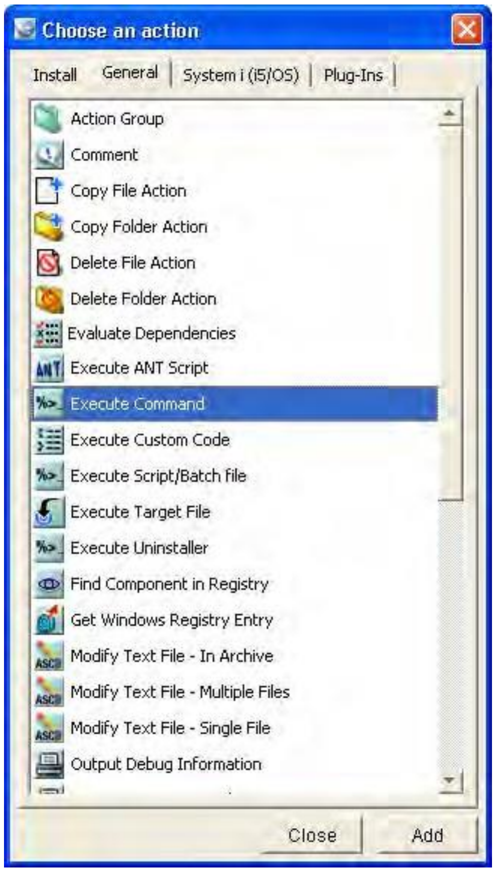


Figure 7-5: General Actions

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. Without a group, you would need to apply those rules to each individual action.

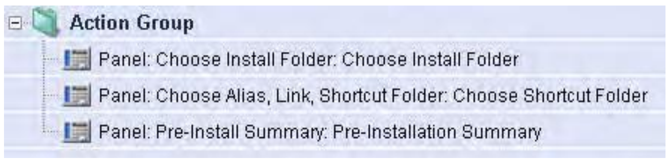

Action Group
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

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.

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 Panel
- Set System Environment Variable

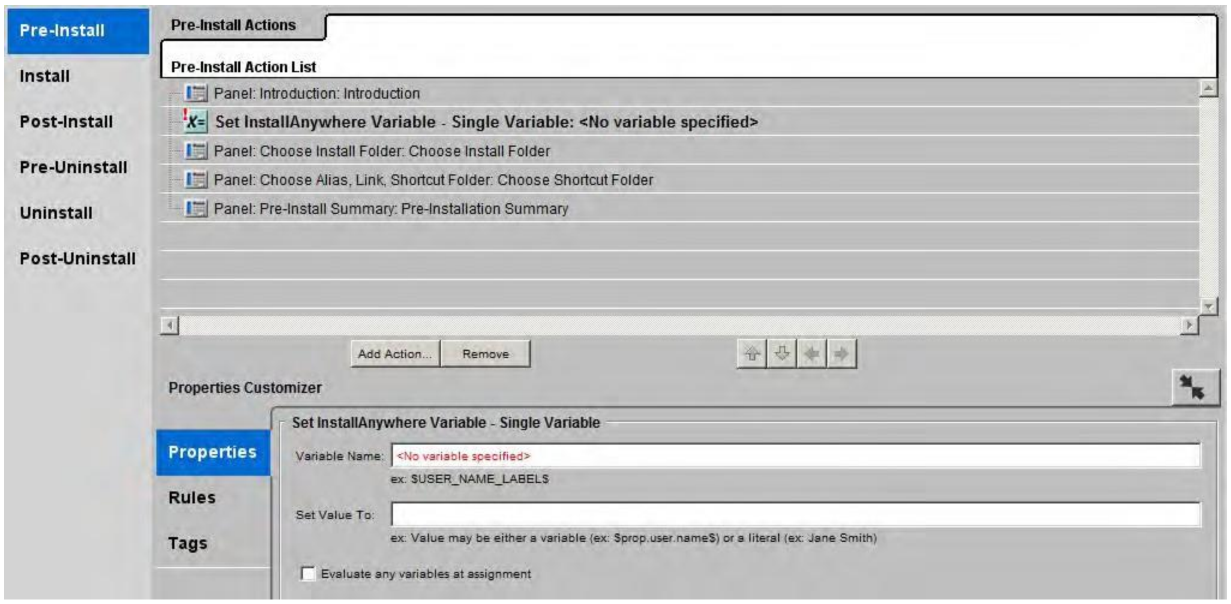
Set InstallAnywhere Variable Action

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.



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

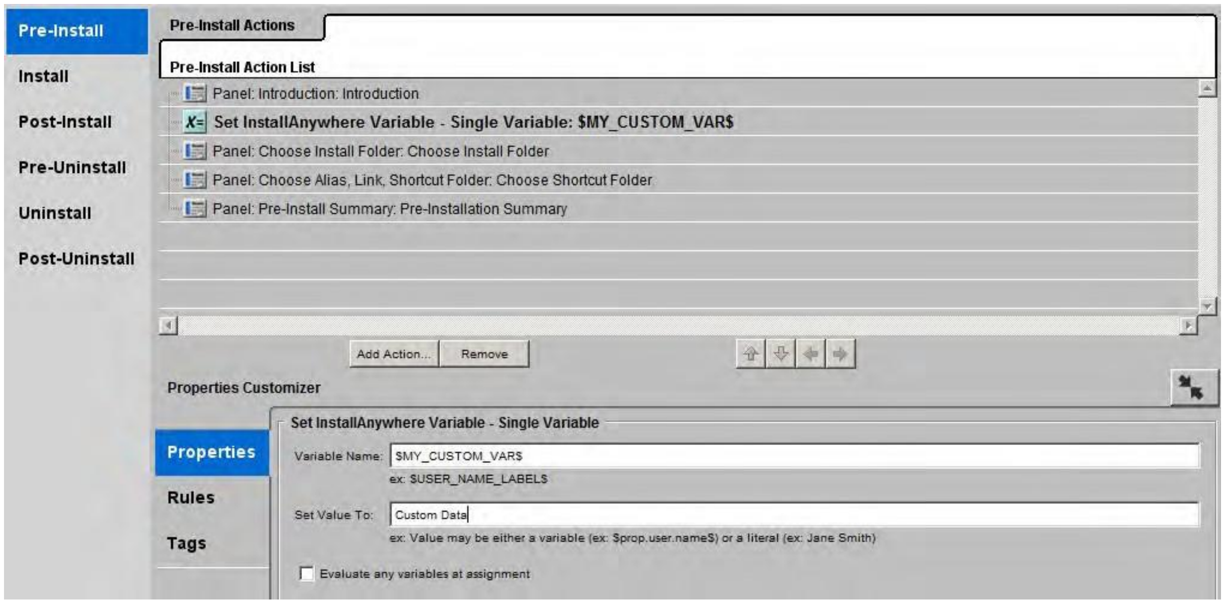
Set InstallAnywhere Variable - Single Variable
Variable Name: <No variable specified>
ex: $USER_NAME_LABEL$
Set Value To:
ex: Value may be either a variable (ex: $prop.user.name$) or a literal (ex: Jane Smith)
☐ Evaluate any variables at assignment

Figure 7-7: Customizer for Set InstallAnywhere Variable Action

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.



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

Set InstallAnywhere Variable - Single Variable
Variable Name: MY_CUSTOM_VAR
ex: $USER_NAME_LABEL$
Set Value To: Custom Data
ex: Value may be either a variable (ex: $prop.user.name$) or a literal (ex: Jane Smith)
☐ Evaluate any variables at assignment

Figure 7-8: Setting Variable Name and Value

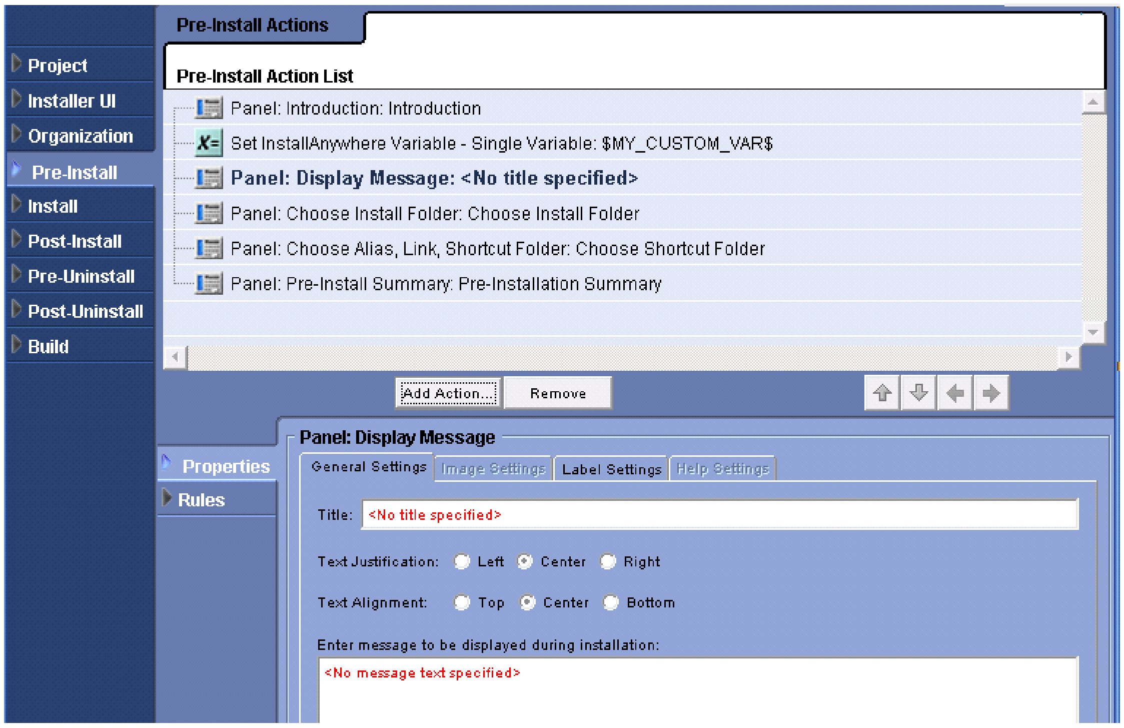
You can then use the expression `$MY_CUSTOM_VAR$` in a later action, and the expression will be replaced with the variable's value.

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.



Pre-Install Action List

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

Panel: Display Message

Title: <No title specified>

Text Justification: Left [✓ Center] Right

Text Alignment: Top [✓ Center] Bottom

Enter message to be displayed during installation:

<No message text specified>

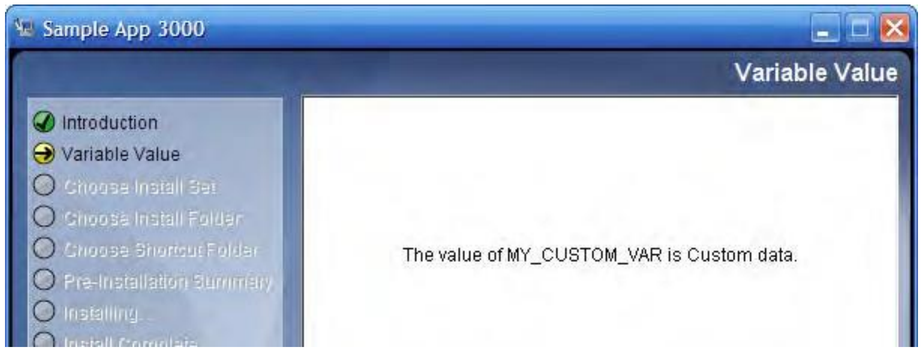
Figure 7-9: Customizing Display Messages

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:



The value of MY_CUSTOM_VAR is Custom data.

Figure 7-10: Variable Value Display

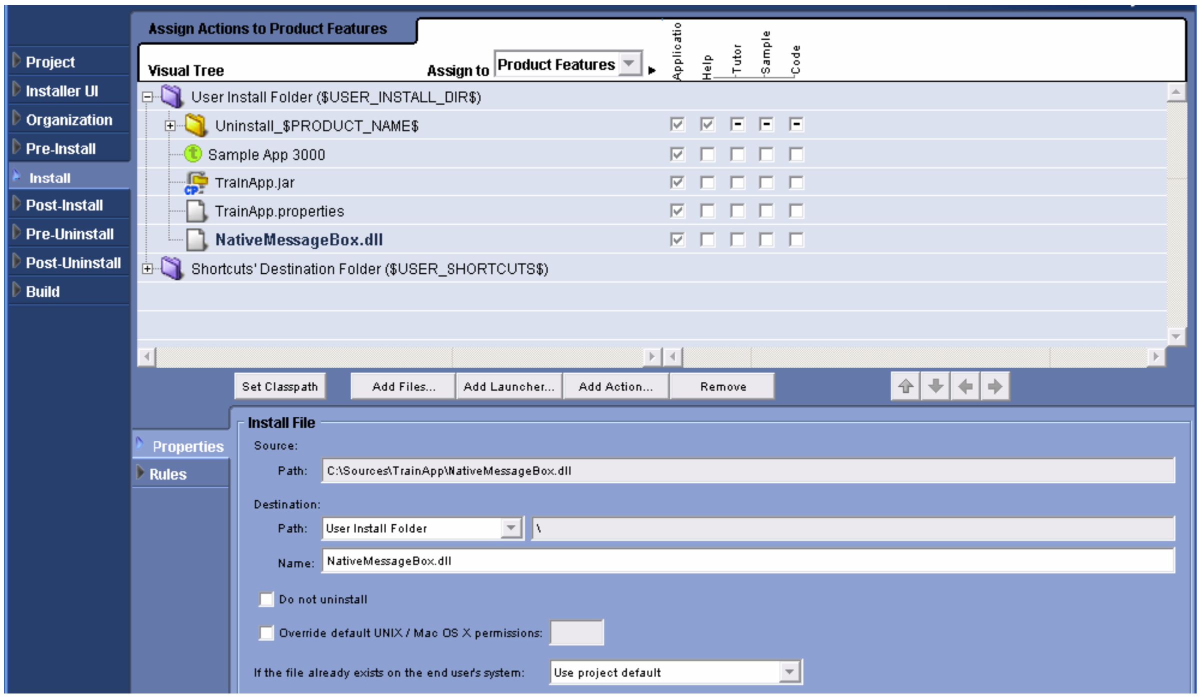
You can use the **Label Settings** tab of the panel's customizer to specify the text that appears in the left-hand list of steps on the panel at run time.

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.



User Install Folder ($USER_INSTALL_DIR$)
Uninstall_$PRODUCT_NAME$
Sample App 3000
TrainApp.jar
TrainApp.properties
NativeMessageBox.dll
Shortcuts' Destination Folder ($USER_SHORTCUTS$)

Install File
Source:
Path: C:\Sources\TrainApp\NativeMessageBox.dll
Destination:
Path: User Install Folder \
Name: NativeMessageBox.dll
Do not uninstall
Override default UNIX / Mac OS X permissions:
If the file already exists on the end user's system: Use project default

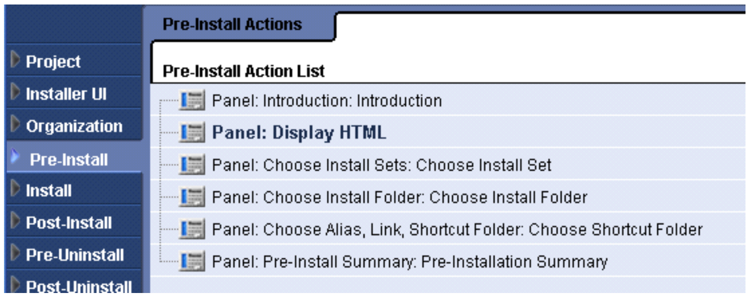
Figure 7-11: Setting System Environment Variables

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.

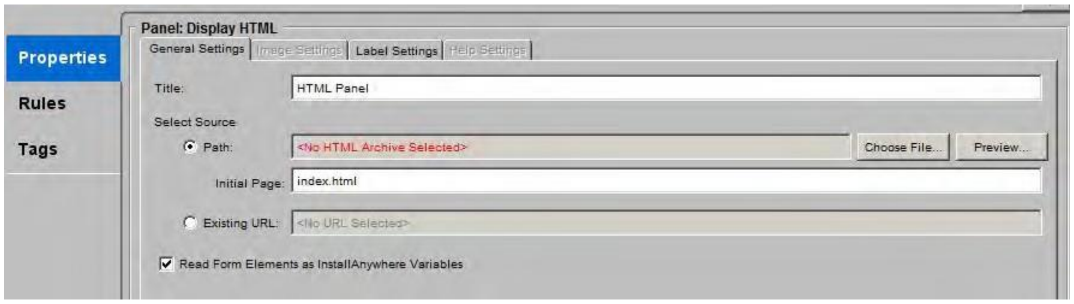
For example, you add the panel to the **Pre-Install** task the same way you add other panels, using the **Add Action** button, if necessary using the up-arrow and down-arrow buttons to schedule the action relative to other actions.



Pre-Install Action List
Panel: Introduction: Introduction
Panel: Display HTML
Panel: Choose Install Sets: Choose Install Set
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

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.



Title: HTML Panel

Select Source

Path: <No HTML Archive Selected>

Initial Page: index.html

Existing URL: <No URL Selected>

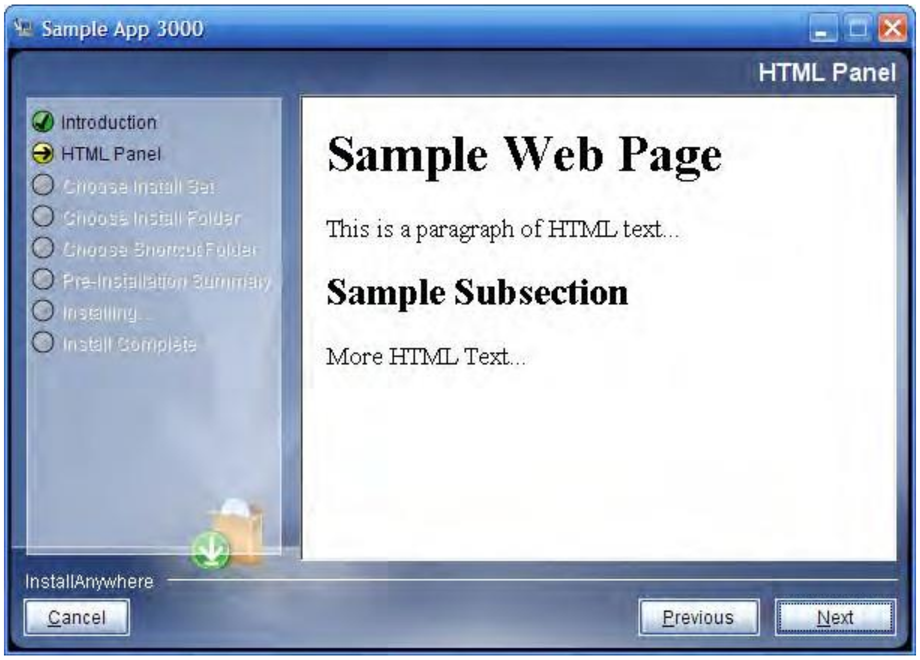
☑ Read Form Elements as InstallAnywhere Variables

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.



Sample Web Page

This is a paragraph of HTML text...

Sample Subsection

More HTML Text...

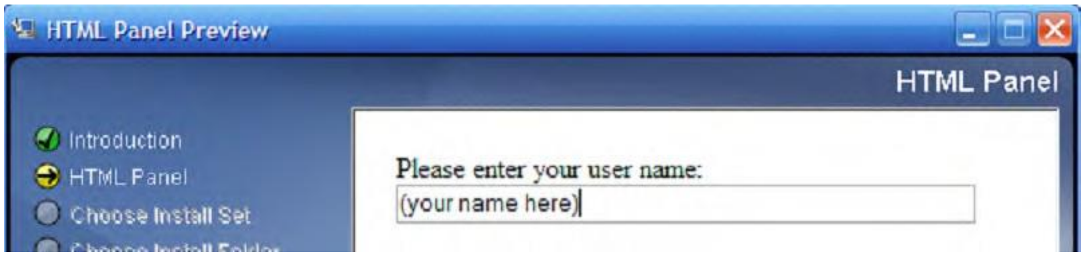
Figure 7-14: Example of a Display HTML Panel

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:



Please enter your user name:

(your name here)

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 is accessible using the expression `$username$`.

Application Servers and Database Servers

InstallAnywhere 2008 Enterprise Edition introduced 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.

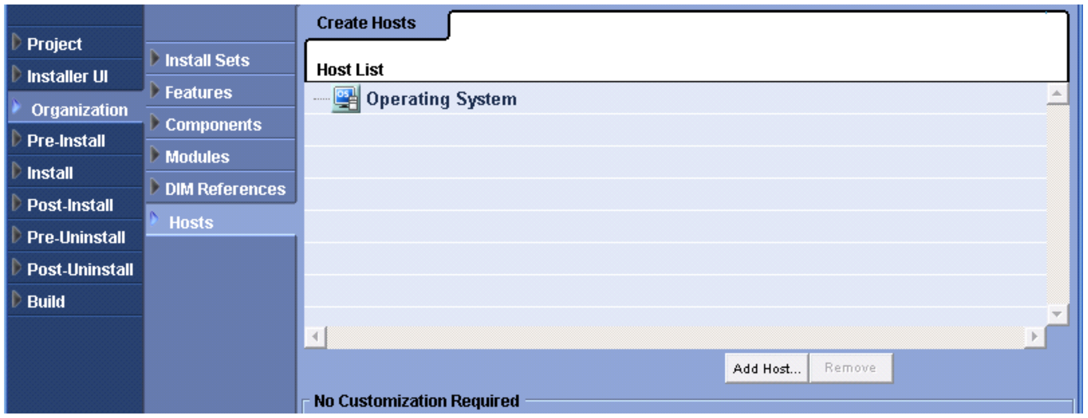


Figure 7-16: Adding Hosts

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.

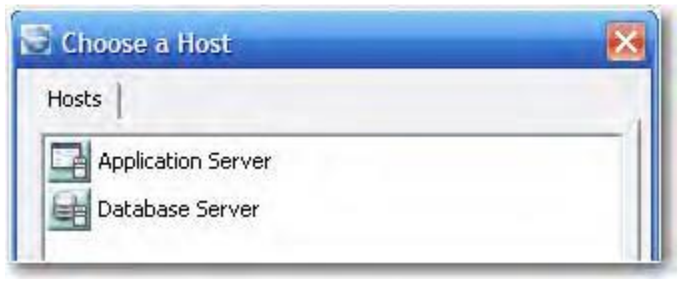
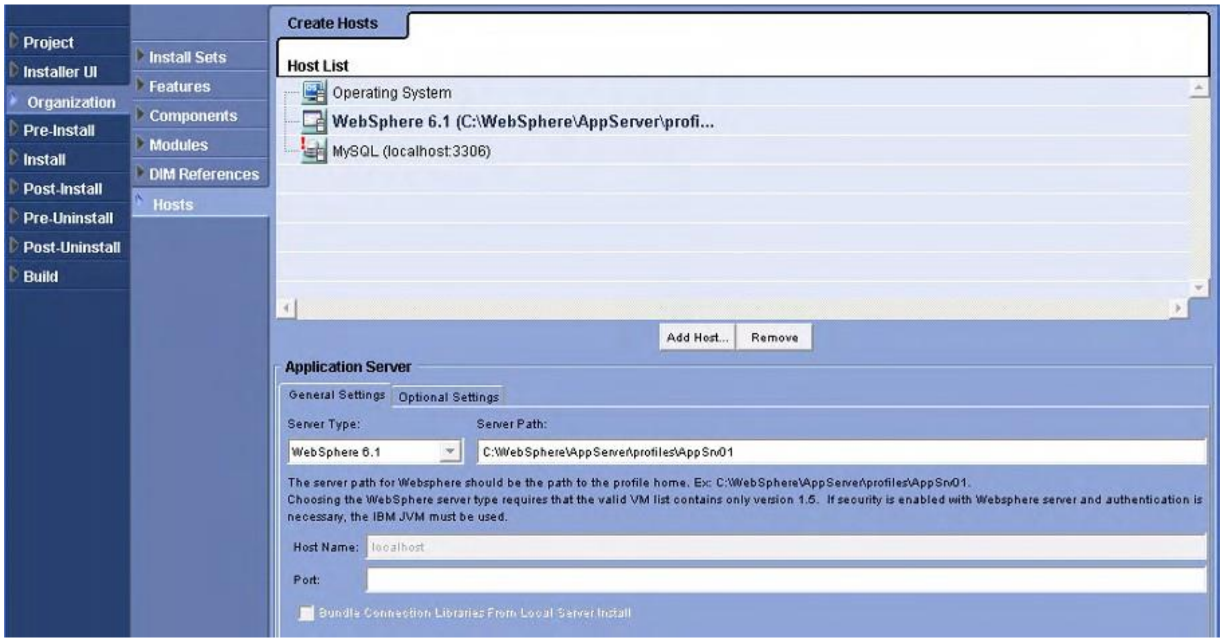


Figure 7-17: Choosing a New Host

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.



Server Type: WebSphere 6.1
Server Path: C:\WebSphere\AppServer\profiles\AppSrv01

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

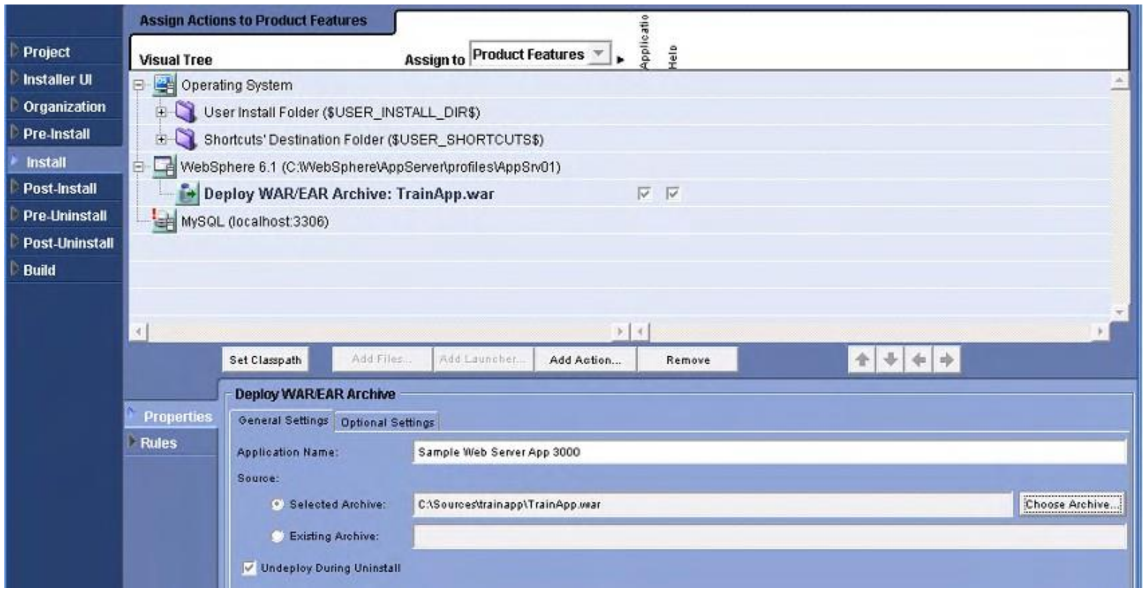
Host Name: localhost
Port:
Bundle Connection Libraries From Local Server Install

Figure 7-18: Configuring an Application Server Host

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 Flexera Software 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.



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

Deploy WAR/EAR Archive
Application Name: Sample Web Server App 3000
Source:
Selected Archive: C:\Sources\trainapp\TrainApp.war
Existing Archive:
Undeploy During Uninstall ✓

Figure 7-19: 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
Comment	Sets the name of the action in the visual tree.
Do not uninstall	Tells an action to not attempt to undo the results of the action at uninstall time.
If file already exists on end user's system	Overrides the default behavior for how to resolve conflicts between installed files and pre-existing files.
In Classpath	Puts the item on the classpath for all LaunchAnywhere executables installed.

Table 7-1 • Common Properties

Property	Description
Installed File/Existing File	Determines whether the file is being installed, or already exists on the end user's system.
Override default Unix/Mac OS X permissions	Sets the file permissions to a specific value for this action.
Path	Shows the path where the action will be installed.
Show Indeterminate Dialog	Brings up an indeterminate progress bar to show progress to the end user while a external process is executing.
Source	Shows the path where the item currently exists on the developer's system (displays the source path if source paths are being used).
Store process exit code in	Sets the value of the InstallAnywhere Variable to the process exit code.
Store process stderr in	Sets the value of the InstallAnywhere Variable to the process standard error.
Store process stdout in	Sets the value of the InstallAnywhere Variable to the process standard out.
Suspend installation until process completes	Pauses the installer until the launched process completes.

LaunchAnywhere Executable

A native executable used to launch a Java application, LaunchAnywhere is InstallAnywhere'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.

A properties file with the .lax extension is responsible for 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. The 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

Table 7-2 • Answers to Quick Quiz

Question	Answer
1	c
2	c

8

Customizing the Uninstaller

InstallAnywhere automatically creates an uninstaller for the project. The Uninstaller, much like the Installer, is a collection of panels, consoles, and actions. The standard Uninstaller uninstalls the application by executing each action's uninstallation procedure.

However, in some situations, you may want additional flexibility and have more control over how the uninstallation is performed. Therefore, you may want to use the **Uninstall** task in the Advanced Designer to customize the Uninstaller by adding, removing or changing some of the uninstall actions. For example, you may want to disable the uninstallation of an entire set of resources, rename files, copy and move files, display additional dialog messages, or execute some custom code at uninstall time.

Information about Uninstaller customization is presented in the following topics:

- About Uninstaller Customization
- Adding Uninstall Categories and Actions to the Uninstall Task
- Preventing an Uninstall Category From Being Uninstalled

About Uninstaller Customization

This section lists some benefits of customizing the Uninstaller and provides an overview of the **Uninstall** task on the Advanced Designer.

- Benefits of Customizing the Uninstaller
- Overview of the Uninstall Task

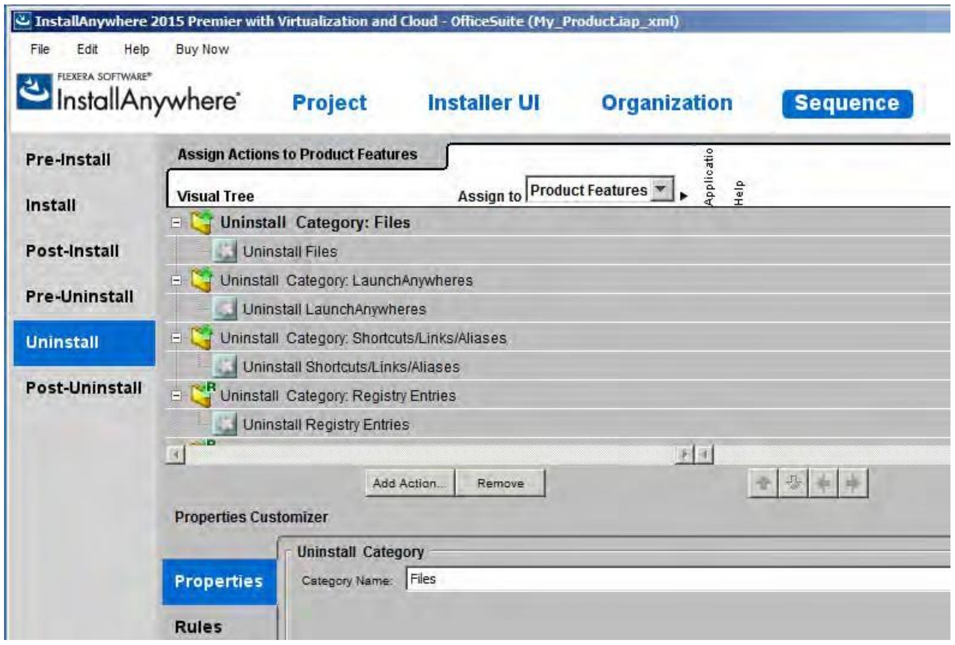
Benefits of Customizing the Uninstaller

By customizing the **Uninstall** phase, you are able to configure how uninstallation will occur by adding, changing or even removing steps of it. That level of flexibility enables you to address scenarios such as the following:

- **Prevent uninstallation of some resources**—You can choose to prevent the uninstallation of specific resources. For example, you could choose to keep some registry entries unchanged without uninstalling them for future reference.
- **Customize the uninstall to provide some actions/custom code before a category is uninstalled**—For example, before having the Uninstaller undeploy a WAR file from an application server, you can choose to run a script to stop the application server.
- **Reorder uninstallation steps**—You can choose to reorder the uninstallation steps. This enables you to run scripts before any resource has been uninstalled. For example, you could choose to run scripts to clean up external resources that were installed in a location other than the installation directory—such as resources that were generated during installation—prior to beginning the main uninstallation steps.
- **Uninstall Merge Modules**—You can choose to uninstall Merge Modules from the base installation during the **Uninstall** phase, rather than adding an **Execute Uninstaller** action to end of the **Pre-Uninstall** phase to perform this task.

Overview of the Uninstall Task

You can customize the uninstaller by using the **Uninstall** task in the Advanced Designer. On the **Uninstall** task, actions are grouped into **Uninstall Categories**.



Uninstall Category: Files
Uninstall Files
Uninstall Category: LaunchAnywheres
Uninstall LaunchAnywheres
Uninstall Category: Shortcuts/Links/Aliases
Uninstall Shortcuts/Links/Aliases
Uninstall Category: Registry Entries
Uninstall Registry Entries

Uninstall Category
Category Name: Files

Figure 8-1: Uninstall Categories and Actions on the Uninstall Task

By default, the following Uninstall Categories are created:

- Files
- LaunchAnywheres
- Shortcuts/Links/Aliases
- Registry Entries
- Native Packages
- Folders
- Others Category

By default, each of these Uninstall Categories contains one Uninstall action that would run the uninstallation of the category. You can reorder items in this list, add additional **General** actions, remove items from the list, and re-add deleted items. To add custom uninstallation behavior, you can add additional custom Uninstall Categories.

Adding Uninstall Categories and Actions to the Uninstall Task

To customize the **Uninstall** phase, you can add **Uninstall** and **General** actions and group those actions into Uninstall Categories.

- About Adding Uninstall Categories and Actions
- Adding Categories and Actions
- Available Uninstall Actions
- Assigning Actions to Product Features and Components

About Adding Uninstall Categories and Actions

Regarding adding Uninstall Categories and actions, note the following:

- **Only Uninstall actions and General actions can be added**—Only **Uninstall** and **General** actions can be added to the Uninstall Categories because they are not bound to Magic Folders and they do not change the product registry.
- **Each Uninstall Category and Uninstall action can be added only once**—Each of the Uninstall Categories and Uninstall actions can be added to the **Uninstall** Visual Tree only once. If they are added more than one time, the second instance is ignored.
- **Rules can be assigned**—You can assign rules to Uninstall Categories and actions in those categories to prevent them from being executed/uninstalled.
- **Uninstall Categories and Uninstall actions cannot be assigned to Components or Features**—While you are permitted to assign **General** actions in the **Uninstall** task to Components or Features, you are not permitted to assign **Uninstall Categories** or **Uninstall** actions to Components or Features. This is because **Uninstall Categories** and **Execute Uninstall Actions** are special actions which are not Feature/Component-specific. These are shared across all Components and Features.

Adding Categories and Actions

To add an Uninstall Category or action to the Uninstall task, perform the following steps.

Task ***To add an Uninstall Category or action to the Uninstall task:***

1. Open the **Uninstall** task in the Advanced Designer. The Visual Tree of the Uninstall task lists Uninstall Categories and actions previously added.

   Note • *By default, all of the Uninstall actions are listed in the Visual Tree of the **Uninstall** task. If you delete one of these Uninstall actions and want to re-add it later, you can select it from the **Uninstall** tab of the **Choose an Action** dialog box.*

2. To add an Uninstall Category, perform the following steps:

   a. Click **Add Action**. The **Choose an Action** dialog box opens, listing two tabs: **Uninstall** and **General**.

   b. On the **Uninstall** tab, select **Uninstall Category** and click **Add**. The new Uninstall Category is added to the list.

   c. In the **Properties** tab of the **Uninstall Category** customizer, enter a name for the new category in the **Category Name** text box.

   d. With the new Uninstall Category selected, using the arrow keys to move the category up or down in the Visual Tree.

3. To add an action, perform the following steps:

   a. Select the Uninstall Category that you want to add an action to.

   b. Click **Add Action**. The **Choose an Action** dialog box opens, listing two tabs: **Uninstall** and **General**.

   c. Select an action on the **Uninstall** or **General** tab and click **Add**. The new action is added to the Visual Tree.

      Note • *Each of the **Uninstall** actions can be added to the Uninstall Visual Tree only once. If they are added more than one time, the second instance is ignored.*

   d. Select an action on the **Uninstall** or **General** tab and click **Add**. The new action is added to the Visual Tree.

   e. If you added a **General** action, you can assign it to a Product Feature or Component by selecting the appropriate check boxes. See Assigning Actions to Product Features and Components.

4. Save and build the project.

Available Uninstall Actions

The following Uninstall actions are available in the **Uninstall** task.

Table 8-1 • Uninstall Actions

Action	Description
Uninstall Category	Groups sets of uninstall actions.
Uninstall Files	Uninstalls all of the files installed or created by the installer during installation.
Uninstall Folders	Uninstalls all of the folders installed or created by the installer during installation.
Uninstall LaunchAnywhere	Uninstalls all of the LaunchAnywheres installed or created by the installer during installation.
Uninstall Native Packages	Uninstalls all of the native packages installed or created by the installer during installation.
Uninstall Registry Entries	Uninstalls all of the registry entries installed or created by the installer during installation.
Uninstall Shortcuts/ Links/Aliases	Uninstalls all of the shortcuts, links, and aliases installed or created by the installer during installation.
Uninstall Software Databases	Uninstalls all of the software databases installed or created by the installer during installation.

Assigning Actions to Product Features and Components

To assign actions on the **Uninstall** task to product features or components, use the **Assign to** option.

Task ***To assign an action to a product feature or component:***

1. Open a project in the Advanced Designer and open the **Uninstall** task.
2. Select an action in the Visual Tree.
3. From the **Assign to** list above the Visual Tree, select **Product Features** or **Components**. All of the project features or components are listed.
4. Select the product features or components that you want to assign the selected action to.
5. Save the project.

Preventing an Uninstall Category From Being Uninstalled

When an end user launches the Uninstaller, it displays a list of the Uninstall Categories and starts uninstalling all of them one by one. If you want to prevent an entire Uninstall Category of actions from being uninstalled during uninstallation, you can remove an action from that Uninstall Category so that that category is skipped during uninstallation, or you can assign a rule so that the Uninstall Category is not executed.

Removing the Uninstall Action from an Uninstall Category

To remove an Uninstall Action from an Uninstall Category, to prevent all of the actions in that category from being uninstalled during uninstallation, perform the following steps:

Task ***To remove an Uninstall Action from an Uninstall Category:***

1. Open the **Uninstall** task in the Advanced Designer. The Visual Tree of the **Uninstall** task lists Uninstall Categories and the **Uninstall** and **General** actions previously added.
2. Locate and select the **Uninstall** action in the Uninstall Category that you want to prevent from being uninstalled and click **Remove**. The **Uninstall** action is no longer listed in the Visual Tree.
3. Save and build the project.

Assigning a Rule

To assign a rule to an Uninstall Category, to prevent all of the actions in that category from being uninstalled during uninstallation, perform the following steps:

Task ***To assign a Rule to an Uninstall Category:***

1. Open the **Uninstall** task in the Advanced Designer. The Visual Tree of the **Uninstall** task lists Uninstall Categories and the **Uninstall** and **General** actions previously added.
2. Locate and select the Uninstall Category that you want to prevent from being uninstalled.
3. Open the **Rules** tab and click **Add Rule**. The **Choose a rule** dialog box opens.
4. Select a rule and click **Add**. The rule is now listed in the **Rules List** and its customizer is now displayed.
5. Specify the rule conditions that must be met in order for this Uninstall Category to be executed.
6. Save the project.

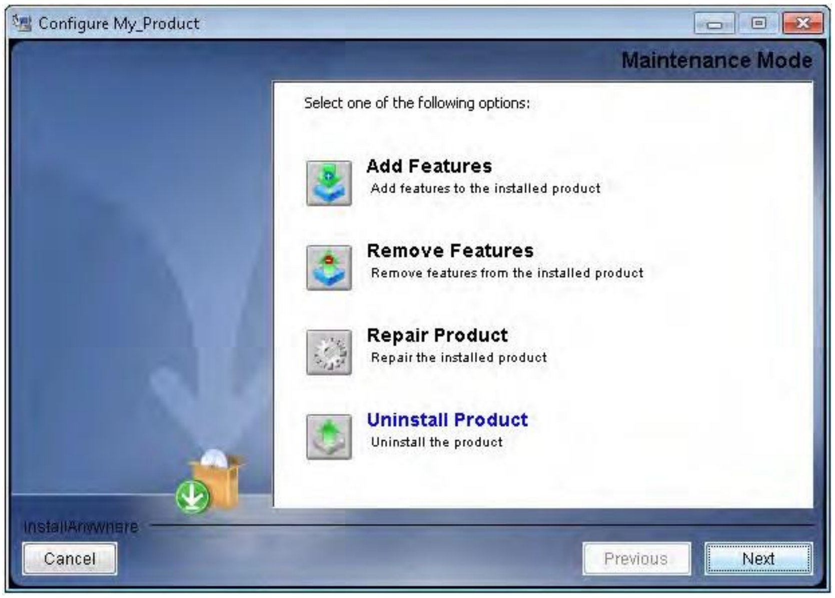
9

Implementing Maintenance Mode

You can choose to implement Maintenance Mode in an installer so that end users are able to add or remove features to previously installed products as well as repair broken installations. The end user will be able to launch Maintenance Mode by:

- Executing the **Change Installation** launcher.
- Selecting **Add or Remove Programs** from the Windows Control Panel (Microsoft Windows OS only).

When an end user launches Maintenance Mode, they will be prompted to select from the listed options, which may include **Add Features**, **Remove Features**, **Repair Product**, and **Uninstall Product**.



Select one of the following options:

Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

Figure 9-1: Maintenance Mode Panel

Information about implementing Maintenance Mode is presented in the following sections:

Table 9-1 • Implementing Maintenance Mode

Section	Description
Enabling and Configuring Maintenance Mode	Explains how to enable Maintenance Mode support in an InstallAnywhere project, specify the options to include, and configure how Maintenance Mode will be implemented.
Maintenance Mode End User Experience	Provides examples of the different end user experiences for the various Maintenance Mode options.
About the Uninstaller / Maintenance Mode Launcher	Explains the methods the end user can use to launch Maintenance Mode and the Uninstaller.

Enabling and Configuring Maintenance Mode

To include Maintenance Mode support in your installation project, you need to first enable the Maintenance Mode options you want to include, and then customize those options for the product being installed.

Table 9-2 • Steps to Enable Maintenance Mode

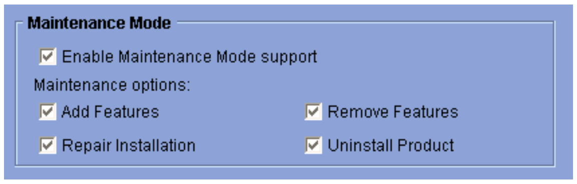
Step	Description
Enabling Maintenance Mode	Explains how to enable Maintenance Mode support in a project and the purpose of each of the Maintenance Mode options.
About Maintenance Mode Action Groups	Describes the panels that are automatically added to your project when you select Maintenance Mode options, and explains how to customize those panels.
About Check Running Mode Rules	Explains the purpose of the **Check Running Mode** rules that are automatically added to your installation project when Maintenance Mode is enabled, and how to use them to customize the events that occur when an end user launches Maintenance Mode.
Specifying Instance Management Options	Explains how to configure an installation project to specify whether multiple instances of a product can be installed on the same machine.

Enabling Maintenance Mode

You can enable Maintenance Mode support and identify an installer's supported Maintenance Mode options on the **Project > Advanced** subtask of the Advanced Designer.

Task ***To enable Maintenance Mode support and identify supported options:***

1. Open an installation project in the Advanced Designer.
2. On the **Project** task, click **Advanced**. The **Project > Advanced** subtask opens.
3. Under **Maintenance Mode**, select **Enable Maintenance Mode support**. The Maintenance Mode options are then enabled.


Maintenance Mode
☑ Enable Maintenance Mode support
Maintenance options:
☑ Add Features ☑ Remove Features
☑ Repair Installation ☑ Uninstall Product

Important • *After selecting* ***Enable Maintenance Mode support****, you are required to select at least one of the four* ***Maintenance options****. Failure to do so will result in build failure.*

Note • *For all new or migrated projects, the* ***Enable Maintenance Mode support*** *option will be disabled by default.*

4. Select one (or more) of the following options:

Option	Description
Add Features	Select this option to enable the end user to use Maintenance Mode to install previously uninstalled product features. An end user can use **Add Features** to: ● **Install** product features that they did not choose to install during the initial installation. ● **Re-install** product features that had originally been installed but were removed using the **Remove Features** option of Maintenance Mode. When the end user launches Maintenance Mode and chooses **Add Features**, the installer will advance to the Pre-Install phase of the installation and then proceed as a regular installation. ***Note*** • *The set of actions executed when the end user selects **Add Features** do not include the whole set of actions in Pre-Install, but a smaller subset for which the Check Running Mode Rule is set to **Pre-Installation**. See About Check Running Mode Rules.* ***Note*** • *For more information, see Adding a Feature User Experience.*
Remove Features	Select this option to enable the end user to use Maintenance Mode to remove previously installed product features. ***Tip*** • *The InstallAnywhere Uninstaller also provides you with the ability to selectively remove individual features.* When the end user launches Maintenance Mode and chooses **Remove Features**, the installer will advance to the Pre-Uninstall phase of the uninstallation and then proceed with the uninstallation of the selected features. ***Note*** • *The set of actions executed when the end user selects **Remove Features** do not include the whole set of actions in Pre-Uninstall, but a smaller subset for which the Check Running Mode Rule is set to **Remove Features**. See About Check Running Mode Rules.* ***Note*** • *For more information, see Removing a Feature User Experience.*

Option	Description
Repair Installation	Select this option to enable the end user to use Maintenance Mode to repair a previously installed product. When the end user launches Maintenance Mode and chooses **Repair Product**, all of those actions in the Pre-Install phase with a Check Running Mode Rule set to **Repair Installation** will be executed (not just those actions that exist in the **Repair Installation** Action Group). See About Check Running Mode Rules. ***Note*** • *For more information, see Repairing a Product User Experience.*
Uninstall Product	Select this option to enable the end user to use Maintenance Mode to uninstall a previously installed product. ***Tip*** • *The InstallAnywhere Uninstaller also provides you with the ability to uninstall a product.* When the end user launches Maintenance Mode and chooses **Uninstall Product**, the installer will advance to the Pre-Uninstall phase of the uninstallation and proceed with the uninstallation of the product. ***Note*** • *For more information, see Uninstalling a Product User Experience.*

Tip • *You can assign a **Check Running Mode** rule to associate elements of your installation project with the Add/Remove/Repair/Uninstall Maintenance Mode options. For example, you may want to add a Check Running Mode rule to an action that should not be executed during the Repair phase. For more information, see About Check Running Mode Rules.*

5. Save the InstallAnywhere project.

About Maintenance Mode Action Groups

When you enable Maintenance Mode support, the Advanced Designer adds new Action Groups to the default **Pre-Install** and **Pre-Uninstall** tasks. Depending on the Maintenance Mode options you selected (**Add Features**, **Remove Features**, **Repair Installation**, **Uninstall Product**), separate Action Groups will be created to run when Maintenance Mode is launched.

- Pre-Install Task
- Pre-Uninstall Task

Important • *This section details the Action Groups and panels that are automatically added to your project when you enable Maintenance Mode support. However, you may want to customize these panels.*

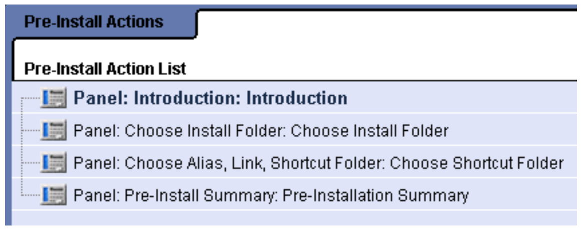
Pre-Install Task

When Maintenance Mode is enabled, the following Action Groups are created in the **Pre-Install** task:

- Pre-Installation Action Group
- Add Features Action Group
- Repair Installation Action Group

Pre-Installation Action Group

When you create a new project, and Maintenance Mode is not selected, the **Pre-Install** Action List consists of several panels:


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

Note • *In the case of InstallAnywhere projects created using previous versions which are migrated to InstallAnywhere, all panels and actions present in the Pre-Install phase would, by default, be grouped under this new Pre-Installation Action Group. The Check Running Mode Rule would be set to **Pre-Installation** to enable the existing panels to run as-is.*

As soon as you enable Maintenance Mode, those panels are grouped into a new Action Group named **Pre-Installation**, and a Check Running Mode rule with a value of **Pre-Installation** is assigned to that group.

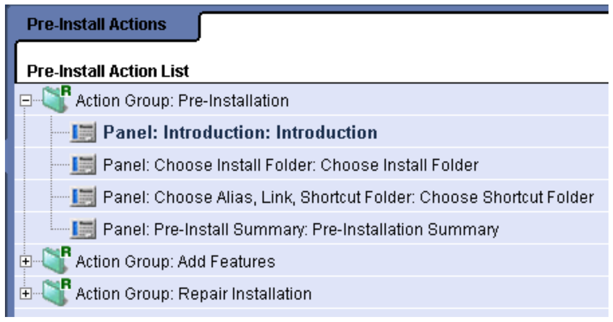

Pre-Install Actions
Pre-Install Action List
Action Group: Pre-Installation
Panel: Introduction: Introduction
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary
Action Group: Add Features
Action Group: Repair Installation

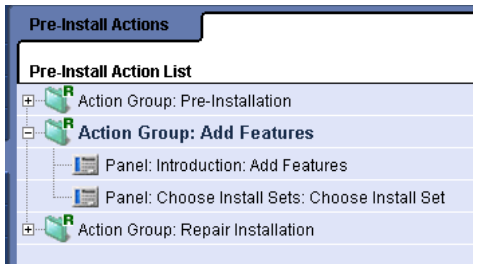
Figure 9-2: Pre-Installation Action Group on the Pre-Install Action List

These default Action Groups are provided as a convenience; you are not required to group actions into Action Groups in order to assign a Check Running Mode rule to an action or panel. You are permitted to assign Check Running Mode Rules directly to individual actions or panels. You can choose to keep these default Action Groups or delete them, but remember to add the appropriate Check Running Mode Rule to those actions and panels that you want to be executed during Maintenance Mode.

***Note** • Because a Check Running Mode rule with the value of **Pre-Installation** is assigned to the Pre-Installation Action Group, these panels will only be executed during installation, not during the Add Features or Repair Installation phases of Maintenance Mode. For more information, see About Check Running Mode Rules.*

Add Features Action Group

If you select the **Add Features** Maintenance Mode option, the **Add Features** Action Group is added to the **Pre-Install** Action List, and a Check Running Mode rule with a value of **Add Features** is assigned to that group.



Pre-Install Actions

Pre-Install Action List
- Action Group: Pre-Installation
- Action Group: Add Features
  - Panel: Introduction: Add Features
  - Panel: Choose Install Sets: Choose Install Set
- Action Group: Repair Installation

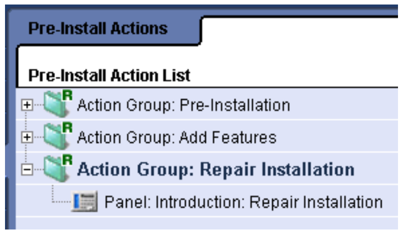
Figure 9-3: Add Features Action Group on the Pre-Install Action List

By default, the **Add Features** Action Group consists of an **Introduction** and a **Choose Install Sets** panel.

***Note** • Because a Check Running Mode rule with the value of **Add Features** is assigned to the Add Features Action Group, these panels will only be executed during the Add Features phase of Maintenance Mode. For more information, see About Check Running Mode Rules.*

Repair Installation Action Group

If you select the **Repair Installation** Maintenance Mode option, the **Repair Installation** Action Group is added to the **Pre-Install** Action List, and a Check Running Mode rule with a value of **Repair Installation** is assigned to that group.



Pre-Install Actions

Pre-Install Action List
- Action Group: Pre-Installation
- Action Group: Add Features
- Action Group: Repair Installation
  - Panel: Introduction: Repair Installation

Figure 9-4: Repair Installation Action Group on the Pre-Install Action List

By default, the **Repair Installation** Action Group consists of an **Introduction** panel.

By default, when an end user chooses the Repair Installation option, the product would be reinstalled. If, at the same time, you would like to perform some other kind of repair operations, such as running a script that repairs a database, you may want to add an additional action to the **Repair Installation** Action Group.

***Note** • Because a Check Running Mode rule with the value of **Repair Installation** is assigned to the Repair Installation Action Group, these panels will only be executed during the Repair Installation phase of Maintenance Mode. For more information, see About Check Running Mode Rules.*

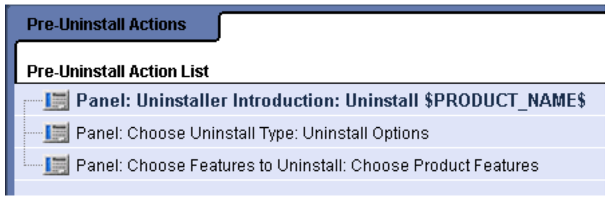
Pre-Uninstall Task

When Maintenance Mode is enabled, the following Action Groups are created in the **Pre-Uninstall** task:

- Pre-Uninstallation Action Group
- Remove Features Action Group

Pre-Uninstallation Action Group

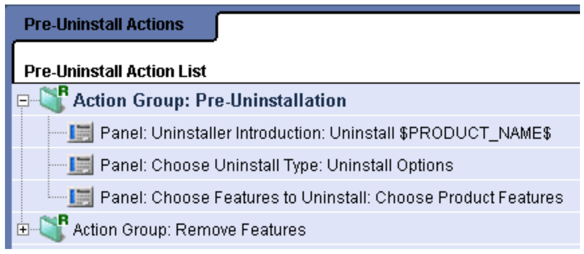
When you create a new project, and Maintenance Mode is not selected, the **Pre-Uninstall** task consists of several panels:



Pre-Uninstall Action List
- Panel: Uninstaller Introduction: Uninstall $PRODUCT_NAME$
- Panel: Choose Uninstall Type: Uninstall Options
- Panel: Choose Features to Uninstall: Choose Product Features

Figure 9-5: Default Panels on the Pre-Uninstall Action List

As soon as you enable Maintenance Mode, those panels are grouped into a new Action Group named **Pre-Uninstallation**, and a Check Running Mode rule with a value of **Pre-Uninstallation** is assigned to that group.



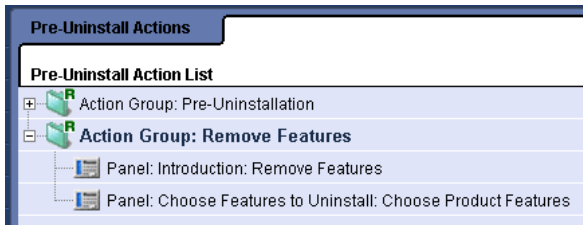
Pre-Uninstall Action List
- Action Group: Pre-Uninstallation
  - Panel: Uninstaller Introduction: Uninstall $PRODUCT_NAME$
  - Panel: Choose Uninstall Type: Uninstall Options
  - Panel: Choose Features to Uninstall: Choose Product Features
- Action Group: Remove Features

Figure 9-6: Pre-Uninstallation Action Group on the Pre-Uninstall Action List

***Note** • Because a Check Running Mode rule with the value of **Pre-Uninstallation** is assigned to the Pre-Uninstallation Action Group, these panels will only be executed during uninstallation, not during the **Remove Features** phase of Maintenance Mode. For more information, see About Check Running Mode Rules.*

Remove Features Action Group

If you select the **Remove Features** Maintenance Mode option, the **Remove Features** Action Group is added to the **Pre-Uninstall Action List**, and a Check Running Mode rule with a value of **Remove Features** is assigned to that group.


Pre-Uninstall Actions

Pre-Uninstall Action List
- Action Group: Pre-Uninstallation
- Action Group: Remove Features
  - Panel: Introduction: Remove Features
  - Panel: Choose Features to Uninstall: Choose Product Features

Figure 9-7: Remove Features Action Group on Pre-Uninstall Action List

By default, the **Remove Features** Action Group consists of an **Introduction** and a **Choose Features to Uninstall** panel.

Note • *Because a Check Running Mode rule with the value of **Remove Features** is assigned to the Remove Features Action Group, these panels will only be executed during the **Remove Features** phase of Maintenance Mode. For more information, see About Check Running Mode Rules.*

About Check Running Mode Rules

As described in About Maintenance Mode Action Groups, when you enable Maintenance Mode, **Check Running Mode** rules are automatically added to the Action Groups that are created in the **Pre-Install** and **Pre-Uninstall** tasks. However, you can also manually apply a Check Running Mode rule to any Action Group, Action, or Panel in your installation project to specify whether or not that element should be executed when Maintenance Mode is run. This enables you to customize the events that occur when an end user launches Maintenance Mode.

Important • *When a Check Running Mode rule is assigned to an Action Group, it is applied to all panels and actions in that Action Group.*

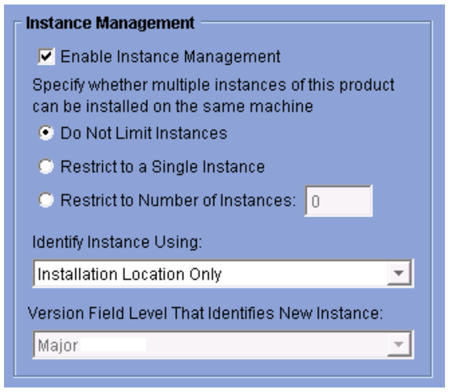
Specifying Instance Management Options

InstallAnywhere's Maintenance Mode support includes an **Instance Management** feature that enables you to specify whether multiple instances of a product can be installed on the same machine.

If an installation includes both Maintenance Mode support and support for multiple instances of the product on the same machine, when an end user launches Maintenance Mode, they are able to specify which instance of the product they want to perform maintenance on.

Note • *The instance count is only as secure as the security of the InstallAnywhere product registry file.*

The **Instance Management** options are on the **Project > Advanced** subtask of the Advanced Designer.


Instance Management

☑ Enable Instance Management

Specify whether multiple instances of this product can be installed on the same machine

● Do Not Limit Instances
○ Restrict to a Single Instance
○ Restrict to Number of Instances: 0

Identify Instance Using:
Installation Location Only

Version Field Level That Identifies New Instance:
Major

Figure 9-8: Instance Management Options on Project > Advanced Subtask

Important • *The **Instance Management** options are only enabled if the **Enable Maintenance Mode support** option is selected. However, you can provide Maintenance Mode support without enabling Instance Management support.*

To specify Instance Management options, perform the following steps:

Task ***Specifying Instance Management options:***

1. Open an installation project in the Advanced Designer.
2. On the **Project** task, click **Advanced**. The **Project > Advanced** subtask opens.
3. Under **Maintenance Mode**, select **Enable Maintenance Mode Support**. The Maintenance Mode and **Enable Instance Management** options are then enabled.

   Note • *For all new or migrated projects, the **Enable Maintenance Mode Support** option will be disabled by default.*

4. Set your desired Maintenance Mode options, as described in Enabling Maintenance Mode.
5. Under **Instance Management**, select **Enable Instance Management**. The Instance Management options are enabled.

6. Select one of the following options:

Option	Description
Do not limit instances	Select this option to enable end users to install an unlimited number of instances of this product on a machine. End users may want to install multiple instances on the same machine if they want to configure each instance differently (each with different features, configuration settings, environments) and want to launch each instance independently (such as launching multiple instances on WebSphere/Tomcat using different JVM versions). If an installer was created with the **Do not limit instances** option selected, the following occurs: ● When an end user attempts to install a second instance of that product, the **Manage Instances** dialog box opens, prompting the user to specify that they want to `Install a New Instance` (rather than **Modify an Existing Instance**). ● When an end user has installed multiple instances of a product on a machine and then launches Maintenance Mode, the **Manage Instances** dialog box opens, prompting the user to select the instance that they want to modify. ***Note** • For more information, see Maintenance Mode User Experience on a Machine With Multiple Installed Instances.*
Restrict to a Single Instance	Select this option to restrict end users to be able to install only one instance of the product on a machine. By selecting the **Restrict to a Single Instance** option, you are preventing the end user from being able to do either of the following: ● Installing additional instances of the product on the same machine. ● Overwriting an existing instance of the product. If an installer was created with the **Restrict to a Single Instance** option selected, when an end user attempts to install a second instance of that product, the **Manage Instances** dialog box opens, prompting the user to modify the existing instance, rather than to install a new instance.
Restrict to Number of Instances	Select this option and enter a number in the text box to restrict end users to be able to install only a specified number of instances of the product on a machine.
Identify Instance Using	Select one of the following options to specify how to identify an instance of an application: ● **Installation Location Only**—Identify an instance by examining the installation location. No filters are applied to the detected instances. ● **Installation Location and Version**—Identify an instance by examining the installation location and the application version of the installed application. When this option is selected, the **Version Field Level That Identifies New Instance** option is enabled, which enables you to identify what level of version change constitutes a new version.

Option	Description
Version Field Level That Identifies New Instance	When you select **Installation Location and Version** from the **Identify Instance Using** option, this option is enabled. You are prompted to specify what level of version change constitutes a new version by selecting one of the following options: ● **Major** ● **Minor** ● **Revision** ● **Subrevision** ***Note*** • *Versions are conventionally represented in the following format: [Major].[Minor].[Revision].[Subrevision], such as:* `1.0.2.1047`. For example, if you select the version level of **Minor** to identify a new version, versions 1.5.4 and 1.5.6 would be considered to be the same version because both the **Major** and **Minor** version are identical (even though the **Revision** version is different), while versions1.5.4 and 1.6.2 would be considered to be two different versions because the **Minor** version does not match. ***Note*** • *Reinstalling an application to the same location as an existing application would result in only one instance, irrespective of the version.* ***Note*** • *If you choose the **Installation Location and Version** option from the **Identify Instance Using** list, the installation does not recognize those instances which are from a more recent version, because an installer of a previous version might not be able to successfully manage an instance of the installer of a more recent version. For example, a Version 2.0.0.0 installer will not consider Version 3.x instances, but will consider all Version 1.x instances and Version 2.0.0.0 instances.*

7. Save your installation project.

Important • *When an installer that has the Instance Management feature enabled is launched, it needs to determine how many instances of the application have already been installed. To do this, the installer queries the product registry to locate the uninstaller executable for that application. By default, the uninstaller executable is named Change* `ProductName Installation`*.exe (in InstallAnywhere) or* `Uninstall ProductName`*.exe (in versions prior to InstallAnywhere 2010). However, if you have chosen to rename the executable file of the uninstaller, that file will not be located and the Instance Management feature will not work properly.*

***Note** • When using the Instance Management feature, note the following:*

- *Instance Management uses the* `global/local zerog` *registries. If these directories are altered, then the Instance Management feature will not work properly.*
- *The **InstallAnywhere Uninstall Component**, which is a standard component in an InstallAnywhere project, should have the **Key File** set to the Uninstaller executable file, which is the default setting. However, if the Key File is modified, then the Instance Management feature may not be in a position to manage instances because it would be unable to find the Uninstaller.*

  *The **Key File** for the **InstallAnywhere Uninstall Component** is set on the **Properties > Component** subtab of the **Organization > Components** subtask.*

About the Uninstaller / Maintenance Mode Launcher

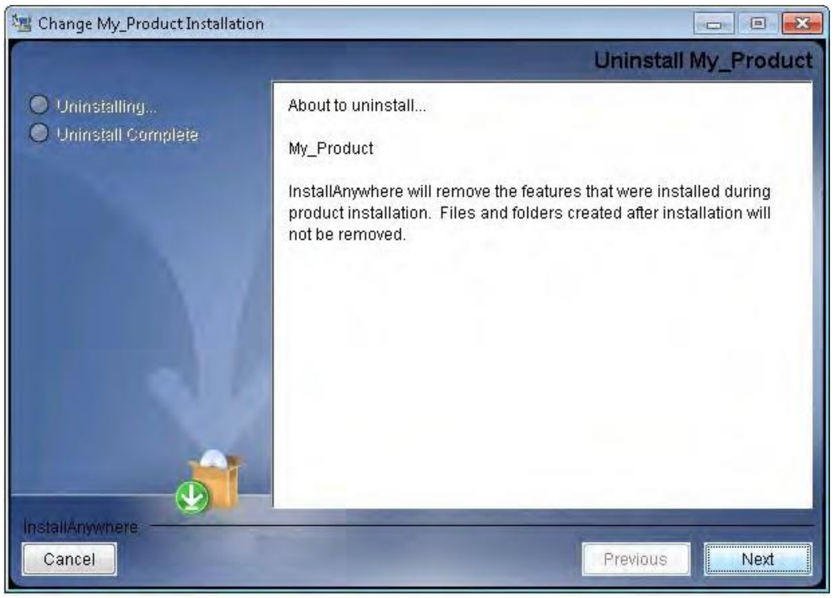
By default, an InstallAnywhere project is configured to create one Uninstaller / Maintenance Mode executable named `Change` *`Product_Name`* `Installation.exe`.

***Note** • You can modify the name of this executable by selecting the **Uninstall** launcher on the **Install** task and editing the **Name** field on the **Properties** tab of the **Create Uninstaller** customizer.*

***Note** • If you are migrating a project from an earlier version of InstallAnywhere prior to InstallAnywhere 2010, the launcher will retain the name that it had in that earlier version. In previous releases, the default name for this launcher was* `Uninstall ProductName.exe`. *You might want to consider changing the name of the uninstall launcher to match the default name assigned to projects created in InstallAnywhere: Change Product_Name Installation.exe.*

The behavior of this launcher varies depending upon whether or not you have implemented Maintenance Mode:

- **Maintenance Mode not enabled**—If you have not enabled Maintenance Mode, opening this launcher launches the standard Uninstall runtime.



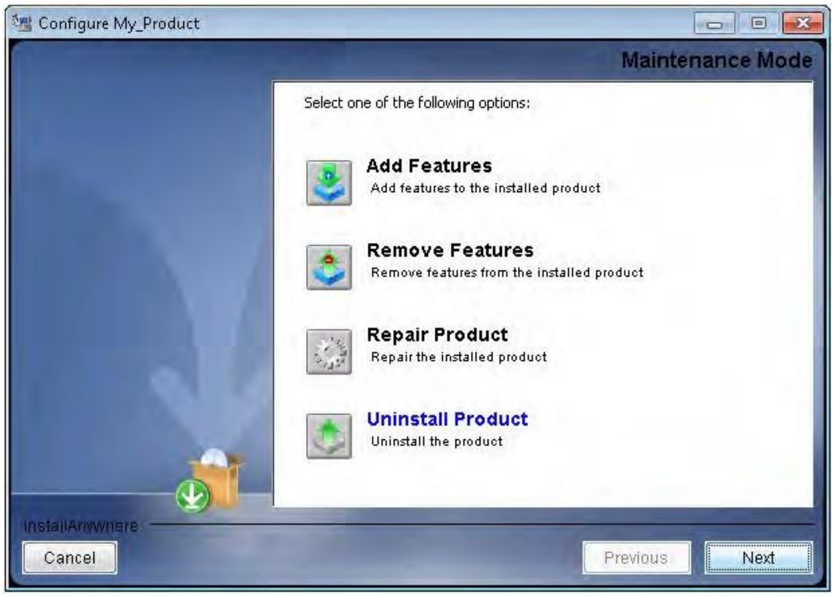
About to uninstall...

My_Product

InstallAnywhere will remove the features that were installed during product installation. Files and folders created after installation will not be removed.

Figure 9-9: Uninstall Panel of the Change Installation Launcher

- **Maintenance Mode enabled**—If you have enabled Maintenance Mode, running this launcher opens the **Configure Maintenance Mode** panel, which prompts the end user to select from the listed options (which may include **Add Features**, **Remove Features**, **Repair Product**, and **Uninstall Product**).



Select one of the following options:

Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

Figure 9-10: Maintenance Mode Panel of the Change Installation Launcher

Maintenance Mode End User Experience

When an end user launches a Change Installation launcher created using the Maintenance Mode option, the user experience varies depending upon the Maintenance Mode options you specified on the **Project > Advanced** subtask. This section describes those user experiences.

- Initial Experience
- Repairing a Product User Experience
- Adding a Feature User Experience
- Removing a Feature User Experience
- Uninstalling a Product User Experience
- Maintenance Mode User Experience on a Machine With Multiple Installed Instances

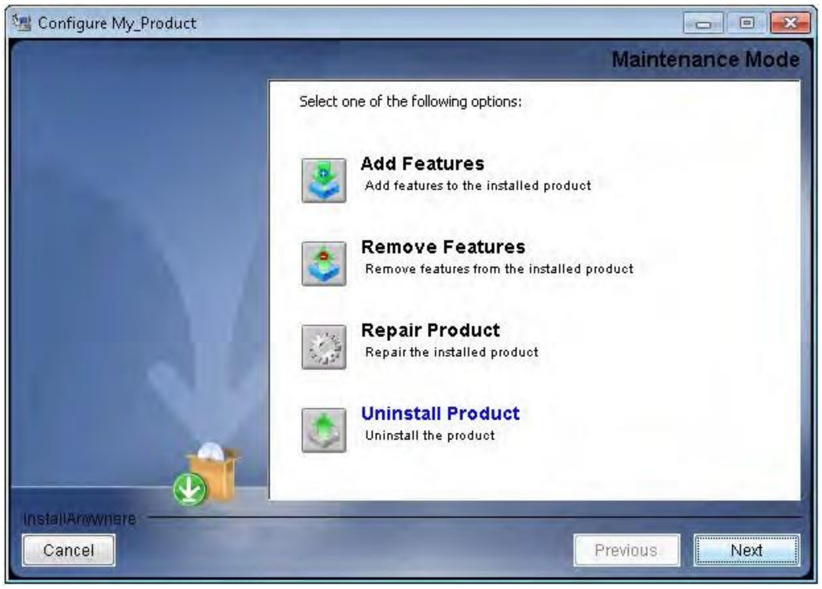
Initial Experience

When an end user launches a Change Installation launcher created using the Maintenance Mode option, their user experience begins as follows:

Task ***To launch the "Change Installation" Maintenance Mode launcher:***

1. Launch the Maintenance Mode by doing one of the following:
    - Opening Change *`Product_Name`* `Installation.exe`.
    - Selecting **Add or Remove Programs** from the Windows Control Panel (Microsoft Windows OS only).

   The **Maintenance Mode** panel opens, and the options that were selected on the **Product > Advanced** subtask of the Advanced Designer are displayed:



Select one of the following options:

Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

***Note** • The Maintenance Mode panel appears only once. After the user selects an option and clicks next, they will be unable to return to this panel by clicking the **Previous** button.*

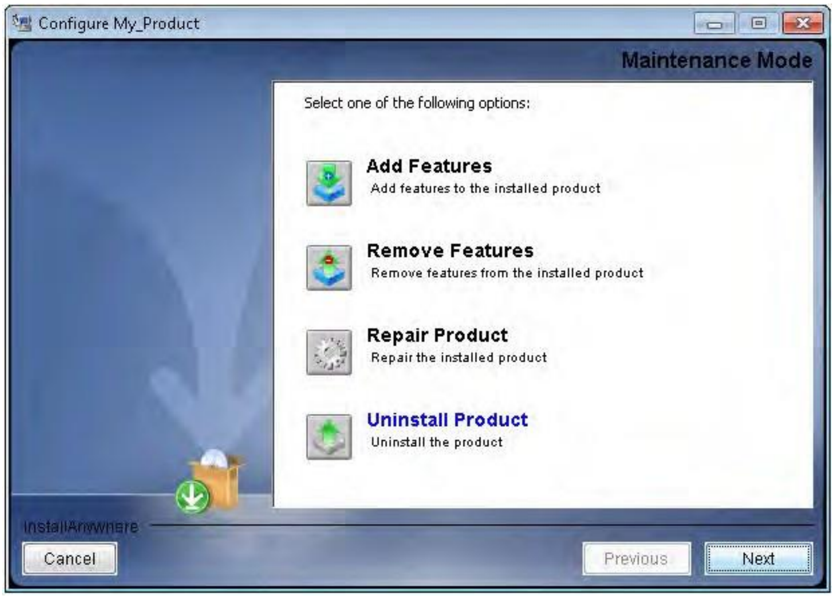
2. Select one of the supported options. The Maintenance Mode runtime executes the selected option, as described in the following topics:
    - Adding a Feature User Experience
    - Removing a Feature User Experience
    - Repairing a Product User Experience
    - Uninstalling a Product User Experience
    - Maintenance Mode User Experience on a Machine With Multiple Installed Instances

Adding a Feature User Experience

This section explains what an end user will see when they launch the Change Installation launcher and choose the **Add Feature** option.

Task ***To add features to an installed product:***

1. Open the Change Product Installation launcher, as described in Initial Experience. The **Maintenance Mode** panel opens.



Select one of the following options:

Add Features
Add features to the installed product

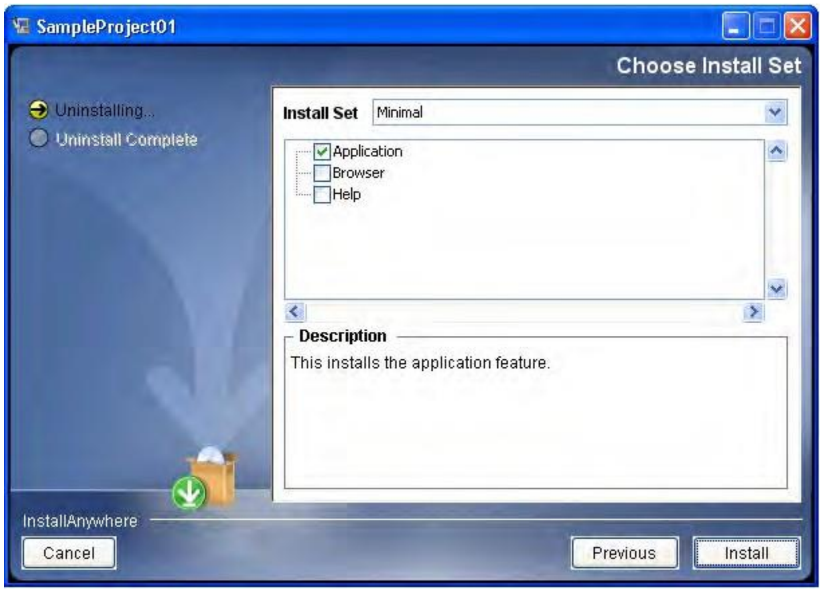
Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

2. Select **Add Features** and click **Next**. The Maintenance Mode runtime displays the panels in the **Add Features** Action Group on the **Pre-Install** Action List and proceeds as a regular installation.

The **Choose Install Set** panel displays the features that are already installed and only allows the selection of the ones that are not yet installed.



This installs the application feature.

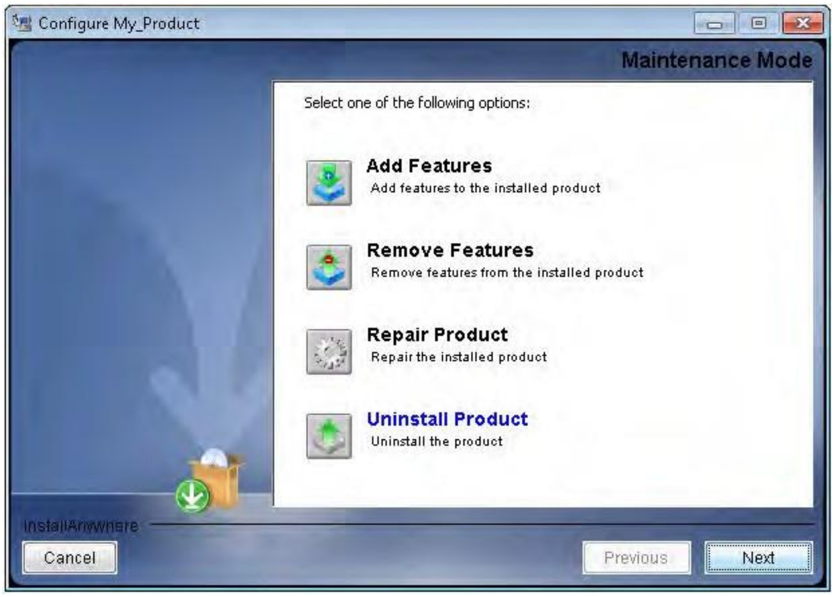
***Note** • Only those Action Groups, Actions, and Panels that have been assigned a **Check Running Mode** rule with a value of **Add Features** will be executed.*

Removing a Feature User Experience

This section explains what an end user will see when they launch the Change Installation launcher and choose the **Remove Features** option

Task ***To remove features to an installed product:***

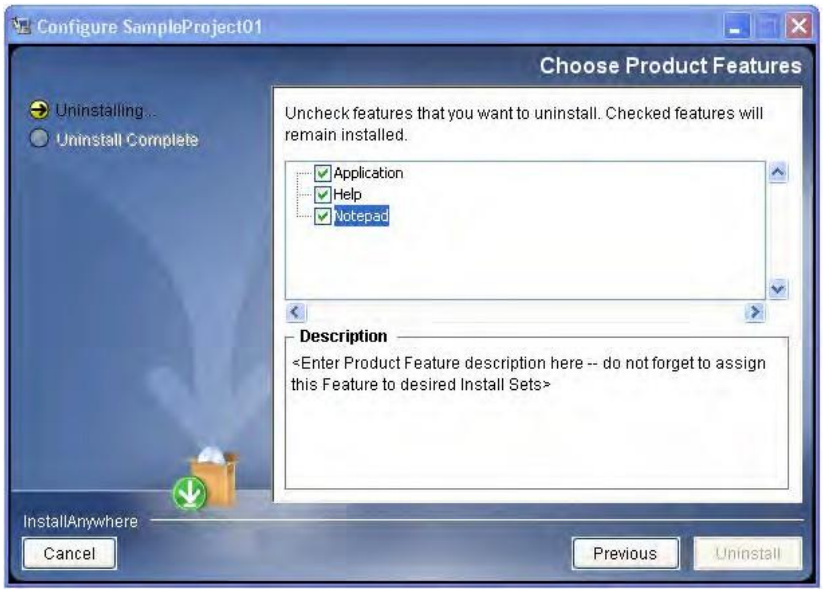
1. Open the Change Installation launcher, as described in Initial Experience. The **Maintenance Mode** panel opens.



2. Select **Remove Features** and click **Next**. The Maintenance Mode runtime displays the panels in the **Remove Features** Action Group on the **Pre-Uninstall** Action List and proceeds as a regular installation.

Note • *Only those Action Groups, Actions, and Panels that have been assigned a **Check Running Mode** rule with a value of **Remove Features** will be executed.*

3. The **Choose Product Features** panel opens, and you are prompted to uncheck the features that you want to uninstall.


Configure SampleProject01

Choose Product Features

Uninstalling...
Uninstall Complete

Uncheck features that you want to uninstall. Checked features will remain installed.

✓ Application
✓ Help
✓ Notepad

Description
<Enter Product Feature description here -- do not forget to assign this Feature to desired Install Sets>

InstallAnywhere
Cancel | Previous | Uninstall

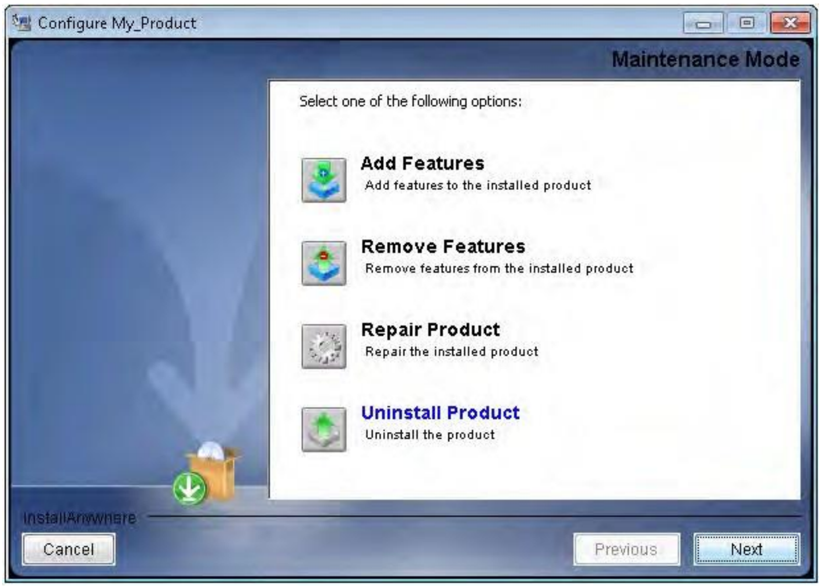
4. Select the features to uninstall and click **Uninstall** to proceed.

Repairing a Product User Experience

This section explains what an end user will see when they launch the Change Installation launcher and choose the **Repair Product** option

Task ***To repair an installed product:***

1. Open the Change Installation launcher, as described in Initial Experience. The **Maintenance Mode** panel opens.


Configure My_Product

Maintenance Mode

Select one of the following options:

Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

InstallAnywhere
Cancel | Previous | Next

2. Select **Repair Product** and click **Next**. The Maintenance Mode runtime displays the panels in the **Repair Installation** Action Group on the **Pre-Install** Action List and proceeds as a regular installation.

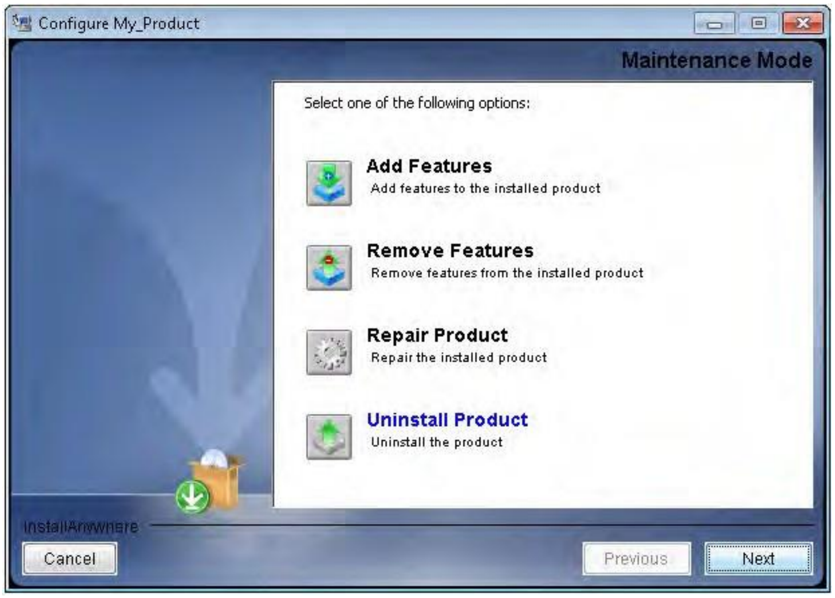
   ***Note** • Only those Action Groups, Actions, and Panels that have been assigned a **Check Running Mode** rule with a value of **Repair Installation** will be executed.*

Uninstalling a Product User Experience

This section explains what an end user will see when they launch the Change Installation launcher and choose the **Uninstall Feature** option

Task ***To uninstall a product:***

1. Open the Change Installation launcher, as described in Initial Experience. The **Maintenance Mode** panel opens.



Select one of the following options:

Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

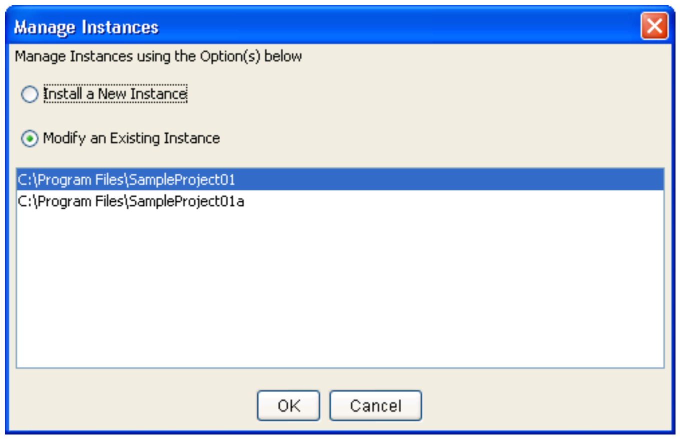
2. Select **Uninstall Product** and click **Next**. The Maintenance Mode runtime displays the panels in the **Pre-Uninstallation** Action Group on the **Pre-Uninstall** Action List and proceeds as a regular uninstallation.

   ***Note** • Only those Action Groups, Actions, and Panels that have been assigned a **Check Running Mode** rule with a value of **Pre-Uninstallation** will be executed.*

Maintenance Mode User Experience on a Machine With Multiple Installed Instances

As described in Specifying Instance Management Options, you can create an installer that permits an end user to install multiple instances of a product on the same machine.

If multiple instances of a product are installed on the same machine, when the end user launches Maintenance Mode, the **Manage Instances** dialog box opens, where they are prompted to either choose to install a new instance or select the instance that they want to modify.


Manage Instances

Manage Instances using the Option(s) below

Install a New Instance [✓ Modify an Existing Instance]

C:\Program Files\SampleProject01
C:\Program Files\SampleProject01a

Figure 9-11: Manage Instances Dialog Box

10
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 that you plan, and create it using a number of InstallAnywhere development concepts.

Task ***To create a new project:***

1. Build an installer according to the following guidelines:

Name	Value
Magic Folders	Your installer must include at least one magic folder, in addition to the core `USER_INSTALL_DIR` and `SHORTCUTS`.
LaunchAnywhere	Your installer must contain at least one LaunchAnywhere launcher.
InstallAnywhere Variables	Your installer must display an understanding and reasonable management of InstallAnywhere variables.
InstallAnywhere Rules	You must implement rules that control installer behavior and install-path options.
Implement the following actions and panels	● Panel: Choose Alias, Link, or Shortcut ● Panel: Choose File or Choose Folder ● Panel: Choose Install Folder ● Panel: Display Message ● Panel: Get Password ● Panel: Install Failed ● Panel: Install Success ● Panel: Install Summary ● Panel: Introduction ● Panel: Show License Agreement
Implement the following InstallAnywhere actions	● Add Comment ● Create LaunchAnywhere for Java Application ● Create Alias, Link, or Shortcut ● Install File ● Install Folder ● Install Uninstaller ● Output Text to Console ● Create Folder ● Set System Environment Variable ● Execute Target File ● Get Windows Registry ● Set InstallAnywhere Variable ● Set Windows Registry - Single Entry ● Show Message Dialog

2. When you have completed your installer project, test your installer to see if it meets your expectations.

Note • *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.

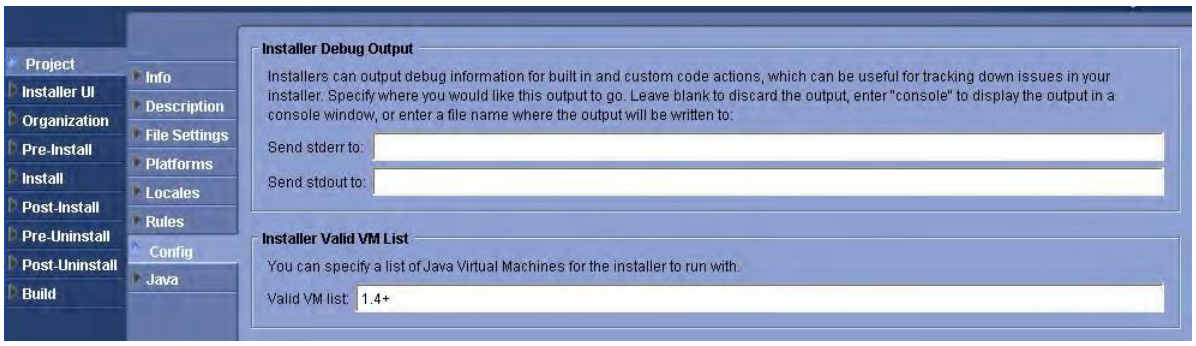
Most InstallAnywhere installers utilize InstallAnywhere's LaunchAnywhere technology. Along with many convenient features for end users (double-clickable launchers, native-like user experience), LaunchAnywhere launchers provide many built-in debugging features.

During Installer Development

InstallAnywhere provides a project-specific debugging feature that enables you to create a debug file for each installer.

Task ***To activate the project-specific debugging feature:***

1. In the InstallAnywhere Advanced Designer, select **Project > Config**.



Installer Debug Output

Installers can output debug information for built in and custom code actions, which can be useful for tracking down issues in your installer. Specify where you would like this output to go. Leave blank to discard the output, enter "console" to display the output in a console window, or enter a file name where the output will be written to:

Send stderr to:

Send stdout to:

Installer Valid VM List

You can specify a list of Java Virtual Machines for the installer to run with.

Valid VM list: 1.4+

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 file options to the following values:

Option	Value
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.

Starting with InstallAnywhere 2009, you can also use LAX environment-variable expressions of the form `$lax.nl.env.var_name$` or `$lax.nl.env.exact_case.var_name$`, Java system properties using the form `$prop.property_name$`, and the platform-independent directory separator expression `$/$` in destinations for redirecting stdout and stderr. (The use of LAX environment-variable expressions is not supported on Mac OS systems or for pure Java installers.)

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 Windows Installer

To view or capture the debug output from a Windows 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, there is an indirect method available.

Task ***To debug a Windows installer:***

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 whose name begins with the letter "I" followed by several digits, as in "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` file that 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 is 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.

Task ***To generate debug output:***

1. In the InstallAnywhere Advanced Designer, highlight the launcher.
2. Click the **Edit Properties** button.
3. Alter the values for the following variables:

   `lax.stderr.redirect=` and `lax.stdout.redirect=`

   to be:

   `lax.stderr.redirect=output.txt` and `lax.stdout.redirect=output.txt`
4. After a normal installation, edit the `.lax` file as described in the preceding instructions.
5. Repeat this procedure 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 2009
Version: 10.0

Fri Oct 31 25:00:00 CST 2008

Total Memory: 24575 KB
Free Memory: 22581 KB

No arguments.

java.class.path:
  C:\Program Files\InstallAnywhere 2009 Enterprise\resource
  C:\Program Files\InstallAnywhere 2009 Enterprise\resource\swingall.jar
  C:\Program Files\InstallAnywhere 2009 Enterprise\resource\compiler.zip
  C:\Program Files\InstallAnywhere 2009 Enterprise\IAClasses.zip
  C:\Program Files\InstallAnywhere 2009 Enterprise\lax.jar
  C:\Program Files\InstallAnywhere 2009 enterprise\jre\lib\rt.jar

ZGUtil.CLASS_PATH:
  C:\Program Files\InstallAnywhere 2009 Enterprise\resource
  C:\Program Files\InstallAnywhere 2009 Enterprise\resource\swingall.jar
  C:\Program Files\InstallAnywhere 2009 Enterprise\resource\compiler.zip
  C:\Program Files\InstallAnywhere 2009 Enterprise\IAClasses.zip
  C:\Program Files\InstallAnywhere 2009 Enterprise\lax.jar

java.version == 1.5.0
java.vendor == Sun Microsystems Inc.
java.home == C:\Program Files\InstallAnywhere 2009 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:

- Add a rule that states: `Install Only If $prop.os.name$=Solaris`

  The install continues, and the action assigned this rule does not execute.

- So, add a display message panel. The message is: The `prop.os.name is: $prop.os.name$`

When 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.

11

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
- Adding Source Paths
- Updating the Location of Files and Resources
- Managing Source Files
- The Resource Manager
- Adding Source Path Management Capability to Your Installer Project
- Quick Quiz

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:

```
$LONG_PATH$ = D:\sources\SampleApp\3000
$SHORT_PATH$ = D:\sources
```

Then, when you add a file such as `D:\sources\SampleApp\3000\readme.txt`, the file will be referenced by `$LONG_PATH$/readme.txt`, since *$LONG_PATH$* 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.

InstallAnywhere provides some predefined source paths that exist in any project. They cannot be changed or edited. They are:

Table 11-1 • Predefined Source Paths

Source Path	Description
`$IA_HOME$`	Location on the system where InstallAnywhere is running. A common location is `C:\Program Files\InstallAnywhere`.
`$IA_PROJECT$`	Location on the system where the InstallAnywhere project is located.
`$USER_HOME$`	The User Home folder.

Adding Source Paths

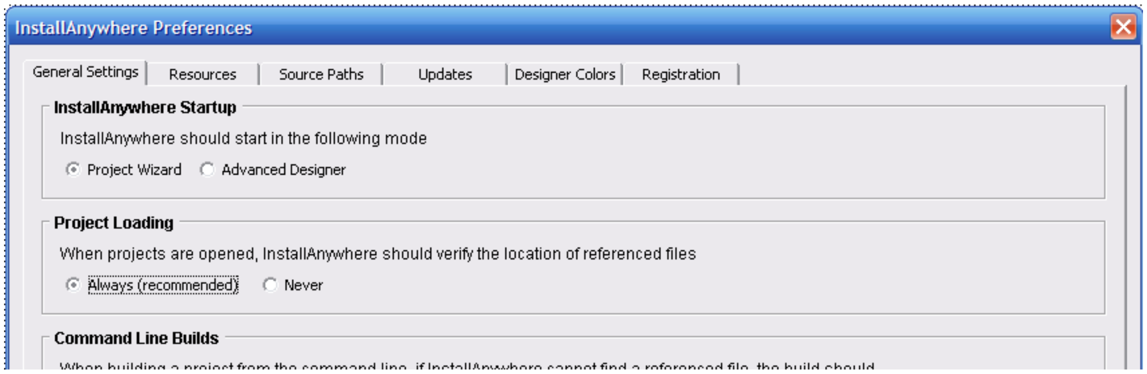
Source paths are added through the use of source path variables. This can be done in Advanced Designer or using environment variables.

InstallAnywhere Preferences

To set InstallAnywhere preferences, perform the following steps.

Task ***To set InstallAnywhere preferences:***

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



InstallAnywhere Startup

InstallAnywhere should start in the following mode

[✓ Project Wizard] Advanced Designer

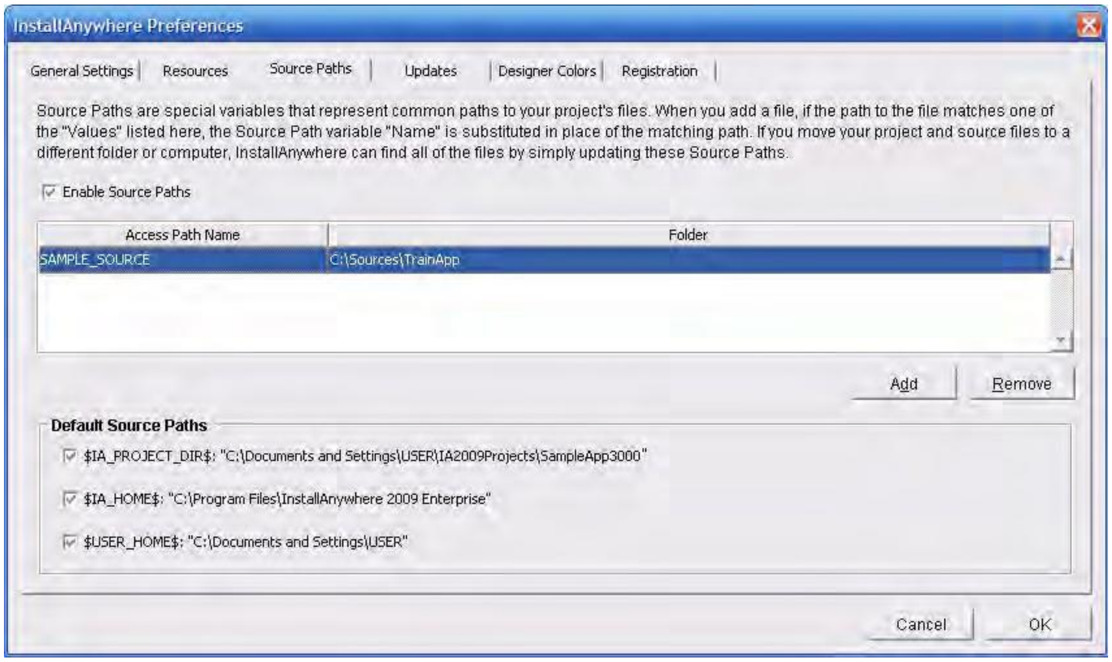
Project Loading

When projects are opened, InstallAnywhere should verify the location of referenced files

[✓ Always (recommended)] Never

Command Line Builds

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.



Source Paths are special variables that represent common paths to your project's files. When you add a file, if the path to the file matches one of the "Values" listed here, the Source Path variable "Name" is substituted in place of the matching path. If you move your project and source files to a different folder or computer, InstallAnywhere can find all of the files by simply updating these Source Paths.

✓ Enable Source Paths

Access Path Name	Folder
SAMPLE_SOURCE	C:\Sources\TrainApp

Default Source Paths

✓ $IA_PROJECT_DIR$: "C:\Documents and Settings\USER\IA2009Projects\SampleApp3000"

✓ IA_HOME: "C:\Program Files\InstallAnywhere 2009 Enterprise"

✓ $USER_HOME$: "C:\Documents and Settings\USER"

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

To set system environment variables, perform the following steps.

Task ***To set system environment variables:***

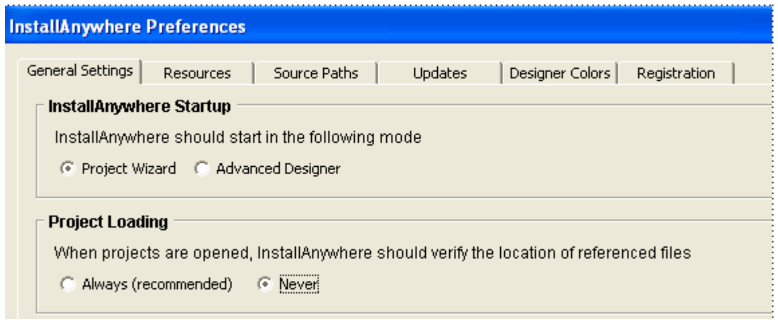
1. Access the environment variables.
   - On a Windows system, right-click **My Computer** on the Desktop, choose **Properties**, choose **Advanced**, and click **Environment Variables**.
   - On Unix or Mac OS X, modify the proper shell configuration file or set the variable directly using the shell.
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**.



InstallAnywhere Startup

InstallAnywhere should start in the following mode

[✓ Project Wizard] Advanced Designer

Project Loading

When projects are opened, InstallAnywhere should verify the location of referenced files

Always (recommended) [✓ Never]

Figure 11-1: 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 Manger 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:

Table 11-2 • Default Source Paths

Source Path	Description
`$IA_HOME$`	Location on your system where you are running InstallAnywhere. A common location may be `C:\Program Files\InstallAnywhere`.
`$IA_PROJECT$`	Location on your system where your InstallAnywhere project is located.
`$USER_HOME$`	The User Home directory on all platforms.

Adding Source Paths

Source paths can be added through the use of variables. Variables are defined on the **InstallAnywhere Preferences** dialog box, in the `PathManager.properties` file, or at the operating system level.

InstallAnywhere Preferences

To add source paths to the InstallAnywhere Preferences dialog box, perform the following steps.

Task ***To add source paths to the Preferences***

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:\Sources\TrainApp`, or browse to the file location.

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

Set System Environment Variables

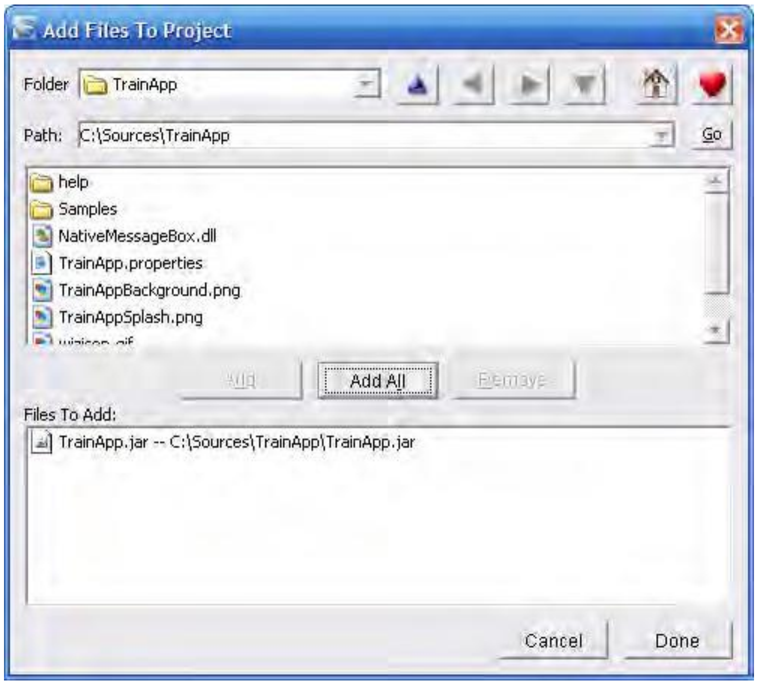
To set system environment variables, perform the following steps.

Task ***To 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

In the **Install** task, add the file `TrainApp.jar` from your TrainApp directory.


Add Files To Project

Folder: TrainApp

Path: C:\Sources\TrainApp

help
Samples
NativeMessageBox.dll
TrainApp.properties
TrainAppBackground.png
TrainAppSplash.png

Files To Add:
TrainApp.jar -- C:\Sources\TrainApp\TrainApp.jar

Figure 11-2: Adding Files from the Install Task

The source path you defined should appear in the action's customizer.

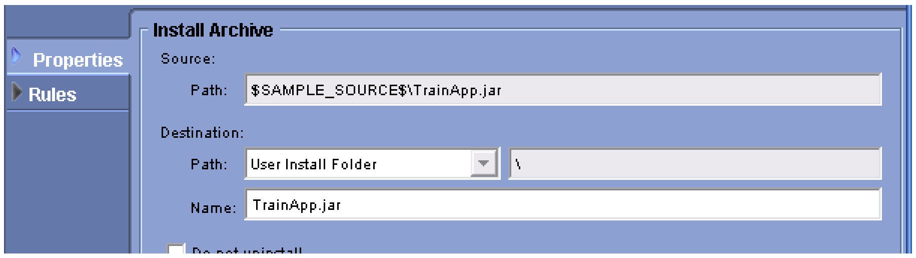

Install Archive
Source:
Path: $SAMPLE_SOURCE$\TrainApp.jar
Destination:
Path: User Install Folder \
Name: TrainApp.jar

Figure 11-3: Source Path in Action Properties

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_`

Table 11-3 • Answers to Quick Quiz

Question	Answer
1	d
2	b

12

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, Basic Installer Customization, 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:

Table 12-1 • Additional Installation-Related Concepts

Concept	Description
Console Installers	Explains using console-mode for command-line-driven installations.
Silent Installers	Explains using silent installations for working with limited or no user interaction.
Uninstallation	Explains using customized uninstallers to control the removal of products and features.
Setting Installation Rollback Options	Explains how to enable installation rollback to avoid any problems that are caused when an end user cancels an installation before it has completed, or if a fatal error occurs during the installation.

Console Installers

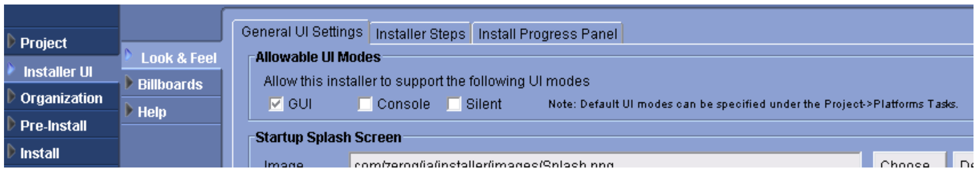
In enterprise-level environments, it is not uncommon for end users to install applications to servers and other remote systems. In these cases, a rich graphical user interface (GUI) such as that provided by InstallAnywhere's standard installer modes is not always desirable. You may find that your end 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.



Allowable UI Modes

Allow this installer to support the following UI modes

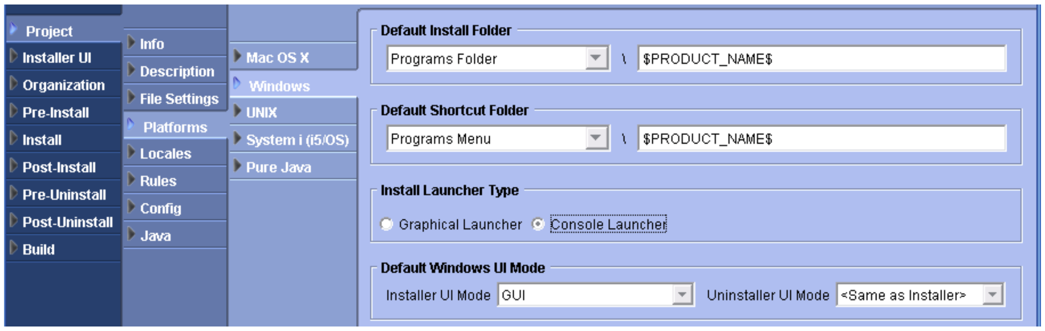
[✓ GUI] Console Silent

Note: Default UI modes can be specified under the Project->Platforms Tasks.

Startup Splash Screen

Figure 12-1: 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. Starting with InstallAnywhere 2009, you can set the default UI mode of the uninstaller independently of the installer.



Default Install Folder: Programs Folder \ $PRODUCT_NAME$

Default Shortcut Folder: Programs Menu \ $PRODUCT_NAME$

Install Launcher Type: Graphical Launcher [✓ Console Launcher]

Default Windows UI Mode

Installer UI Mode GUI Uninstaller UI Mode <Same as Installer>

Figure 12-2: Install Launcher Type

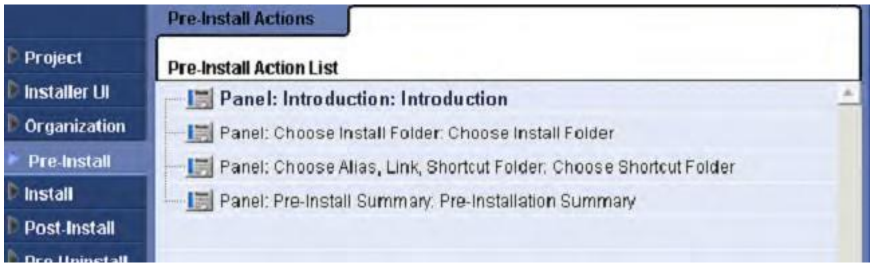
To run an installer in console mode, the user runs the installer with the `-i console` argument, as in the following:

```
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.

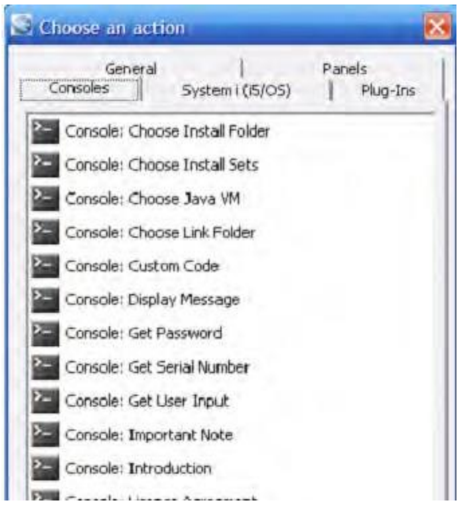
For example, the **Pre-Install** task normally includes graphical panels such as **Introduction**, **Choose Install Folder**, and **Pre-Install Summary**.



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

Figure 12-3: Pre-Install Action List

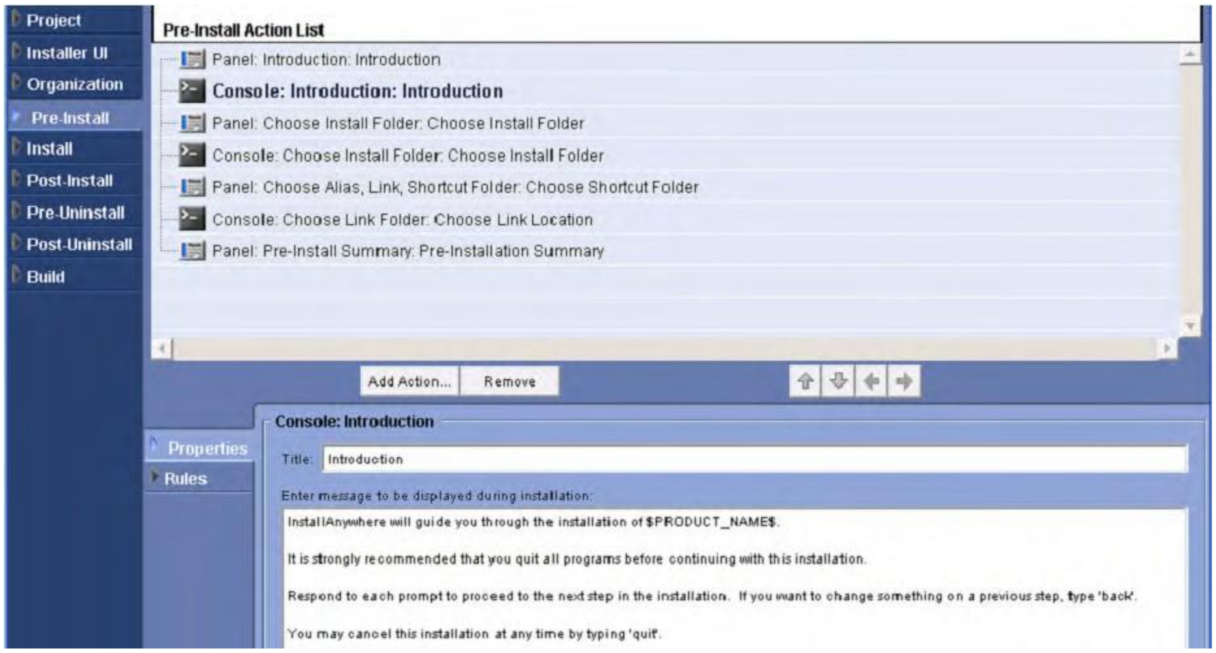
To add console equivalents to these graphical panels, begin by clicking the **Add Action** button and selecting the **Consoles** tab.



Console: Choose Install Folder
Console: Choose Install Sets
Console: Choose Java VM
Console: Choose Link Folder
Console: Custom Code
Console: Display Message
Console: Get Password
Console: Get Serial Number
Console: Get User Input
Console: Important Note
Console: Introduction

Figure 12-4: 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.



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

Console: Introduction

Title: Introduction

Enter message to be displayed during installation:

InstallAnywhere will guide you through the installation of $PRODUCT_NAME$.

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'.

Figure 12-5: 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.

```
# Fri Oct 31 25:00:00 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 the specified property 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:

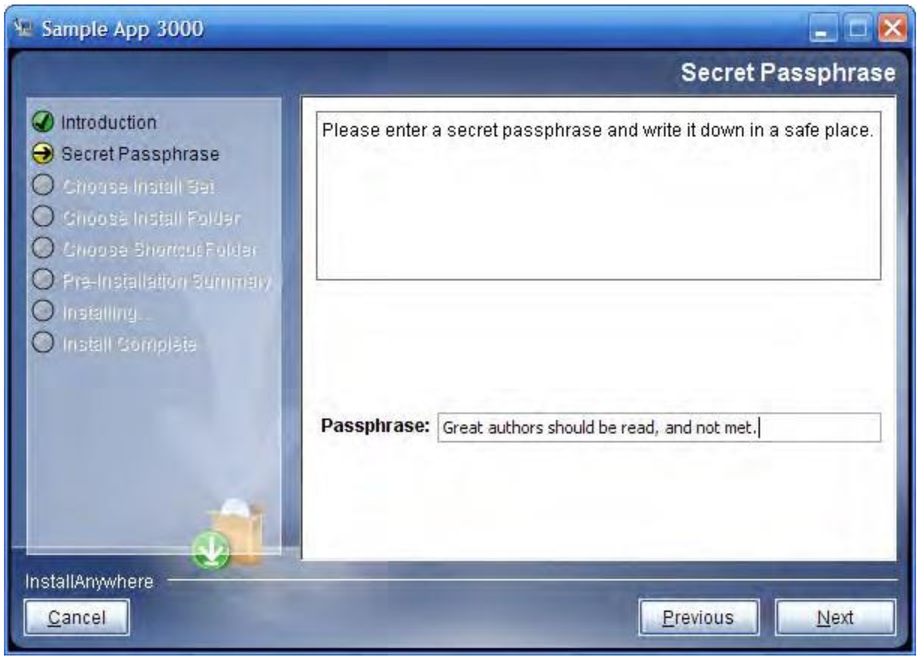
```
./install.bin -i silent -f /usr/tmp/SampleResponseFile.properties
```

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.

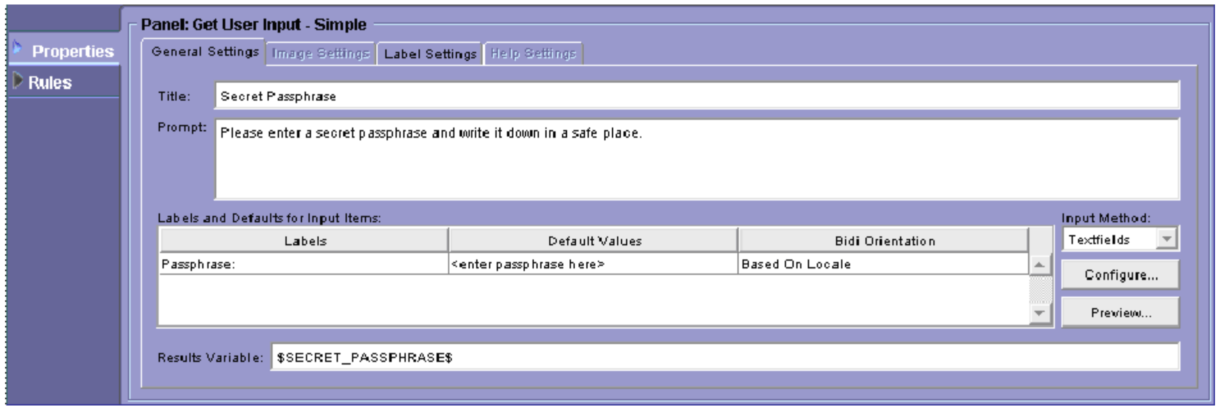


Please enter a secret passphrase and write it down in a safe place.

Passphrase: Great authors should be read, and not met.

Figure 12-6: 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.



Title: Secret Passphrase

Prompt: Please enter a secret passphrase and write it down in a safe place.

Labels and Defaults for Input Items:

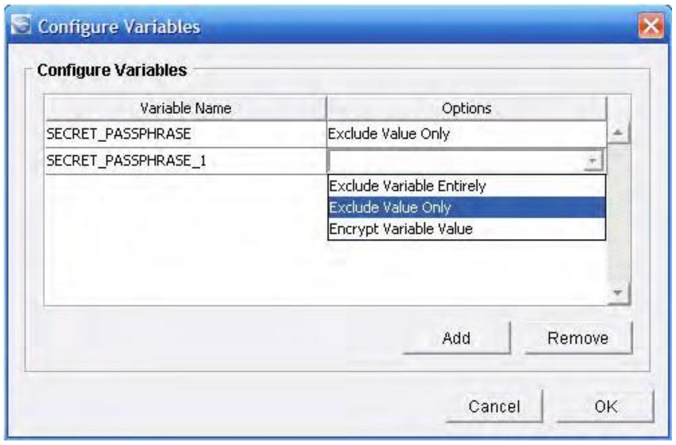
Labels	Default Values	Bidi Orientation
Passphrase:	<enter passphrase here>	Based On Locale

Input Method: Textfields

Results Variable: $SECRET_PASSPHRASE$

Figure 12-7: 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).



Variable Name	Options
SECRET_PASSPHRASE	Exclude Value Only
SECRET_PASSPHRASE_1	

Figure 12-8: Adding Options to Variables

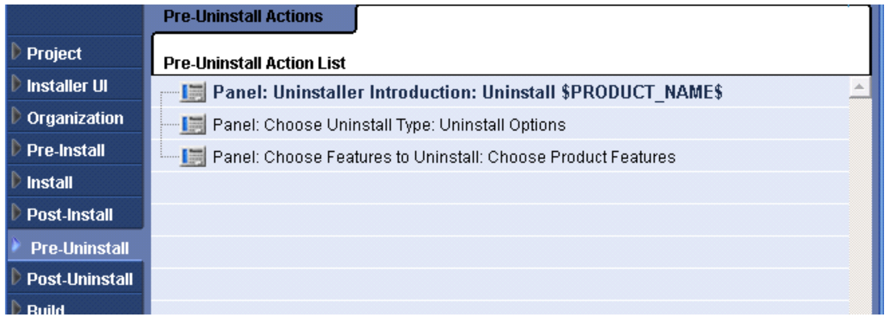
If you select **Exclude Value Only**, a user who generates a response file by running `install -r` receives the following entries for the corresponding panel:

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

***Note** • For additional information on user input panels, refer to Chapter 17, Custom Panels and Consoles.*

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, 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.



Pre-Uninstall Action List
- Panel: Uninstaller Introduction: Uninstall $PRODUCT_NAME$
- Panel: Choose Uninstall Type: Uninstall Options
- Panel: Choose Features to Uninstall: Choose Product Features

Figure 12-9: Pre-Install and Post-Install Tasks

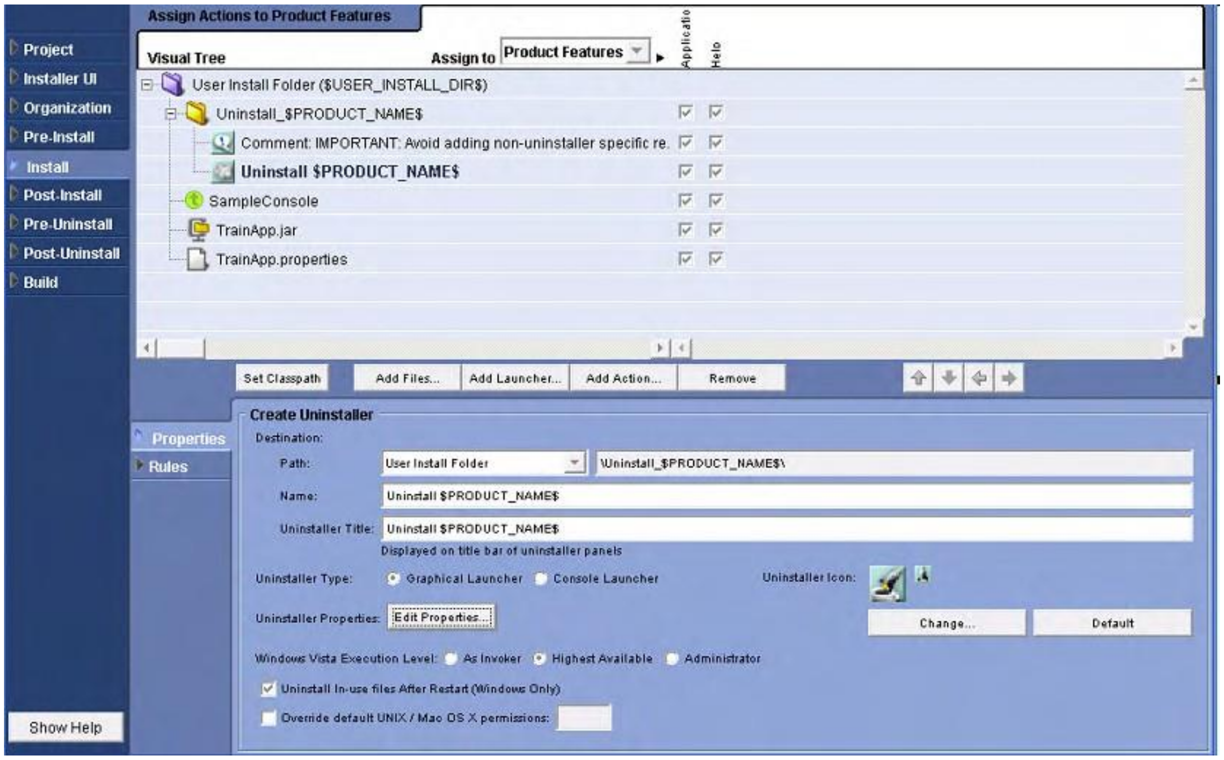
Starting with InstallAnywhere 2009, you can set the variable *$SKIP_UNINSTALL$* to true to prevent the Uninstall step from being performed.

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.



User Install Folder ($USER_INSTALL_DIR$)
Uninstall_$PRODUCT_NAME$
Comment: IMPORTANT: Avoid adding non-uninstaller specific re.
Uninstall $PRODUCT_NAME$
SampleConsole
TrainApp.jar
TrainApp.properties

Create Uninstaller
Destination:
Path: User Install Folder \Uninstall_$PRODUCT_NAME$\
Name: Uninstall $PRODUCT_NAME$
Uninstaller Title: Uninstall $PRODUCT_NAME$
Displayed on title bar of uninstaller panels
Uninstaller Type: [✓ Graphical Launcher] Console Launcher
Uninstaller Icon:
Uninstaller Properties:
Windows Vista Execution Level: As Invoker [✓ Highest Available] Administrator
[✓] Uninstall In-use files After Restart (Windows Only)
[ ] Override default UNIX / Mac OS X permissions:

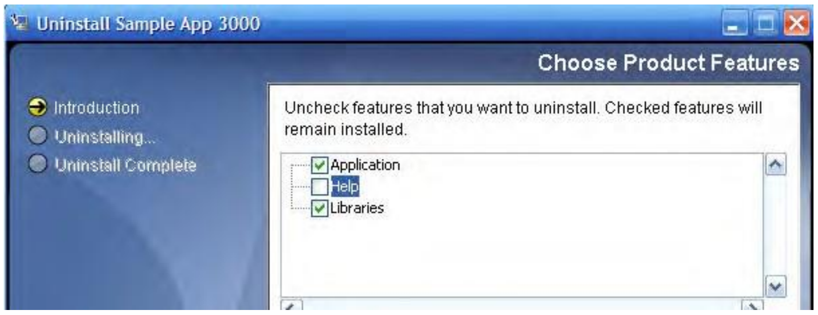
Figure 12-10: Customizing the Uninstall

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.



Uninstall Sample App 3000
Choose Product Features
Uncheck features that you want to uninstall. Checked features will remain installed.
[✓] Application
[ ] Help
[✓] Libraries

Figure 12-11: Choosing Features to Uninstall

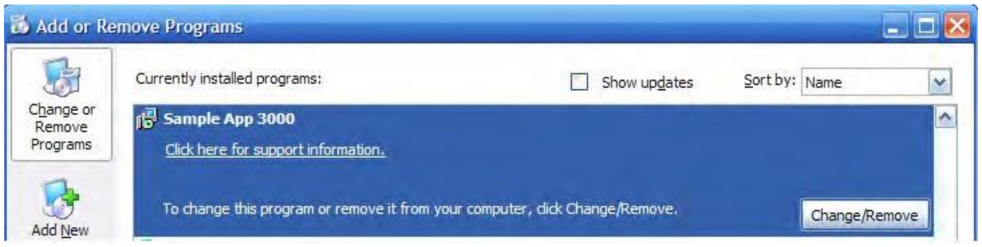
In the settings for the **Choose Features to Uninstall** panel, you can specify to use the opposite behavior, that installed features are displayed un-selected, and selecting a feature causes it to be uninstalled.

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.



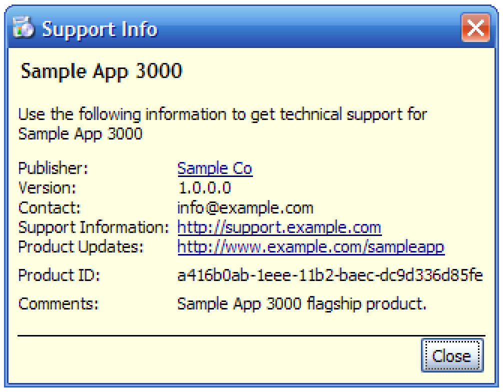
Sample App 3000

Click here for support information.

To change this program or remove it from your computer, click Change/Remove.

Figure 12-12: 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.



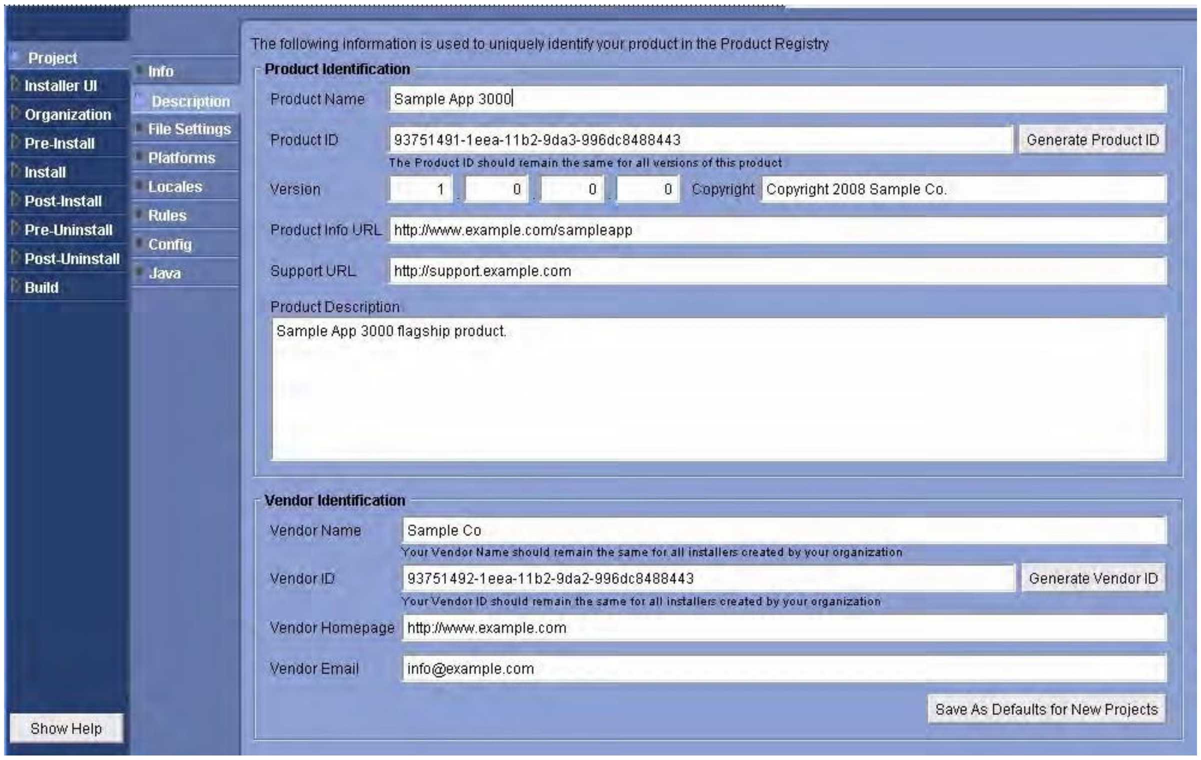
Sample App 3000

Use the following information to get technical support for Sample App 3000

Publisher:	Sample Co
Version:	1.0.0.0
Contact:	info@example.com
Support Information:	http://support.example.com
Product Updates:	http://www.example.com/sampleapp
Product ID:	a416b0ab-1eee-11b2-baec-dc9d336d85fe
Comments:	Sample App 3000 flagship product.

Figure 12-13: Support Information Panel

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



The following information is used to uniquely identify your product in the Product Registry

Product Identification

Product Name: Sample App 3000

Product ID: 93751491-1eea-11b2-9da3-996dc8488443

The Product ID should remain the same for all versions of this product

Version: 1 . 0 . 0 . 0 Copyright: Copyright 2008 Sample Co.

Product Info URL: http://www.example.com/sampleapp

Support URL: http://support.example.com

Product Description: Sample App 3000 flagship product.

Vendor Identification

Vendor Name: Sample Co

Your Vendor Name should remain the same for all installers created by your organization

Vendor ID: 93751492-1eea-11b2-9da2-996dc8488443

Your Vendor ID should remain the same for all installers created by your organization

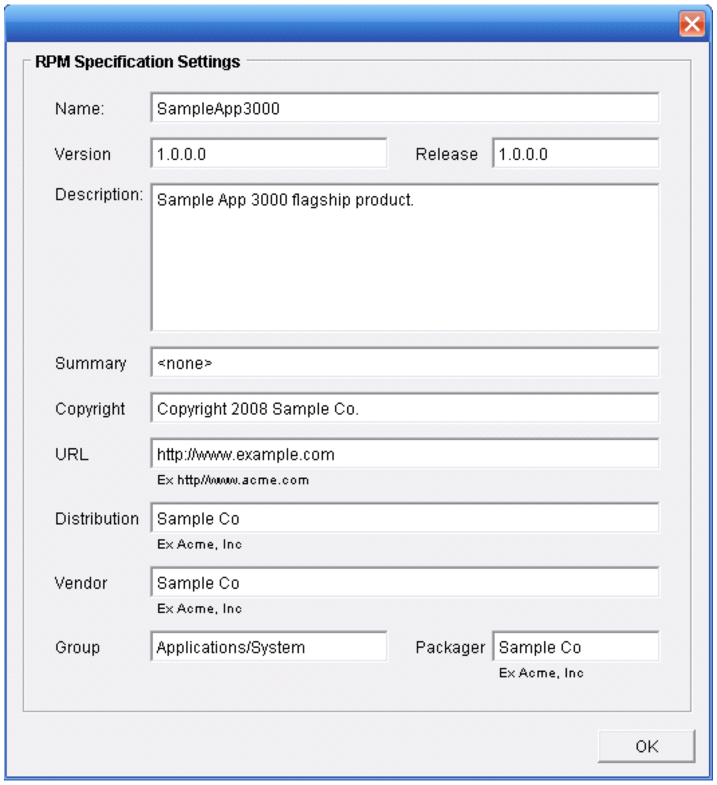
Vendor Homepage: http://www.example.com

Vendor Email: info@example.com

Figure 12-14: 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, Introduction to the Advanced Designer, 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.


RPM Specification Settings

Name: SampleApp3000

Version 1.0.0.0 Release 1.0.0.0

Description: Sample App 3000 flagship product.

Summary <none>

Copyright Copyright 2008 Sample Co.

URL http://www.example.com
Ex http://www.acme.com

Distribution Sample Co
Ex Acme, Inc

Vendor Sample Co
Ex Acme, Inc

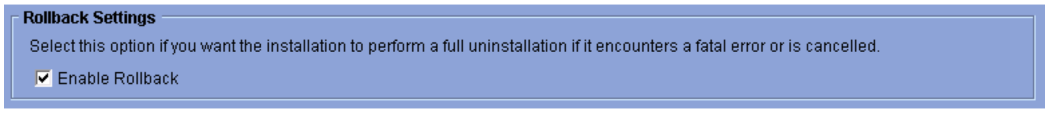
Group Applications/System Packager Sample Co
Ex Acme, Inc

Figure 12-15: RPM Specification Settings

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

Setting Installation Rollback Options

If an end user cancels an installation before it has completed, or if a fatal error occurs during the installation, the result can be an incomplete, corrupt application. To avoid this problem, you can enable installation rollback by selecting the **Enable Rollback** option on the **Project > Advanced** subtask.


Rollback Settings

Select this option if you want the installation to perform a full uninstallation if it encounters a fatal error or is cancelled.

☑ Enable Rollback

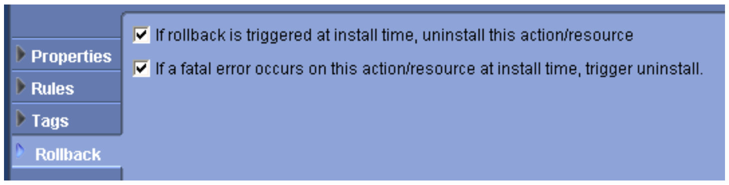
Figure 12-16: Rollback Settings on the Project > Advanced Subtask

***Note** • The **Enable Rollback** option is selected by default for all new projects.*

If you select the **Enable Rollback** option, and the end user cancels an installation or a fatal error occurs, the installer will automatically revert what has been altered or added to the system. The installation log will contain all status messages, including the action that triggered the rollback.

Using the Rollback Subtab of the Install Task to Fine-Tune a Rollback

On the **Rollback** subtab of action customizers in the **Install** task, you can specify rollback options on a file-by-file basis. You can fine tune how the installer treats individual project elements during a rollback and can specify which project elements would trigger a rollback if the installation of that element fails.



☑ If rollback is triggered at install time, uninstall this action/resource
☑ If a fatal error occurs on this action/resource at install time, trigger uninstall.

Figure 12-17: Rollback Subtab of Install Task

If the **Enable Rollback** option is selected on the **Project > Advanced** task, you can select the options on the **Rollback** tab of action customzers in the **Install** task to specify rollback options for the selected project element:

- **If rollback is triggered at install time, uninstall this action/resource**—Select this option if you want the selected project element to be uninstalled if the product installation is cancelled or encounters a fatal error. If this option is not selected and a rollback occurs, the project element will not be uninstalled.
- **If a fatal error occurs on this action/resource at install time, trigger uninstall**—Select this option if you want a rollback to be triggered if the selected project element (execution of the action results in a fatal error) fails to execute properly or to be installed properly. If this option is not selected and the execution of this action results in a fatal error during installation, a rollback will not be triggered.

By default, both options are selected for all project elements.

Note • *In previous releases, InstallAnywhere had support for custom code that provided a Rollback Handler that would be called if an end user cancelled his installation. Now that this new Rollback functionality has been added to InstallAnywhere 2010, the Rollback Handlers custom code will be executed only when the Enable Rollback option on the* ***Project > Advanced*** *subtask is not selected.*

Using Trigger Rollback Actions

You can also choose to add a **Trigger Rollback** action to the **Install** task to specifically initiate a rollback if a certain rule or condition is met.

You can add a **Trigger Rollback** action to the **Install** task to specifically initiate a rollback if a certain rule or condition is met. For example, suppose that you are trying to install an application that interacts with a database. You could add a Trigger Rollback action to the installation so that if the installer detects that the database is not accessible from the machine on which the installation is in progress, the installation could be rolled back.

To add a Trigger Rollback action to the Install task, perform the following steps:

Task ***To add a Trigger Rollback action to the Install task:***

1. Open a project in the Advanced Designer.
2. Open the **Install** task.
3. Click **Add Action**. The **Choose an action** dialog box opens.

4. Select **Trigger Rollback Action** and click **Add**.
5. Click **Close** to close the **Choose an action** dialog box.
6. Select the **Trigger Rollback Action** in the **Visual Tree**.
7. In the **Trigger Rollback Action** customizer, select the **Rules** subtab.
8. Click **Add Rule**.
9. Add the rule or rules that you want to use to define the conditions for rollback. If the conditions defined in the rules are not met, the installation will be rolled back.

13

Creating and Editing Build Configurations

Each InstallAnywhere project can have multiple Build Configurations, each representing how the installer will be built for particular set of platforms, files, build distributions, JVMs, locales, and other settings. You can create and modify Build Configurations on the **Build Configurations** tab of the Advanced Designer's **Build** task. On this tab, you can add, rename, copy, or delete a Build Configuration.

This section explains how to perform the following tasks:

- About Build Configurations
- Creating a New Build Configuration
- Creating Migrated Build Configurations
- Adding a Build Configuration to the Project Build
- Using Tags to Customize Build Configurations

About Build Configurations

In some situations, you may need to generate multiple installers for the same application, each with a slightly different configuration. For example:

- You may want to provide different locale support for different platforms.
- You may want to create two different editions of an application—with both editions containing most of the same content but diverging in a few select files.

One way you could do this would be to create multiple InstallAnywhere projects, each with different settings. However, a much more efficient way of accomplishing this is to use the InstallAnywhere Build Configuration feature.

Benefits of Using Build Configurations

You can create multiple Build Configurations in the same InstallAnywhere project, each with different settings. Each Build Configuration can define how an installer will be built for particular set of platforms, build distributions, JVMs, locales, and other settings. You can store as many Build Configurations with a project as necessary.

When building your InstallAnywhere project, you can choose to build a selected Build Configuration, all Build Configurations that exist in the project, or just a specified set of Build Configurations.

For each Build Configuration that you build, a separate installer, each in its own directory, is generated.

Default Build Configurations for New and Migrated Projects

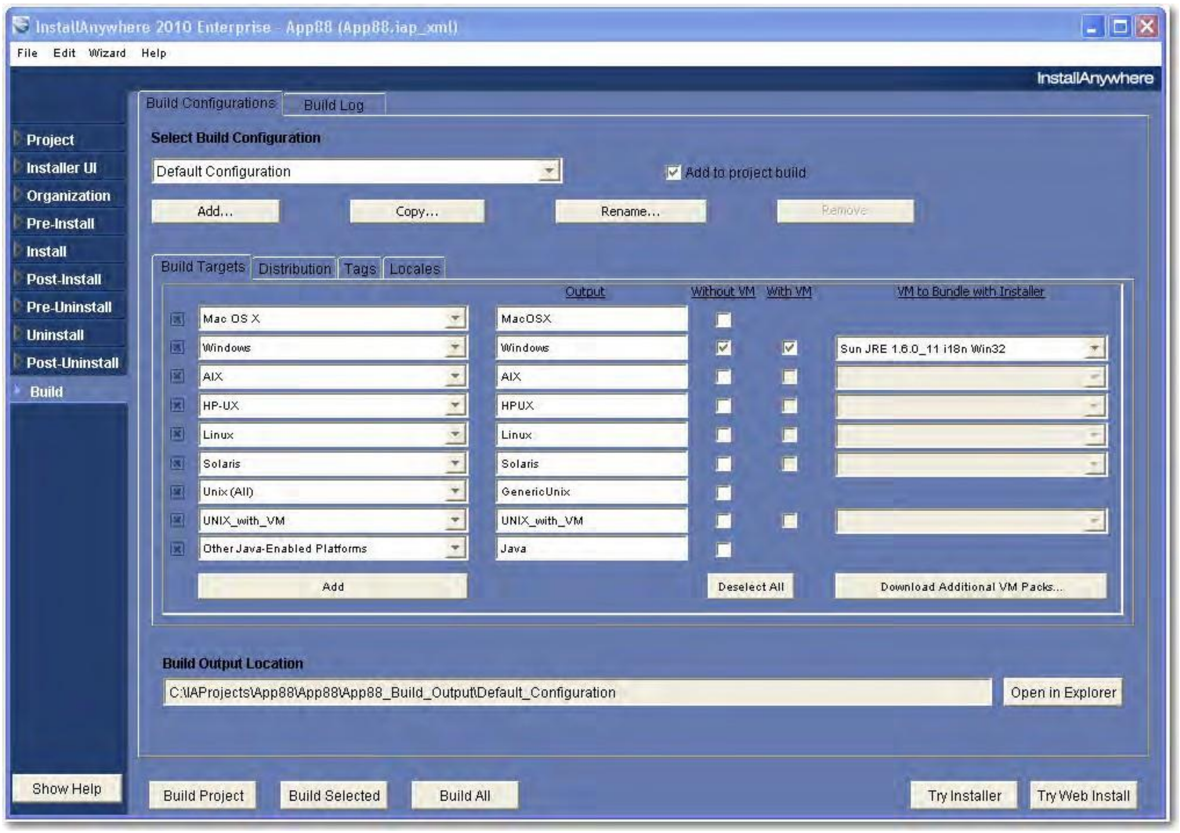
A new InstallAnywhere project will automatically have a Build Configuration named `Default Configuration`.

If you migrate an InstallAnywhere project created using a previous version or edition of InstallAnywhere where the Build Configuration features was unavailable, the migrated project will automatically have a Build Configuration named `Migrated Configuration`, which will have the same build settings that were found in the original project.

Note • *If you migrate an InstallShield MultiPlatform project to InstallAnywhere 2010 using a Project Manifest, a Build Configuration named* `Migrated Configuration` *is created.*

Build Configuration Options in the InstallAnywhere User Interface

You define Build Configurations on the **Build Configurations** tab of the Advanced Designer **Build** task.



Select Build Configuration: Default Configuration

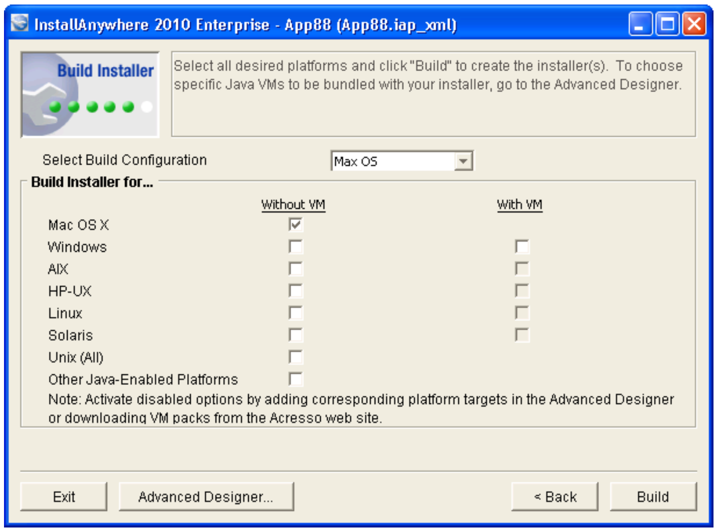
☑ Add to project build

		Output	Without VM	With VM	VM to Bundle with Installer
☒	Mac OS X	MacOSX	☐		
☒	Windows	Windows	☑	☑	Sun JRE 1.6.0_11 i18n Win32
☒	AIX	AIX	☐	☐	
☒	HP-UX	HPUX	☐	☐	
☒	Linux	Linux	☐	☐	
☒	Solaris	Solaris	☐	☐	
☒	Unix (All)	GenericUnix	☐		
☒	UNIX_with_VM	UNIX_with_VM	☐	☐	
☒	Other Java-Enabled Platforms	Java	☐		

Build Output Location: C:\IAProjects\App88\App88\App88_Build_Output\Default_Configuration

Figure 13-1: Build Configurations Tab of the Build Task

When using the Project Wizard interface, you cannot create or edit Build Configurations, but you can select an existing Build Configuration to build by making a selection from the **Select Build Configuration** list on the **Build Installer** panel.


InstallAnywhere 2010 Enterprise - App88 (App88.iap_xml)

Select all desired platforms and click "Build" to create the installer(s). To choose specific Java VMs to be bundled with your installer, go to the Advanced Designer.

Select Build Configuration: Max OS

Build Installer for...

	Without VM	With VM
Mac OS X	☑	
Windows	☐	☐
AIX	☐	☐
HP-UX	☐	☐
Linux	☐	☐
Solaris	☐	☐
Unix (All)	☐	
Other Java-Enabled Platforms	☐	

Note: Activate disabled options by adding corresponding platform targets in the Advanced Designer or downloading VM packs from the Acresso web site.

Figure 13-2: Select Build Configuration Option on the Build Installer Panel of the Project Wizard

Using Build Configuration Tags

You can use Tags to bundle different sets of actions, panels, features, and components into Build Configurations. Using Tags involves three main steps:

- **Create Tags**—First, create a set of Tags that you can use to group project elements.
- **Assign Tags to project elements**—Next, assign an appropriate Tag to all of those project elements that you want to include in some Build Configurations but exclude from others.
- **Associate Tags with Build Configurations**—Then, for each Build Configuration, specify which Tags you want to include and which Tags you want to exclude.

When a Build Configuration that has been customized using Tags is built, it results in an installer that includes:

- **All untagged project elements**—The installer will include all untagged project elements: project elements that do not have any Tags listed in the **Associated Tags** list on the **Tags** subtab of its customizer:
- **Project elements associated with a Tag that is also associated with the Build Configuration**—The installer will include those project elements that have been associated with one or more Tags, one of which is also associated with the selected Build Configuration.

All other project elements will be excluded. In other words, if a project element is associated with one or more Tags, none of which is associated with the selected Build Configuration, that project element will be excluded from the installer.

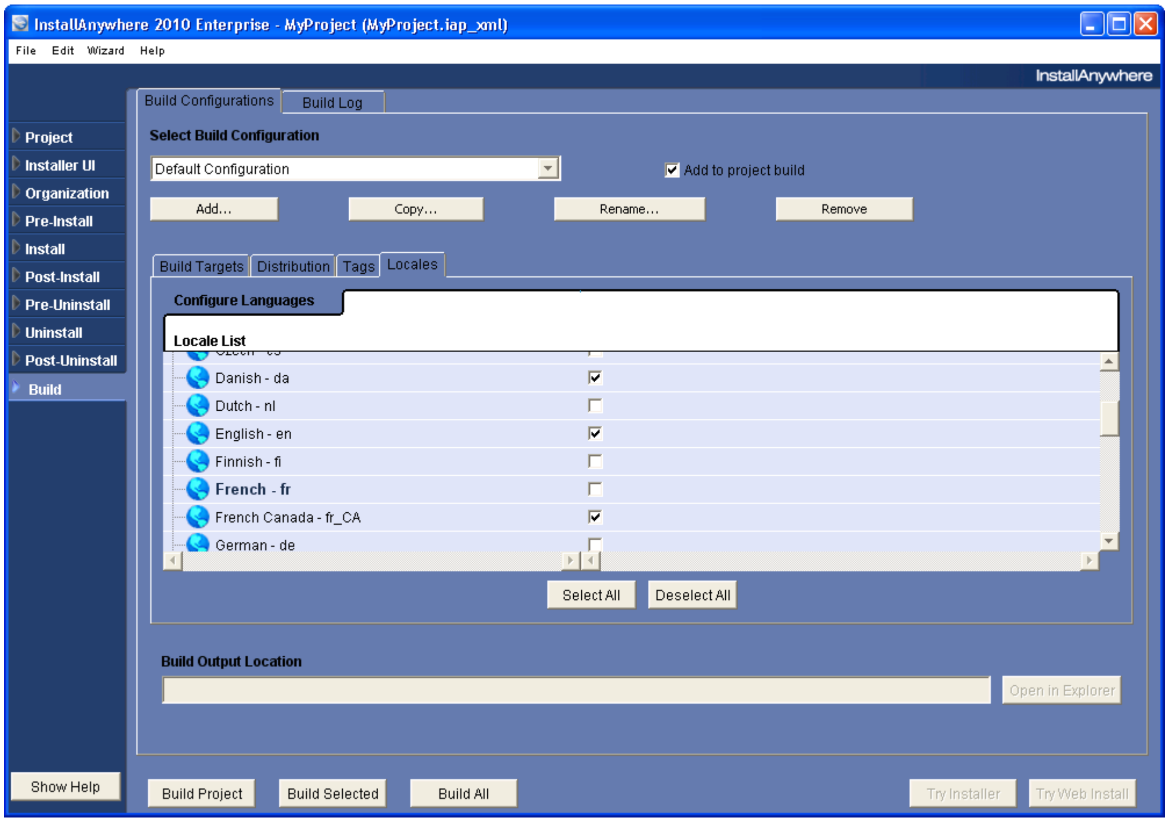
Note • *For more information, see Using Tags to Customize Build Configurations.*

Important • *If a project element is not associated with any Tag, then that project element is included in all Build Configurations by default.*

Important • *When you create a new Tag, it is automatically associated with all existing Build Configurations.*

Locale Settings

For each Build Configuration, you can specify the languages for which the installer will be created. Languages are selected on the **Locales** subtab of the **Build Configurations** tab of the Advanced Designer **Build** task. A locale is enabled when it is checked.



Select Build Configuration: Default Configuration

[✓] Add to project build

Configure Languages

Locale List

Danish - da	✓
Dutch - nl	
English - en	✓
Finnish - fi	
French - fr	
French Canada - fr_CA	✓
German - de	

Build Output Location

Figure 13-3: Build Configurations Tab of the Build Task / Locales Subtab

Important • *In previous releases, Locales could only be specified on a project-level basis on the **Project > Locales** subtask, and these Locale settings were applied to all installers created by this project. Starting with InstallAnywhere 2010, you specify Locale settings on the **Locales** subtab of the **Build Configurations** tab, which enables you to specify different Locale settings for each Build Configuration.*

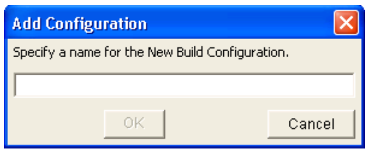
Creating a New Build Configuration

Each InstallAnywhere project can have multiple Build Configurations, each representing how the installer will be built for particular set of platforms, files, build distributions, JVMs, and other settings.

To add a new Build Configuration to a project, perform the following steps.

Task ***To add a new Build Configuration to a project:***

1. Open an installer project in the Advanced Designer.
2. Open the **Build** task.
3. On the **Build Configurations** tab, click **Add**. The **Add Configuration** dialog box opens, prompting you to enter a name for the new Build Configuration.


Add Configuration
Specify a name for the New Build Configuration.

4. Enter a name and click **OK**.
   - The name must be 60 characters or less.
   - Do not use an asterisk (*), question mark (?), period (.), back slash (\), or forward slash(/) in the name.

   The new Build Configuration is created and is selected in the **Select Build Configuration** list, but it is not yet saved to the project.
5. Specify settings for this Build Configuration.
6. On the **File** menu, click **Save** to save this new Build Configuration into the project.

Creating Migrated Build Configurations

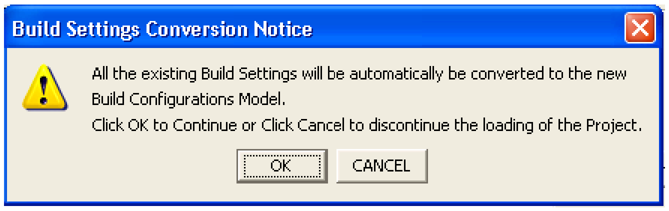
When a project created using a version of InstallAnywhere prior to InstallAnywhere 2010 is opened, all of the build settings of the earlier project are migrated to the new Build Configuration model. These migrated build settings are saved as a migrated Build Configuration named **Migrated Configuration**.

To create a migrated Build Configuration, perform the following steps:

Task ***To create a migrated Build Configuration:***

1. Launch InstallAnywhere. The InstallAnywhere **About** dialog box opens.
2. Click **Open Existing**. The **Open Project File** dialog box opens.
3. Locate and select the project that was created using an earlier version of InstallAnywhere.

4. Click **Open**. The **Build Settings Conversion Notice** dialog box opens, informing you that the existing build settings will automatically be converted to use the new Build Configurations model.



All the existing Build Settings will be automatically be converted to the new Build Configurations Model.
Click OK to Continue or Click Cancel to discontinue the loading of the Project.

5. Click **OK**. The project opens in the Advanced Designer, and **Migrated Configuration** is selected in the **Select Build Configuration** list on the **Build Configurations** tab of the **Build** task.

***Note** • The **Add to project build** option for the migrated Build Configuration is automatically selected.*

6. Save the project.
7. You can then modify the settings of this migrated Build Configuration.

Adding a Build Configuration to the Project Build

You can add a Build Configuration to the project build by selecting its **Add to project build** option. If a Build Configuration is added to a project build, that Build Configuration will be built each time one of the following occurs:

- You click the **Build Project** or **Build All** buttons on the **Build Configurations** tab of the **Build** task.
- You build the project from the command line without specifying any Build Configurations, such as:

```
build.exe MyProject.iap_xml
```

To add a Build Configuration to the project build, perform the following steps.

Task ***To add a Build Configuration to the project build:***

1. Open an installer project in the Advanced Designer.
2. Open the **Build** task.
3. On the **Build Configurations** tab, select the Build Configuration that you want to add to the project build from the **Select Build Configuration** list.
4. Select the **Add to project build** option.
5. Select **Save** on the **File** menu to save the project.

***Note** • If you click the **Build All** button on the **Build Configurations** tab (or use the `-all` command line argument), all existing Build Configurations for that project will be built regardless of whether their **Add to project build** option has been selected.*

Using Tags to Customize Build Configurations

You can use Tags to bundle different sets of actions, panels, features, and components with Build Configurations. Using Tags involves three main steps:

Table 13-1 • Steps to Customize Build Configurations Using Tags

Step	Description
Create Tags	Define a set of Tags to use with Build Configurations. See Creating New Tags.
Assign Tags to Project Elements	Assign an appropriate Tag to all of those project elements that you want to include in some Build Configurations but exclude from others. See Assigning Tags to Project Elements.
Associate Tags With Build Configurations	For each Build Configuration, specify which Tags you want to include and which Tags you want to exclude. See Associating Tags to Build Configurations.

Determining Whether a Project Element is Included in a Build Configuration

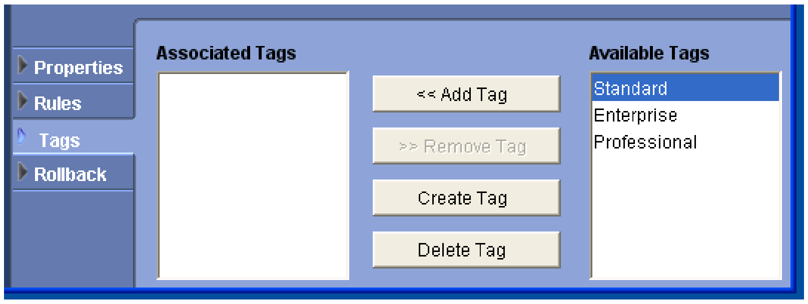
Whether a project element (action, panel, feature, component) is included in a Build Configuration depends what is selected on the **Tags** subtab of each project element's customizer.

- Tagging Project Elements
- Rules for Applying Build Tags to Build Configurations

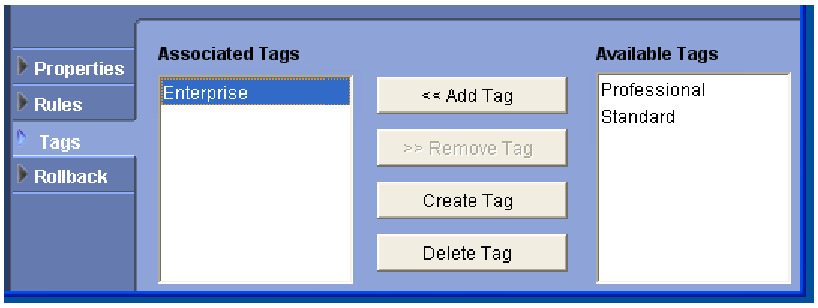
Tagging Project Elements

All project elements are either *tagged* or *untagged*:

- Untagged—A project element is untagged when it does not have any Tags listed in the **Associated Tags** list:



- **Tagged**—A project element is tagged when it has one or more Tags listed in the **Associated Tags** list:



Rules for Applying Build Tags to Build Configurations

When a Build Configuration is built, the project elements that are included depends upon both whether the Build Configuration has any associated Tags, and which (if any) Tags are associated with a project element.

When building a Build Configuration, follow these rules to determine which project elements are included in the resulting installer:

Table 13-2 • Rules for Including Project Elements in a Build Configuration

Build Configuration	Included Project Elements	Excluded Project Elements
No Associated Tags	All project elements (both tagged and untagged). ***Important*** • *When a Build Configuration that has no associated Tags is built, it results in an installer that includes all project elements, both those with associated Tags and those without any associated Tags.*	None
With Tags	• All untagged project elements, AND • Project elements that are tagged with one or more of the Tags that are associated with the Build Configuration. ***Note*** • *For example, if a project element is associated with a Tag named **Professional**, and the Build Configuration is also associated with a Tag named **Professional**, the project element will be included in the installer.*	Project elements that are tagged with non-matching Tags (Tags that are not associated with the Build Configuration). ***Note*** • *For example, if a project element is associated with a Tag named **Standard**, but the Build Configuration is associated with a Tag named **Professional** (but not **Standard**), the project element will be excluded from the installer.*

To summarize, the following rules are enforced:

- A project element which has no Tags associated with it will be included in the installer for all Build Configurations.
- A project element which has Tags associated will be included in the installer if, and only if, the Build Configuration has at least one matching Tag.
- A Build Configuration which has no associated Tags includes all project elements.

Important • *When you create a new Tag, it is automatically associated with all existing Build Configurations.*

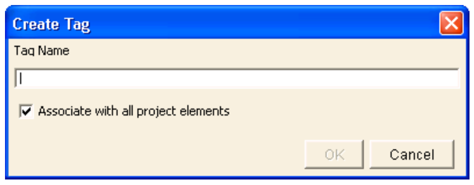
Creating New Tags

You can create new project Tags on the **Project > Advanced** subtask or on the **Tags** subtab of a project element customizer.

Note • *Each Tag can be associated with multiple Build Configurations.*

Task ***To create a new Build Configuration tag:***

1. Do one of the following:
   - Open the **Project > Advanced** subtask of the Advanced Designer.
   - Open the **Tags** subtab of a project element customizer.
2. In the **Manage Tags** area, click **Create Tag**. The **Create Tag** dialog box opens.



3. Enter a name for the new Tag.
4. If you want this new Tag to be automatically associated with all existing project elements, select the A**ssociate with all project elements** option.

   Note • *If you select the **Associate with all project elements** option when creating a new Tag, that Tag is automatically associated with all **existing** project elements, but it is not automatically associated with additional project elements that are added to the project from that point forward.*

5. Click **OK** to close the **Create Tag** dialog box.
6. Select **Save** on the **File** menu to save the project.

Assigning Tags to Project Elements

You can use Tags to bundle different sets of actions, panels, features, and components with Build Configurations. After creating a set of Tags for a project, you need to assign an appropriate Tag to all of those project elements that you want to include in some Build Configurations but exclude from others.

When a group of project elements are associated with a given Tag, and then that Tag is associated with a Build Configuration, then all of those project elements are built for that Build Configuration.

Important • *If a project element is not associated with any Tag, then that project element is included in all Build Configurations by default.*

Considerations When Assigning Build Configuration Tags

In a project, Features take precedence over Components, which take precedence over Actions. When you are assigning Tags to project elements, to avoid conflicts, you should pick one project element level to filter by: Features, Components, or Actions. Mixing and matching is generally not recommended.

For example, if you assign a Tag to a Feature, do not assign a Tag to an action installed with that Feature. If you attempt to assign a Tag to a project element that “belongs” to a higher-level project element that is already assigned a different Tag, a warning message will appear on the **Tags** subtab of that project element’s customizer.

Assigning a Tag to a Project Element

To assign a Tag to a project element, perform the following steps:

Task ***To assign a Tag to a project element.***

1. Open a project in the Advanced Designer.
2. Select a project element in the **Pre-Install**, **Install**, **Post-Install**, **Pre-Uninstall**, **Uninstall**, or **Post-Uninstall** tasks.
3. In the project element customizer, open the **Tags** subtab. All of the Tags defined in this project are listed. Tags associated with this project element are listed in the **Associated Tags** list, while those Tags not associated with this project element are listed in the **Available Tags** list.
4. To associate a Tag with this project element, select a Tag in the **Available Tags** list and then click **Add Tag** to move it to the **Associated Tags** list.
5. To remove a Tag from the **Associated Tags** list, select it then click **Remove Tag**.
6. Save the project.

Associating Tags to Build Configurations

You associate Tags with Build Configurations on the **Tags** subtab of the **Build** task’s **Build Configurations** tab.

Task ***To associate Tags to a Build Configuration:***

1. Open an installer project in the Advanced Designer.
2. Open the **Build** task.
3. On the **Build Configurations** tab, select the Build Configuration that you want to edit from the **Select Build Configuration** list.
4. Open the **Tags** subtab. All of the Tags defined in this project are listed. Tags associated with this Build Configuration are listed in the **Associated Tags** list, while those Tags not associated with this Build Configuration are listed in the **Available Tags** list.
5. To associate a Tag with a Build Configuration, select a Tag in the **Available Tags** list and then click **Add Tag** to move it to the **Associated Tags** list.
6. To remove a Tag from the **Associated Tags** list, select it then click **Remove Tag**.
7. Save the project.

Searching for Tags

You can search a project to locate all references to a given Tag. You can choose to search for a selected Tag in any of the installation tasks/phases (Pre-Install, Install, Post-Install, Pre-Uninstall, Uninstall, Post-Uninstall) and can choose to search Features and/or Components. Search results are displayed on the **Tag Search Results** dialog box.

Task ***To search for a Build Configuration Tag:***

1. Open a project in the Advanced Designer.
2. On the **File** menu, click **Search Tags** (or press Ctrl + F). The **Tag Search Results** dialog box opens.
3. From the **Select Tag** list, select a Build Configuration Tag.
4. Under **Search for Selected Tag in**, select the installation phases (**Pre-Install**, **Install**, **Post-Install**, **Pre-Uninstall**, **Uninstall**, and/or **Post-Uninstall**) and product elements (**Features** and/or **Components**) that you want to search.
5. Click **Search**. Project elements associated with the selected Tag are listed, grouped by category.

14

Advanced Organizational Concepts

This chapter contains information on:

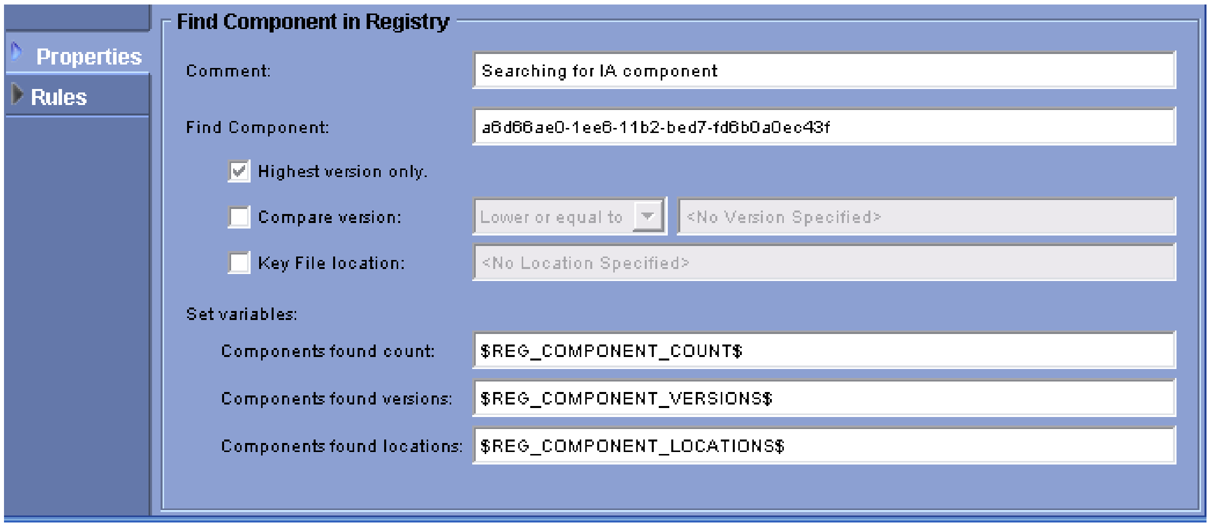
- Using the Find Component in Registry Action
- Merge Modules and Templates
- Importing a Design-Time Merge Module
- Advanced Topic: Importing ISMP Manifests
- InstallAnywhere Collaboration and DIMs
- FLEXnet Connect
- Quick Quiz

Using the 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** (referring to the InstallAnywhere registry and 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. (If you have a project that contains the desired component, the Unique ID value is visible in the component's customizer. Otherwise, if a product has installed the component on a system, the component ID can be found in the InstallAnywhere registry.) 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 you specify.

The following figure shows the customizer for the **Find Component in Registry** action.



Find Component in Registry

Comment:	Searching for IA component
Find Component:	a6d66ae0-1ee6-11b2-bed7-fd6b0a0ec43f
[✓] Highest version only.	
[ ] Compare version:	Lower or equal to <No Version Specified>
[ ] Key File location:	<No Location Specified>
Set variables:	
Components found count:	$REG_COMPONENT_COUNT$
Components found versions:	$REG_COMPONENT_VERSIONS$
Components found locations:	$REG_COMPONENT_LOCATIONS$

Figure 14-1: Find Component in Registry Properties

After the action runs, you can use the resulting variables in subsequent actions and rules. For example, to display a panel only if a component is not found, you can add a Display Message panel with a rule that uses the value of *$REG_COMPONENT_COUNT$*.

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.

Benefits of Using Merge Modules

Merge Modules provide many benefits and provide solutions to complex installation requirements. For instance:

- **Use to create suite installer**—Combine several Merge Modules from different products to create a “Suite Installer.”
- **Use for different components**—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.
- **Self-contained units of installer functionality**—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.
- **Simplify future installer creation**—Save common installer functionality, such as **License Agreement** panels and **Custom User Input** panels, into Merge Modules to simplify future installer project creation.
- **Combine third party merge modules**—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 as starting point for new projects**—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**—Any installer project can be built into a Merge Module. And any Merge Module can be used within any other installer project.
- **Merge module creation during build process**—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).

Merging Merge Modules Into an Existing Installer

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.

Integrating Merge Modules Into a Project

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

Other important facts about Merge Modules are:

- **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.
- **Install Merge Module action**—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.

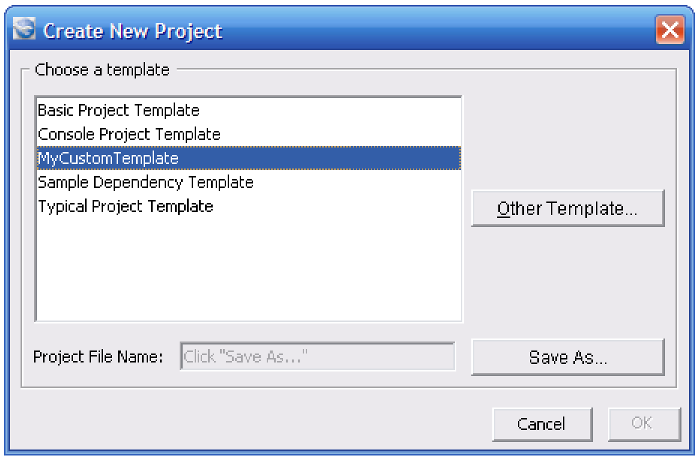


Figure 14-2: Creating a New Project from a Template

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.

Merge Module Types

There are the following merge module types:

- Design-Time Merge Modules
- Build-Time Merge Modules
- Install-Time Merge Modules
- Dynamic Merge Modules

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

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.

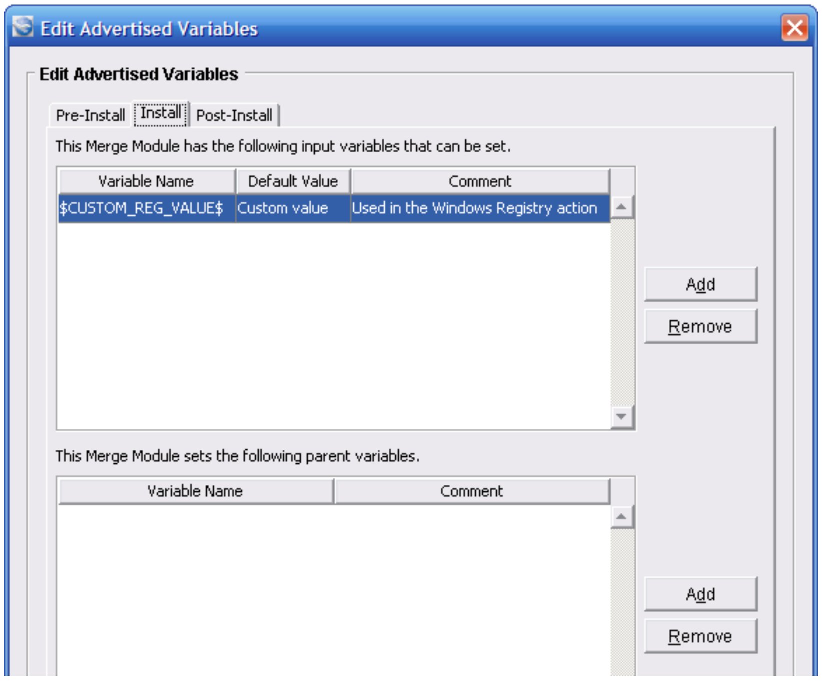
Creating Merge Modules as Read Only

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.

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 set up your variables.



This Merge Module has the following input variables that can be set.

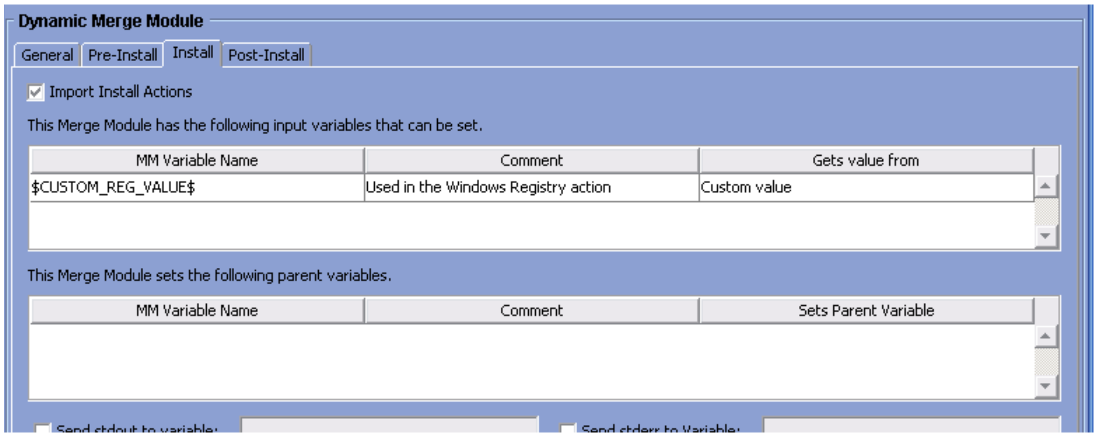
Variable Name	Default Value	Comment
$CUSTOM_REG_VALUE$	Custom value	Used in the Windows Registry action

This Merge Module sets the following parent variables.

Variable Name	Comment

Figure 14-3: Define Advertised 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 required, and these will be passed to the Merge Module at install time.



☑ Import Install Actions

This Merge Module has the following input variables that can be set.

MM Variable Name	Comment	Gets value from
$CUSTOM_REG_VALUE$	Used in the Windows Registry action	Custom value

This Merge Module sets the following parent variables.

MM Variable Name	Comment	Sets Parent Variable

☐ Send stdout to variable: ☐ Send stderr to Variable:

Figure 14-4: Advertised Variables in Parent Project

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 value would both be *`$IA_PROJECT_DIR$`*.

Note • *Variables in InstallAnywhere must be expressed with dollar signs ($) on either side.*

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.

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.

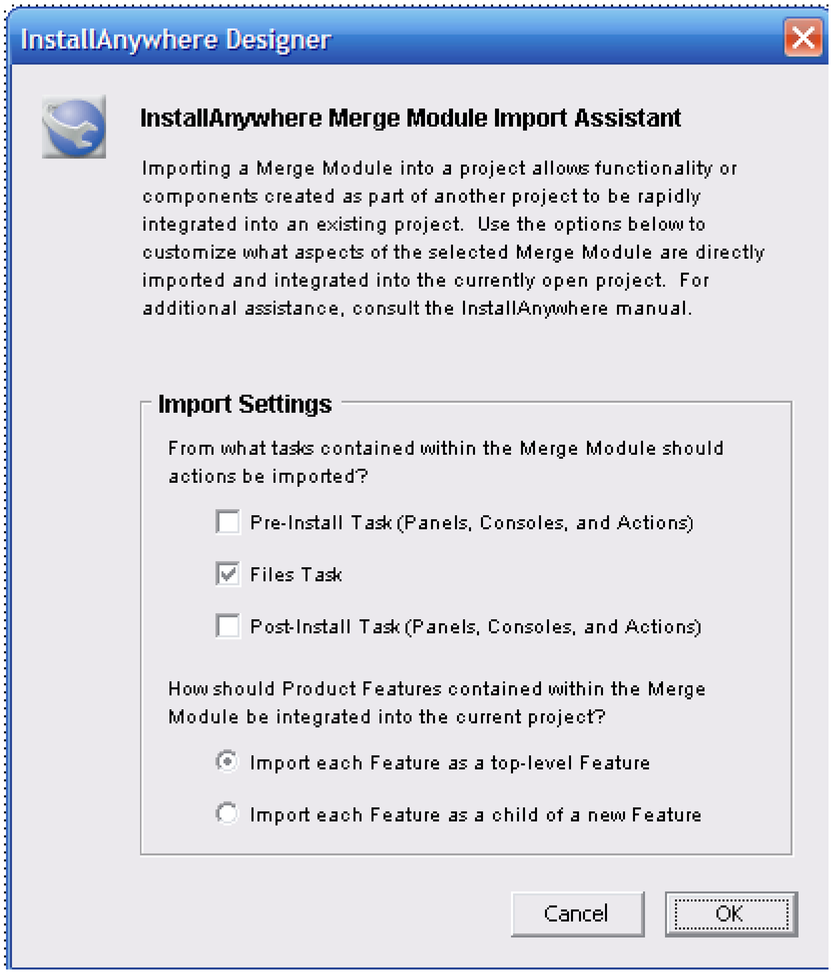
Importing a Design-Time Merge Module

To import a design-time merge module, perform the following steps.

Task ***To import a design-time 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.



InstallAnywhere Merge Module Import Assistant

Importing a Merge Module into a project allows functionality or components created as part of another project to be rapidly integrated into an existing project. Use the options below to customize what aspects of the selected Merge Module are directly imported and integrated into the currently open project. For additional assistance, consult the InstallAnywhere manual.

Import Settings

From what tasks contained within the Merge Module should actions be imported?

- [ ] Pre-Install Task (Panels, Consoles, and Actions)
- [✓] Files Task
- [ ] Post-Install Task (Panels, Consoles, and Actions)

How should Product Features contained within the Merge Module be integrated into the current project?

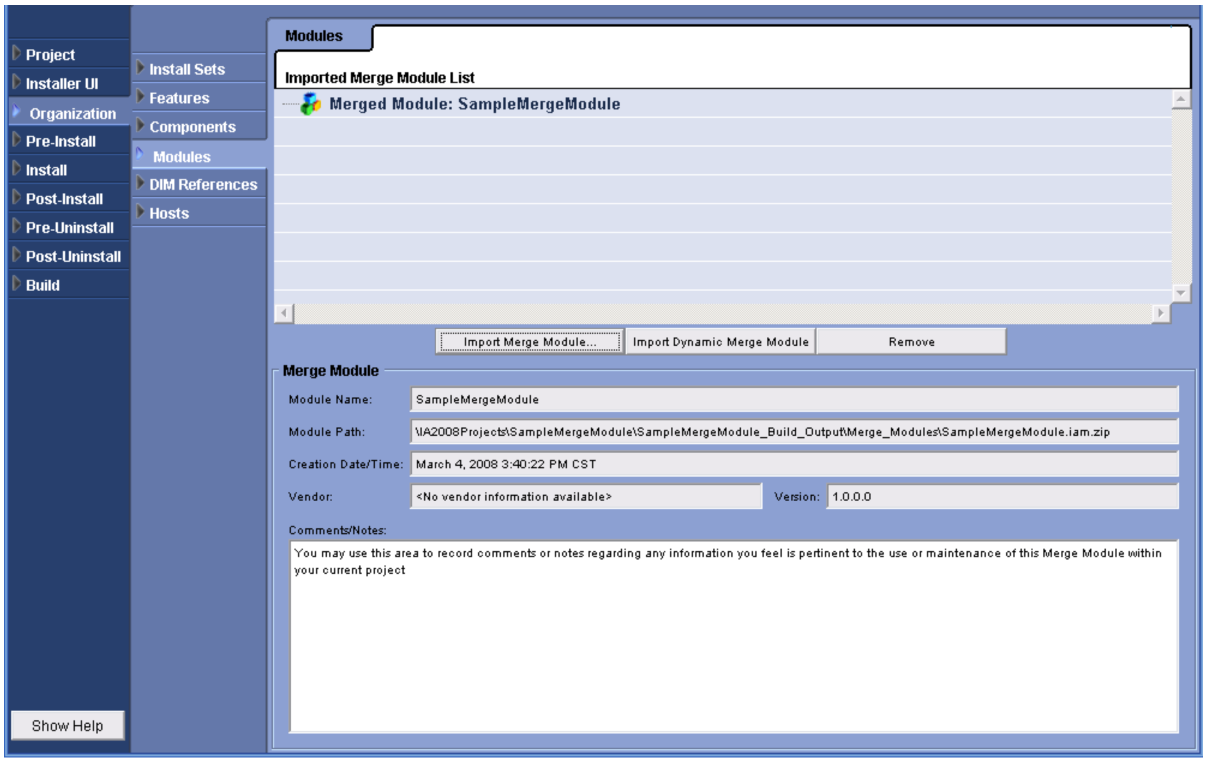
- [✓ Import each Feature as a top-level Feature]
- Import each Feature as a child of a new Feature

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. It is recommended 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.



Merged Module: SampleMergeModule

Merge Module

Module Name: SampleMergeModule

Module Path: \IA2008Projects\SampleMergeModule\SampleMergeModule_Build_Output\Merge_Modules\SampleMergeModule.iam.zip

Creation Date/Time: March 4, 2008 3:40:22 PM CST

Vendor: <No vendor information available> Version: 1.0.0.0

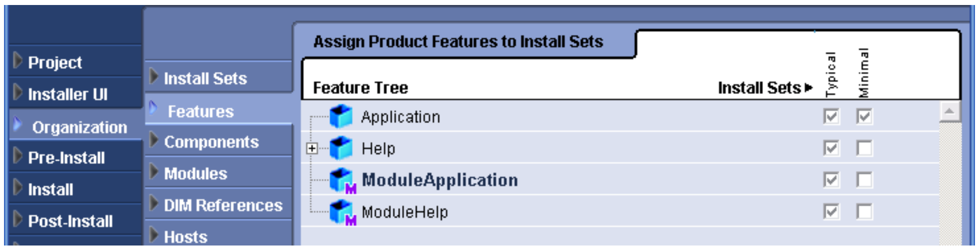
Comments/Notes:

You may use this area to record comments or notes regarding any information you feel is pertinent to the use or maintenance of this Merge Module within your current project

   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.

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.



Feature Tree	Install Sets ▸ Typical	Minimal
Application	✓	✓
Help	✓	
ModuleApplication	✓	
ModuleHelp	✓	

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 Flexera Software 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.

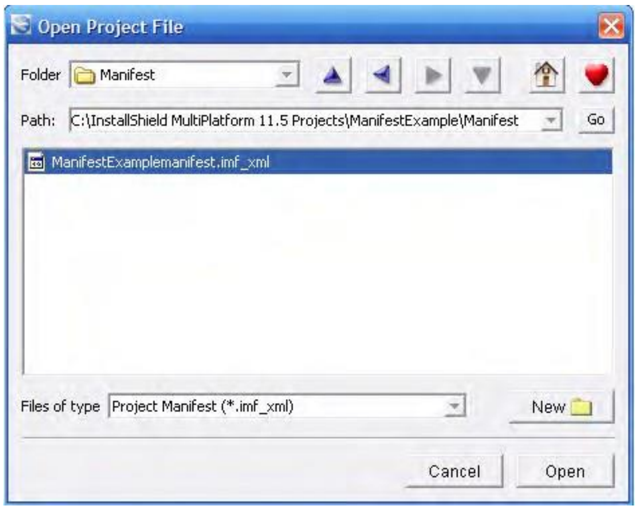
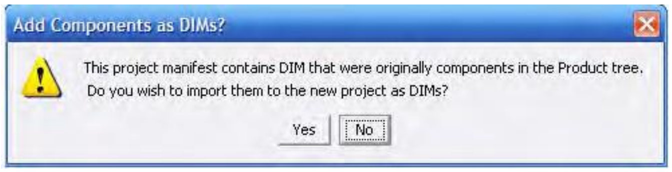

Open Project File
Folder
Manifest
Path:
C:\InstallShield MultiPlatform 11.5 Projects\ManifestExample\Manifest
Go
ManifestExamplemanifest.imf_xml
Files of type
Project Manifest (*.imf_xml)
New
Cancel
Open

Figure 14-5: 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.

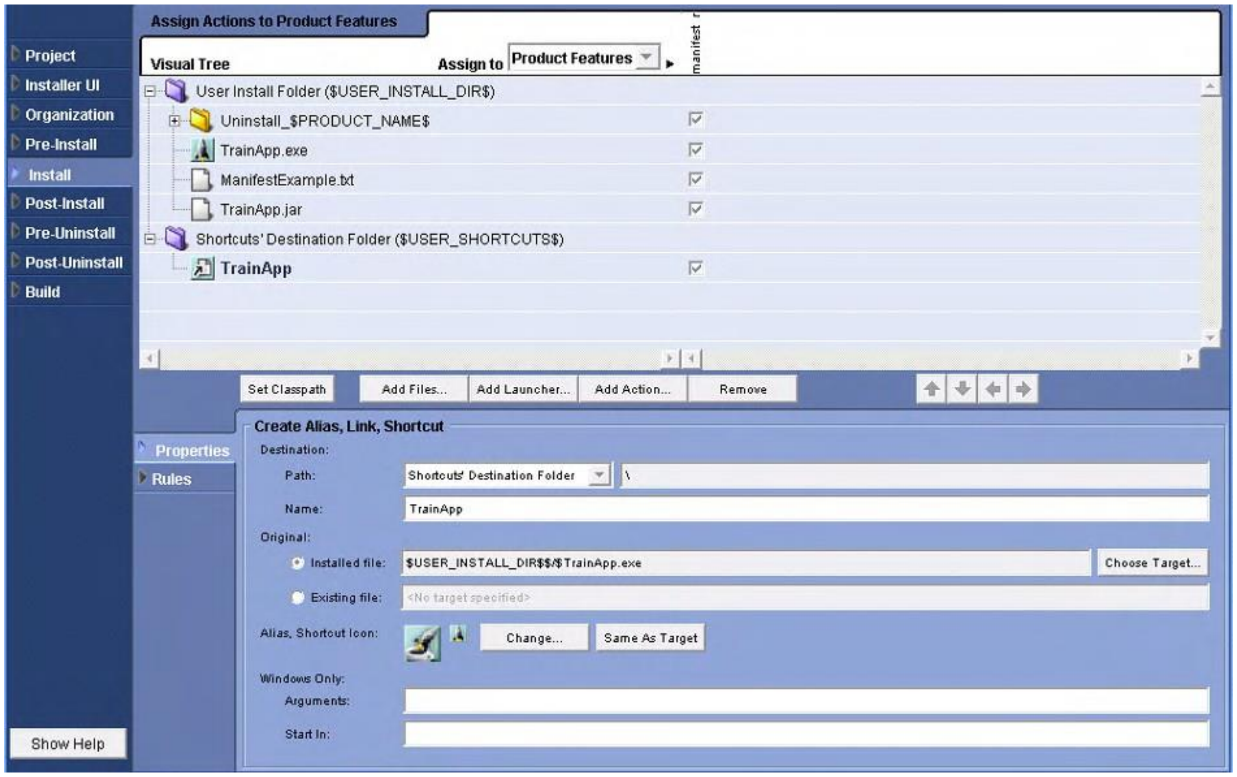

Add Components as DIMs?

This project manifest contains DIM that were originally components in the Product tree.
Do you wish to import them to the new project as DIMs?

Figure 14-6: 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.



Destination:
Path: Shortcuts' Destination Folder \
Name: TrainApp
Original:
Installed file: $USER_INSTALL_DIR$$/$TrainApp.exe
Existing file: <No target specified>
Alias, Shortcut Icon:
Windows Only:
Arguments:
Start In:

Figure 14-7: 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 Chapter 16, Custom Code and Chapter 17, Custom Panels and Consoles.

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 Flexera Software Web site at:

http://www.flexerasoftware.com

Creating a DIM

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

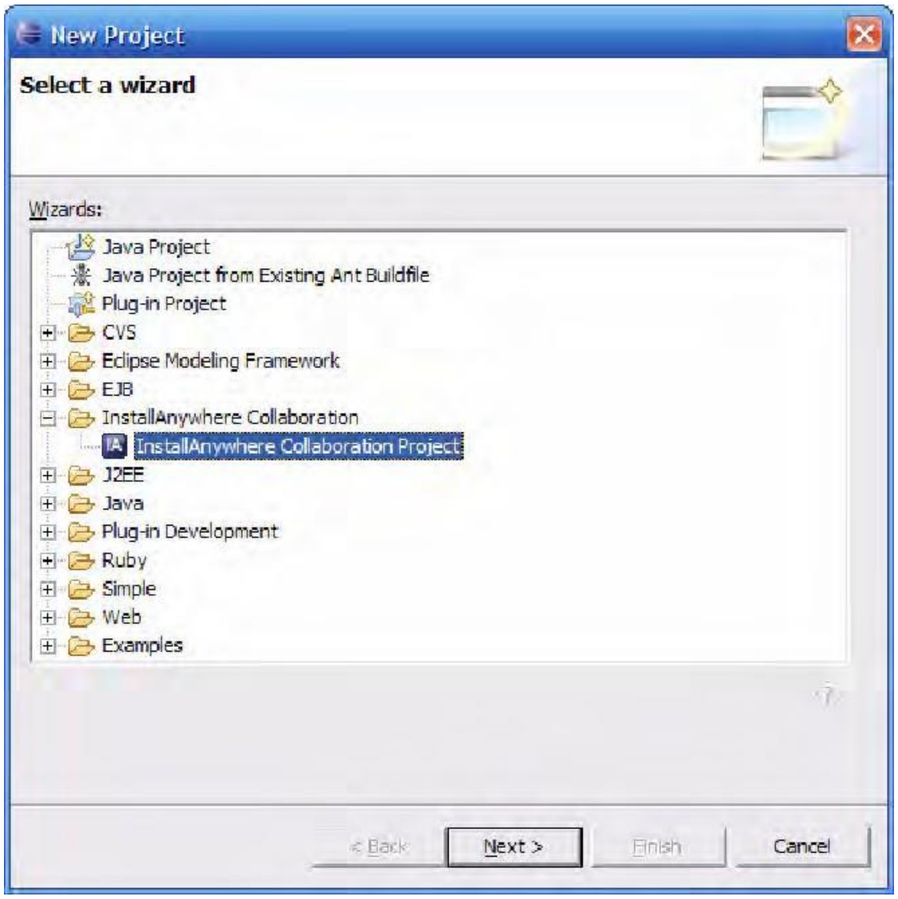
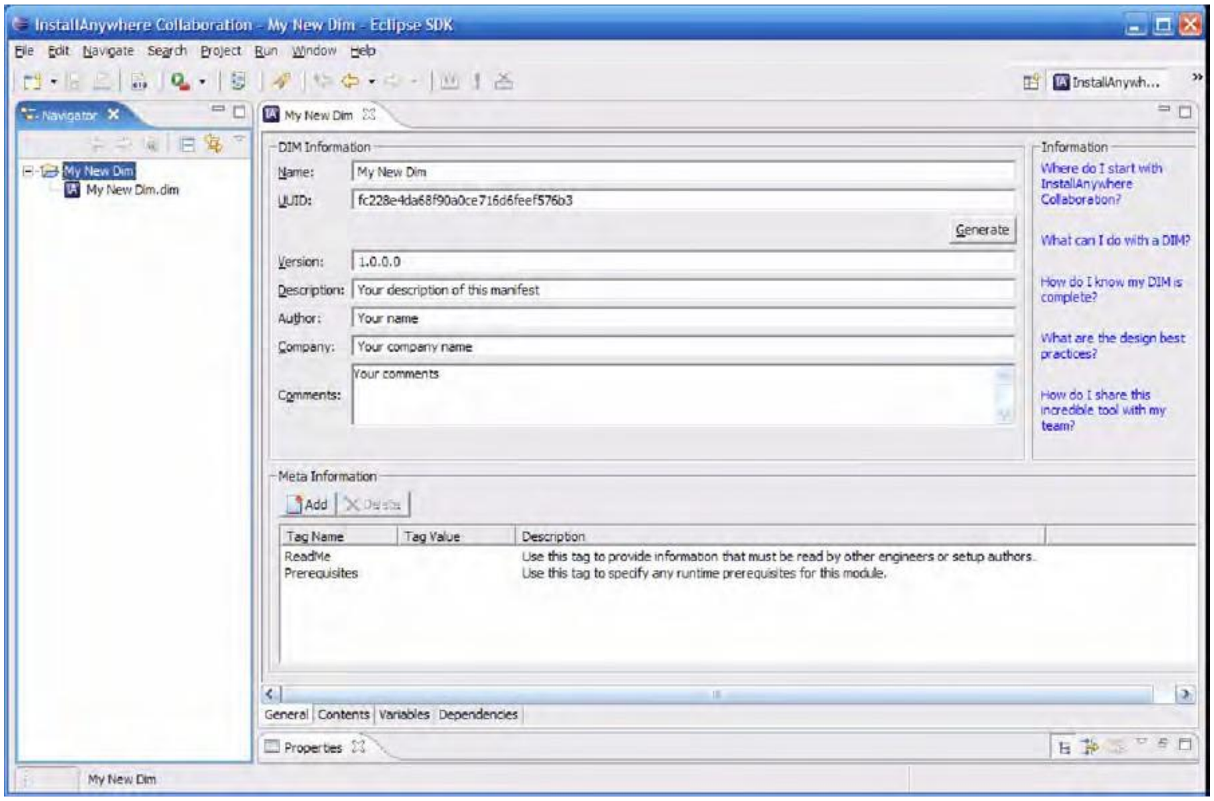


Figure 14-8: New Project Window

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**


InstallAnywhere Collaboration - My New Dim - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Navigator
My New Dim
My New Dim.dim
My New Dim
Generate
Add
Delete
General Contents Variables Dependencies
Properties

DIM Information

Name: My New Dim
UUID: fc228e4da68f90a0ce716d6feef576b3
Version: 1.0.0.0
Description: Your description of this manifest
Author: Your name
Company: Your company name
Comments: Your comments

Information

Where do I start with InstallAnywhere Collaboration?
What can I do with a DIM?
How do I know my DIM is complete?
What are the design best practices?
How do I share this incredible tool with my team?

Meta Information

Tag Name	Tag Value	Description
ReadMe		Use this tag to provide information that must be read by other engineers or setup authors.
Prerequisites		Use this tag to specify any runtime prerequisites for this module.

Figure 14-9: New DIM Tab

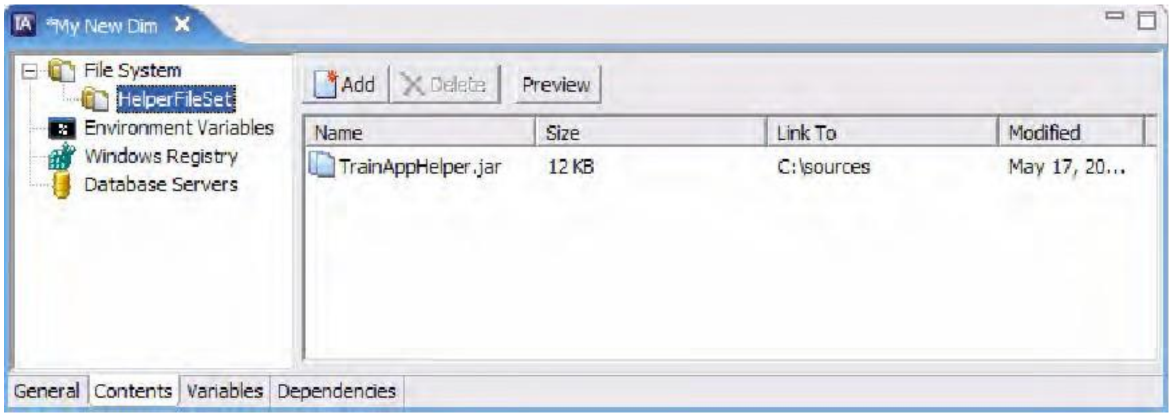
The Eclipse figures displayed here use the InstallAnywhere Collaboration perspective. To specify this perspective, click the **Open Perspective** toolbar button, select **Other**, and select **InstallAnywhere Collaboration**.

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**.

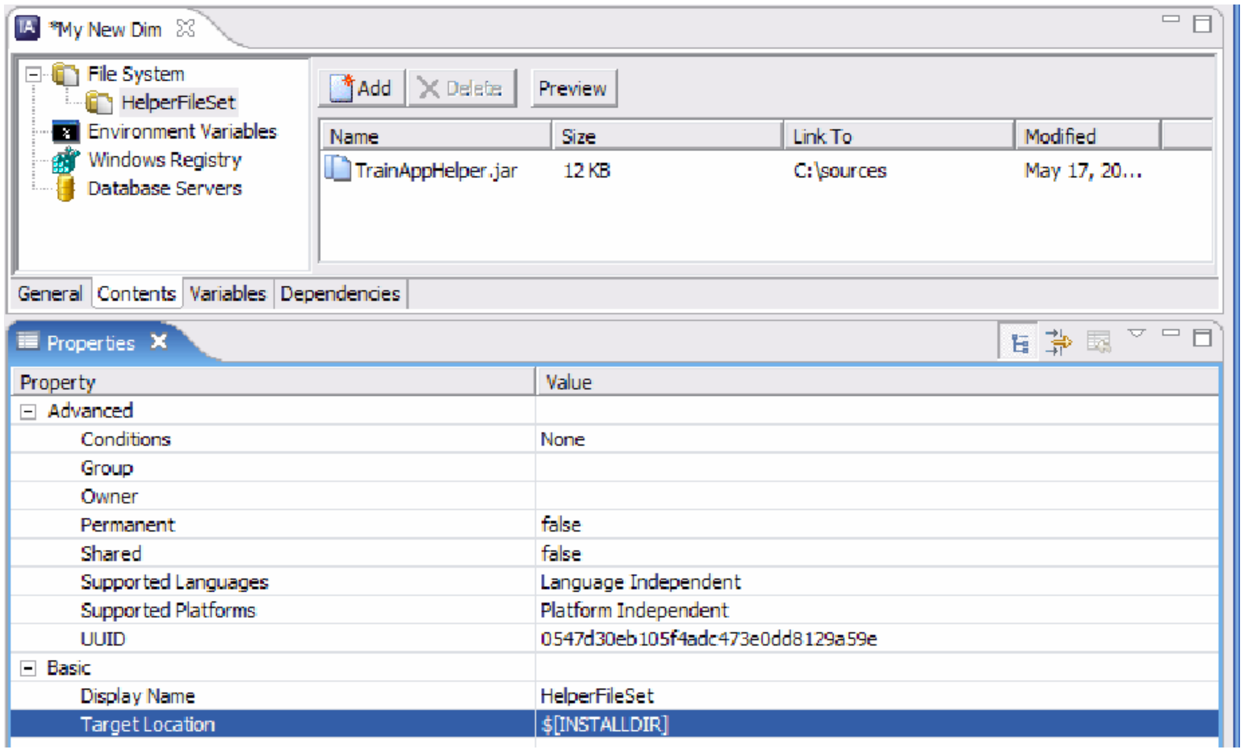


Name	Size	Link To	Modified
TrainAppHelper.jar	12 KB	C:\sources	May 17, 20...

Figure 14-10: 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.



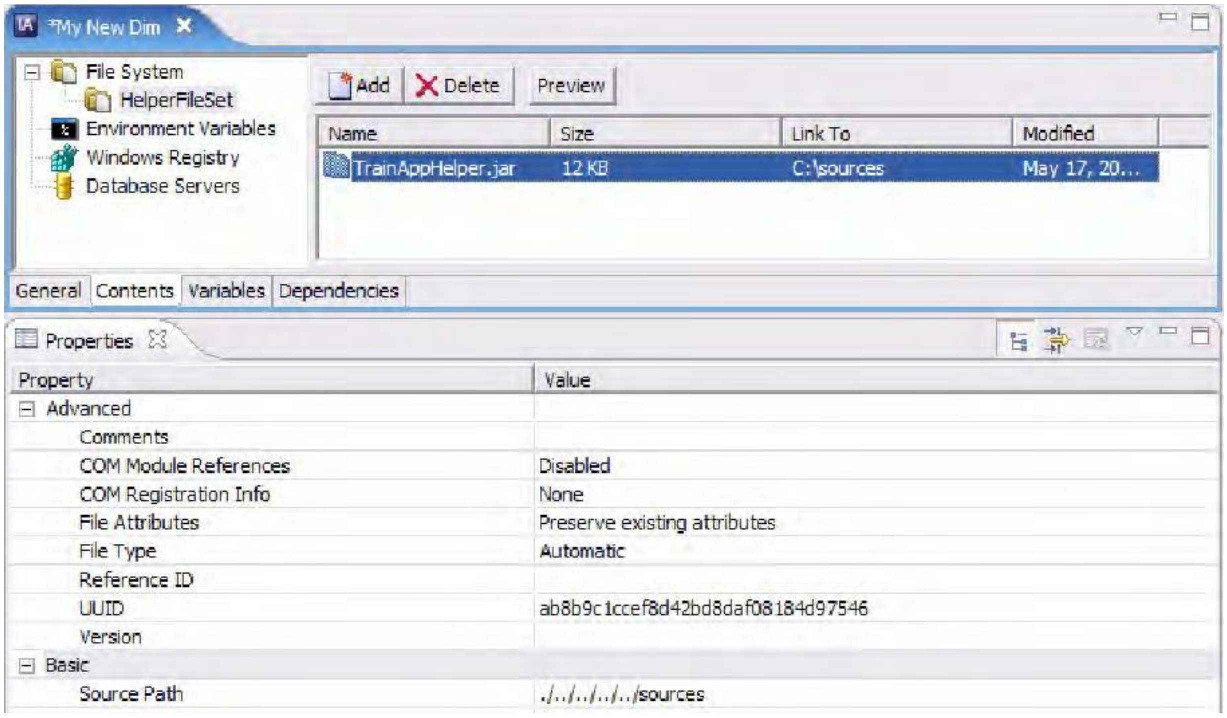
Name	Size	Link To	Modified
TrainAppHelper.jar	12 KB	C:\sources	May 17, 20...

Property	Value
Advanced	
Conditions	None
Group	
Owner	
Permanent	false
Shared	false
Supported Languages	Language Independent
Supported Platforms	Platform Independent
UUID	0547d30eb105f4adc473e0dd8129a59e
Basic	
Display Name	HelperFileSet
Target Location	$[INSTALLDIR]

Figure 14-11: 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.



Name	Size	Link To	Modified
TrainAppHelper.jar	12 KB	C:\sources	May 17, 20...

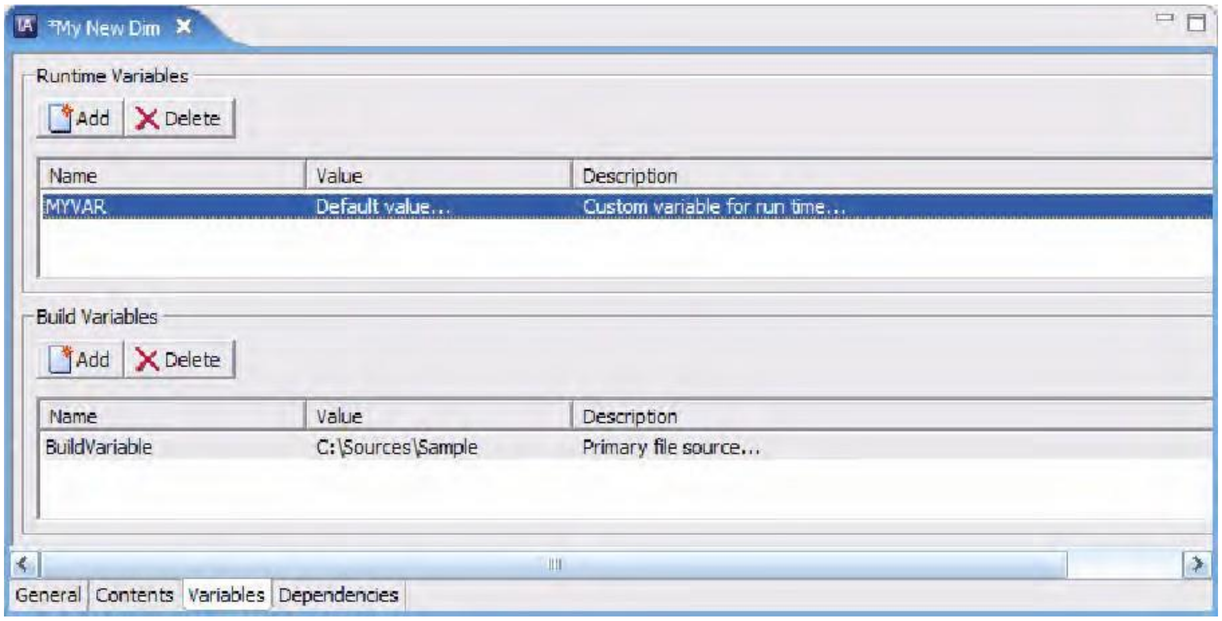
Property	Value
Advanced	
Comments	
COM Module References	Disabled
COM Registration Info	None
File Attributes	Preserve existing attributes
File Type	Automatic
Reference ID	
UUID	ab8b9c1ccef8d42bd8daf08184d97546
Version	
Basic	
Source Path	./../../../../sources

Figure 14-12: 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.



Name	Value	Description
MYVAR	Default value...	Custom variable for run time...

Name	Value	Description
BuildVariable	C:\Sources\Sample	Primary file source...

Figure 14-13: Variables View

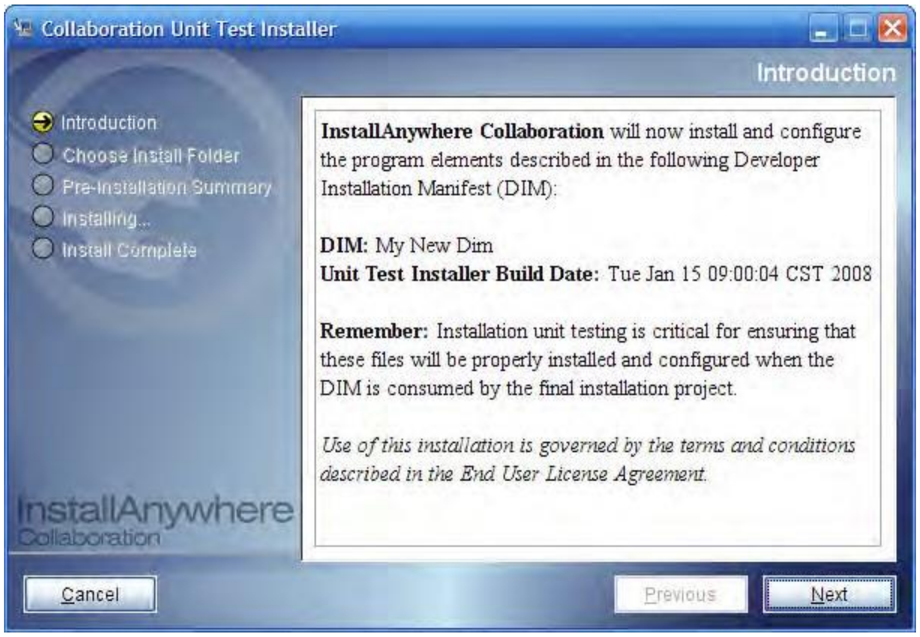
InstallAnywhere Collaboration uses either relative or absolute paths to source files. A recommended procedure is to treat DIMs as source code, meaning that they should be included in your version control system.

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**.



InstallAnywhere Collaboration will now install and configure the program elements described in the following Developer Installation Manifest (DIM):

DIM: My New Dim
Unit Test Installer Build Date: Tue Jan 15 09:00:04 CST 2008

Remember: Installation unit testing is critical for ensuring that these files will be properly installed and configured when the DIM is consumed by the final installation project.

Use of this installation is governed by the terms and conditions described in the End User License Agreement.

Figure 14-14: 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.

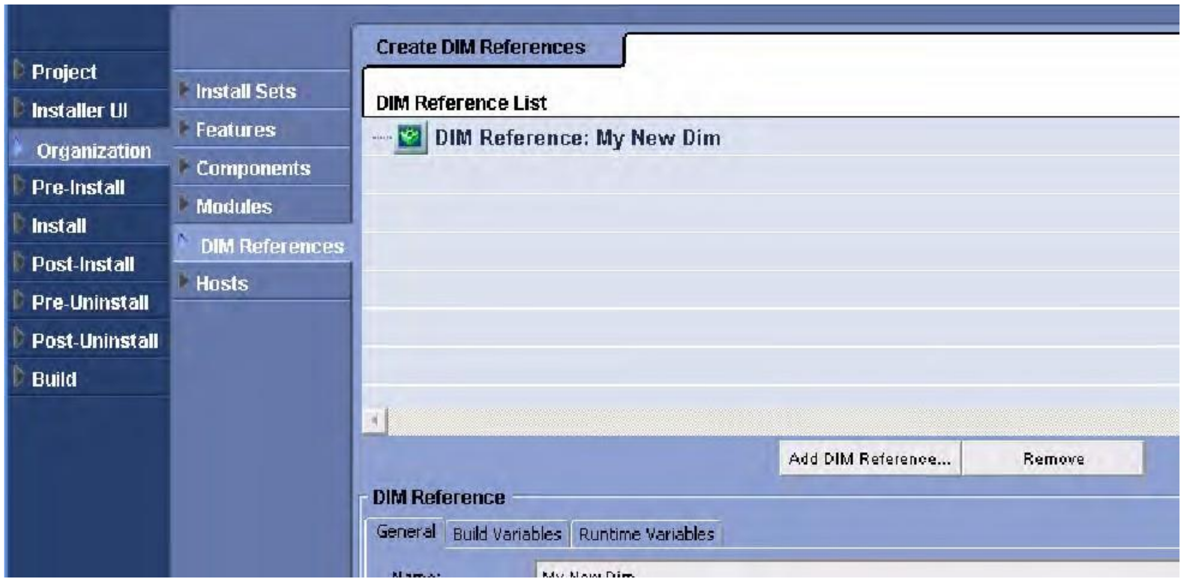
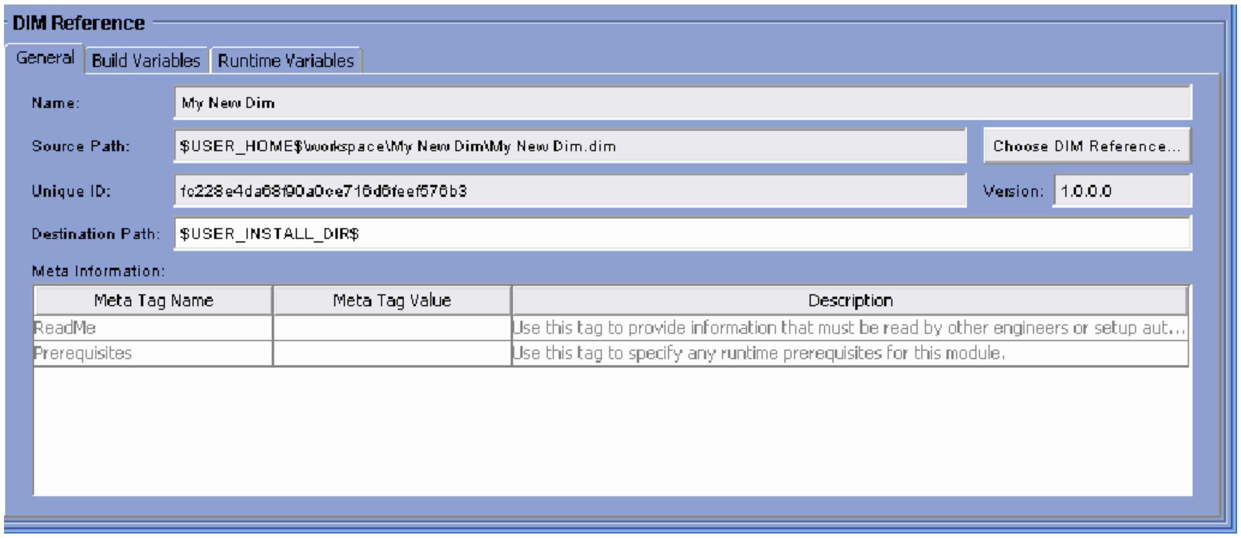


Figure 14-15: DIM References

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.

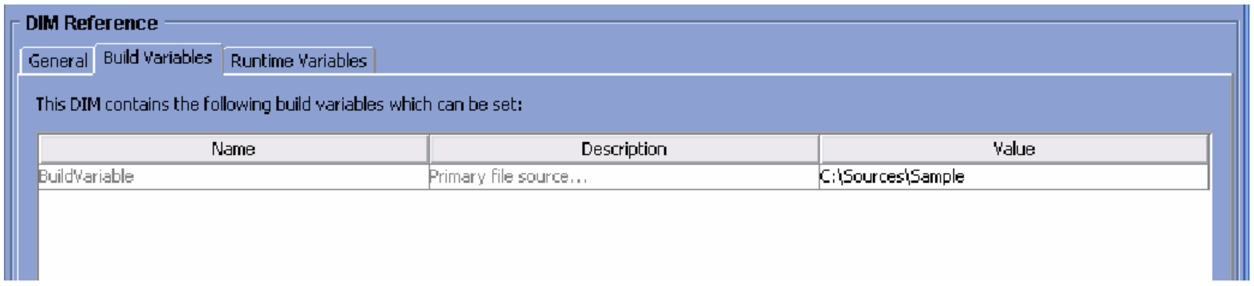


Name: My New Dim
Source Path: $USER_HOME$\workspace\My New Dim\My New Dim.dim
Unique ID: fc228e4da68f90a0ce716d6fef576b3
Version: 1.0.0.0
Destination Path: $USER_INSTALL_DIR$
Meta Information:

Meta Tag Name	Meta Tag Value	Description
ReadMe		Use this tag to provide information that must be read by other engineers or setup aut...
Prerequisites		Use this tag to specify any runtime prerequisites for this module.

Figure 14-16: 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.



This DIM contains the following build variables which can be set:

Name	Description	Value
BuildVariable	Primary file source...	C:\Sources\Sample

Figure 14-17: 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.

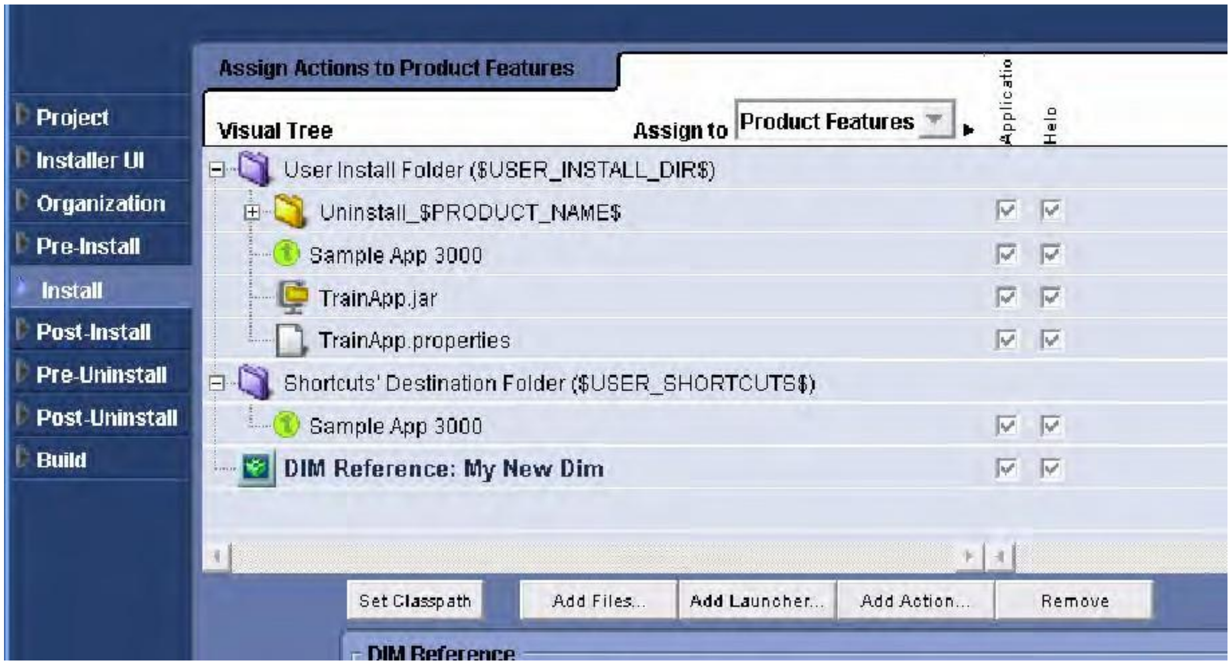


Figure 14-18: Viewing DIMs From the Install Task

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.

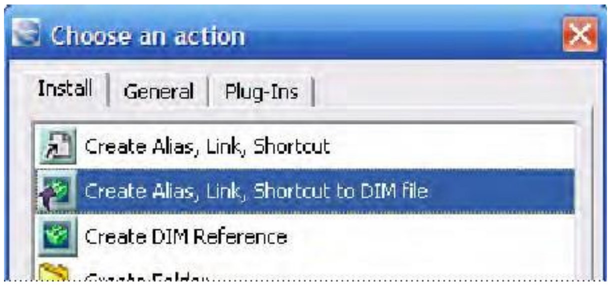

Choose an action
Install
General
Plug-Ins
Create Alias, Link, Shortcut
Create Alias, Link, Shortcut to DIM file
Create DIM Reference

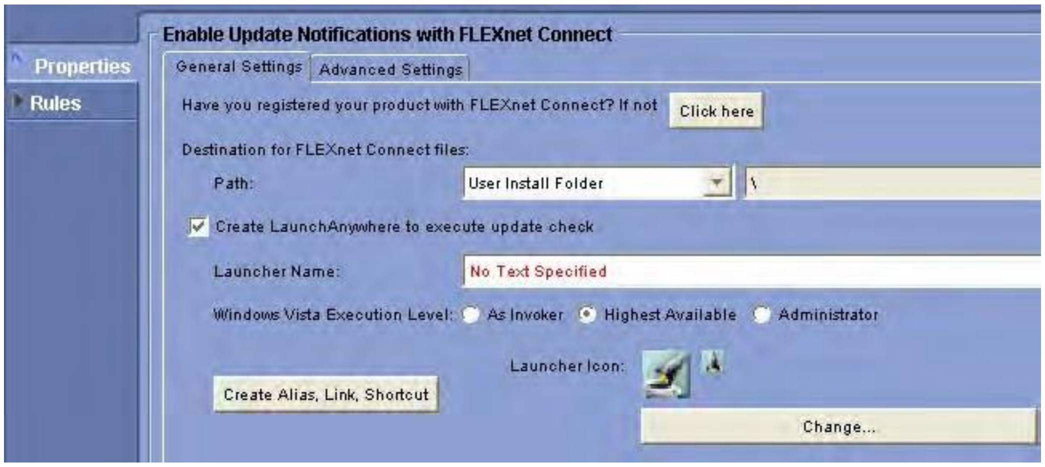
Figure 14-19: Choosing the Shortcut Action

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.



Enable Update Notifications with FLEXnet Connect

Have you registered your product with FLEXnet Connect? If not

Destination for FLEXnet Connect files:

Path: User Install Folder \

[✓] Create LaunchAnywhere to execute update check

Launcher Name: No Text Specified

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

Launcher Icon:

Figure 14-20: 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

Table 14-1 • Answers to Quick Quiz

Question	Answer
1	a
2	a
3	d
4	d

15

Integrating InstallAnywhere with Automated Build Environments

This chapter contains information on:

- InstallAnywhere Command-Line Build Facility
- Advanced Topic: Project Automation
- 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 `build-as-invoker.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 want 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 without opening the graphical environment.

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
- `wv` 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 introduced 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.

Advanced Topic: Project Automation

InstallAnywhere 2009 introduces a Project Automation API, with which you can modify your project using Java code, without requiring use the graphical Advanced Designer. The Project Automation API is provided with the Enterprise Edition of InstallAnywhere.

The methods used in the Project Automation API are contained in the package `com.zerog.ia.auto.project`, and the classes are contained in the file `IAClasses.zip`, which must therefore be available to your Java compiler.

The framework of a simple Java application to open a project file, read some project settings, and then close and save the project file might appear as follows:

```
import com.zerog.ia.auto.project.*;
public class ProjectApiTest
{
    public static void main(String[] args)
    {
        // change to point to your project file
        Project iap = ProjectAutomation.loadProject(
            "C:\\IA2010Projects\\SampleApp3000\\SampleApp3000.iap_xml");
        // access the settings in Project > Description task
        ProjectDescription desc = iap.getProjectDescription( );
        // Version object gives access to version fields
        Version ver = desc.getProductVersion( );
        System.out.println(
            "Product name: " + desc.getProductName( ) +
            "\nVersion: " + ver.getMajor( ) + "." + ver.getMinor( ));
        // save and close the project
        ProjectAutomation.saveProject(iap);
    }
}
```

After compiling the code (with the equivalent of `javac -classpath IAClasses.zip ProjectApiTest`) and running it, you should see the product name and version displayed on the console. To modify information in your project file, the `ProjectDescription` object used in the previous code provides methods such as `setProductName`, `setProductVersion`, and so forth.

Beyond the project-wide settings exposed in the `ProjectDescription` and similar classes, the Project Automation API provides classes that represent your project's hosts (as described in Chapter 7, Introduction to Advanced Actions and Panel Actions): `OSHost` for your project's Operating System host, `DBHost` for your project's Database Server hosts (if any), and `JEEHost` for your project's Application Server hosts (if any). These hosts contain the relevant files and actions delivered by your installer. Most items contained in these hosts have corresponding classes in the Project Automation API.

For Javadoc documentation and further information about the Project Automation API, see the project-auto subdirectory of your InstallAnywhere distribution.

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 though "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`. The 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\InstallAnywhere 2009 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.”

16 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 variables and resource-loading functionality. 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 16-1 • Types of Custom Code

Custom Code Type	Description
Actions	These actions run within InstallAnywhere’s action framework, alongside the default InstallAnywhere actions.
Rules	These rules are evaluated when the action they are associated with is about to be executed. Custom code rules return a Boolean value defining whether the rule succeeds, and therefore whether the associated action should be performed.
Panels	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.
Consoles	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 17, Custom Panels and Consoles.

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.

Information about custom code is presented in the following sections:

- Writing Custom Code
- Custom Code Actions
- Accessing Properties and Variables
- Example: Displaying the Current Time
- Custom Code Rules
- InstallShield MultiPlatform Services
- Advanced Action Methods
- Writing Custom Errors in the Installation Log
- Working with Java Native Interface (JNI)
- Quick Quiz

Writing Custom Code

The steps you follow when using custom code are:

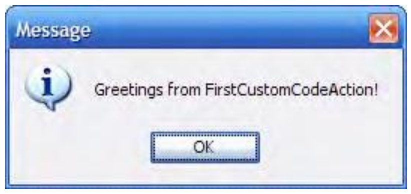
- **Step 1**—Write and compile your custom Java code.
- **Step 2**—Package the custom class and any required resources into a .jar file.
- **Step 3**—Execute the class in your InstallAnywhere project.

Custom Code Actions

Custom Code Actions enable you to create non-graphical actions that can manipulate data on your target system, read or set 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.



Greetings from FirstCustomCodeAction!

Figure 16-1: Custom Code Message Box

Note • *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 in your installer using an Execute Custom Code action, as described in the following section.

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.installanywhere.training.AnotherCustomAction**, you would copy the class file into the subdirectory structure `com/installanywhere/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.

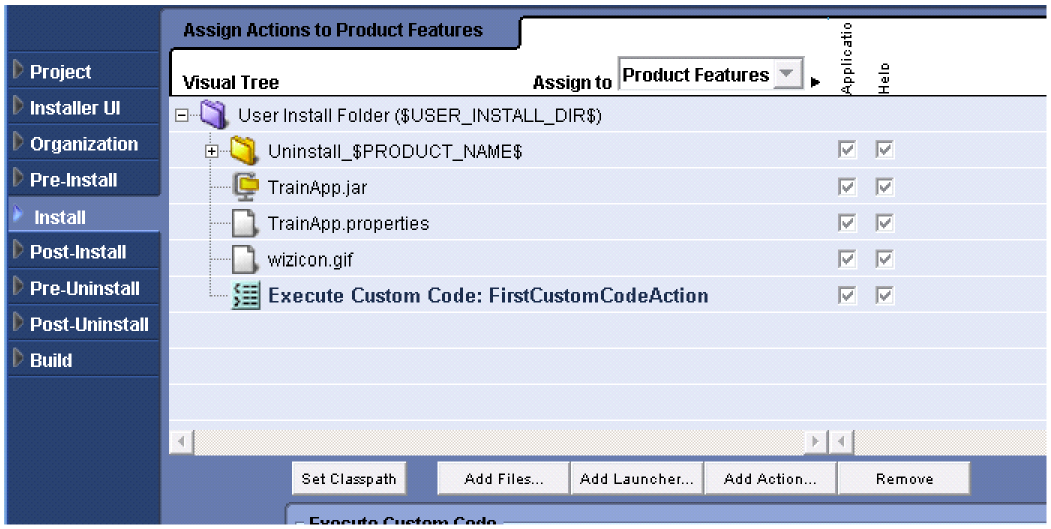


Figure 16-2: Execute Custom Code Action

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.installanywhere.training.ActionName`.

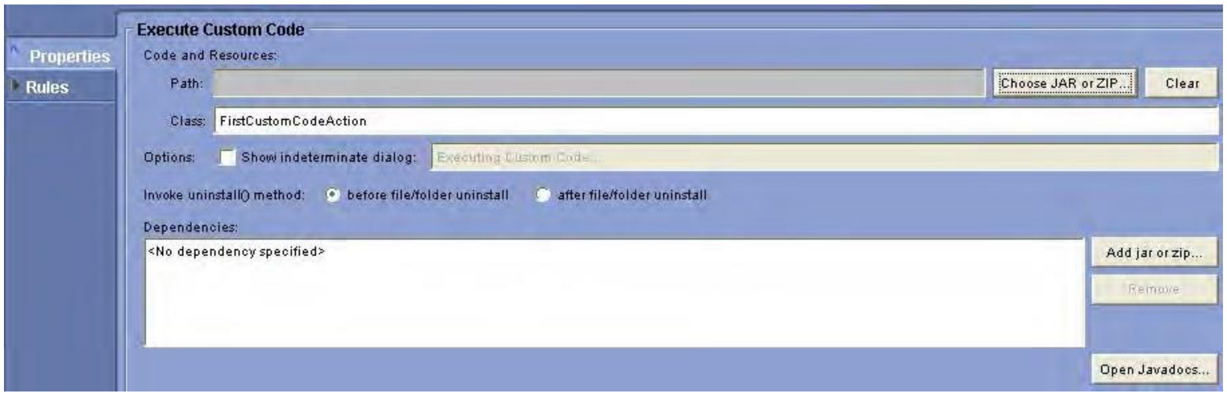
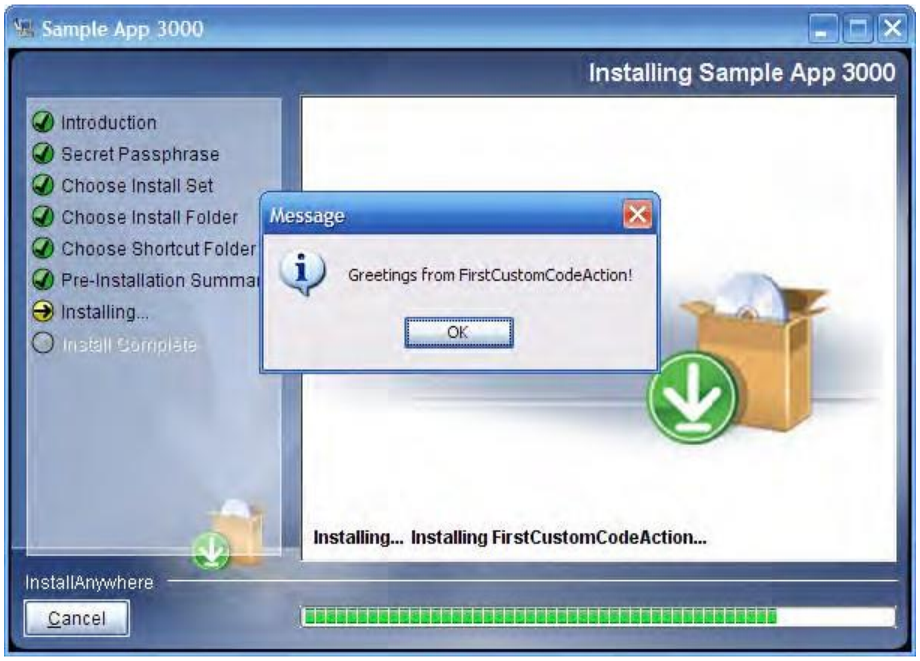

Execute Custom Code
Code and Resources:
Path:
Class: FirstCustomCodeAction
Options: Show indeterminate dialog: Executing Custom Code...
Invoke uninstall() method: [✓ before file/folder uninstall] after file/folder uninstall
Dependencies:
<No dependency specified>

Figure 16-3: 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.



Greetings from FirstCustomCodeAction!

Installing... Installing FirstCustomCodeAction...

Figure 16-4: 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:

Table 16-2 • customCode.properties

Entry	Description
`plugin.main.class=`*`classname`*	The `classname` value should be the full name of your action class, such as `FirstCustomCodeAction` or `com.sampleco.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 | panel | console`	This value should be set to the code's type.

Additional optional entries include:

Table 16-3 • Additional Optional Entries for customCode.properties

Entry	Description
`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 now lists **First Custom Code Action** along with the sample plug-ins that ship with InstallAnywhere:

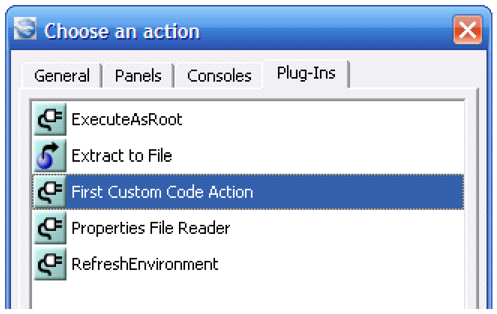
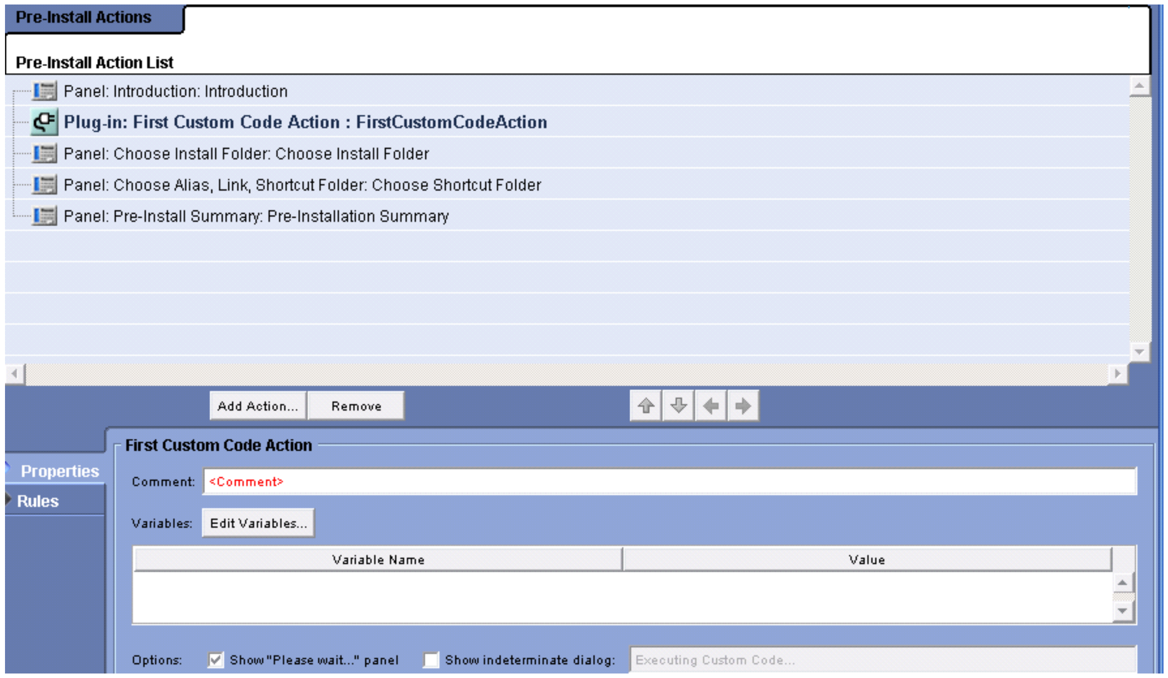


Figure 16-5: Inserting a Custom Code Action as a Plug-in

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



Panel: Introduction: Introduction
Plug-in: First Custom Code Action : FirstCustomCodeAction
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

First Custom Code Action
Comment: <Comment>
Variables:
Options: ✓ Show "Please wait..." panel Show indeterminate dialog: Executing Custom Code...

Figure 16-6: Custom Code Properties

Adding Plug-in 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.

Using Custom Code Actions in Uninstall

The custom code action framework provides a method—`uninstall`—that 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 implement these tasks in a custom code action using 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 the console or command prompt. 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...
```

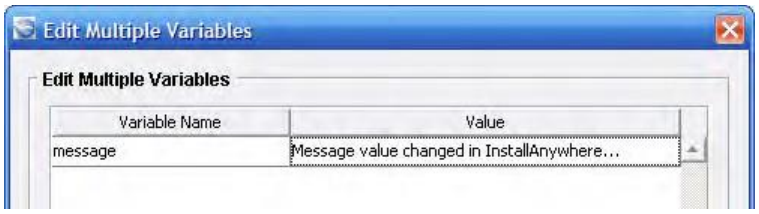
Inside the code, you can use the `substitute` method of the `InstallerProxy` object passed to the `install` method:

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

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.

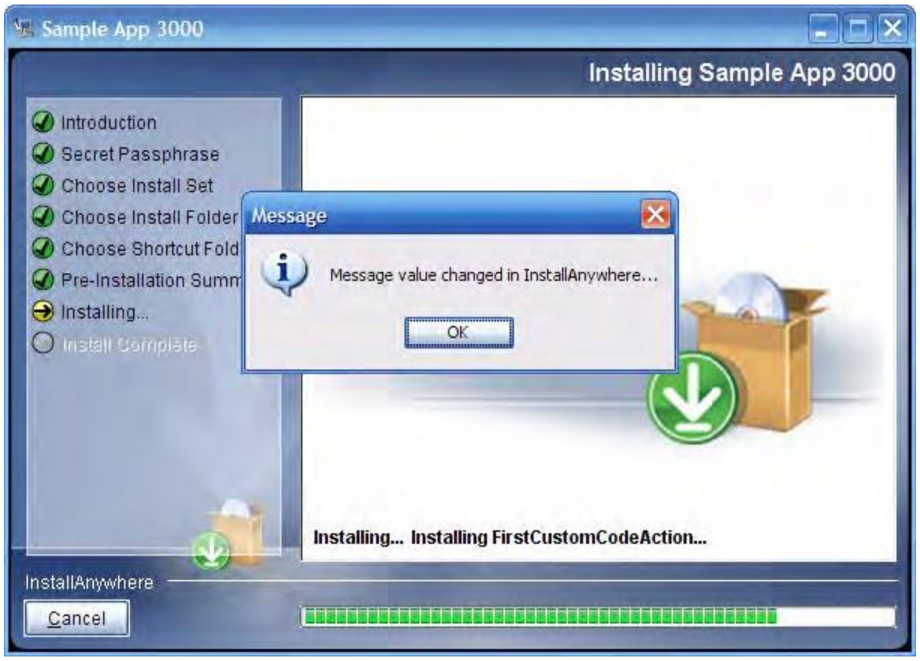
The developer-user of your action can double-click the variable's value to change it.



Variable Name	Value
message	Message value changed in InstallAnywhere...

Figure 16-7: 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:



Message value changed in InstallAnywhere...

Installing... Installing FirstCustomCodeAction...

Figure 16-8: 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 12, Advanced Installer Concepts) or installer log. Beginning with InstallAnywhere 2008 Value Pack 1, you can use the `ReplayVariableService` class 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 registers a custom variable called *$MYVAR$* to be written to the response file when the user runs the command `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 can 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 `substitute` 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, `getVariable` returns `null`.

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("$USER_INSTALL_DIR$")` returns the MagicFolder object for the current install location, while `substitute("$USER_INSTALL_DIR$")` returns 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.

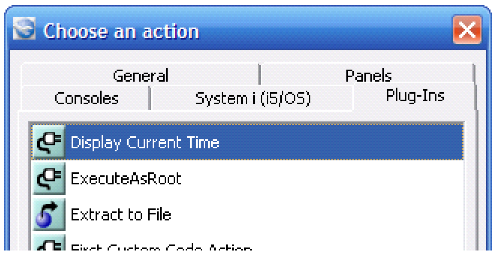


Figure 16-9: Adding the DisplayTime Plug-in

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

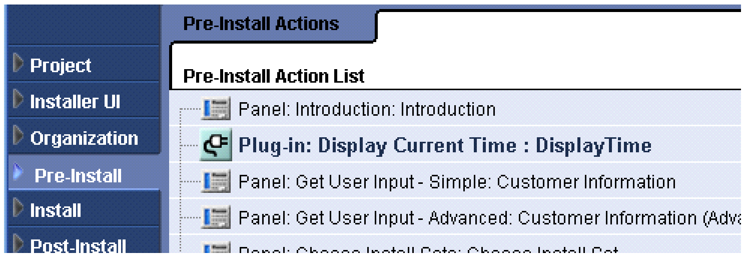

Pre-Install Actions
Project
Installer UI
Organization
Pre-Install
Install
Post-Install
Pre-Install Action List
Panel: Introduction: Introduction
Plug-in: Display Current Time : DisplayTime
Panel: Get User Input - Simple: Customer Information
Panel: Get User Input - Advanced: Customer Information (Adva

Figure 16-10: DisplayTime as a Plug-in

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

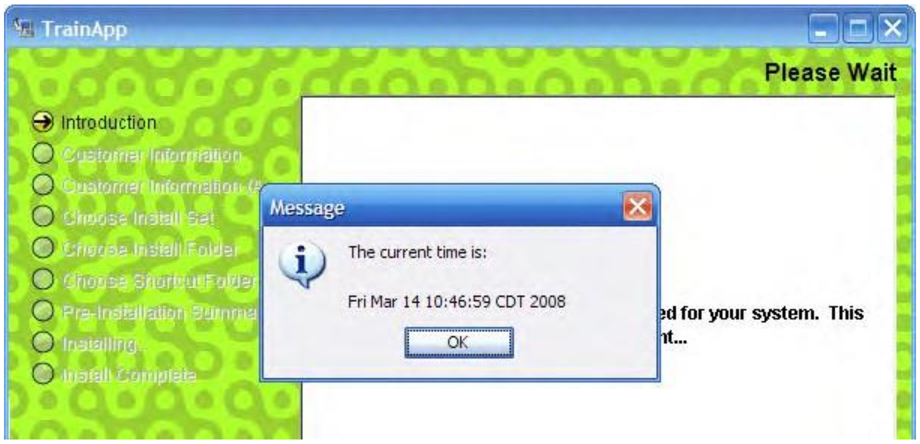


Figure 16-11: DisplayTime at Run Time

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.

For another example, suppose you want to ensure the target system screen resolution is above a given minimum. A custom code rule that succeeds if the target system's resolution is at least 1024 by 768 might appear as follows.

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

public class CheckScreenDimensionsRule extends CustomCodeRule
{
    private static final int MIN_WIDTH = 1024;
    private static final int MIN_HEIGHT = 768;

    public boolean evaluateRule( )
    {
        Toolkit tk = Toolkit.getDefaultToolkit( );
        Dimension d = tk.getScreenSize( );

        // fail if either dimension is below minimum
        if ((d.width < MIN_WIDTH) || (d.height < MIN_HEIGHT))
            return false;
        else
            return true;
    }
}
```

As before, you compile the class and package it in a `.jar` or `.zip` file. You can then use the condition in a project by selecting an action and clicking the **Rules** tab in its customizer. When you click **Add Rule**, select **Evaluate Custom Rule** from the **Choose a rule** panel.

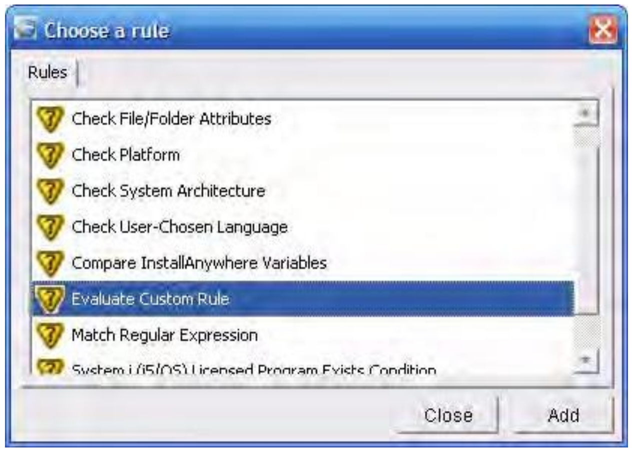
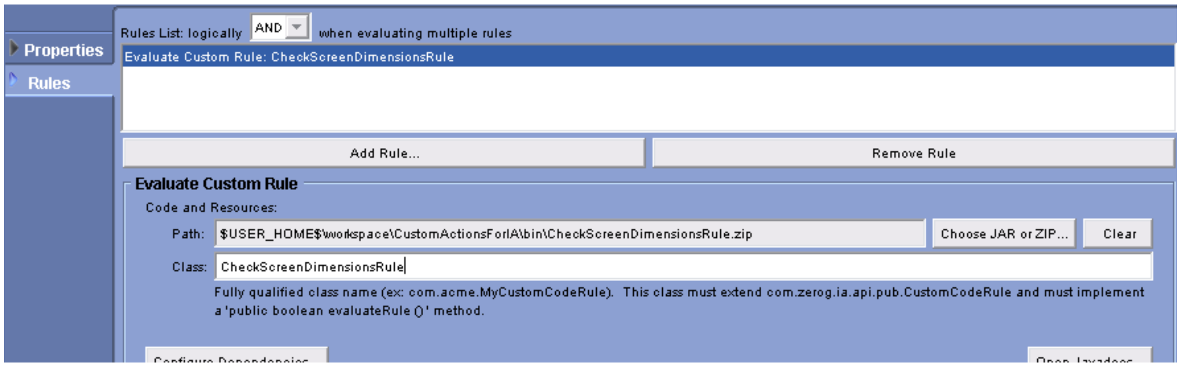

Choose a rule
Rules
Check File/Folder Attributes
Check Platform
Check System Architecture
Check User-Chosen Language
Compare InstallAnywhere Variables
Evaluate Custom Rule
Match Regular Expression
System i (i5/OS) Licensed Program Exists Condition
Close
Add

Figure 16-12: Adding a Custom Rule

In the **Evaluate Custom Rule** customizer, browse for the `.jar` or `.zip` file and enter the custom rule class name.



Rules List: logically AND when evaluating multiple rules

Evaluate Custom Rule: CheckScreenDimensionsRule

Evaluate Custom Rule

Code and Resources:

Path: $USER_HOME$\workspace\CustomActionsForIA\bin\CheckScreenDimensionsRule.zip

Class: CheckScreenDimensionsRule

Fully qualified class name (ex: com.acme.MyCustomCodeRule). This class must extend com.zerog.ia.api.pub.CustomCodeRule and must implement a 'public boolean evaluateRule ()' method.

Figure 16-13: Custom Rule Properties

After you build and run the installer, the action containing the custom code rule will be skipped if the target system does not meet the minimum screen resolution.

The `CustomCodeRule` class provides the proxy variable `ruleProxy` (of type `CustomCodeRuleProxy`). With `ruleProxy` you can obtain the values of InstallAnywhere variables with the same methods `getVariable` and `substitute` available to custom code actions. For example, a refinement for the previous rule might be to enable the rule to automatically succeed if the installer is running in silent mode or console mode, using the `$INSTALLER_UI$` variable.

Reading the InstallAnywhere Product Registry

As described in Chapter 3, Introduction to the Advanced Designer, by default InstallAnywhere writes information about every installed product, feature, and component into the global InstallAnywhere product registry. Starting with InstallAnywhere 2009, you can use the `ProductRegistryService` interface in your custom code to read information about products previously installed using InstallAnywhere. For example, you can use methods from `ProductRegistryService` in a custom code rule to detect if a particular earlier version of your product is installed on a target system.

You must import the package `com.zerog.ia.api.pub.registry` in your custom code to use methods from `ProductRegistryService`.

For example, the following custom code action displays summary information—product name, version, and install location—for every product in the InstallAnywhere product registry.

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

public class ProductRegistryTest extends CustomCodeAction
{
    public void install(InstallerProxy ip) throws InstallException
    {
        // get a handle to the product registry service
        ProductRegistryService prs =
            (ProductRegistryService)ip.getService(ProductRegistryService.class);

        // specify which objects to return: the search criteria can include specific names,
        // versions, locations, or unique IDs, either as strings or as Java regular expressions
        SoftwareObjectSearchCriteria search = new SoftwareObjectSearchCriteria( );

        // return array of products matching the search criteria
```

```
            Product[] products = prs.getProducts(search);

            // for each product, display the summary information
            for (int i = 0; i < products.length; i++)
            {
                javax.swing.JOptionPane.showMessageDialog(null,
                    products[i].getName( ) + " " + products[i].getVersion( ) + ": " +
                    products[i].getLocation( ));
            }
        }

        public String getInstallStatusMessage( ) { return "Querying IA registry..."; }
        public String getUninstallStatusMessage( ) { return "..."; }
        public void uninstall(UninstallerProxy up) throws InstallException { /* do nothing */ }
    }
```

After compiling and packaging the custom code action and inserting it into one of your installation tasks, the custom code displays information such as:

```
FLEXnet Operations 12.0.0.0: C:\Program Files\FLEXnet\publisher\ops

InstallAnywhere 2010 Enterprise 11.0.0.0: C:\Program Files\InstallAnywhere 2010 Enterprise
```

In addition to providing the `getProducts` method used in this example, the product registry service provides the following methods:

- `getProductsByVendor`, returning all products from a particular vendor identified by the Vendor ID value in the **Project > Description** task
- `getProductsWithComponent`, returning all products containing a particular component with a given Unique ID setting
- `getComponents`, returning all components matching the given search criteria (as described in Chapter 6, in some cases you can set up component dependencies or use the Find Component in Registry action to detect a given component)

The `Product` object returned by the various `getProducts` methods gives access to that product’s feature information.

For more information about the `ProductRegistryService`, `SoftwareObjectSearchCriteria`, and other related classes, see the Javadoc help for `ProductRegistryService`.

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. Support for these services was introduced in InstallAnywhere 8.

The following ISMP services are available:

Table 16-4 • Available ISMP Services

Service	Description
FileService	Manipulates files and directories and their privileges.
SecurityService	Manipulates system users and groups.
SystemUtilService	Works with environment variables, reboots, and startup commands.
Win32RegistryService	Queries and 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
```

(The Windows-only interfaces `Win32RegistryService` and `Win32Service` additionally require `resource\services\ppk\windowsppk.jar` on the compiler class path.) 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.

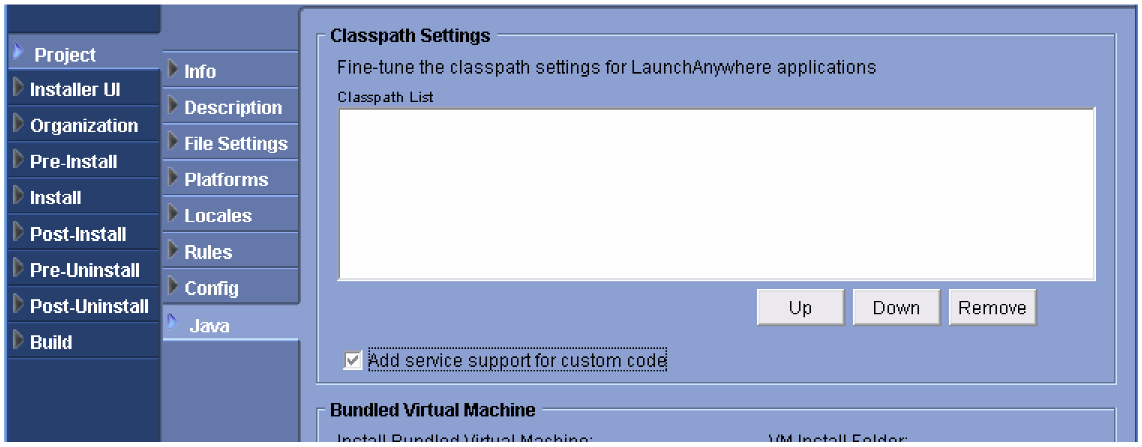


Figure 16-14: Adding Service Support for Custom Code

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: Service Differences Between 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 17, Custom Panels and Consoles); the `InstallerResources` class, which provides access to some resources and data inside an installer; the `ReplayVariableService` class, 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

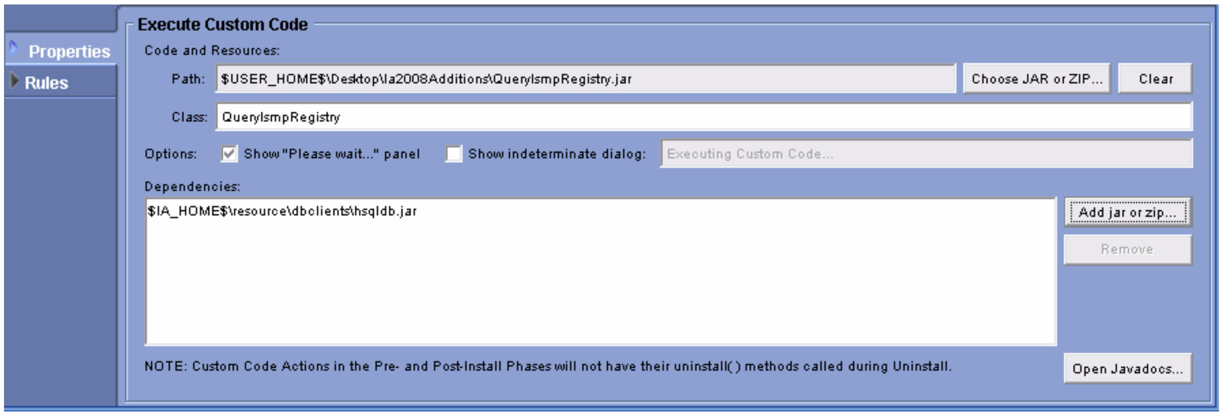
Similar to InstallAnywhere’s product registry is the InstallShield MultiPlatform product registry, which keeps track of the ISMP products, features, and components installed on a target system. To enable you to query the ISMP information—to determine if a product installed with ISMP is present or where 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.



Code and Resources:
Path: $USER_HOME$\Desktop\Ia2008Additions\QueryIsmpRegistry.jar
Class: QueryIsmpRegistry
Options: ✓ Show "Please wait..." panel ☐ Show indeterminate dialog: Executing Custom Code...
Dependencies:
IA_HOME\resource\dbclients\hsqldb.jar
NOTE: Custom Code Actions in the Pre- and Post-Install Phases will not have their uninstall( ) methods called during Uninstall.

Figure 16-15: 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 be displayed only if the installer or uninstaller is run in debug mode.

Refer to Chapter 10, Applying Basic and Intermediate Development Concepts, for more information on working with debug mode.

Tip • *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.

Table 16-5 • Advanced Action Methods

Method	Description
`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`.

Table 16-5 • Advanced Action Methods (cont.)

Method	Description
`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. Calling `setJavaVMList` enables the development of actions that can validate version information and so forth about a system VM. Setting the VM list prior to the display of the Choose Java VM panel will change the VMs listed in the Choose Java VM panel, as well as the values returned by `getJavaVMList`.

Writing Custom Errors in the Installation Log

The Custom Error class provides access to the InstallAnywhere error logging and end-user installation log features to Custom Actions, Panels, and Consoles.

Given an instance of `InstallerProxy`, `CustomCodePanelProxy`, or other proxy object called proxy, the following code enables you to write information to the installation log.

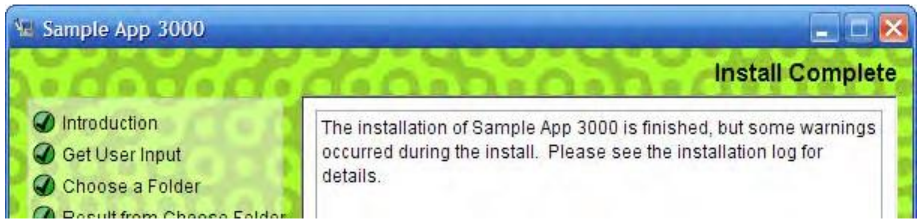
```
CustomError error = (CustomError)proxy.getService(CustomError.class);

// short description of action
error.setLogDescription("FirstCustomCodeAction");

// explain the error or warning; constants are WARNING, ERROR, and FATAL_ERROR
error.appendError(
    "Action failed for lack of ZZZ resources.",
    CustomError.WARNING);

// string indicating how user can avoid error or warning
error.setRemedialText("To avoid this warning, free some ZZZ resources.");

// required to add information to install log
error.log( );
```

The code above logs the following text in the installation log:

```
Installation: Successful

49 Successes
1 Warnings
```

```
0 NonFatalErrors
0 FatalErrors

Action Notes:

FirstCustomCodeAction: To avoid this warning, free some ZZZ resources.

Install Log Detail:

FirstCustomCodeAction
    Status: WARNING
    Additional Notes: WARNING - Action failed for lack of ZZZ resources.
```

Moreover, the warning causes the Install Complete panel to display the following text.



The installation of Sample App 3000 is finished, but some warnings occurred during the install. Please see the installation log for details.

Figure 16-16: Warning Notification in Install Complete Panel

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.

Task ***To call JNI code:***

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.

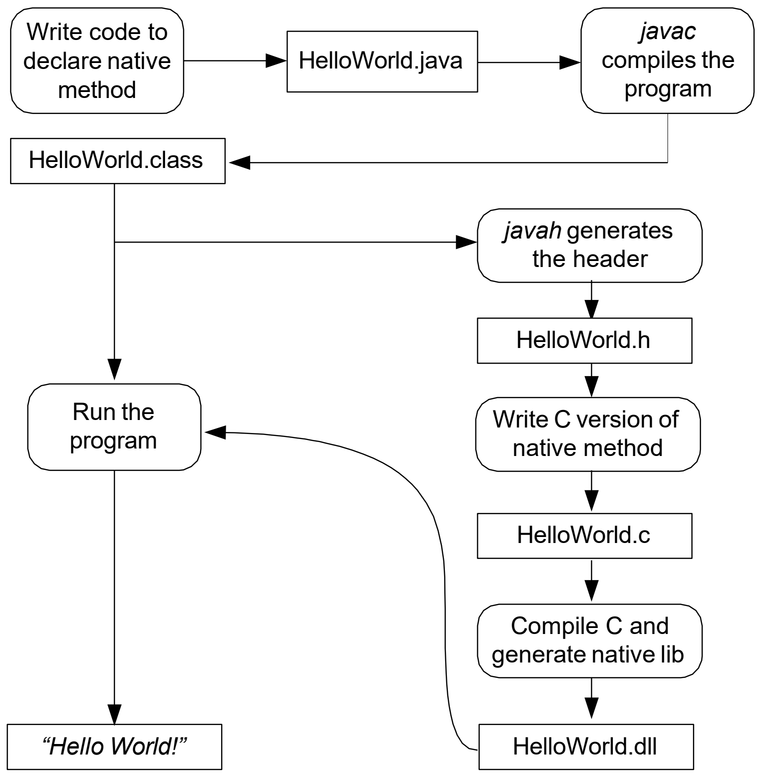

Write code to declare native method
HelloWorld.java
javac compiles the program
HelloWorld.class
javah generates the header
HelloWorld.h
Run the program
Write C version of native method
HelloWorld.c
Compile C and generate native lib
"Hello World!"
HelloWorld.dll

Figure 16-17: 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 the command `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 text 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 `javah`. From the command line, enter:

```
javah -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);

#ifdef __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
```

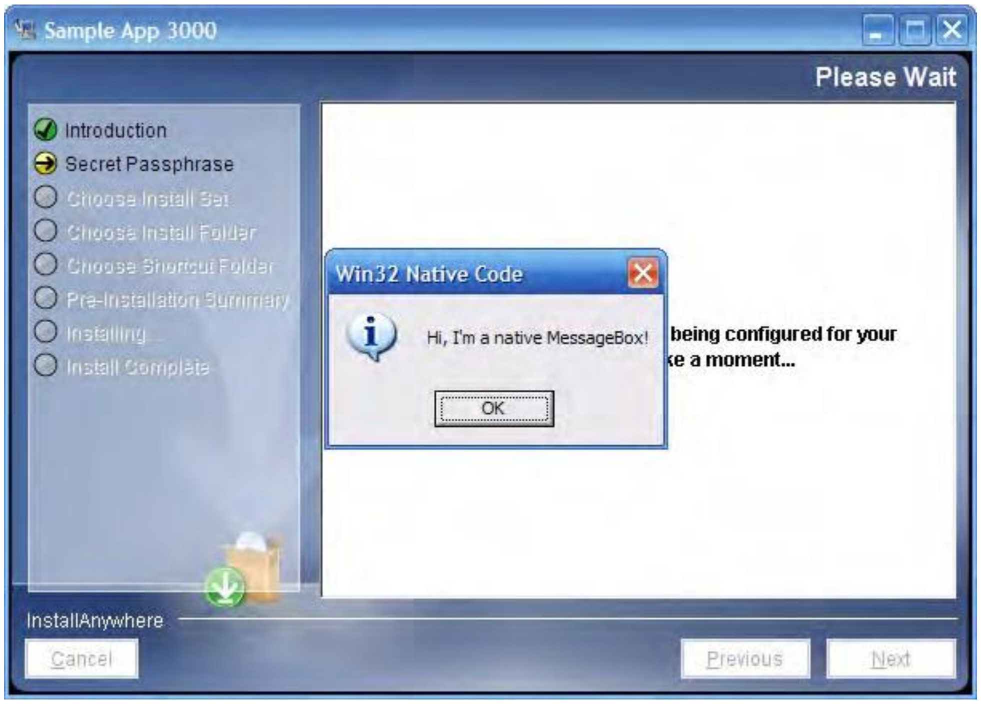
Accessing the Library at Run Time

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.



Hi, I'm a native MessageBox!

being configured for your
ke a moment...

Figure 16-18: 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 disadvantages 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`

Table 16-6 • Answers to Quick Quiz

Question	Answer
1	b
2	a
3	a

17

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
- Custom Code Panels
- Custom Code Consoles
- Quick Quiz

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.

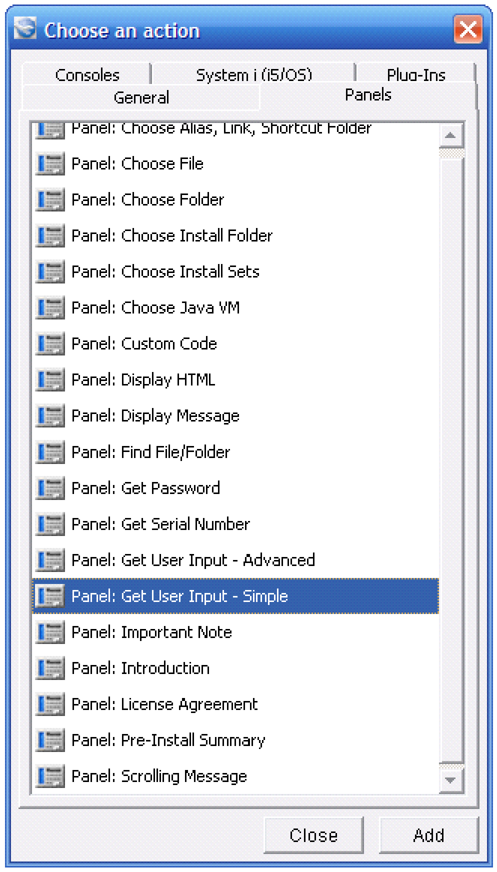


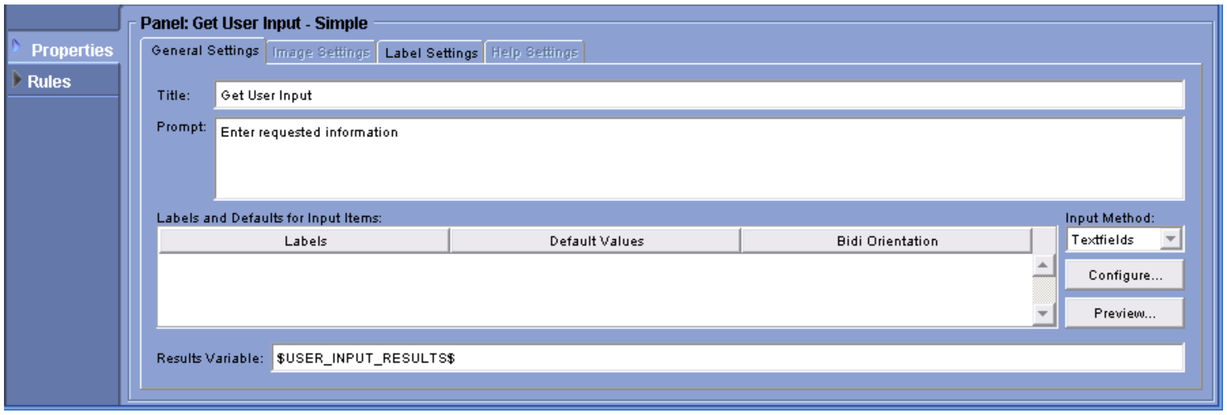
Figure 17-1: Adding a Get User Input Panel

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.



Title: Get User Input
Prompt: Enter requested information
Labels and Defaults for Input Items:

Labels	Default Values	Bidi Orientation

Input Method: Textfields
Results Variable: $USER_INPUT_RESULTS$

Figure 17-2: 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 open the **Configure Input Items** dialog box and begin adding controls.

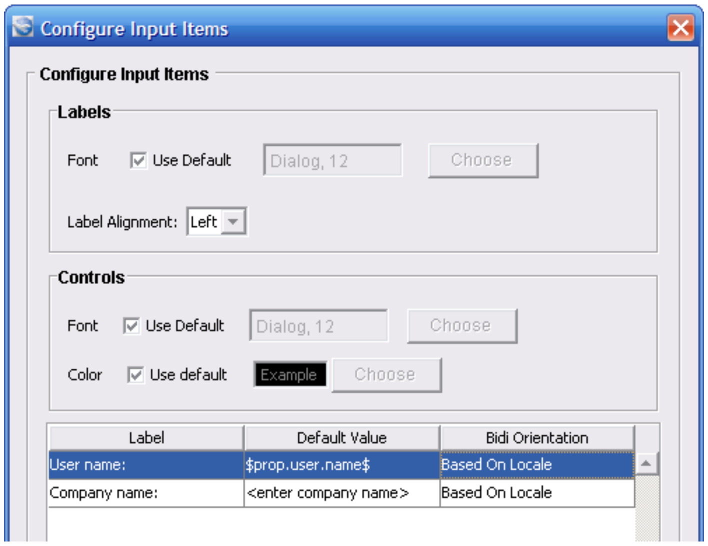
In the **Configure Input Items** dialog box, click **Add** to add a user-input field (a text field, in this example) label and the default value to display.

Note • *On the **Configure Input Items** dialog box, you can also specify the control's orientation when running in a bidirectional locale; localization issues are discussed in Chapter 18, Localizing and Internationalizing Installers.)*

For this example, two input items are used:

Table 17-1 •

Label	Default Value
User name:	`$prop.user.name$` ***Note*** • *The variable expands to the value of the Java system property user.name.*
Company name:	`<enter company name>`

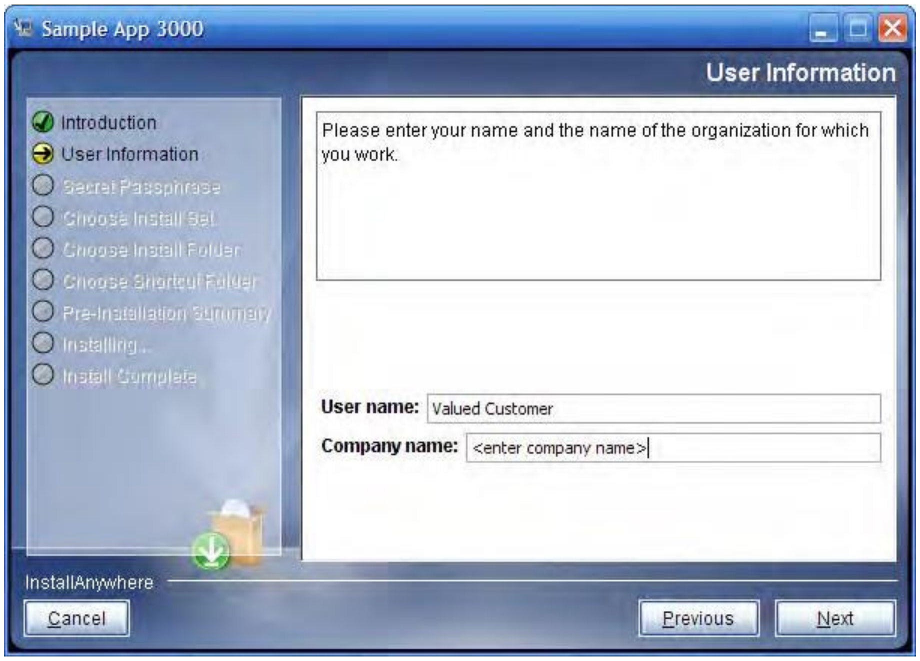


Label	Default Value	Bidi Orientation
User name:	$prop.user.name$	Based On Locale
Company name:	<enter company name>	Based On Locale

Figure 17-3: Configuring Text Fields

You can also specify fonts and colors for labels and controls displayed on the User Input panel.

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:



Please enter your name and the name of the organization for which you work.

User name: Valued Customer

Company name: <enter company name>

Figure 17-4: Simple User Input Panel at 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 `$CUSTOMER_INFO_RESULTS$`.

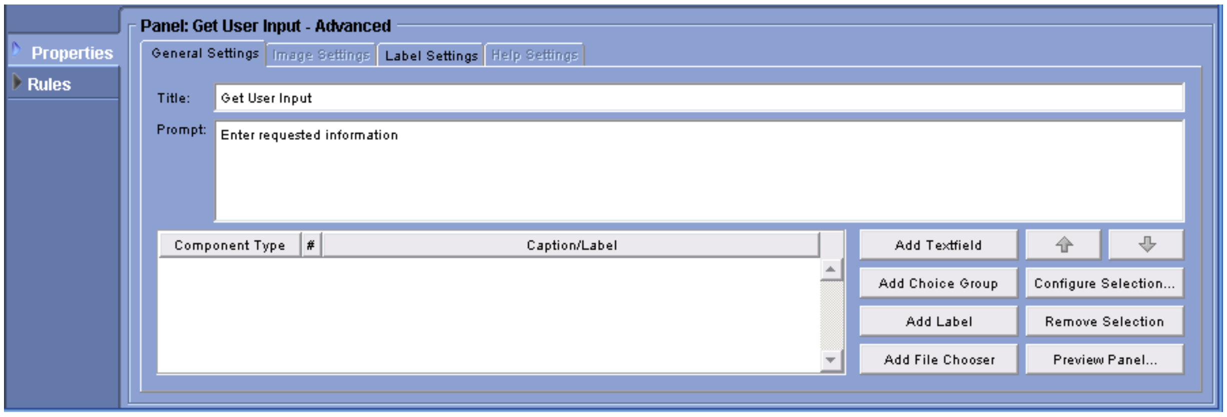
After the panel is displayed, the specified variable (`$CUSTOMER_INFO_RESULTS$`) contains a comma-separated sequence of all of the user input, while the contents of individual fields are stored in numbered variations of the variable. For this example, the contents of the first (User name) field are stored 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 "Get User Input Panels" topic in the InstallAnywhere Help Library 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.



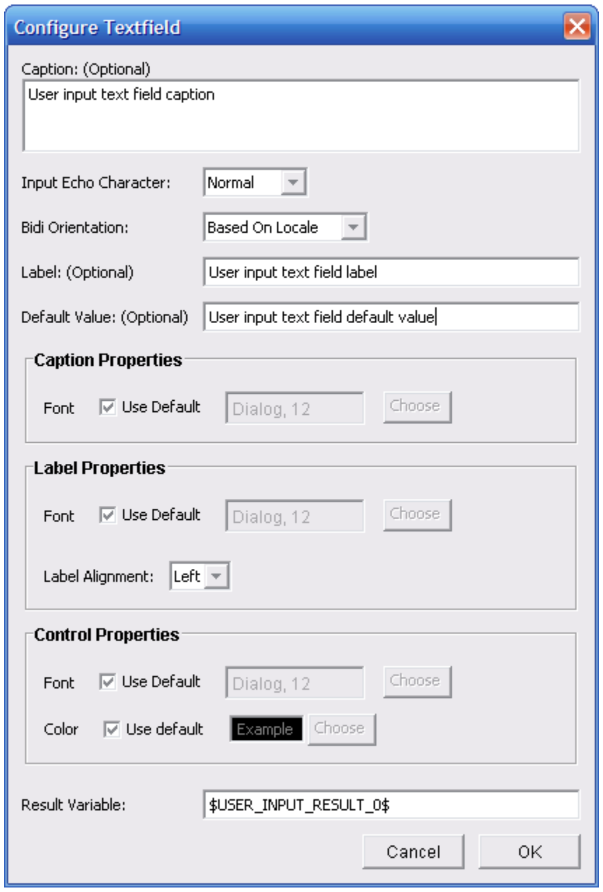
Title: Get User Input

Prompt: Enter requested information

Figure 17-5: 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. InstallAnywhere 2009 adds the options for specifying fonts and colors for items displayed on your User Input panels.


Configure Textfield

Caption: (Optional)
User input text field caption

Input Echo Character: Normal
Bidi Orientation: Based On Locale
Label: (Optional) User input text field label
Default Value: (Optional) User input text field default value

Caption Properties
Font ✓ Use Default Dialog, 12

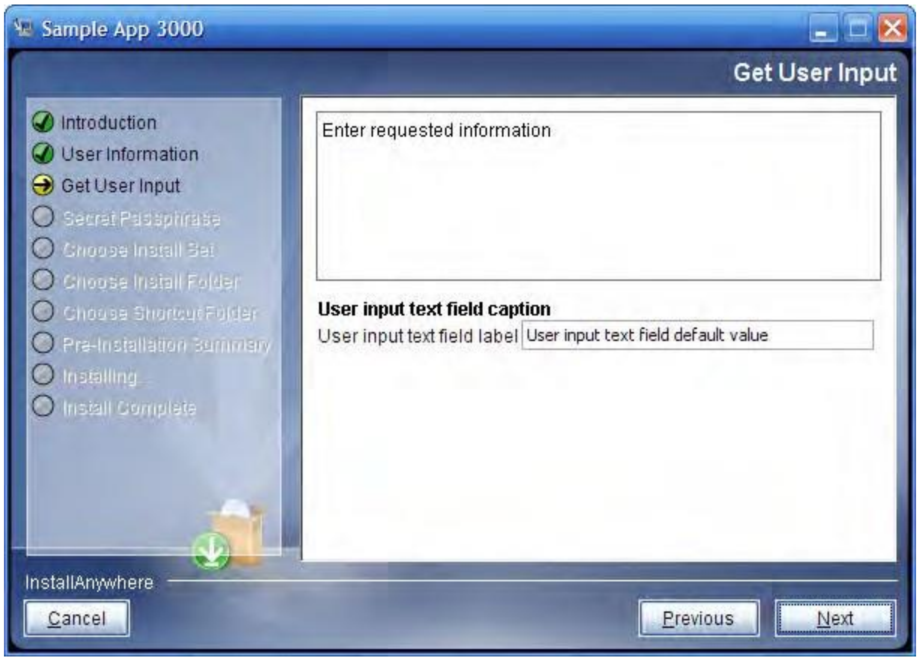
Label Properties
Font ✓ Use Default Dialog, 12
Label Alignment: Left

Control Properties
Font ✓ Use Default Dialog, 12
Color ✓ Use default Example

Result Variable: $USER_INPUT_RESULT_0$

Figure 17-6: Configuring a Text Field

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



Enter requested information

User input text field caption

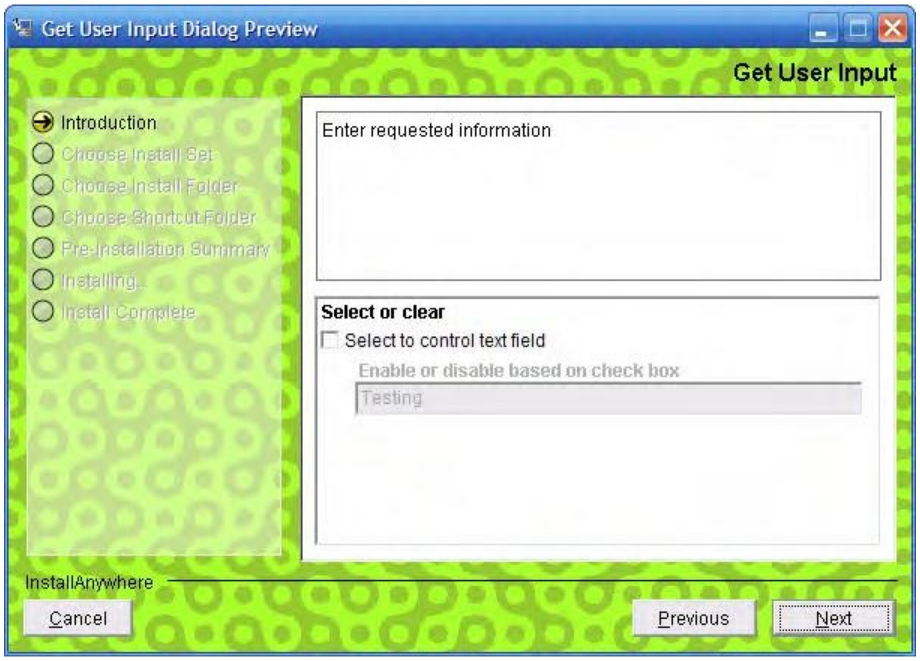
User input text field label: User input text field default value

Figure 17-7: 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 12, Advanced Installer Concepts, 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.*

You can configure a choice group such as check boxes or radio buttons to have dependent subcomponents. A dependent subcomponent is another dialog box control that you can disable and enable (or hide and show) based on the parent control’s selection state. For example, the following figure illustrates a text field dependent on the selection state of a check box.



Enter requested information

Select or clear

Select to control text field

Enable or disable based on check box

Testing

Figure 17-8: Check box with dependent subcomponent

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.

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:

Table 17-2 • Methods to Override

Method	Description
`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"; }
}
```

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);
```

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**.

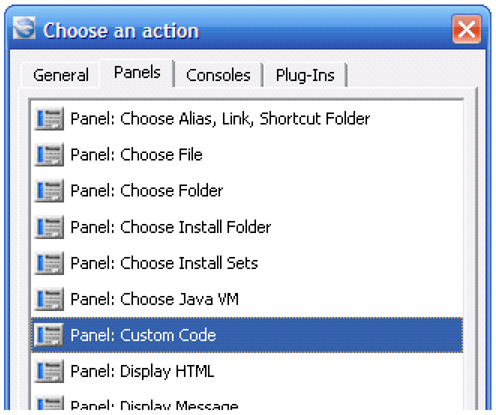
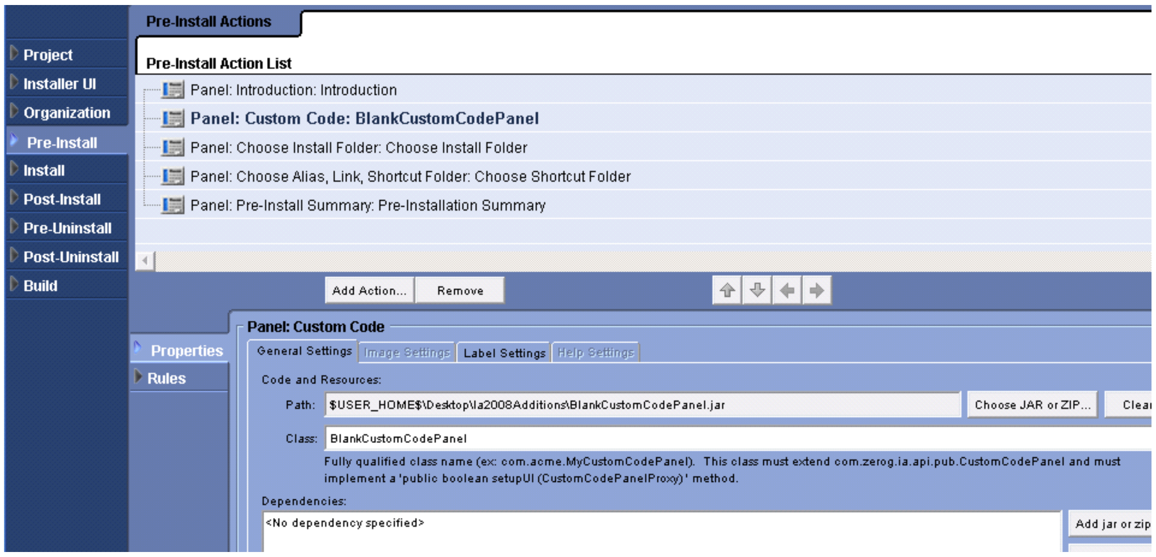


Figure 17-9: Adding a Custom Code Panel

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`).



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

Panel: Custom Code

Code and Resources:

Path: $USER_HOME$\Desktop\ia2008Additions\BlankCustomCodePanel.jar

Class: BlankCustomCodePanel

Fully qualified class name (ex: com.acme.MyCustomCodePanel). This class must extend com.zerog.ia.api.pub.CustomCodePanel and must implement a 'public boolean setupUI (CustomCodePanelProxy)' method.

Dependencies:

<No dependency specified>

Figure 17-10: 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.

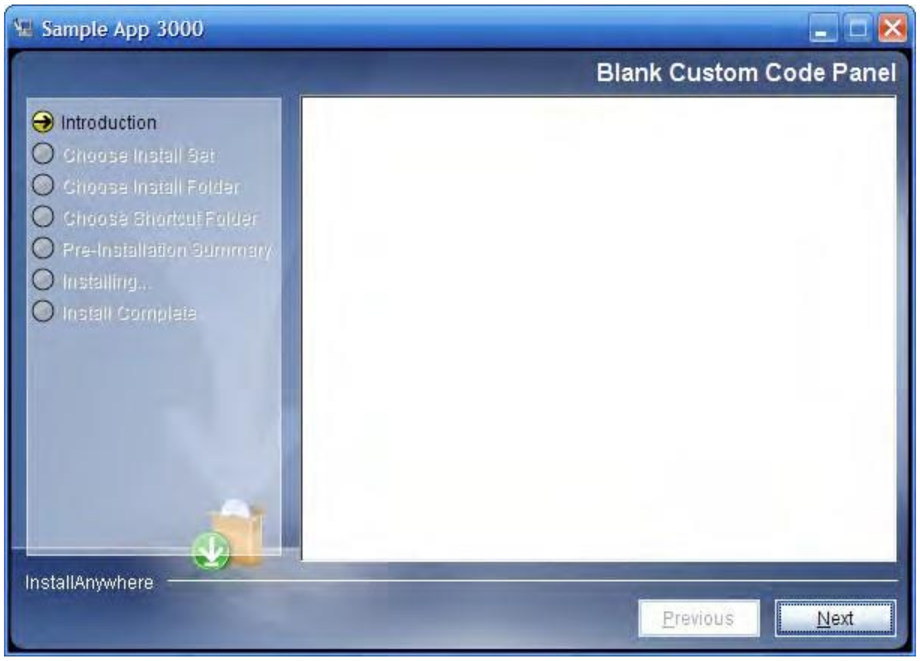


Figure 17-11: Custom Code Panel at Run-time

Adding Controls to the Custom Code Panel

The content area of your custom code panels is where you place user-interface controls that display information to the user and prompt the user for input.

Starting with InstallAnywhere 2009, you can use objects in the package `com.zerog.ia.api.pub.controls` as your user-interface components. The objects provided in that package are `IALabel` for displaying static text; `IATextField` for prompting the user for a line of text or a password; `IAChoiceGroup` for displaying a collection of options as check boxes, radio buttons, a list, or a combo box; `IAButton` for a push button; and `IAFileChooser` to prompt the user to browse for a file or directory. The advantage to using these user-interface components is that they look and act similarly to the components used in built-in InstallAnywhere panels. For information about these controls, see the InstallAnywhere help library and Javadoc help.

As an alternative—or when using a version prior to InstallAnywhere 2009—you can use any Swing or AWT controls available to Java in your custom code panels. 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.*;
import com.zerog.ia.api.pub.controls.*;

public class SimpleCustomCodePanel extends CustomCodePanel
{
    public boolean setupUI(CustomCodePanelProxy ccpp)
    {
        setLayout(new GridLayout(0, 2, 10, 10));
        IALabel[] labels = new IALabel[10];

        for (int i = 0; i < 10; i++)
        {
            labels[i] = new IALabel("Label " + i);
            labels[i].setBorder(BorderFactory.createLineBorder(Color.BLACK));
            add(labels[i]);
        }
    return true;
}
// ...omitting getTitle, 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.

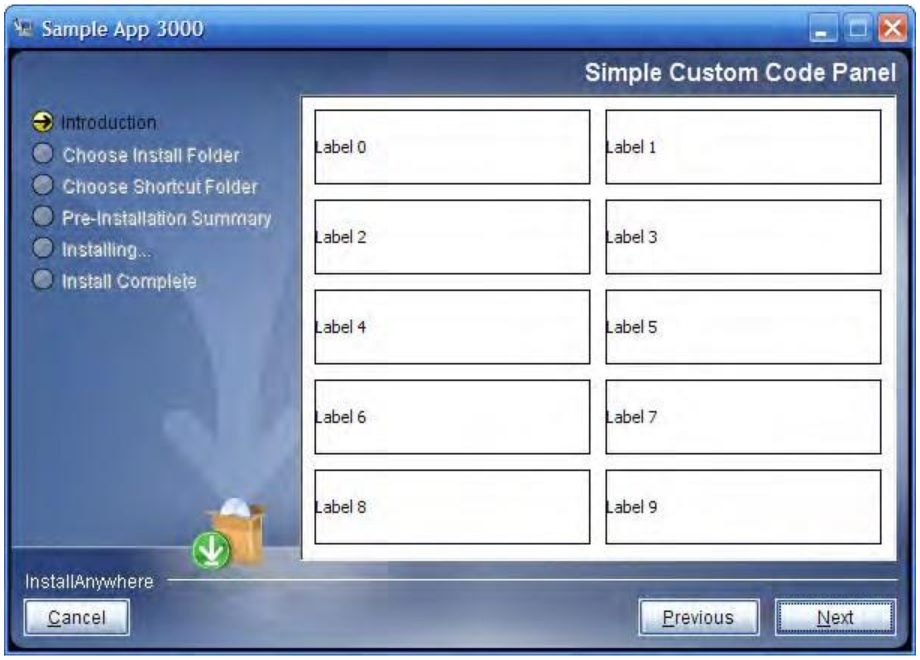


Figure 17-12: Custom Code Panel with Grid Layout

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));
    IALabel[] labels = new IALabel[10];

    for (int i = 0; i < 10; i++)
    {
        labels[i] = new IALabel("Label " + i);
        labels[i].setBorder(BorderFactory.createLineBorder(Color.BLACK));
        add(labels[i]);
    }

    return true;
}
```

At run time, the panel appears as follows.

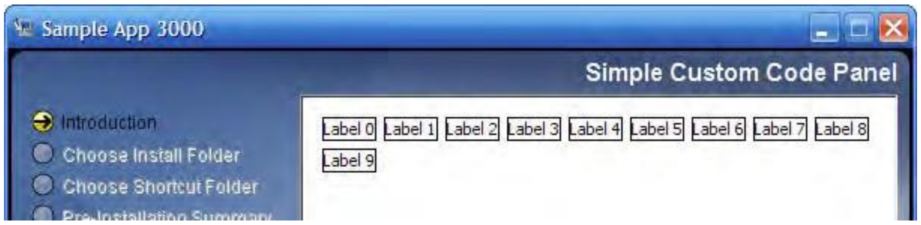


Figure 17-13: Custom Code Panel with Flow Layout

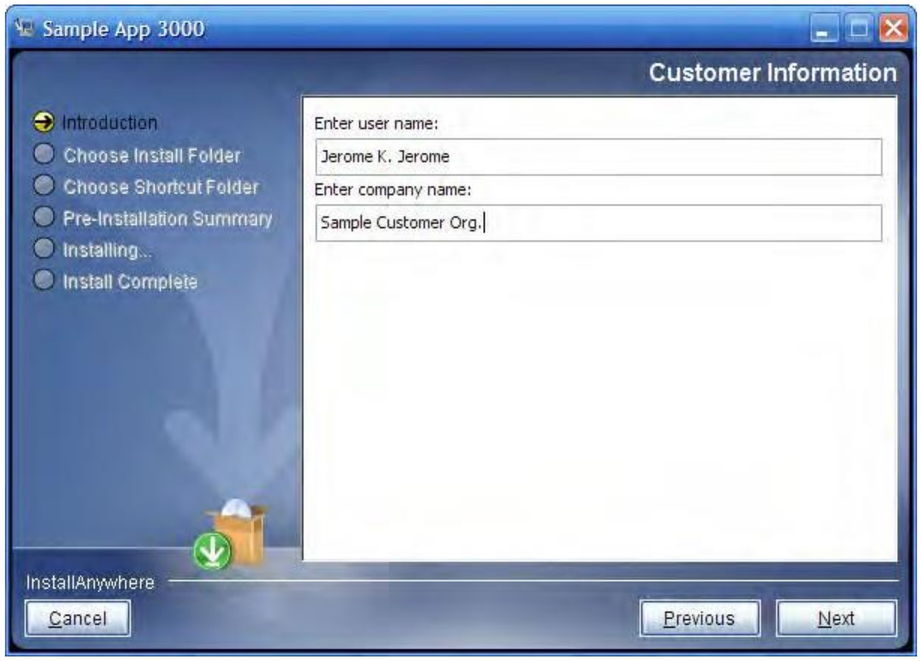
Working with Variables

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 `CustomCodePanel`, the `customCodePanelProxy` static variable and the `CustomCodePanelProxy` object passed to `setupUI` give access to variables, services, and so forth.

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

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

    private IATextField userNameField;
    private IATextField 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 IATextField(System.getProperty("user.name"));
        companyField = new IATextField("Your company name");

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

        add(new IALabel("Enter user name:"));
        add(userNameField);
        add(new IALabel("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:



Enter user name:

Jerome K. Jerome

Enter company name:

Sample Customer Org.

Figure 17-14: 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, beginning 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 com.zerog.ia.api.pub.controls.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

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

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

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

        // create button, add action listener
        button = new IAButton("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:

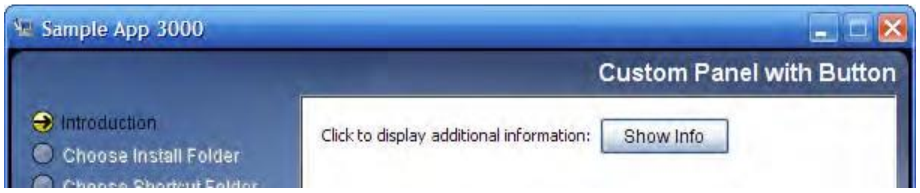


Figure 17-15: Custom Panel Containing a Button

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:

Table 17-3 • CustomCodeConsoleAction Methods

Method	Description
`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.

Table 17-4 • ConsoleUtils Methods

Method	Description
`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.

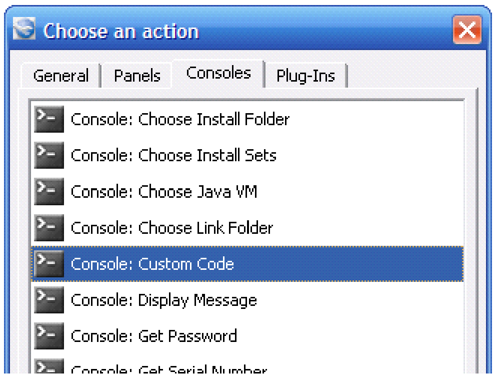


Figure 17-16: Console Actions

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 12, Advanced Installer Concepts.

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.

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

Table 17-5 • Answers to Quick Quiz

Question	Answer
1	a
2	b

18

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:

- Bidirectional Text Support
- Dynamic and Static Text
- Localization and the Internationalized Designer
- Specific Localization Concerns
- Localizable Elements

To generate multi-language installers, open the **Build** task in the Advanced Designer, and use the checkboxes to select the appropriate languages for the selected Build Configuration:

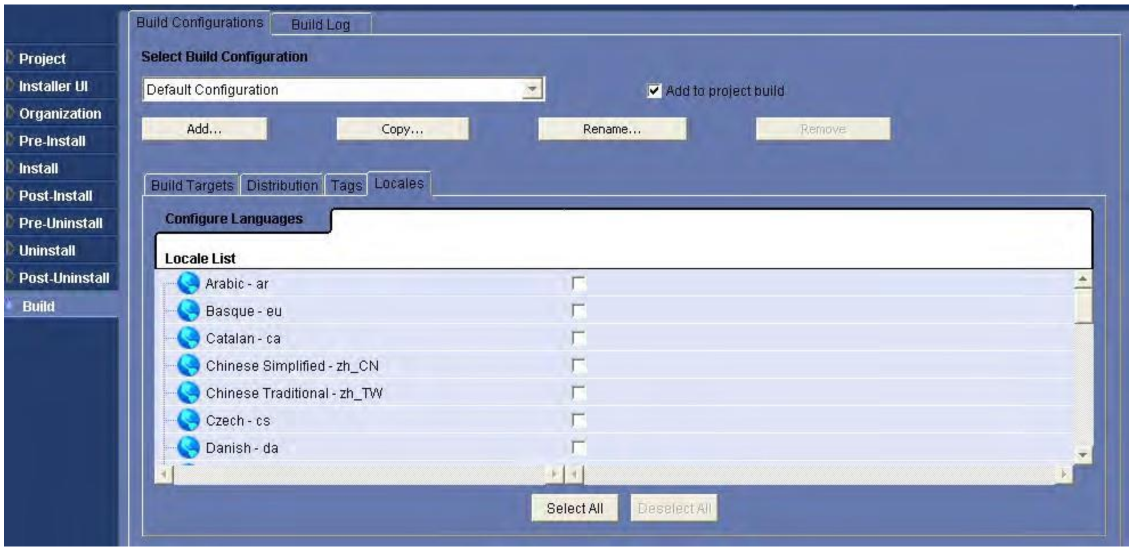


Figure 18-1: Locale List

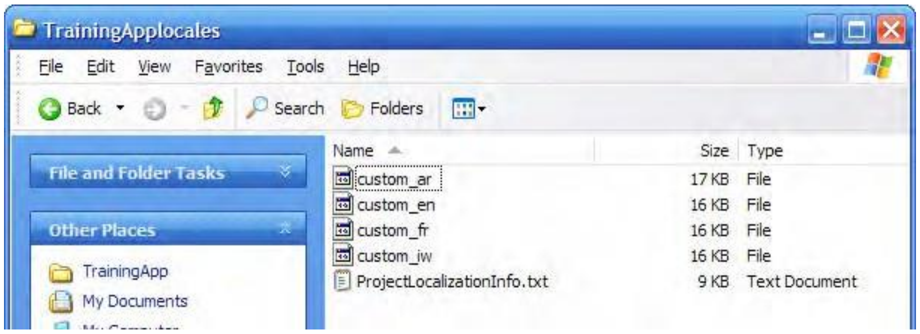
***Tip** • You can specify the locale in which the installer should run by specifying the -l (lowercase L) switch followed by the desired locale abbreviation. For example, to run the installer in French (assuming the project is built to support French), you can run the command install -l fr.*

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 `projectnamelocales`, 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 `projectnamelocales` folder. The locale files are generated as `custom_`*`localecode`*, so for English, which has a locale code of en, the name of the locale file will be `custom_en`.



Name	Size	Type
custom_ar	17 KB	File
custom_en	16 KB	File
custom_fr	16 KB	File
custom_iw	16 KB	File
ProjectLocalizationInfo.txt	9 KB	Text Document

Figure 18-2: 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.

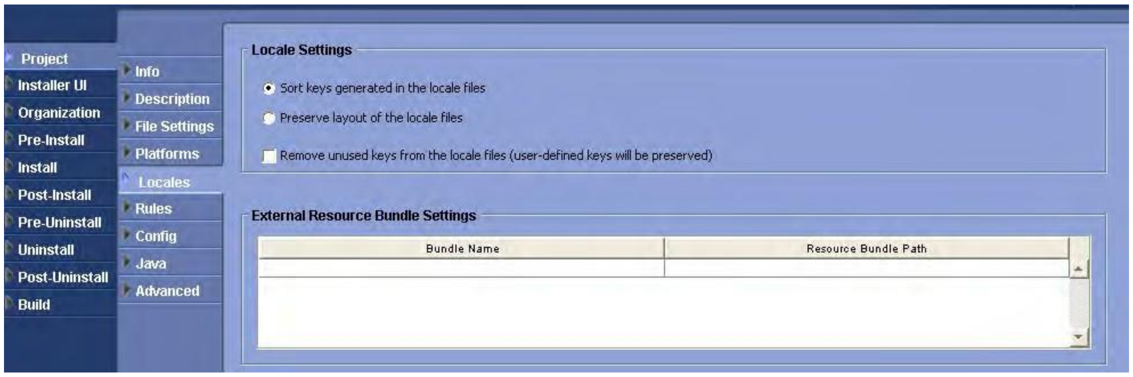
Comments inside the locale files describe the format and contents of the file, along with a timestamp and information about the project associated with the file. For example, the following are two key-value pairs:

```
InstallSet.a2ff402ca1cd.installSetName=Typical
InstallSet.a2ff402ca1cd.description=The most common application features will be installed.
This option is recommended for most users.
```

The `ProjectLocalizationInfo.txt` file contains the mapping between the actions in the project and their keys in the locale files. Review the `ProjectLocalizationInfo.txt` file for any questions regarding to which action the key refers.

If a locale file includes any special (non-ASCII) characters, you should process the file with the Java SDK tool `native2ascii`, which converts special characters (such as é or ñ) into their platform-independent Unicode representations (such as \u00e9 or \u00f1).

You can use the options in the **Locale Settings** area of the **Project > Locales** task 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.



Locale Settings

[✓ Sort keys generated in the locale files] Preserve layout of the locale files

☐ Remove unused keys from the locale files (user-defined keys will be preserved)

External Resource Bundle Settings

Bundle Name	Resource Bundle Path

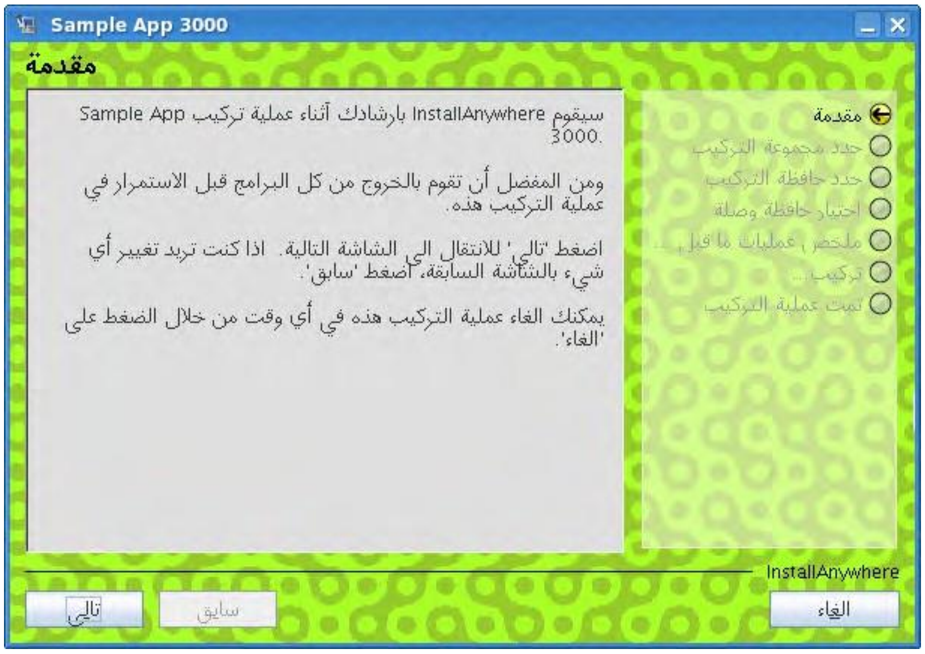
Figure 18-3: Project > Locales Task

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 Flexera Software Knowledge Base article Q113630.

Bidirectional Text Support

InstallAnywhere provides 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.



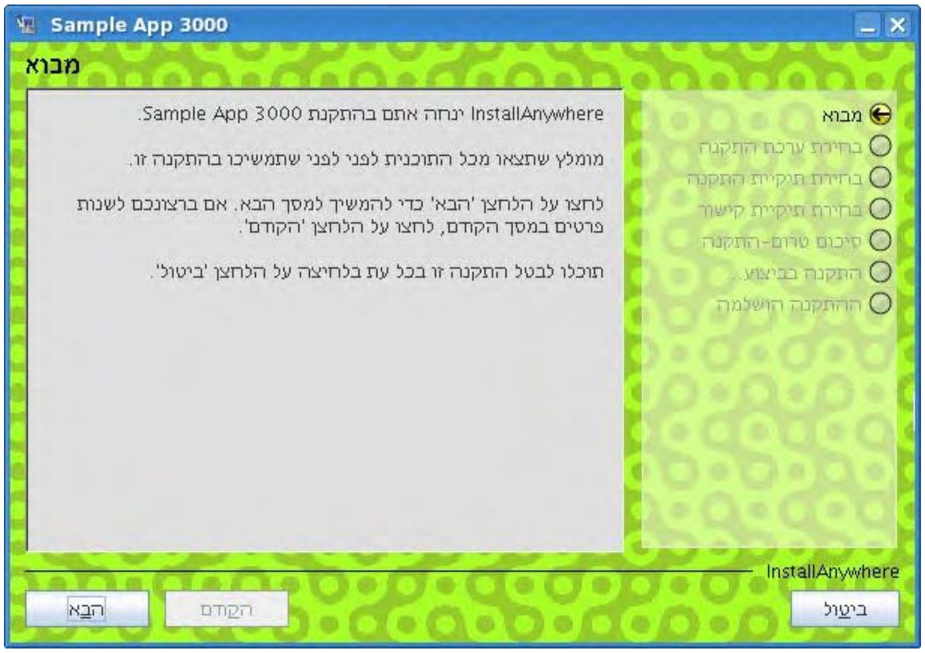
سيقوم InstallAnywhere بارشادك أثناء عملية تركيب Sample App 3000.

ومن المفضل أن تقوم بالخروج من كل البرامج قبل الاستمرار في عملية التركيب هذه.

اضغط 'تالي' للانتقال الى الشاشة التالية. اذا كنت تريد تغيير أي شيء بالشاشة السابقة، اضغط 'سابق'.

يمكنك الغاء عملية التركيب هذه في أي وقت من خلال الضغط على 'الغاء'.

Figure 18-4: Arabic Translation of the Introduction Panel



InstallAnywhere ינחה אתם בהתקנת Sample App 3000.

מומלץ שתצאו מכל התוכנית לפני לפני שתמשיכו בהתקנה זו.

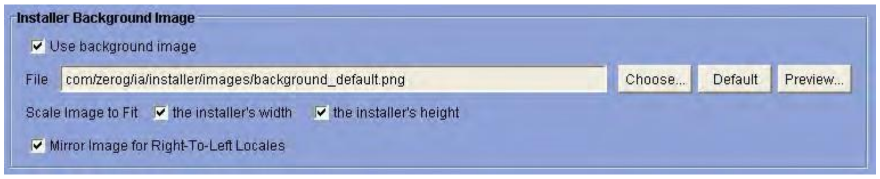
לחצו על הלחצן 'הבא' כדי להמשיך למסך הבא. אם ברצונכם לשנות פרטים במסך הקודם, לחצו על הלחצן 'הקודם'.

תוכלו לבטל התקנה זו בכל עת בלחיצה על הלחצן 'ביטול'.

Figure 18-5: 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 **Installer UI > Look & Feel** task.


Installer Background Image

☑ Use background image

File com/zerog/ia/installer/images/background_default.png

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

☑ Mirror Image for Right-To-Left Locales

Figure 18-6: Installer Background Image Dialog

You can also use the same task to specify the title and prompt of the console-mode locale-selection panel.

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

The following localization concerns are addressed in this section:

- Localizing Resources
- Localizing the Splash Screen
- Localizing Custom Installer Labels
- Using External Resource Bundles
- Localizing Custom Code
- Best Practices for Localizing
- Changing Localized Text
- Modifying Localized Text
- Changing Default Translations Provided in Language Packs

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, perform the following steps:

Task ***To localize the License Agreement panel:***

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 the Splash Screen

You can localize your installer's splash screen. As with resources described in the previous section, you should add all of your localized images to the **Install** task in the `DO NOT INSTALL` magic folder. Similarly, in each locale file, modify the entries named **`Installer.#.splashScreenGUIImageName`** and **`Installer.#.splashScreenImagePath`** so that they refer to the corresponding image file.

For a multi-language project, the default splash screen appears as follows.

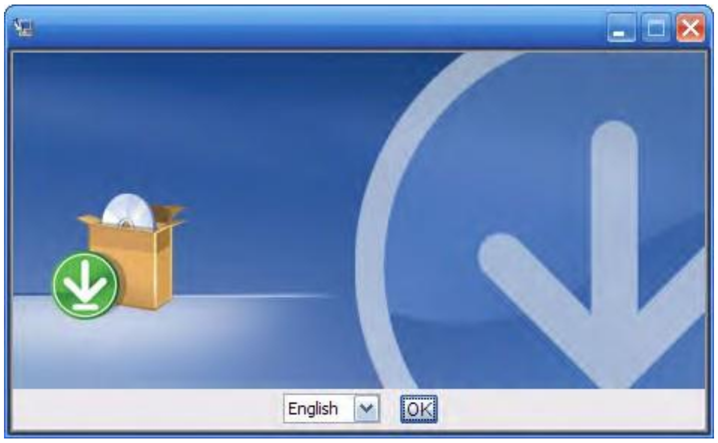
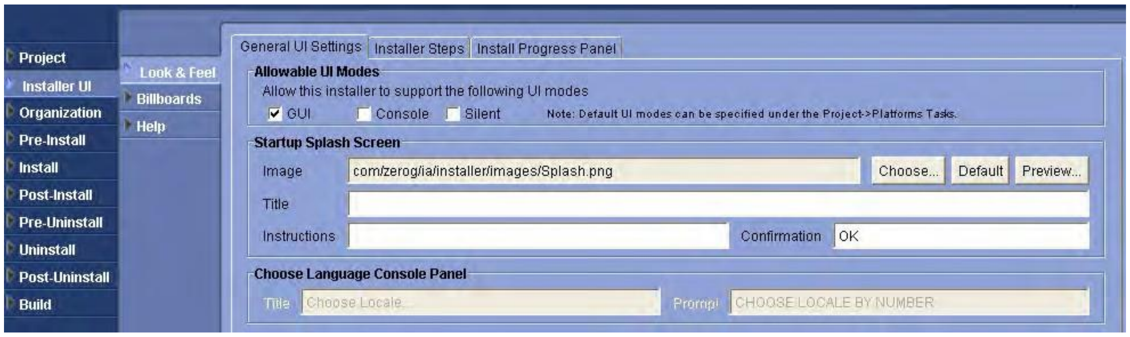


Figure 18-7: Default splash screen for multi-language project

You can use the `Startup Splash Screen` settings in the **Installer UI** > **Look & Feel** task to specify a title to display in the caption bar, instructions to display next to the locale list, and the text displayed on the confirmation button.



Allowable UI Modes
Allow this installer to support the following UI modes
[✓ GUI] Console Silent
Note: Default UI modes can be specified under the Project->Platforms Tasks.

Startup Splash Screen
Image: com/zerog/ia/installer/images/Splash.png
Title:
Instructions:
Confirmation: OK

Choose Language Console Panel
Title: Choose Locale...
Prompt: CHOOSE LOCALE BY NUMBER

Figure 18-8: Startup Splash Screen Settings

For example, if you set the **Title** to **`$PRODUCT_NAME$`** and the **Instructions** to **`Select a locale:`**, the splash screen appears as follows.

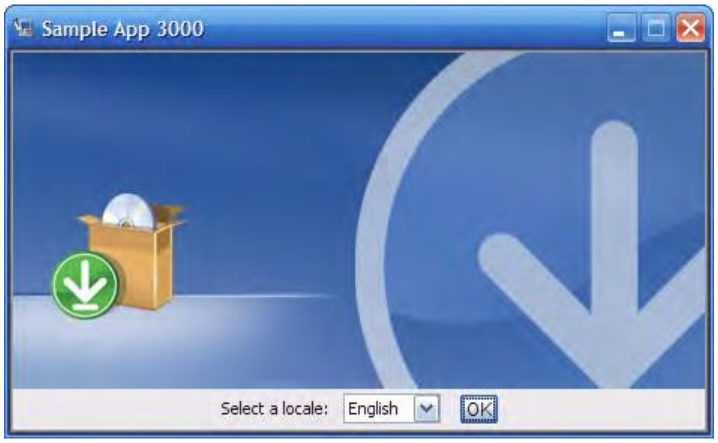


Figure 18-9: Sample Splash Screen

***Note** • `$PRODUCT_NAME$` and `$INSTALLER_TITLE$` are the only variables supported in the splash screen settings.*

To provide translated text for these splash screen elements, in the locale file modify the `Installer.#.splashScreenGUITitle`, `splashScreenGUIInstructions`, and `splashScreenGUIConfirm` values.

Using similar settings, you can modify the text displayed for locale selection in console mode.

Localizing Custom Installer Labels

Custom labels that match the installer panels are not automatically localized. To match labels, perform the following steps.

Task ***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 a plain-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.

Using External Resource Bundles

You can store localized strings in external resource bundles. A resource bundle is a collection of text files containing translated resources, where each locale-specific bundle follows a standard naming convention that identifies the specific language it targets. External resource bundles are supported only in the Enterprise edition of InstallAnywhere.

For example, suppose you want to provide translated strings in a resource bundle called "MyStrings". To assemble the resource bundle, you create properties files named `MyStrings_locale.properties`. You express the locale using the standard Java abbreviations used in other localized elements. The English version of the MyStrings bundle should be named `MyStrings_en.properties`, and the French version should be named `MyStrings_fr.properties`.

The contents of each file is made up of lines reading:

```
resource_key=Translated string
```

If the resource bundle consists of a single string with identifier "greeting", the contents of the English file `MyStrings_en.properties` would be:

```
greeting=Welcome!
```

The contents of a German bundle `MyStrings_de.properties` would be:

```
greeting=Willkommen!
```

In a resource bundle, it is essential that only the values to the right of the equal sign are translated. The resource keys to the left of the equal sign must be the same for every locale.

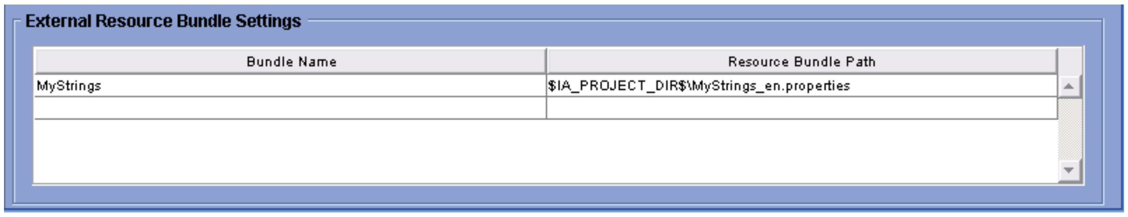
To refer to the translated strings in your project, you can use the following type of expression:

```
$L{bundle_name.resource_key}
```

In the MyStrings example, the expression to add to your project—in, for example, the message displayed by the Introduction panel—would be:

```
$L{MyStrings.greeting}
```

The last step is to add the external resource bundles to your project. In the **Project > Locales** task, the **External Resource Bundle Settings** customizer is where you specify the name of your resource bundles and their location. In the **Resource Bundle Path** field, you need only browse for one resource bundle file; if you followed the bundle naming convention, all of the localized files will be included.

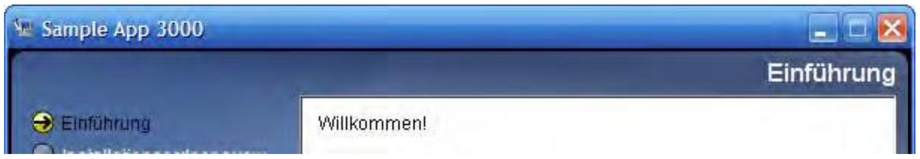


External Resource Bundle Settings

Bundle Name	Resource Bundle Path
MyStrings	$IA_PROJECT_DIR$\MyStrings_en.properties

Figure 18-10: Adding a Resource Bundle

After building your project and running the installer, the translated text from the selected locale will be displayed. For example, the following figure shows the value of the “greeting” key from the MyStrings bundle when the user selects to run the installer in German.



Willkommen!

Figure 18-11: Displaying Text from a Resource Bundle

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)
{
    IALabel myLabel = new IALabel( );
    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 files change, it may require costly re-translation work.

- Use the default text whenever possible. All of the default text is already translated, saving the team time and effort.
- Test the installers on systems running in the foreign locale. This will help reveal 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

To modify localized text, perform the following steps.

Task ***To modify localized text:***

1. Include any localized files (license agreements, graphics for billboards, and so forth) 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

To change the default translations provided in language packs, perform the following steps.

Task ***To change the default translations 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 Flexera Software Support.

Table 18-1 • Language Resource Properties

Property	Definition
`Installer.#.ProductName`	Name of product displayed on installer title bar.
`Installer.#.RulesFailedMessage`	Message displayed if specified rules prevent the installer from running.
`Installer.#.ShortcutDestinationPathMacOS`	Path to where aliases are created during installation on Mac OS, relative to the end-user-selected alias folder chosen on the Choose Alias Location step.

Table 18-1 • Language Resource Properties

Property	Definition
`Installer.#.ShortcutDestinationPathWin32`	Path to where shortcuts are created during installation on Windows, relative to the end-user-selected shortcut folder chosen on the Choose Shortcut Folder step.
`Installer.#.ShortcutDestinationPathSolaris`	Path to where links are created during installation on Unix, relative to the end-user-selected links folder chosen on the Choose Link Location step.
`InstallSet.#.Description`	Description of one of the installer's features.
`InstallSet.#.InstallSetName`	Name of one of the installer's features.
`IntroAction.#.message`	Text to display on the installer's Introduction step.
`Intro.#.stepTitle`	Title to display on the installer's Introduction step. Note that this is a filename only, and not a fully qualified pathname.
`LicenseAgr.#.Path`	Path name to localized license agreement to be displayed as the installer is preparing itself. (Note that this is only a pathname and does not include the filename.)
`LicenseAgr.#.Title`	Title of License Agreement step in the installer.
`MakeExecutable.#.destination Name`	Name of the LaunchAnywhere Executable to be created on the destination computer.
`MakeRegEntry.#.Value`	Value to be written to the Windows registry.
`ShortcutLocAction.#.macTitle`	Title of Mac OS X Choose Alias Location step in the installer.
`ShortcutLocAction.#.unixTitle`	Title of Unix Choose Link Folder step in the installer.
`ShortcutLocAction.#.WinTitle`	Title of Windows Choose Shortcut Folder step in the installer.
`Billboard.#.ImageName`	Name of billboard image file to be displayed as the installer is preparing itself. (Note that this is a filename only, and not a fully qualified pathname.)
`Billboard.#.ImagePath`	Path name to billboard image to be displayed as the installer is preparing itself. (Note that this is only a pathname and does not include the filename.)
`ChooseInstallSet.#.Title`	Title of Choose Install Set step in the installer.
`ChooseJavaVM.#.Title`	Title of Choose Java Virtual Machine step in the installer.

Table 18-1 • Language Resource Properties

Property	Definition
`CreateShortcut.#.Destination Name`	Name of the shortcut/alias/link to be created on the destination computer.
`human.readable.language.name`	The name of the language represented by the data in this resource file (as in English, Español, and so forth).
`ImportantNote.#.FileName`	Name of text file to be displayed on the Important Note step of the installer. Note that this is a filename only, and not a fully qualified pathname.
`ImportantNote.#.Path`	Path name to text file to be displayed on the Important Note step of the installer. Note that this is only a pathname and does not include the filename.
`ImportantNote.#.Title`	Title of Important Note step in installer.
`InstallBundle.#.BundleName`	Name of component.
`InstallBundle.#.Description`	Description text describing component.
`InstallComplete.#.DisplayText`	Text to display on the Install Complete step of the installer.
`InstallComplete.#.Title`	Title of the Install Complete step in the installer.
`InstallDir.#.Title`	Title of the Choose Installation Directory step of the installer.
`Installer.#.InstallerName`	Name of the installer.

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.

Table A-1 • General Information

Field	Description
Product Name:	
Product Version:	

Target Platforms

Select your target platforms:

Table A-2 • Target Platform

X	Target Platform
	Mac OS X
	Windows
	AIX
	HP-UX
	Linux
	Solaris
	Other Unix:

Table A-2 • Target Platform

X	Target Platform
	Other Java-Enabled Platforms:

Installer Target

Specify the type of person who will be installing your application:

Table A-3 • Installer Target

X	End User
	Technical End User
	Non-Technical End User

Deployment Media

Specify the type of deployment media you will be using:

Table A-4 • Deployment Media

X	Deployment Media
	Web
	CD-ROM/DVD
	Merge Module

Application Type

Identify the type of your application:

Table A-5 • Application Type

X	Application Type
	Native Application
	Java Application
	.NET Application

Java-Specific Options

Specify the Java Virtual Machine(s) version required:

Table A-6 • Java Virtual Machine(s) Version Required

Java Virtual Machine(s) Version Required

Installation Needs

Specify the location(s) on the target system where files are to be installed:

Table A-7 • Installation Location on Target System

Platform	Location
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

List the configuration steps that must be done to the target platform:

Table A-8 • Configuration Steps

Step #	Description
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	

What Information Must Be Collected from the End User?

List the information that must be collected from the end user:

Table A-9 • End User Information

Information

Table A-9 • End User Information

Information

Team Development Options

Will this project be managed by more than one developer?

- [ ] **Yes**
- [ ] **No**

If **Yes**, list the developers and their role.

Table A-10 • Team Development Options

Developer Name	Description of Role

Source Paths

Will any Source Paths need to be defined for file maintenance?

- [ ] **Yes**
- [ ] **No**

If Yes, then what Source Paths will be defined for file maintenance?

Table A-11 • Source Paths

Source Path	Description

Uninstall Options

Will this installation require any special uninstall options?

- [ ] **Yes**
- [ ] **No**

If **Yes**, specify uninstall options:

Pre-Uninstall

List any special Pre-Uninstall options that are required:

Table A-12 • Required Pre-Uninstall Options

Required Pre-Uninstall Options

Post-Uninstall

List any special Post-Uninstall options that are required:

Table A-13 • Required Post-Uninstall Options

Required Post-Uninstall Options

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.

Data Files Used in These Exercises

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.

Table B-1 • InstallAnywhere Training Data Files

Type	Location
OfficeSuite	Installed with InstallAnywhere in the following directory: *`[InstallAnywhere Installation Directory]`*`\OfficeSuiteSourceFiles`
TrainApp	Provided by your instructor or with your course materials in a directory named `TrainApp`.

Setting Windows Folder Options

Before beginning these exercises, it is recommended that you make sure that several Windows folder options are set to optimize folder viewing options.

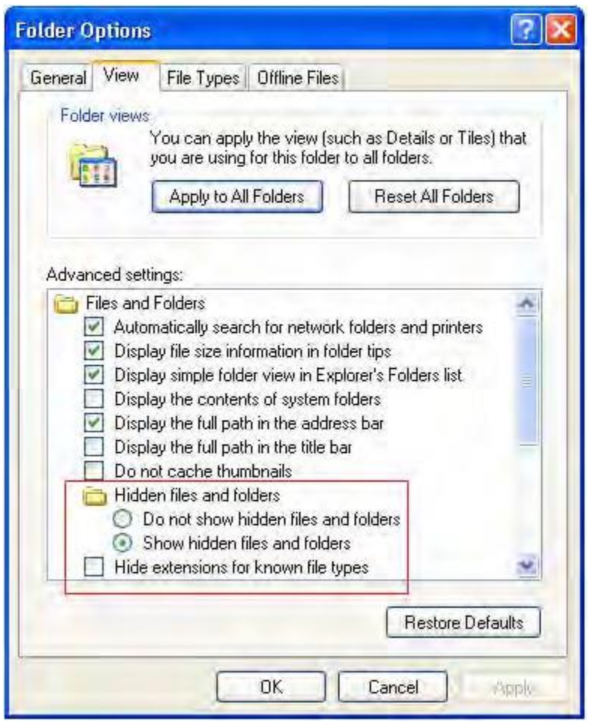
Setting Windows Folder Options in Windows XP

To set Windows folder options in Windows XP, perform the following steps.

Task ***To set Windows XP folder options:***

1. Open the Windows Control Panel.
2. On the **Tools** menu, click **Folder Options**. The **Folder Options** dialog box opens.

3. Open the **View** tab.



You can apply the view (such as Details or Tiles) that you are using for this folder to all folders.

Advanced settings:

Files and Folders
- [✓] Automatically search for network folders and printers
- [✓] Display file size information in folder tips
- [✓] Display simple folder view in Explorer's Folders list
- [ ] Display the contents of system folders
- [✓] Display the full path in the address bar
- [ ] Display the full path in the title bar
- [ ] Do not cache thumbnails

Hidden files and folders
- ○ Do not show hidden files and folders
- ● Show hidden files and folders
- [ ] Hide extensions for known file types

4. In the **Advanced settings** list under **Hidden files and folders**, select **Show hidden files and folders**.

5. Clear the check box labeled **Hide extensions for known file types**.

Setting Windows Folder Options in Windows 7

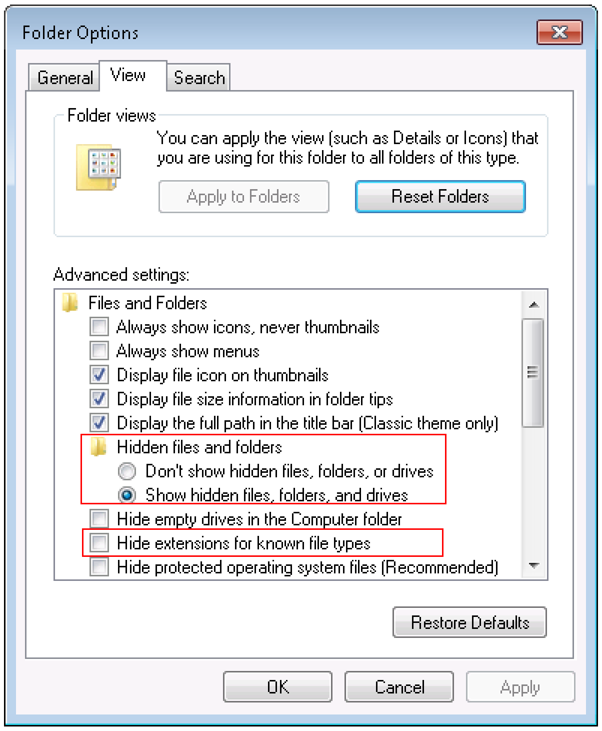
To set Windows folder options in Windows 7, perform the following steps.

Task ***To set Windows 7 folder options:***

1. Open the Windows Control Panel.

2. Click **Appearance and Personalization**. The **Personalization** options are displayed.

3. Under **Folder Options**, click **Show hidden files and folders**. The **Folder Options** dialog box opens.


Folder Options

General | View | Search

Folder views

You can apply the view (such as Details or Icons) that you are using for this folder to all folders of this type.

Apply to Folders | Reset Folders

Advanced settings:

- Files and Folders
  - [ ] Always show icons, never thumbnails
  - [ ] Always show menus
  - [✓] Display file icon on thumbnails
  - [✓] Display file size information in folder tips
  - [✓] Display the full path in the title bar (Classic theme only)
  - Hidden files and folders
    - ○ Don't show hidden files, folders, or drives
    - ● Show hidden files, folders, and drives
  - [ ] Hide empty drives in the Computer folder
  - [ ] Hide extensions for known file types
  - [ ] Hide protected operating system files (Recommended)

Restore Defaults

OK | Cancel | Apply

4. In the **Advanced settings** list under **Hidden files and folders**, select **Show hidden files, folders, and drives**.

5. Clear the check box labeled **Hide extensions for known file types**.

Chapter 2 Exercises: Starting a Project Using the InstallAnywhere Project Wizard

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
- Creating the TrainApp Project

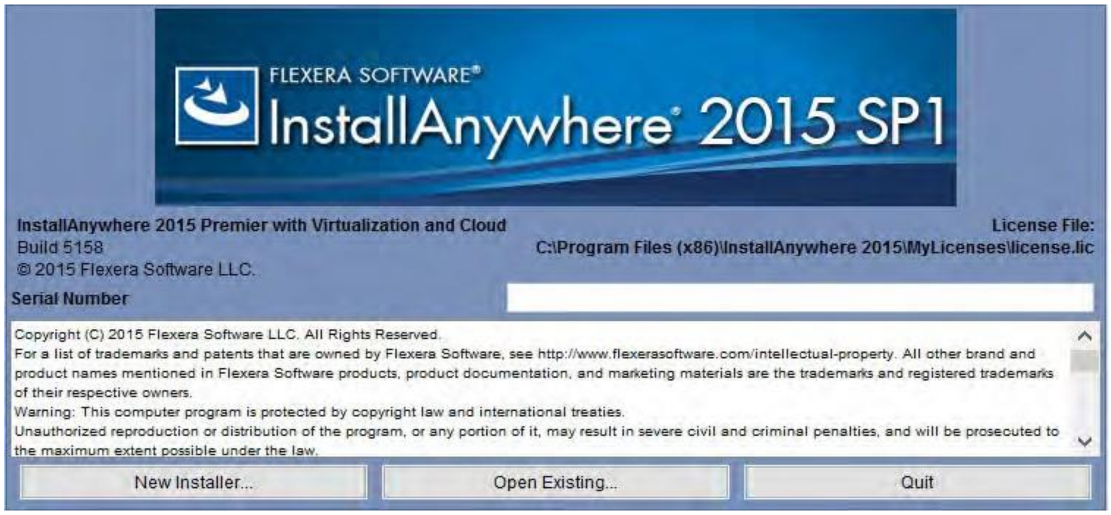
Creating the OfficeSuite Project

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.

Task ***To create a new project:***

1. Launch InstallAnywhere.

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



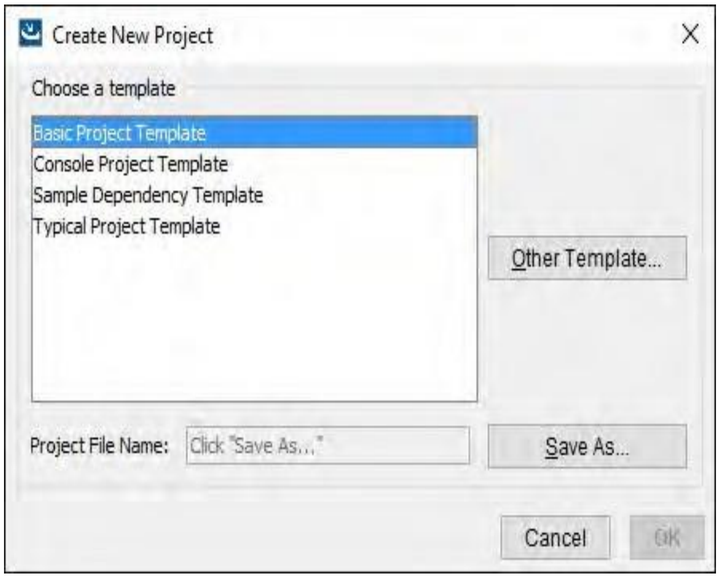
InstallAnywhere 2015 Premier with Virtualization and Cloud
Build 5158
© 2015 Flexera Software LLC.

License File:
C:\Program Files (x86)\InstallAnywhere 2015\MyLicenses\license.lic

Serial Number

Copyright (C) 2015 Flexera Software LLC. All Rights Reserved.
For a list of trademarks and patents that are owned by Flexera Software, see http://www.flexerasoftware.com/intellectual-property. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.
Warning: This computer program is protected by copyright law and international treaties.
Unauthorized reproduction or distribution of the program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

3. Click **Save As** to save and name the project. The **Save New Project As** dialog box opens. By default the project is named **My_Product**.



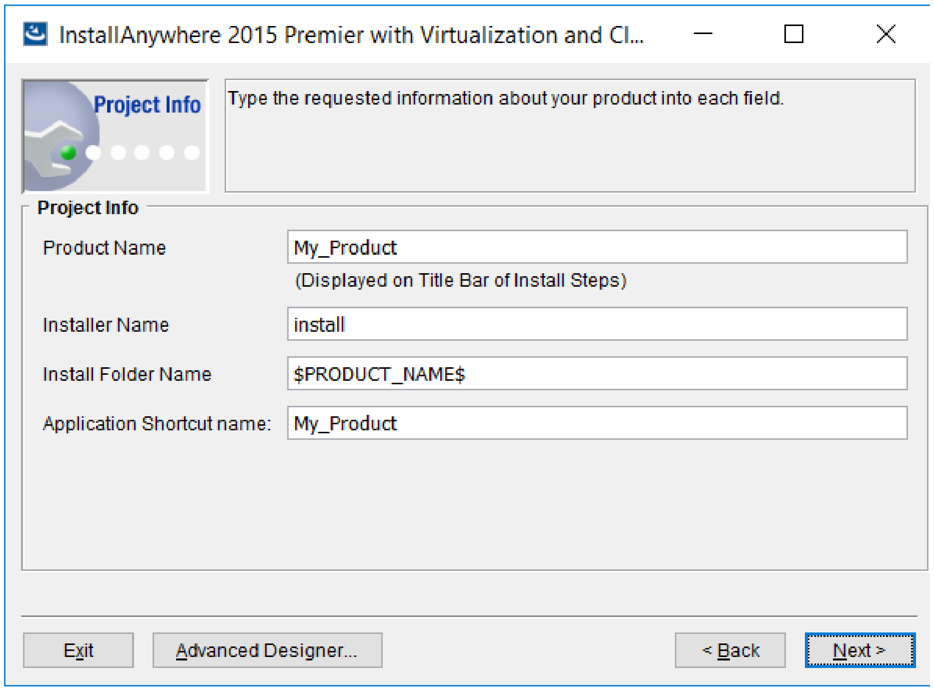
4. Click **Save** to confirm the name and close this dialog box.
5. Click **Next**. The **Project Info** panel opens.
6. Continue with the steps in Setting Project Information.

Setting Project Information

On the **Project Info** panel of the Project Wizard, you define the 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.

Task ***To set project information:***

1. Perform the steps in Creating the OfficeSuite Project. The **Project Info** panel opens.



Type the requested information about your product into each field.

Project Info

Product Name: My_Product
(Displayed on Title Bar of Install Steps)

Installer Name: install

Install Folder Name: $PRODUCT_NAME$

Application Shortcut name: My_Product

2. 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

Note • *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.*

3. Click **Next**. The **Add Files** panel of the Project Wizard opens.

4. Continue with the steps in Adding Files to the Project.

Installing Tasks

This section consists of several steps:

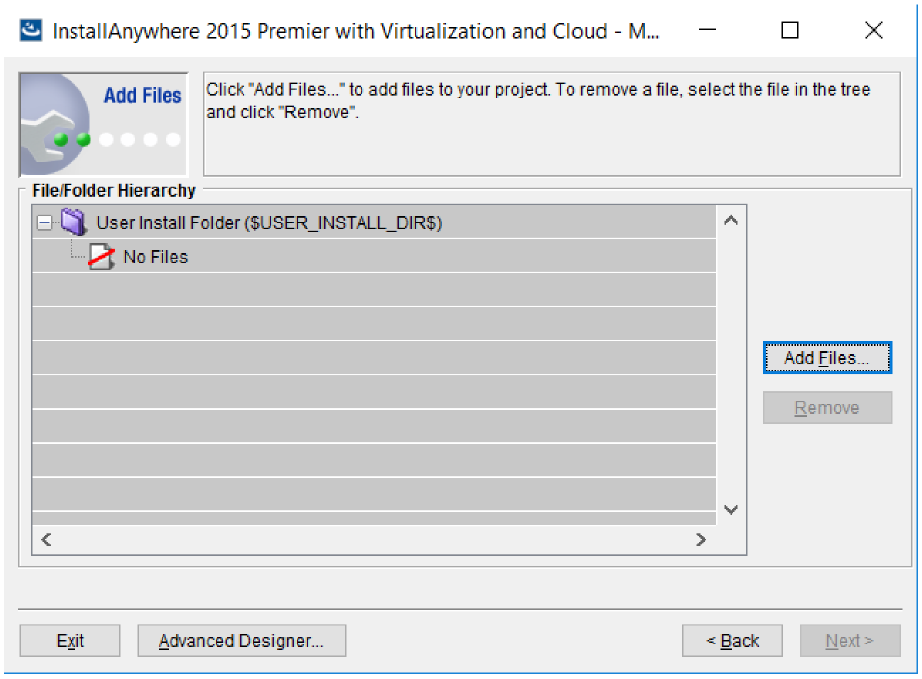
- Adding Files to the Project
- Choosing the Main Java Class for Starting the Application
- Setting the Classpath

Adding Files to the Project

To add files to the project, perform the following steps.

Task ***To add files to the project:***

1. Perform the steps in Setting Project Information. The **Add Files** panel opens.



InstallAnywhere 2015 Premier with Virtualization and Cloud - M...

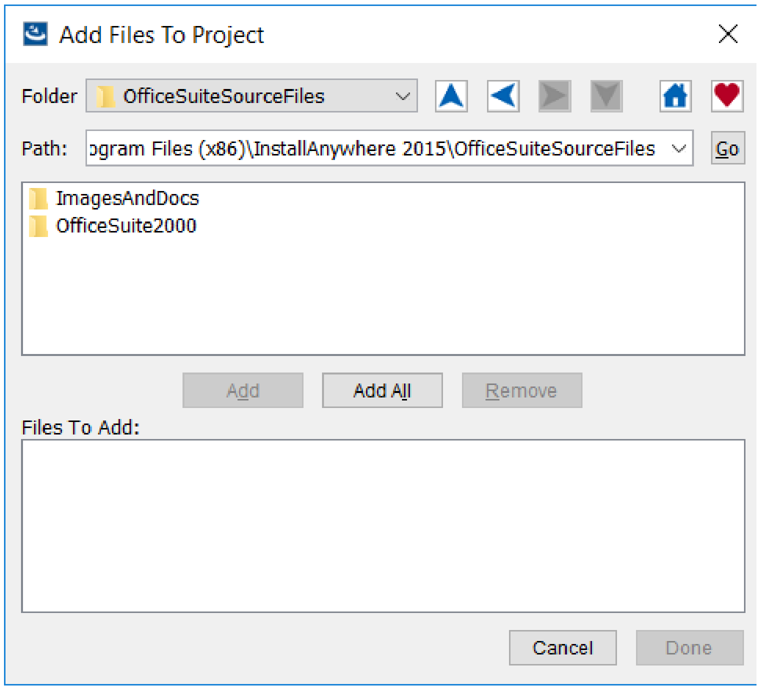
Add Files

Click "Add Files..." to add files to your project. To remove a file, select the file in the tree and click "Remove".

File/Folder Hierarchy

User Install Folder ($USER_INSTALL_DIR$)
No Files

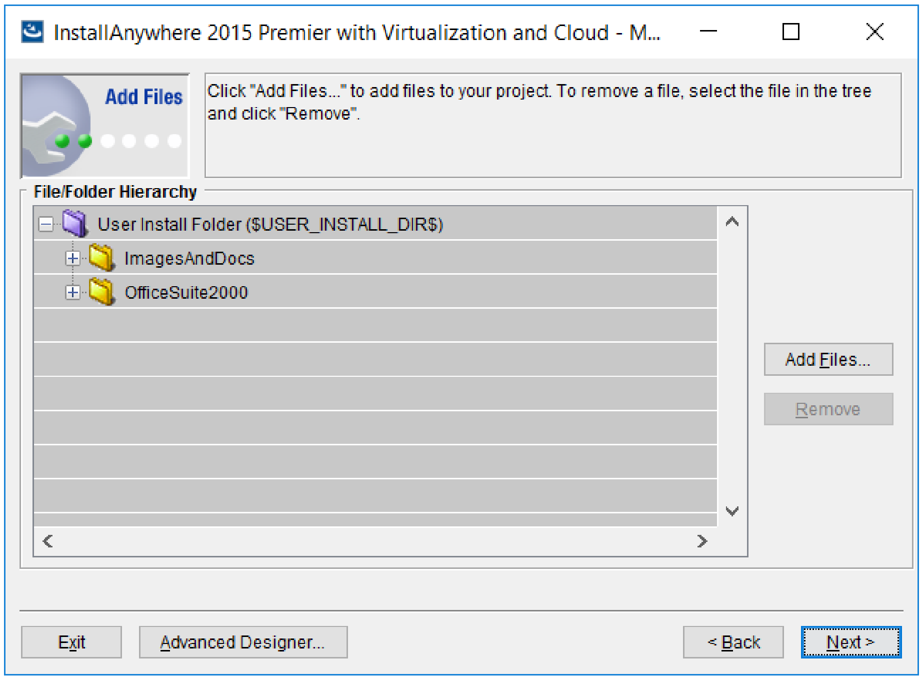
2. Click **Add Files**. The **Add Files to Project** dialog box opens.
3. Browse through the list to find the `OfficeSuiteSourceFiles` folder, located within the InstallAnywhere installation directory.



4. Click **Add All** to add the `ImagesAndDocs` and `OfficeSuite2000` directories, which are inside the `OfficeSuiteSourceFiles` folder. These directories are now listed in the **Files to Add** list.

Note • *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.*

5. Click **Done**. The selected files are now listed in the **File/Folder Hierarchy**.


InstallAnywhere 2015 Premier with Virtualization and Cloud - M...

Click "Add Files..." to add files to your project. To remove a file, select the file in the tree and click "Remove".

***Note** • 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.*

6. Click **Next**. The **Choose Main** panel opens.
7. Continue with the steps in Choosing the Main Java Class for Starting the Application.

Choosing the Main Java Class for Starting the Application

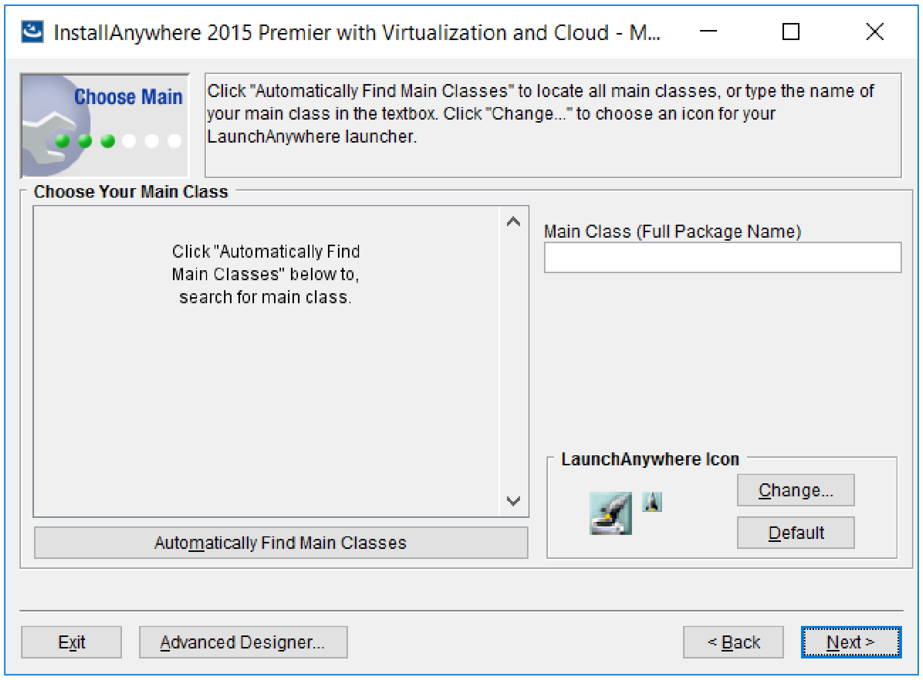
On the **Choose Main** panel of the Project Wizard, you select the starting class for the application. A Java application contains one or more classes that implement a method called `main`. This wizard panel is where you specify the class whose `main` method you want to execute when a user launches your LaunchAnywhere executable.

***Note** • If you are not installing a Java application, you would click **Next** without specifying a main class.*

On the **Choose Main** panel, you can also specify custom icons (in `.gif` format) for the LaunchAnywhere executable file.

Task ***To choose main class:***

1. Perform the steps in Adding Files to the Project. The **Choose Main** panel opens.


InstallAnywhere 2015 Premier with Virtualization and Cloud - M...

Choose Main

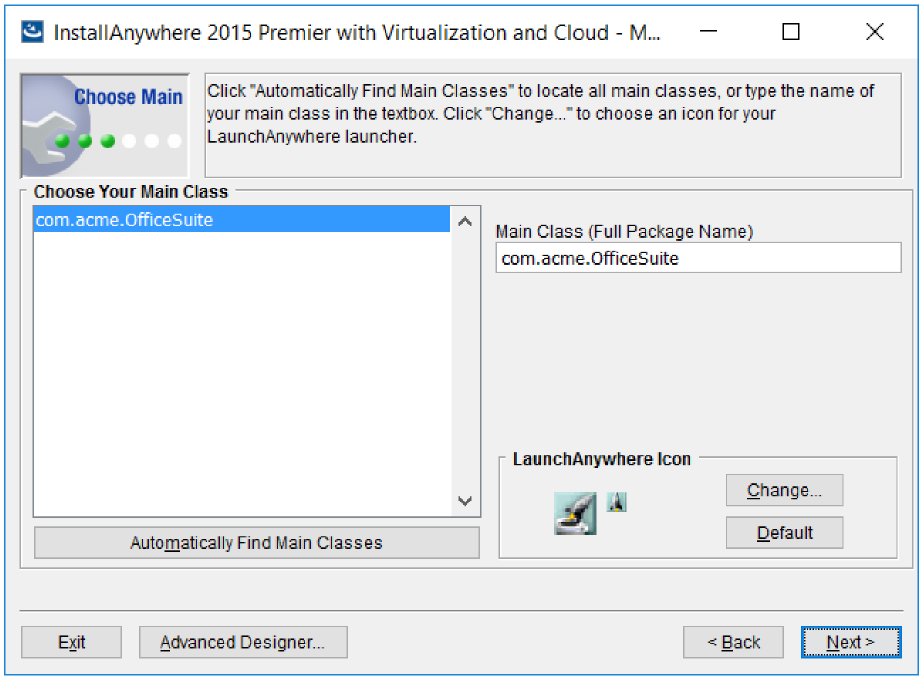
Click "Automatically Find Main Classes" to locate all main classes, or type the name of your main class in the textbox. Click "Change..." to choose an icon for your LaunchAnywhere launcher.

Choose Your Main Class

Click "Automatically Find Main Classes" below to, search for main class.

Main Class (Full Package Name)

2. Click the **Automatically Find Main Classes** button. InstallAnywhere locates all main classes and lists them in the **Choose Your Main Class** list.


InstallAnywhere 2015 Premier with Virtualization and Cloud - M...

Choose Main

Click "Automatically Find Main Classes" to locate all main classes, or type the name of your main class in the textbox. Click "Change..." to choose an icon for your LaunchAnywhere launcher.

Choose Your Main Class

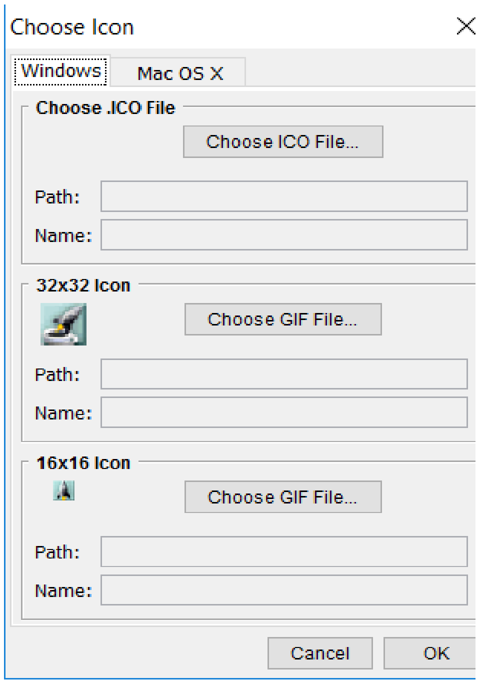
com.acme.OfficeSuite

Main Class (Full Package Name)

com.acme.OfficeSuite

3. Select the main class, which is `com.acme.OfficeSuite`.

4. To specify a custom icon for the LaunchAnywhere executable, click **Change**. The **Choose Icon** dialog box opens, where you can choose a 32-by-32 or a 16-by-16 pixel `.gif` for the application icon.

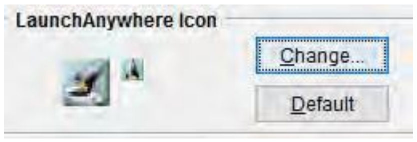


5. Click **Choose GIF File** under **32x32 Icon**, navigate to the `Image and Docs` directory, and choose `OfficeIcon.gif`.

***Note** • Windows-only `.ico` files are not supported.*

6. Click **Choose GIF File** under **16x16 Icon**, navigate to the `Image and Docs` directory and choose `OfficeIconSmall.gif`.

7. Click **OK** to confirm and close the dialog box. The icons you selected are now displayed in the **LaunchAnywhere Icon** area:



8. Click **Next**. The **Classpath** panel opens.

9. Proceed with the steps in Setting the Classpath.

Setting the Classpath

On the **Classpath** panel of the Project Wizard, you configure a Java application's classpath, which is a list of directories and `.jar` files containing classes used by the application. For example, to deploy a Java application packaged as a `.jar` file, the `.jar` file is required on the application's classpath. This is reflected in the command used to manually launch a Java application, such as:

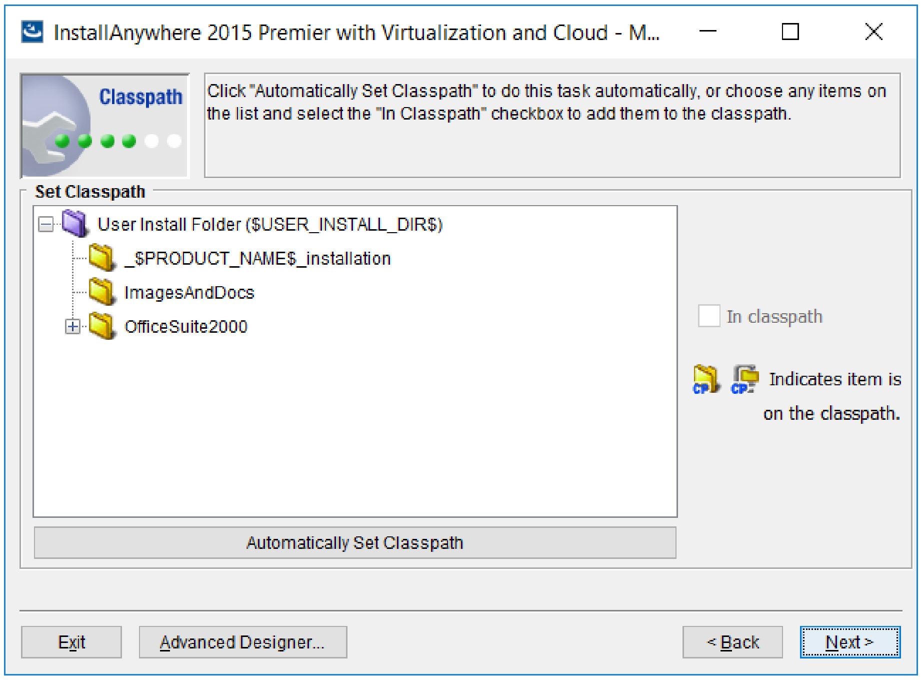
```
java -cp TrainApp.jar TrainingAppMainClass
```

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

In this step, you will have InstallAnywhere automatically set the classpath.

Task ***To set the classpath:***

1. Perform the steps in Choosing the Main Java Class for Starting the Application. The **Classpath** panel opens.


InstallAnywhere 2015 Premier with Virtualization and Cloud - M...

Classpath

Click "Automatically Set Classpath" to do this task automatically, or choose any items on the list and select the "In Classpath" checkbox to add them to the classpath.

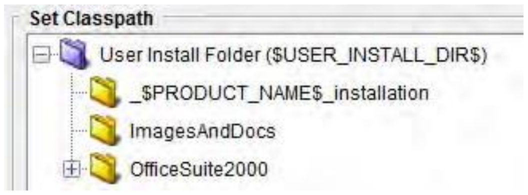
Set Classpath

- User Install Folder ($USER_INSTALL_DIR$)
  - _$PRODUCT_NAME$_installation
  - ImagesAndDocs
  - OfficeSuite2000

In classpath

Indicates item 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.


Set Classpath

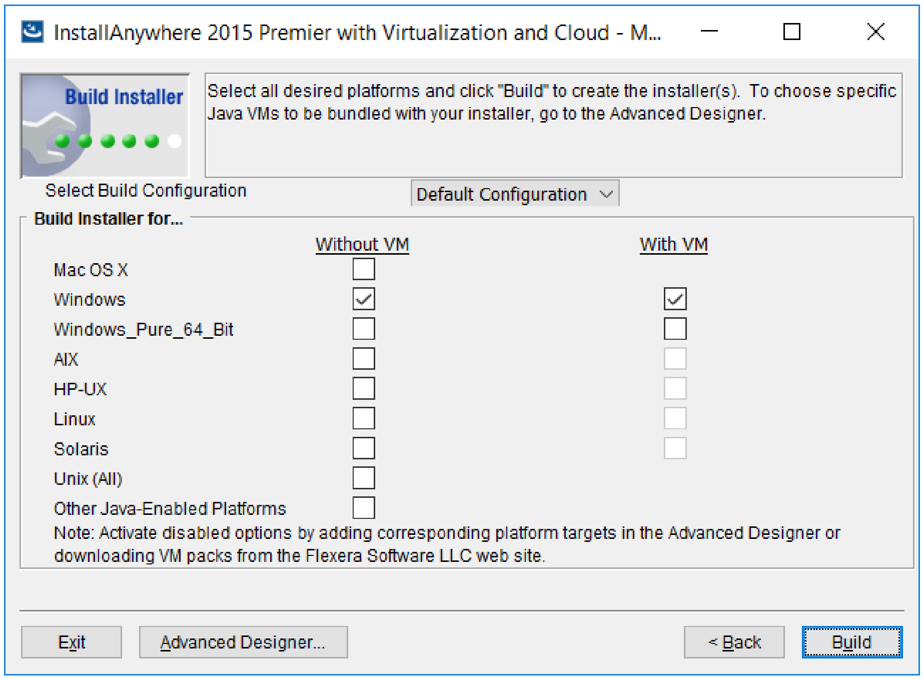
- User Install Folder ($USER_INSTALL_DIR$)
  - _$PRODUCT_NAME$_installation
  - ImagesAndDocs
  - OfficeSuite2000

3. Click **Next**. The **Build Installer** panel opens.

4. Continue with the steps in Building the Installer.

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.


InstallAnywhere 2015 Premier with Virtualization and Cloud - M...
Build Installer

Select all desired platforms and click "Build" to create the installer(s). To choose specific Java VMs to be bundled with your installer, go to the Advanced Designer.

Select Build Configuration: Default Configuration

Build Installer for...

	Without VM	With VM
Mac OS X	☐	
Windows	✓	✓
Windows_Pure_64_Bit	☐	☐
AIX	☐	☐
HP-UX	☐	☐
Linux	☐	☐
Solaris	☐	☐
Unix (All)	☐	
Other Java-Enabled Platforms	☐	

Note: Activate disabled options by adding corresponding platform targets in the Advanced Designer or downloading VM packs from the Flexera Software LLC web site.

Figure B-1: Build Installer Panel of Project Wizard

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.

Note • *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.*

Task ***To build the installer:***

1. Perform the steps in Setting the Classpath. The **Build Installer** panel opens.
2. Choose the desired destination platforms.
3. Click **Build**. InstallAnywhere builds the project and places the installer folder in a sub-directory of the same location as the project file. This location cannot be changed.

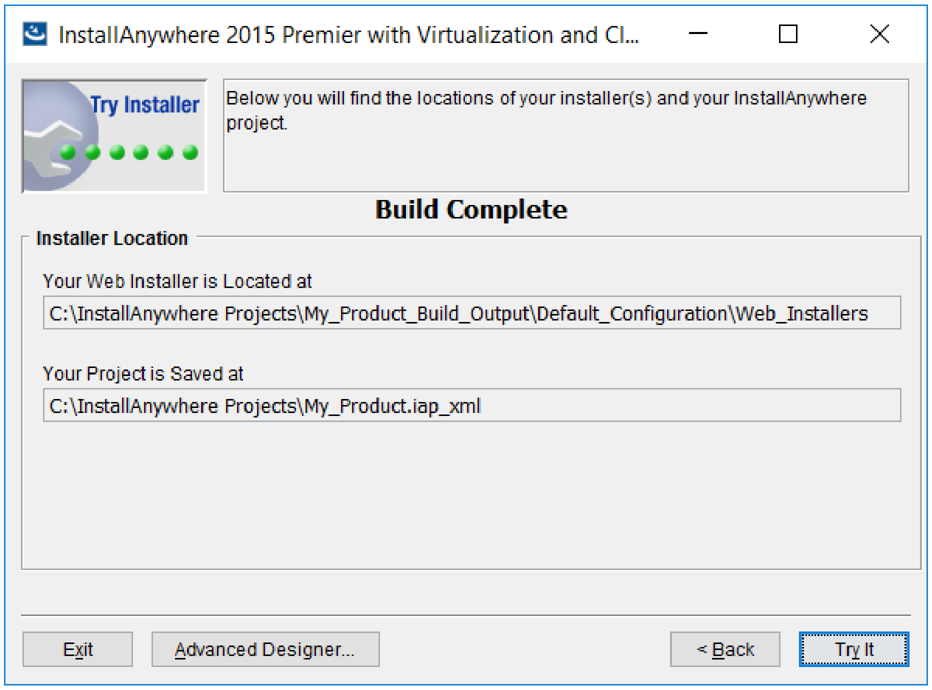
   When the build is complete, the **Try Installer** panel opens.
4. Continue with the steps in Testing the Project.

Testing the Project

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

Task ***To test the project:***

1. Perform the steps in Building the Installer. When the build is complete, the **Try Installer** panel opens.



Below you will find the locations of your installer(s) and your InstallAnywhere project.

Build Complete

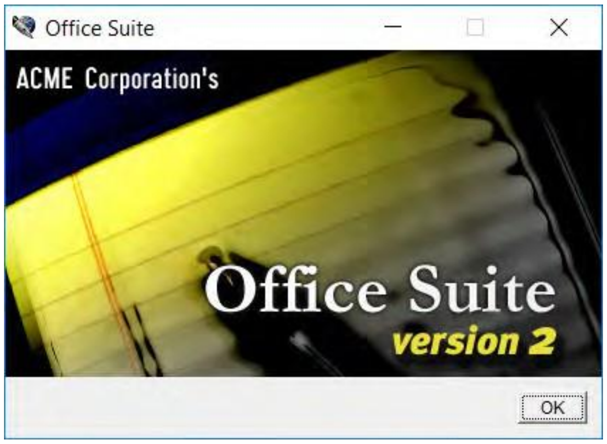
Installer Location

Your Web Installer is Located at
C:\InstallAnywhere Projects\My_Product_Build_Output\Default_Configuration\Web_Installers

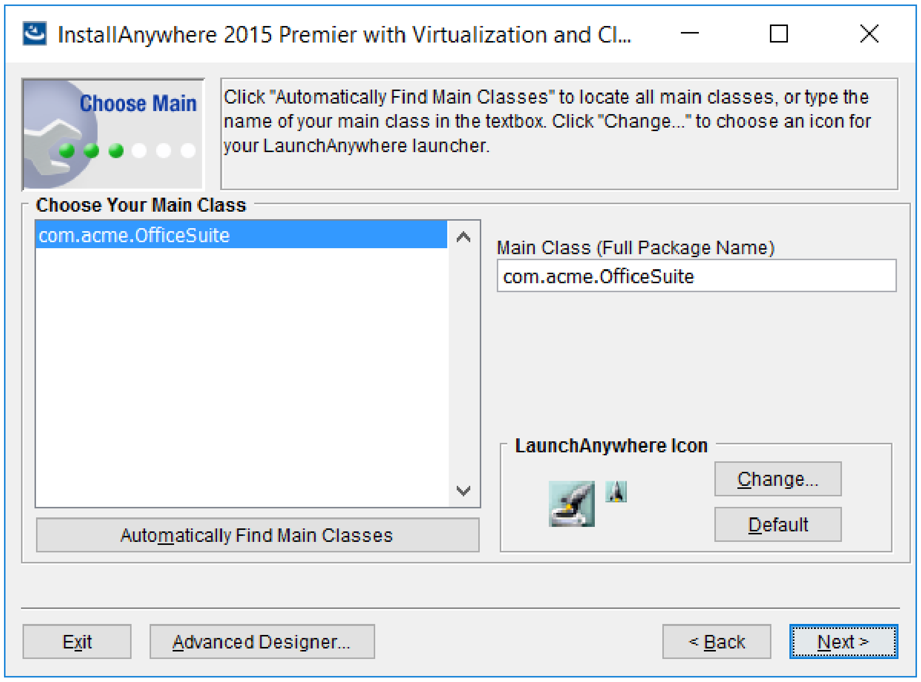
Your Project is Saved at
C:\InstallAnywhere Projects\My_Product.iap_xml

2. Click **Try It**. InstallAnywhere launches the installer.
3. Follow the directions to install the application.
4. After installing the application, do the following to launch the application:
   - Windows—Go to the **OfficeSuite** program group and choose **OfficeSuite**.
   - Unix—Change to the directory where the program was installed and enter **`OfficeSuite`**.
   - Mac OS X—Double-click the **OfficeSuite** icon on the desktop.

   The Office Suite splash screen opens:



5. Click **OK**. **OfficeSuite for Java** opens.


InstallAnywhere 2015 Premier with Virtualization and Cl...
Choose Main

Click "Automatically Find Main Classes" to locate all main classes, or type the name of your main class in the textbox. Click "Change..." to choose an icon for your LaunchAnywhere launcher.

Choose Your Main Class

com.acme.OfficeSuite

Main Class (Full Package Name)

com.acme.OfficeSuite

LaunchAnywhere Icon
Change...
Default
Automatically Find Main Classes
Exit
Advanced Designer...
< Back
Next >

6. 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.*

7. Click **Exit** to close the Project Wizard.

Building for a Platform Other Than That on Which the Installer is Being Developed

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.

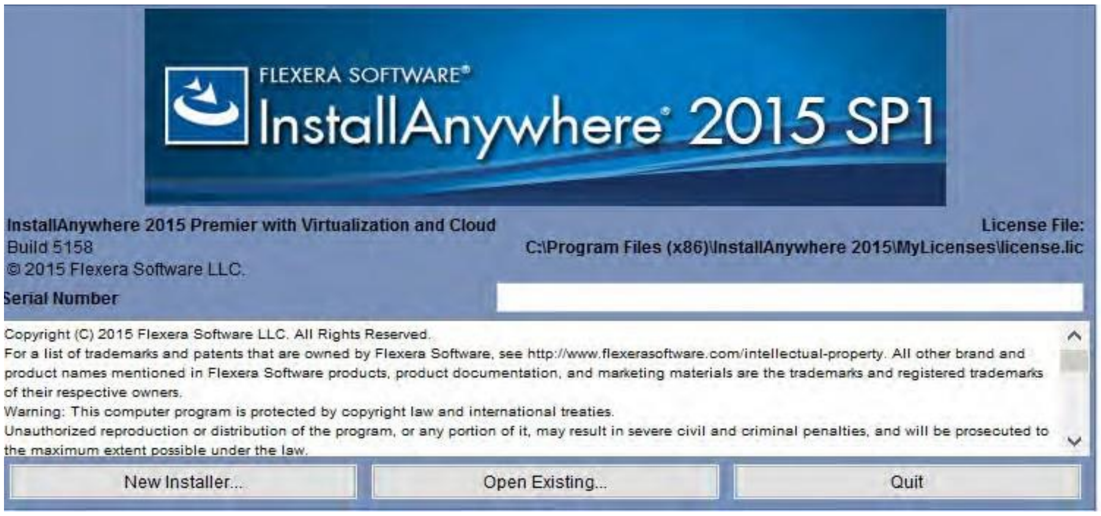
The `Build_Output` folder also contains the `Web_Installers`, and `CDROM_installers`. From within each of these sub-directories, choose the platform to test. For the CDROM installer, transfer the entire contents of the `CDROM_Installers` sub-directory.

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.

Task ***To create the TrainApp project:***

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



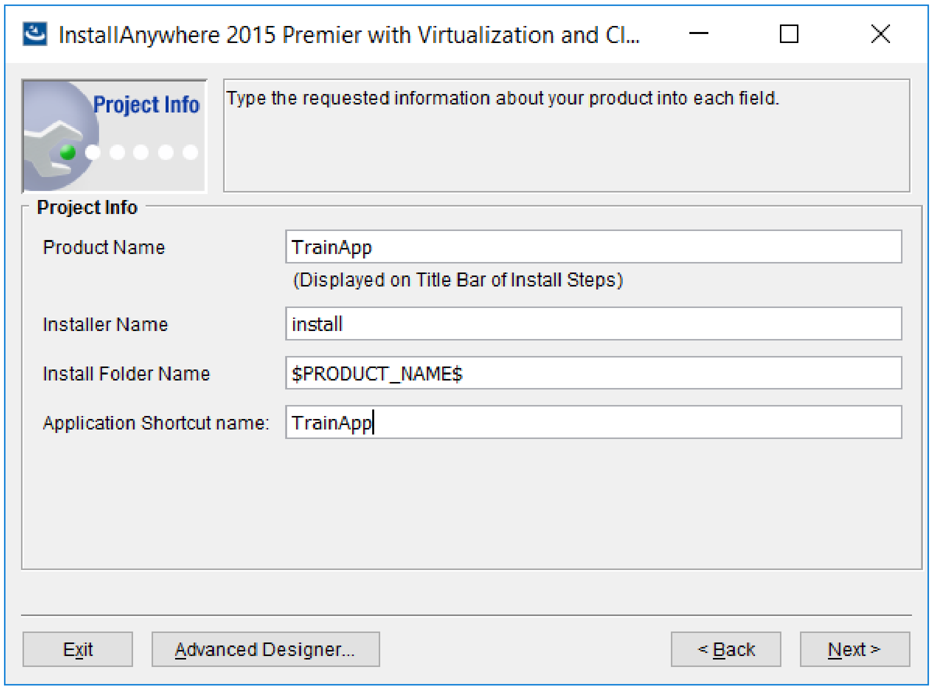
InstallAnywhere 2015 Premier with Virtualization and Cloud
Build 5158
© 2015 Flexera Software LLC.

License File:
C:\Program Files (x86)\InstallAnywhere 2015\MyLicenses\license.lic

Serial Number

Copyright (C) 2015 Flexera Software LLC. All Rights Reserved.
For a list of trademarks and patents that are owned by Flexera Software, see http://www.flexerasoftware.com/intellectual-property. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.
Warning: This computer program is protected by copyright law and international treaties.
Unauthorized reproduction or distribution of the program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

3. Select **Basic Project Template** and then click **Save As**, saving a new project file called `TrainApp.iap_xml` to a directory of your choice.
4. After saving the project, click **Next** to begin the Project Wizard. The **Project Info** panel opens.


InstallAnywhere 2015 Premier with Virtualization and Cl...

Project Info

Type the requested information about your product into each field.

Project Info

Product Name: TrainApp
(Displayed on Title Bar of Install Steps)

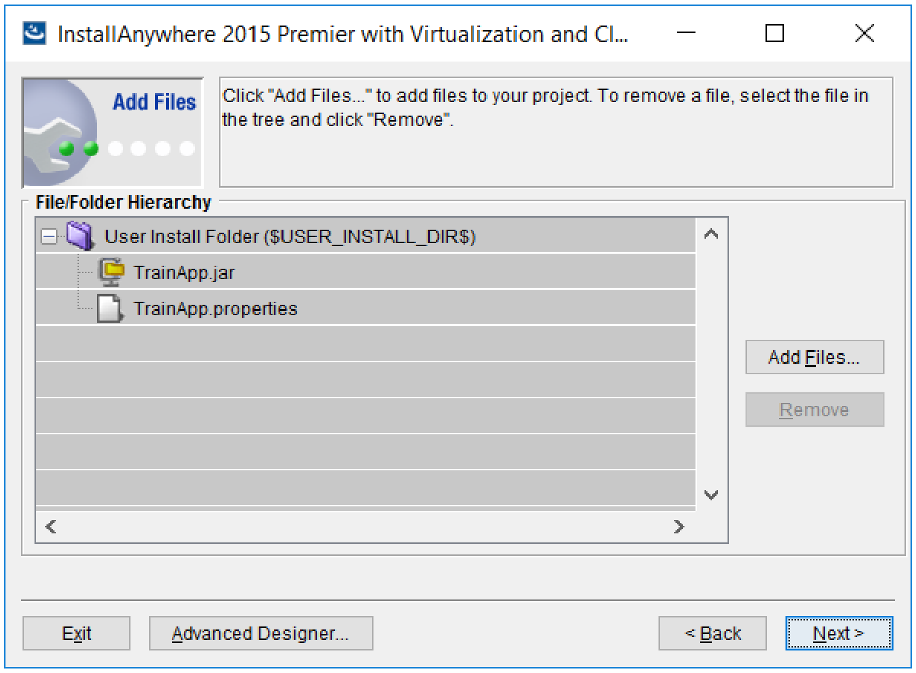
Installer Name: install

Install Folder Name: $PRODUCT_NAME$

Application Shortcut name: TrainApp

5. On the **Project Info** panel, accept the default settings and click **Next**. The **Add Files** panel opens.

6. Click **Add Files** and browse for `TrainApp.jar` and `TrainApp.properties`, installing both to the default installation location `$USER_INSTALL_DIR$`.


InstallAnywhere 2015 Premier with Virtualization and Cl...

Add Files

Click "Add Files..." to add files to your project. To remove a file, select the file in the tree and click "Remove".

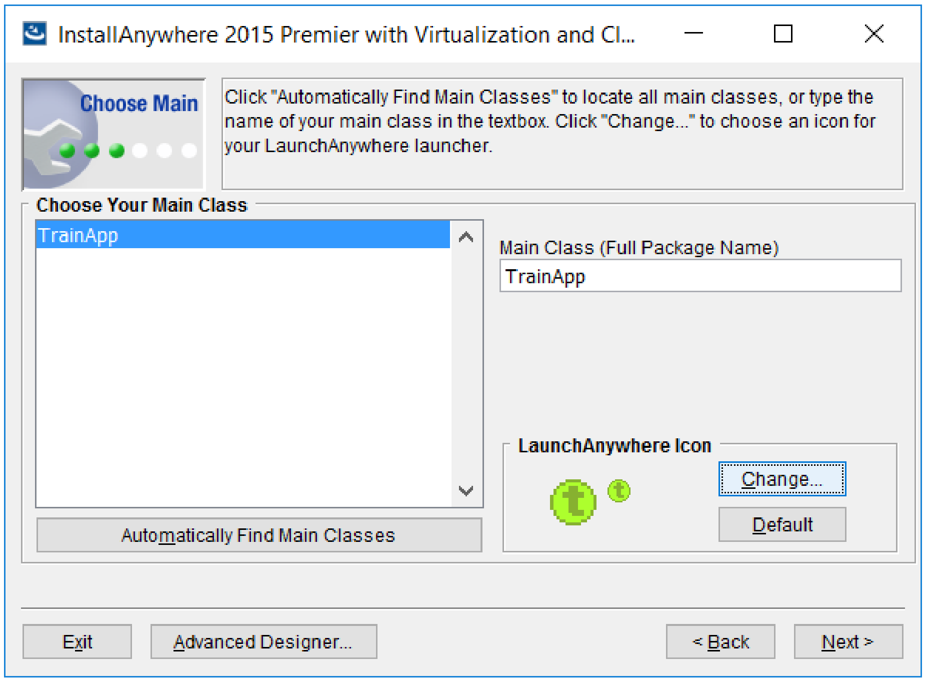
File/Folder Hierarchy

User Install Folder ($USER_INSTALL_DIR$)
TrainApp.jar
TrainApp.properties

7. Click **Next** to continue to the **Choose Main** panel.

8. 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`.

9. In the **LaunchAnywhere Icon** area of the panel, click **Change** and select `wizicon.gif`. The new icon is displayed in the **LaunchAnywhere Icon** area.


InstallAnywhere 2015 Premier with Virtualization and Cl...

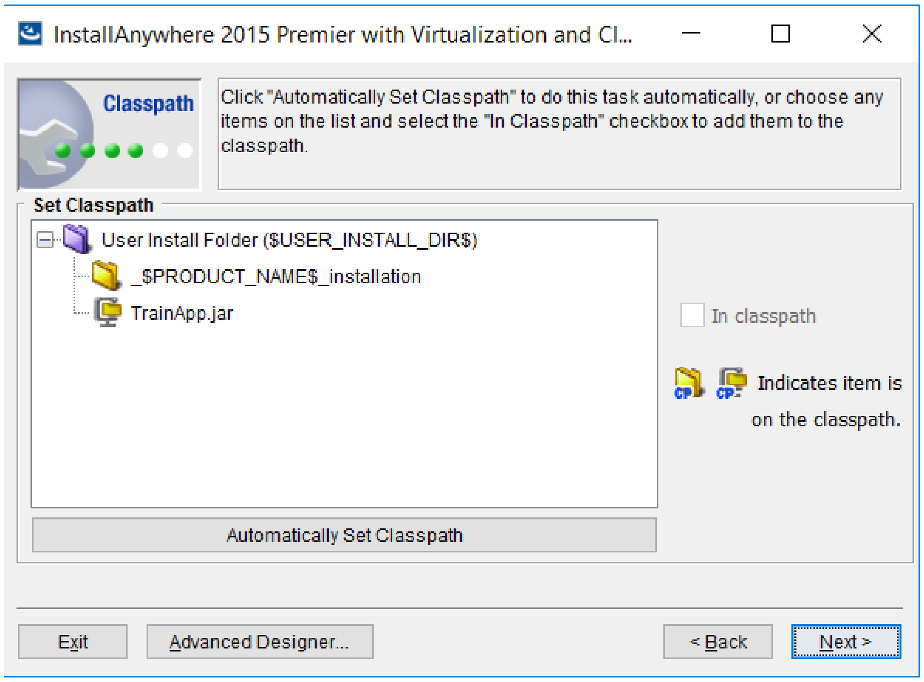
Click "Automatically Find Main Classes" to locate all main classes, or type the name of your main class in the textbox. Click "Change..." to choose an icon for your LaunchAnywhere launcher.

Choose Your Main Class: TrainApp

Main Class (Full Package Name): TrainApp

10. Click **Next** to continue to the **Classpath** panel.

11. Again, because **TrainApp** is a Java application, having its `.jar` file on the class path can be useful. Therefore, click **Automatically Set Classpath** to add `TrainApp.jar` to the class path. A CP overlay icon is added to the `TrainApp.jar` icon to indicates that it is on the class path.


InstallAnywhere 2015 Premier with Virtualization and Cl...
Classpath

Click "Automatically Set Classpath" to do this task automatically, or choose any items on the list and select the "In Classpath" checkbox to add them to the classpath.

Set Classpath

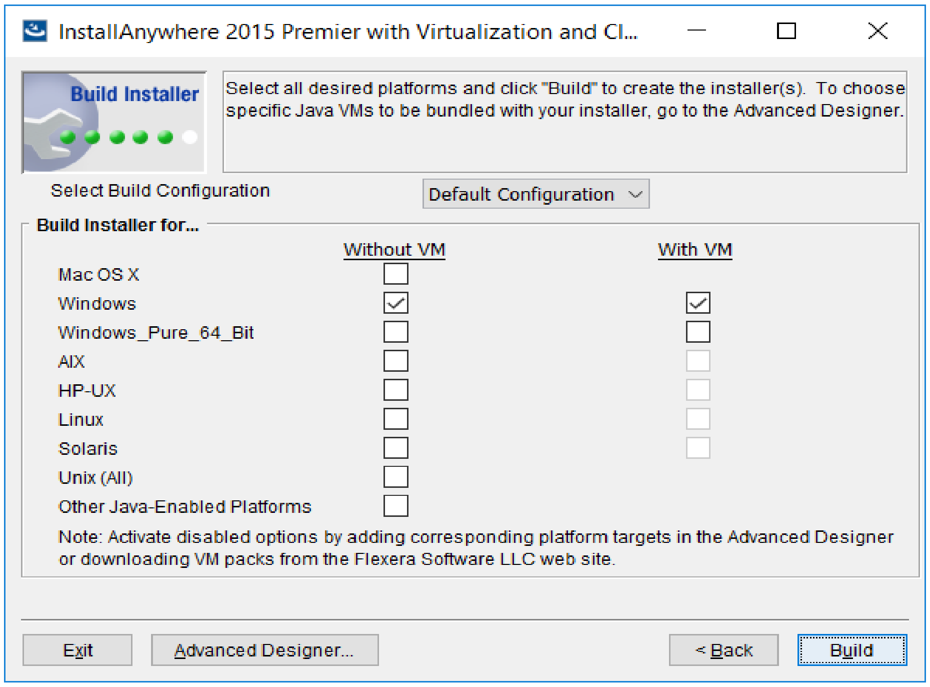
- User Install Folder ($USER_INSTALL_DIR$)
  - _$PRODUCT_NAME$_installation
  - TrainApp.jar

[ ] In classpath

Indicates item is on the classpath.

Automatically Set Classpath
Exit
Advanced Designer...
< Back
Next >

12. Click **Next** to continue to the **Build Installer** panel.
13. If desired, click **Build** to build an installation image for testing.



Select all desired platforms and click "Build" to create the installer(s). To choose specific Java VMs to be bundled with your installer, go to the Advanced Designer.

Select Build Configuration: Default Configuration

Build Installer for...

	Without VM	With VM
Mac OS X	☐	
Windows	✓	✓
Windows_Pure_64_Bit	☐	☐
AIX	☐	☐
HP-UX	☐	☐
Linux	☐	☐
Solaris	☐	☐
Unix (All)	☐	
Other Java-Enabled Platforms	☐	

Note: Activate disabled options by adding corresponding platform targets in the Advanced Designer or downloading VM packs from the Flexera Software LLC web site.

14. 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. Note the following:
    - **Installation location**—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.

- **Shortcut**—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.
- **Uninstallation**—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.

15. Finally, click the **Advanced Designer** button at the bottom of the Project Wizard to prepare for other exercises.

Chapter 3 Exercises: Introduction to the Advanced Designer

In these exercises, you will build an installer with the Advanced Designer and examine the installation project in the Advanced Designer.

- Building an Installer with the Advanced Designer
- Examining the TrainApp Project in the Advanced Designer

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:

- Reviewing the Pre-Install Actions
- Defining the Installation Tasks
- Adding a LaunchAnywhere Executable to the Install Task
- Post-Install Actions
- Testing the Installer

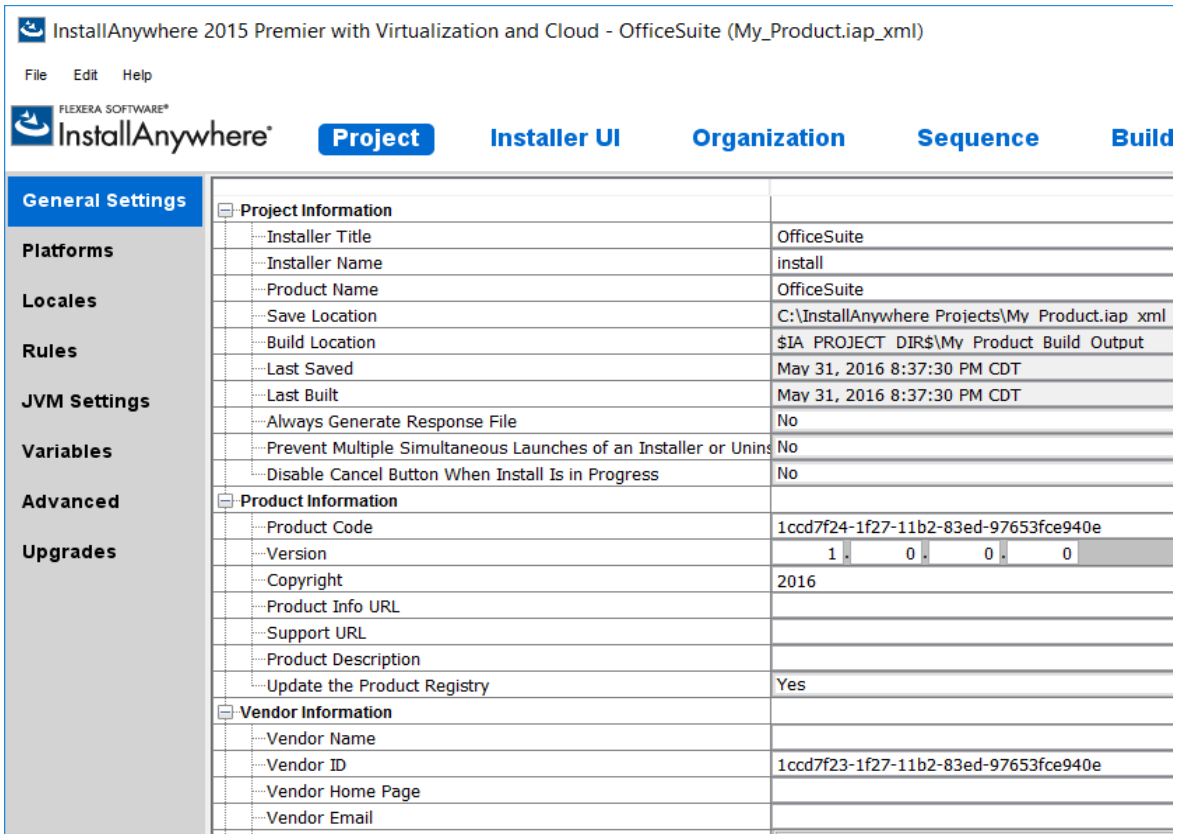
Reviewing the Pre-Install Actions

To review the **Pre-Install** actions, perform the following steps:

Task ***To review the Pre-Install actions:***

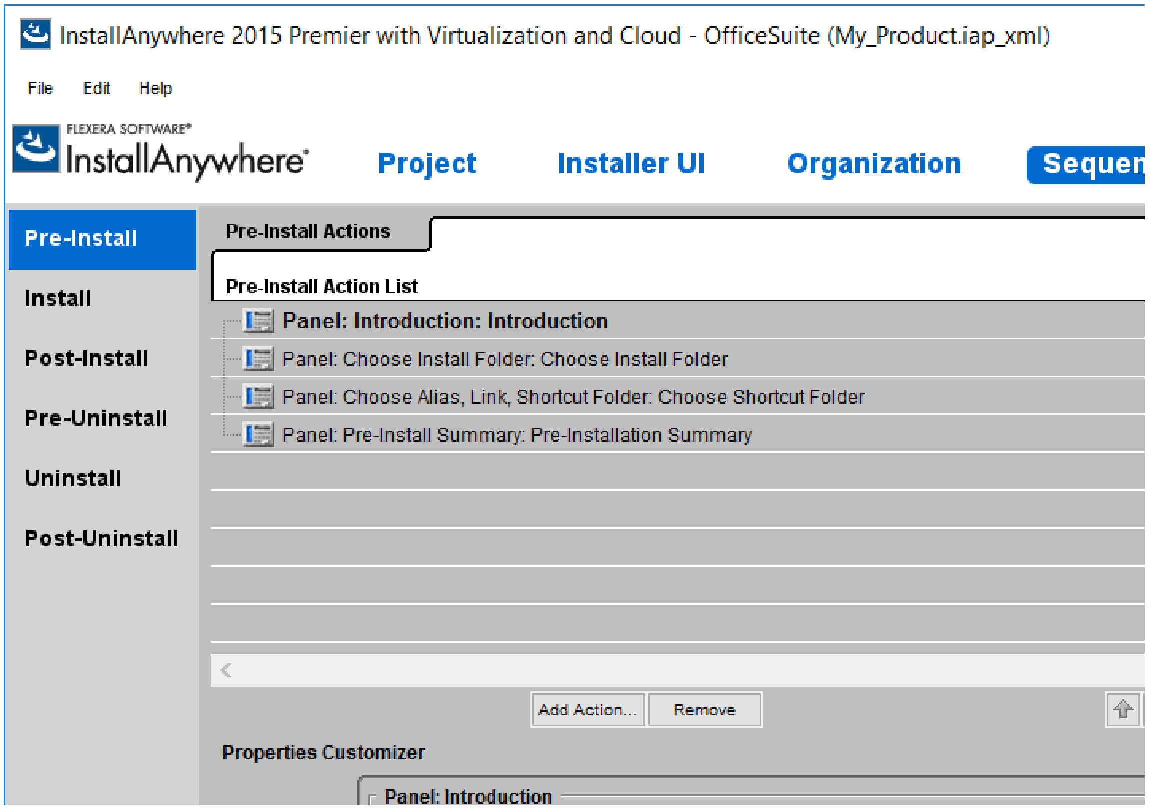
1. Launch InstallAnywhere. The **New Project** dialog box opens.
2. Select the **Open Existing Project** option.
3. Select the `My_Product` project you created in Chapter 2 Exercises: Starting a Project Using the InstallAnywhere Project Wizard.

4. Click **Advanced Designer**. The **Project > Info** subtask of the Advanced Designer opens.



Project Information	
Installer Title	OfficeSuite
Installer Name	install
Product Name	OfficeSuite
Save Location	C:\InstallAnywhere Projects\My_Product.iap_xml
Build Location	$IA_PROJECT_DIR$\My_Product_Build_Output
Last Saved	May 31, 2016 8:37:30 PM CDT
Last Built	May 31, 2016 8:37:30 PM CDT
Always Generate Response File	No
Prevent Multiple Simultaneous Launches of an Installer or Unins	No
Disable Cancel Button When Install Is in Progress	No
Product Information	
Product Code	1ccd7f24-1f27-11b2-83ed-97653fce940e
Version	1 . 0 . 0 . 0
Copyright	2016
Product Info URL	
Support URL	
Product Description	
Update the Product Registry	Yes
Vendor Information	
Vendor Name	
Vendor ID	1ccd7f23-1f27-11b2-83ed-97653fce940e
Vendor Home Page	
Vendor Email	

5. On the top screen, highlight the Sequence option. Open the **Pre-Install** task. The **Pre-Install** task sets the panels and action that occur prior to the installation of files.



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

By default, a new InstallAnywhere project contains the following panels in the **Pre-Install Action List**:

Panel	Description
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 symbolic links (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.

Note the following:

- **Default panel order**—Panel 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.
- **Changing panel order**—The order of panels and actions can be manipulated using the arrow buttons in the middle right of the Advanced Designer screen.
- **Customizing panels**—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.

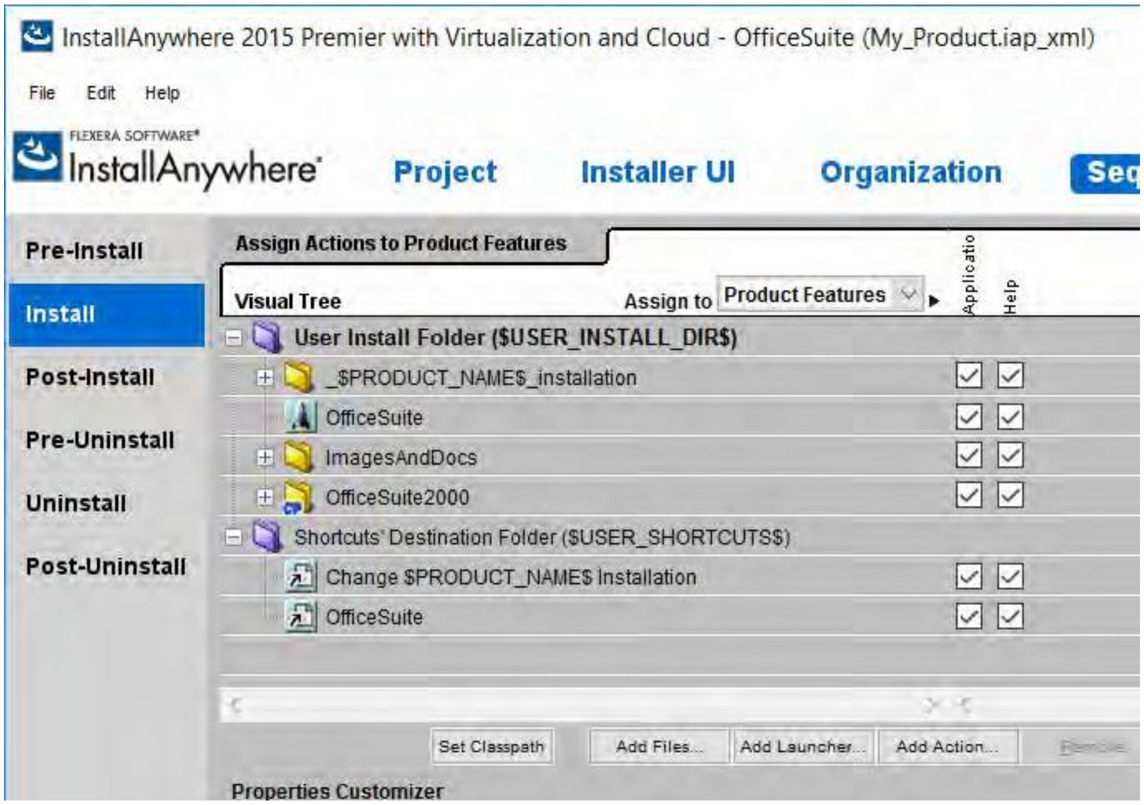
6. Continue with the steps in Defining the Installation Tasks.

Defining the Installation Tasks

In this task, you will define the installation tasks for the TrainApp project.

Task ***To define the installation tasks:***

1. Perform the steps in Reviewing the Pre-Install Actions.
2. Select the **Install** task from the far left side of the Advanced Designer. The **Install** task opens.



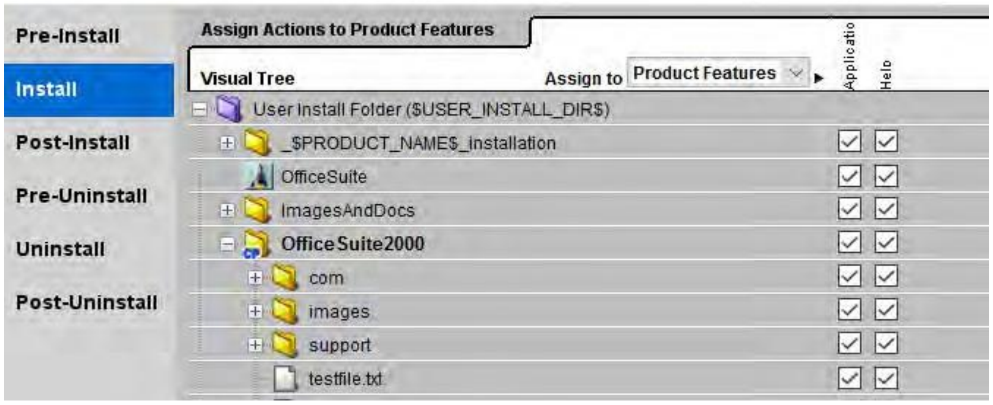
The **Install** task defines the files to install, the folder location to install those files, and the order of the tasks that need to happen as the files are being installed. Note the following regarding the **Install** task:

- **Uninstaller actions**—By default, the InstallAnywhere **Install** task has a folder called `_$PRODUCT_NAME$_installation`, which contains any InstallAnywhere uninstaller actions, and a comment action with instructions pertaining to the uninstaller.
- **Task order**—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 is generally best to have the uninstaller creation action first.*

3. Open the **User Install Folder** to view the `OfficeSuite2000` and `ImagesAndDocs` directories and files, which you added in Adding Files to the Project using the Project Wizard.



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.

4. Continue with the steps in Adding a LaunchAnywhere Executable to the Install Task.

Adding a LaunchAnywhere Executable to the Install Task

In this section, you will add a LaunchAnywhere executable to the Install task by performing the following tasks:

- Select the Add Launcher Button
- Customize the Launcher
- Set the InstallAnywhere Classpath

Select the Add Launcher Button

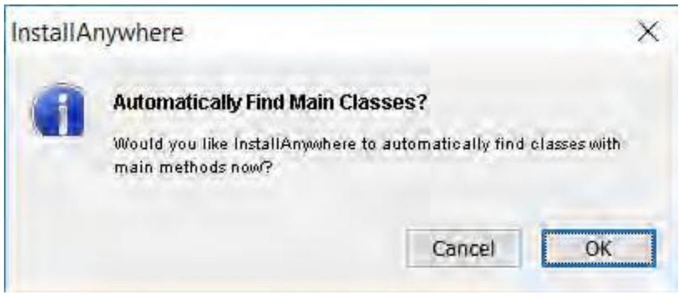
A LaunchAnywhere Executable (LAX) is a unique native executable, created by InstallAnywhere, that is used to launch a Java application. While the InstallAnywhere Wizard specifically asks to select a main class and automatically creates a single launcher, the Advanced Designer allows you to add as many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

There are two ways to add a LaunchAnywhere Launcher to an InstallAnywhere project file. The **Create LaunchAnywhere for Java Application** option can be:

- Selected from the **Add Action** palette, OR
- Added by clicking the **Add Launcher** button on the middle control bar in the Advanced Designer.

Task ***To select the Add Launcher button:***

1. Perform the steps in Defining the Installation Tasks.
2. Highlight the **User Install Folder** in the Advanced Designer and click the **Add Launcher** button. A message dialog box opens asking whether you want InstallAnywhere to automatically find classes with main methods.

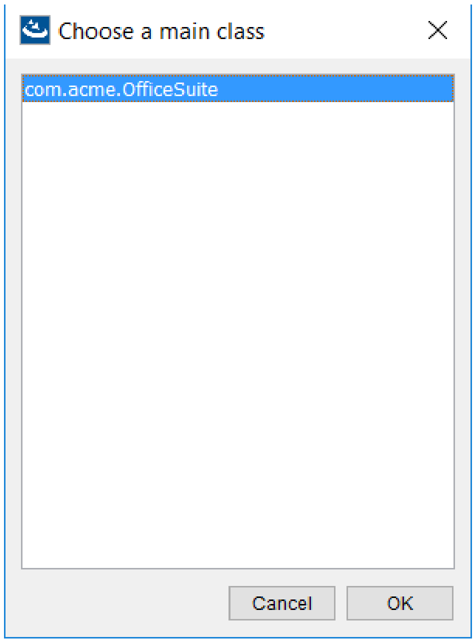

InstallAnywhere

Automatically Find Main Classes?

Would you like InstallAnywhere to automatically find classes with main methods now?

When adding a launcher, InstallAnywhere will automatically inspects the added files (including inspecting the `.jar` and/or `.zip` files) to find class files with Main Methods specified.

3. Click **OK**. The **Choose a main class** dialog box opens.


Choose a main class

com.acme.OfficeSuite

4. Since OfficeSuite is a simple project, you are presented with only the `com.acme.OfficeSuite` class. Choose the `com.acme.OfficeSuite` as the main class for this application.
5. 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.*

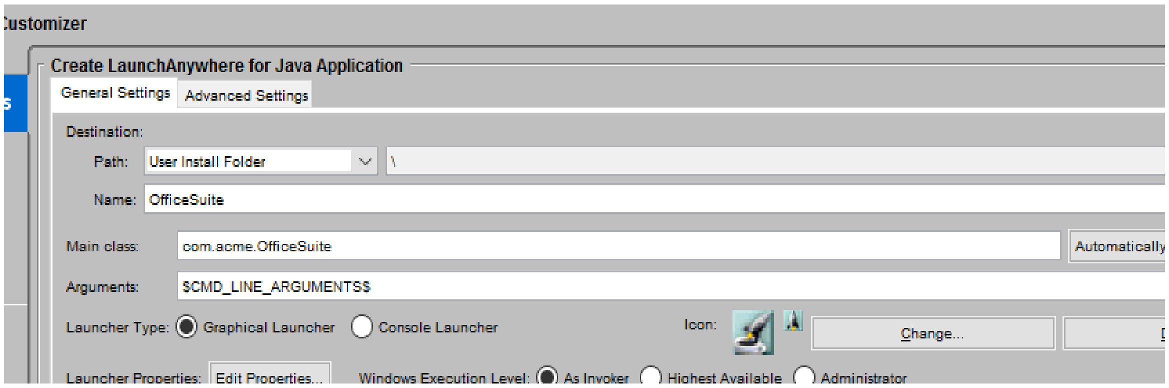
6. Continue with the steps in Customize the Launcher.

Customize the Launcher

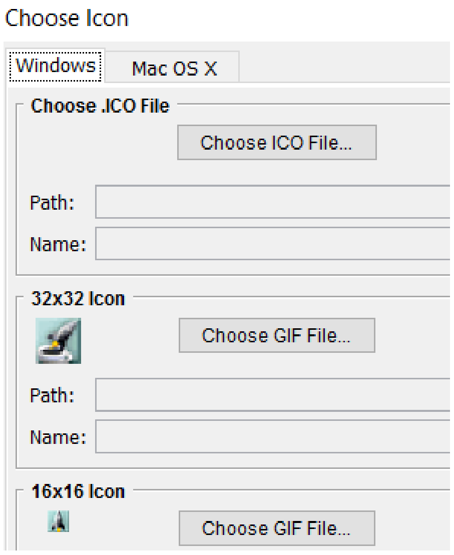
The appearance the launcher will have as a shortcut can now be customized.

Task ***To customize the launcher:***

1. Perform the steps in Select the Add Launcher Button.
2. Select the **OfficeSuite** launcher in the **User Install Folder**. 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.


Customizer
Create LaunchAnywhere for Java Application
General Settings | Advanced Settings
Destination:
Path: User Install Folder \
Name: OfficeSuite
Main class: com.acme.OfficeSuite
Arguments: $CMD_LINE_ARGUMENTS$
Launcher Type: [✓ Graphical Launcher] Console Launcher
Windows Execution Level: [✓ As Invoker] Highest Available Administrator

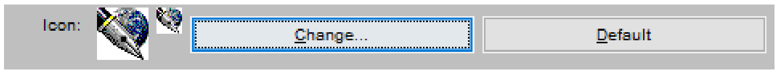
3. The **Icon** field (below the **Arguments** field) displays the icon associated with the launcher. The default icon is a teal tile with a coffee cup, and a rocket ship icon. Click **Change** to alter the icon. The **Choose Icon** dialog box opens.



4. Click **Choose GIF File**. The **Choose a GIF image** dialog box opens.
5. Select the `officeIcon.gif` found in the `OfficeSuiteSourceFiles\ImagesAndDocs` directory.

***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).*

6. Click **OK** to close the **Choose Icon** dialog box. The new icon is now displayed in the **Icon** area of the customizer.



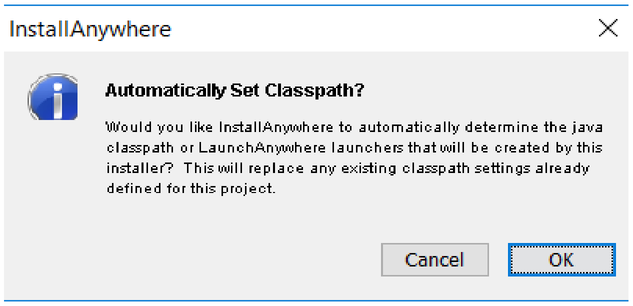
7. Continue with the steps in Set the InstallAnywhere Classpath.

Set the InstallAnywhere Classpath

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

Task ***To set the InstallAnywhere classpath:***

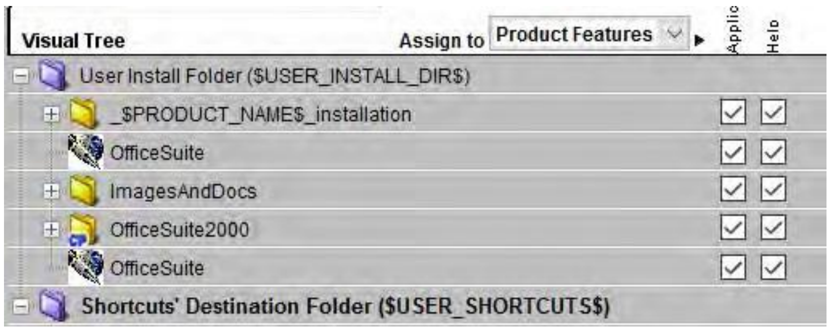
1. Perform the steps in Customize the Launcher.
2. In the **Install** task of the Advanced Designer, click the **Set Classpath** button. The **Automatically Set Classpath?** dialog box opens.


InstallAnywhere

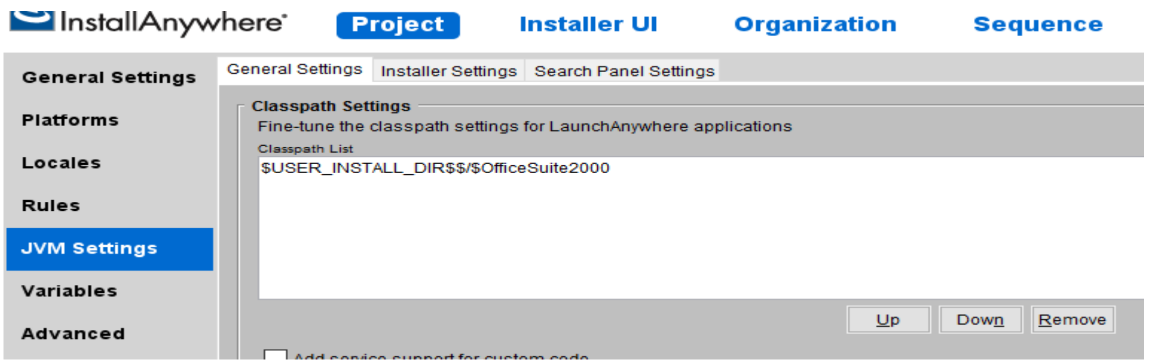
Automatically Set Classpath?

Would you like InstallAnywhere to automatically determine the java classpath or LaunchAnywhere launchers that will be created by this installer? This will replace any existing classpath settings already defined for this project.

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


Visual Tree
Assign to
Product Features
Applic
Help
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
OfficeSuite
ImagesAndDocs
OfficeSuite2000
OfficeSuite
Shortcuts' Destination Folder ($USER_SHORTCUTS$)

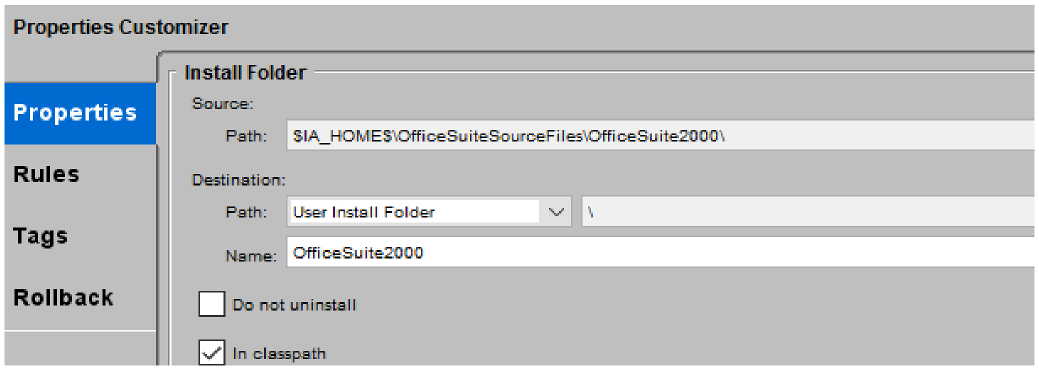
4. Open the **Project > Java** subtask along the left side of the Advanced Designer window to view the classpath as determined by the **Set Classpath** action.



Classpath Settings
Fine-tune the classpath settings for LaunchAnywhere applications
Classpath List
$USER_INSTALL_DIR$$/$OfficeSuite2000

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.

***Note** • If a file is added mistakenly on the **Classpath**, it can be removed by selecting that file in the **Visual Tree** on the **Install** task and un-checking the **In Classpath** option box in the Customizer:*



Install Folder
Source:
Path: IA_HOME\OfficeSuiteSourceFiles\OfficeSuite2000\
Destination:
Path: User Install Folder \
Name: OfficeSuite2000
☐ Do not uninstall
✓ In classpath

5. Continue with the steps in Post-Install Actions.

Post-Install Actions

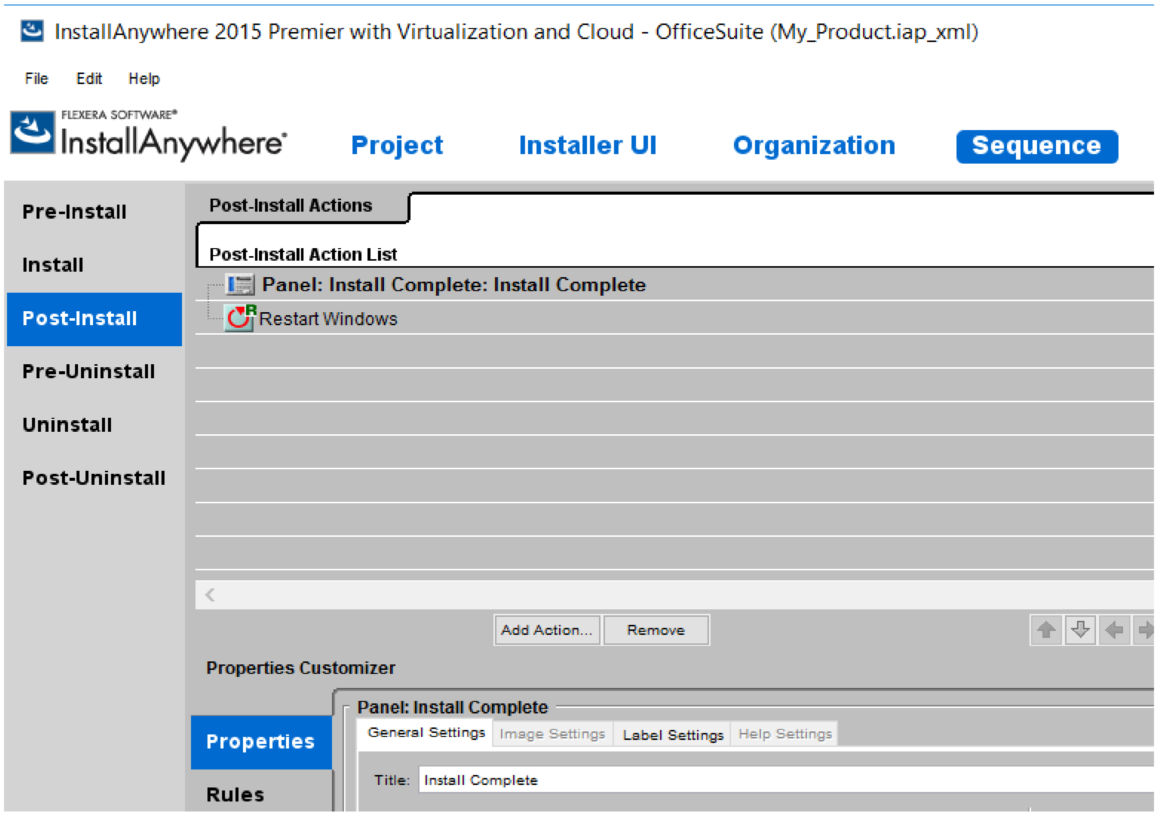
The **Post-Install** task list specifies actions and panels to occur after the installation of files. Like **Pre-Install**, the **Post- Install** step is ordered with the top actions occurring first. By default, InstallAnywhere has added two actions to the InstallAnywhere project. These actions are:

- **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. In this example, you will review and existing rule and add a new rule.

Task ***To define Post-Install actions:***

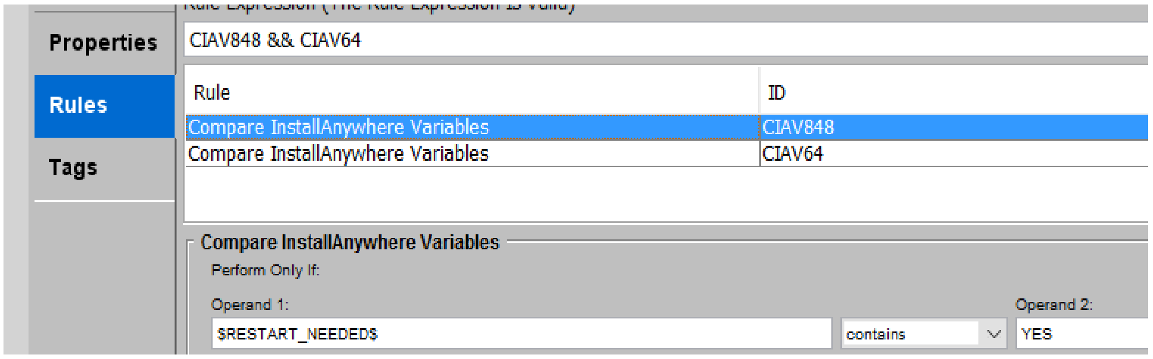
1. Perform the steps in Set the InstallAnywhere Classpath.
2. Open the **Post-Install** task. The **Post-Install** task opens.



Panel: Install Complete: Install Complete
Restart Windows

Panel: Install Complete
Title: Install Complete

3. Select the **Restart Windows** action in the OfficeSuite Project. The **Restart Windows** customizer is displayed.
4. In the customizer, select the **Rules** tab. The **Rules** tab of the **Restart Windows** customizer opens.



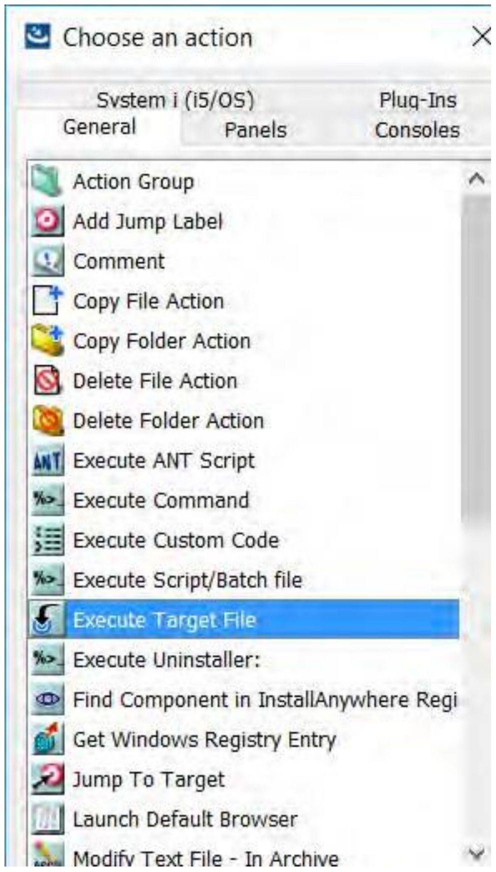
CIAV848 && CIAV64

Rule	ID
Compare InstallAnywhere Variables	CIAV848
Compare InstallAnywhere Variables	CIAV64

Operand 1: $RESTART_NEEDED$ contains Operand 2: YES

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.

5. Click the **Add Action** button. The **Action** palette opens, which is divided by tabs that vary based on the task that is active at the time the palette is called.



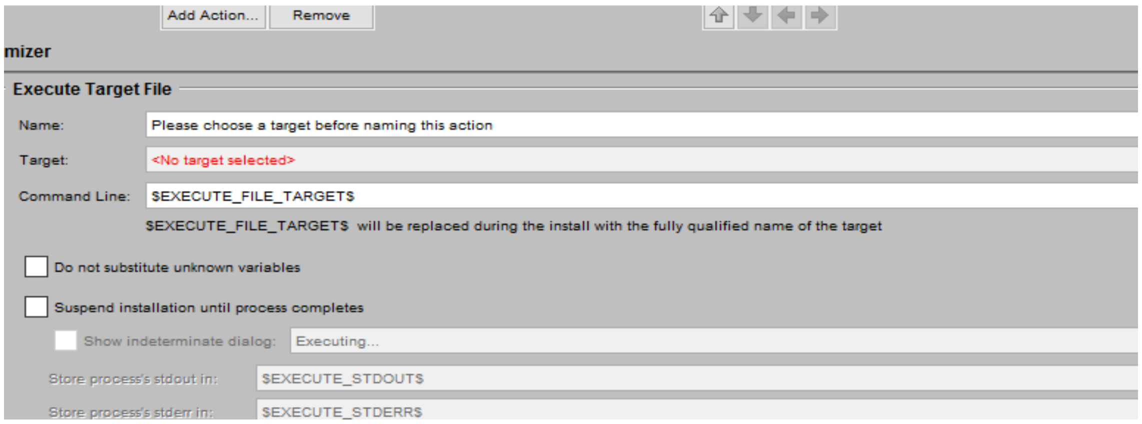
6. Select **Execute Target File** found under the **General** tab.

   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.

7. To add the action, click **Add**. The **Add Action** palette remains open so additional actions may be added.

8. Click **Close** to close the **Action** palette.

9. In the **Post-Install Action List**, select **Execute Target File**. The **Execute Target File** customizer opens.



Execute Target File

Name: Please choose a target before naming this action

Target: <No target selected>

Command Line: $EXECUTE_FILE_TARGET$

$EXECUTE_FILE_TARGET$ will be replaced during the install with the fully qualified name of the target

Do not substitute unknown variables

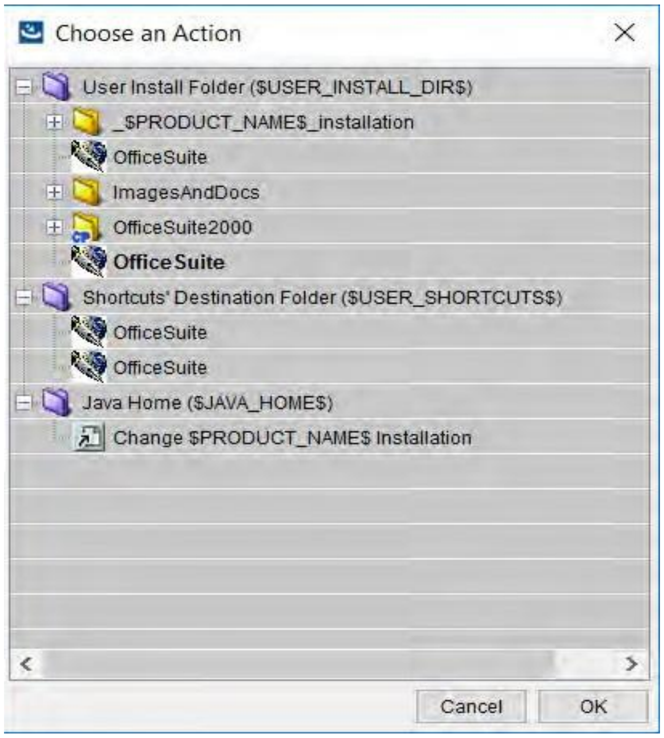
Suspend installation until process completes

Show indeterminate dialog: Executing...

Store process's stdout in: $EXECUTE_STDOUT$

Store process's stderr in: $EXECUTE_STDERR$

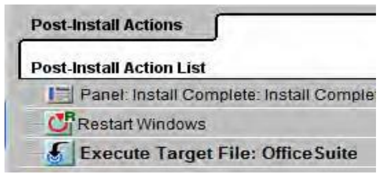
10. On the **Properties** tab of the **Execute Target File** customizer, click the **Choose Target** button. The **Choose an Action** dialog box opens, representing the file installation tree specified in the **Install** task. Files can be executed in this stage.


Choose an Action
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
OfficeSuite
ImagesAndDocs
OfficeSuite2000
Office Suite
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
OfficeSuite
OfficeSuite
Java Home ($JAVA_HOME$)
Change $PRODUCT_NAME$ Installation
Cancel
OK

11. To execute the just-installed **OfficeSuite** application, choose the launcher for that application and click **OK**.

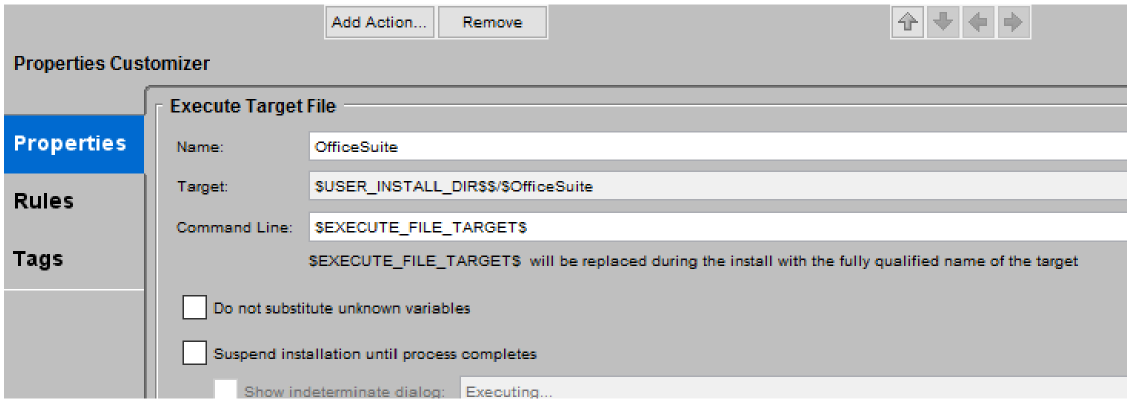
Important *• Choose the actual OfficeSuite Launcher, and not the shortcut (which should share the same Icon). Shortcuts, especially on Windows and Mac OS systems, are pointers and are not inherently executable. InstallAnywhere will not execute a shortcut.*

The **Execute Target File** action is now listed in the **Post-Install Action List**.


Post-Install Actions
Post-Install Action List
Panel: Install Complete: Install Comple
Restart Windows
Execute Target File: OfficeSuite

***Important** • If the **Execute Target File** 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.*

Information about the selected target file is displayed in the **Execute Target File** customizer:



Name: OfficeSuite

Target: $USER_INSTALL_DIR$$/$OfficeSuite

Command Line: $EXECUTE_FILE_TARGET$

$EXECUTE_FILE_TARGET$ will be replaced during the install with the fully qualified name of the target

☐ Do not substitute unknown variables

☐ Suspend installation until process completes

☐ Show indeterminate dialog: Executing...

12. If you want to modify the command line used to execute the file, you can edit the entry in the **Command Line** field, such as adding a handler, or an argument to the execution:

    - To specify a handler, prepend an executable path.
    - To specify an argument, append a file path.
    - These paths MUST be absolute, but the paths can include InstallAnywhere variables.

***Important** • Do not remove or modify the* `$EXECUTE_FILE_TARGET$` *entry, as this represents the file to execute.*

13. The user experience for this action can be tailored by selecting the **Suspend installation until process completes** option. This is particularly useful in cases where a later step in the installation is dependent on the execution.
14. You can also choose the **Show indeterminate dialog** suboption to specify an indeterminate progress bar with a message. This suboption 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).
15. Continue with the steps in Building the Installer.

***Note** • For information on customizing the **Uninstall** task, see Chapter 8 Exercises: Customizing the Uninstaller.*

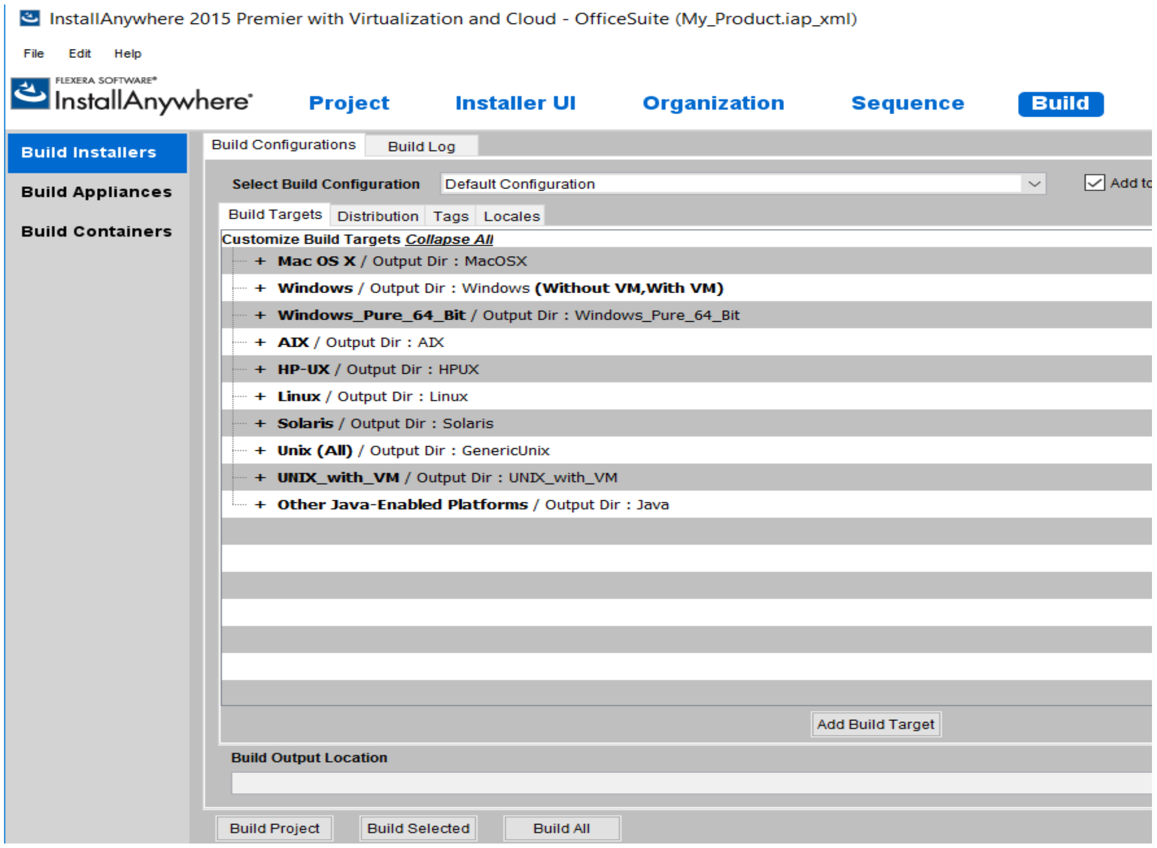
Building the Installer

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 can be set, and platform optimization and installer type can be specified.

***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.*

Task ***To build the installer:***

1. Perform the steps in Post-Install Actions.
2. Open the **Build** task. The **Build** task is displayed.



InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_xml)

+ **Mac OS X** / Output Dir : MacOSX
+ **Windows** / Output Dir : Windows **(Without VM,With VM)**
+ **Windows_Pure_64_Bit** / Output Dir : Windows_Pure_64_Bit
+ **AIX** / Output Dir : AIX
+ **HP-UX** / Output Dir : HPUX
+ **Linux** / Output Dir : Linux
+ **Solaris** / Output Dir : Solaris
+ **Unix (All)** / Output Dir : GenericUnix
+ **UNIX_with_VM** / Output Dir : UNIX_with_VM
+ **Other Java-Enabled Platforms** / Output Dir : Java

3. From the **Select Build Configuration** list on the **Build Configurations** tab, leave **Default Configuration** selected.

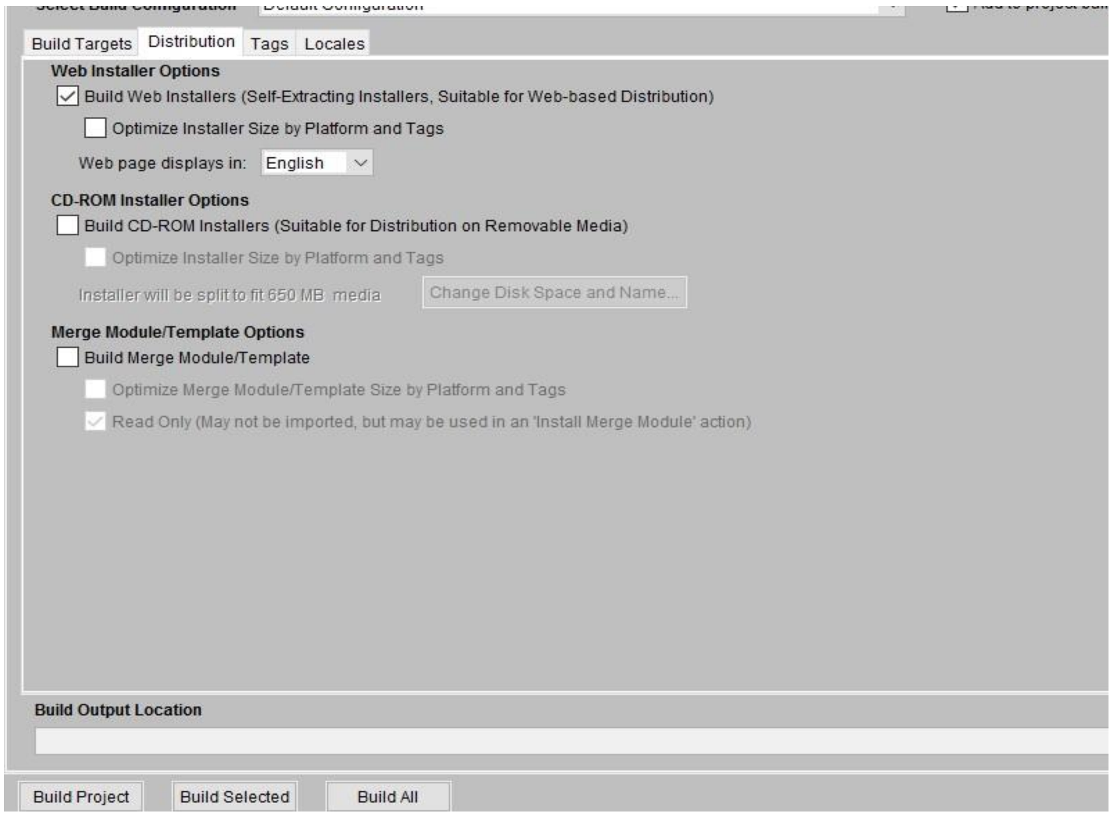
   ***Note** • For information on Build Configurations, see Chapter 13 Exercises: Creating and Editing Build Configurations.*

4. On the **Build Targets** subtab of the **Build Configurations** tab, select the platform(s) that you want to build an installer for by selecting the check boxes in the **Without VM** and/or **With VM** columns. For this tutorial, select **Windows**.

   Note the following regarding selecting a platform:

- If you select the **With VM** option, the installer will be bundled with a VM.
- The **With VM** option 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**.

5. Open the **Distribution** subtab of the **Build Configurations** tab.



Build Targets | Distribution | Tags | Locales

Web Installer Options
- [✓] Build Web Installers (Self-Extracting Installers, Suitable for Web-based Distribution)
  - [ ] Optimize Installer Size by Platform and Tags
  - Web page displays in: English

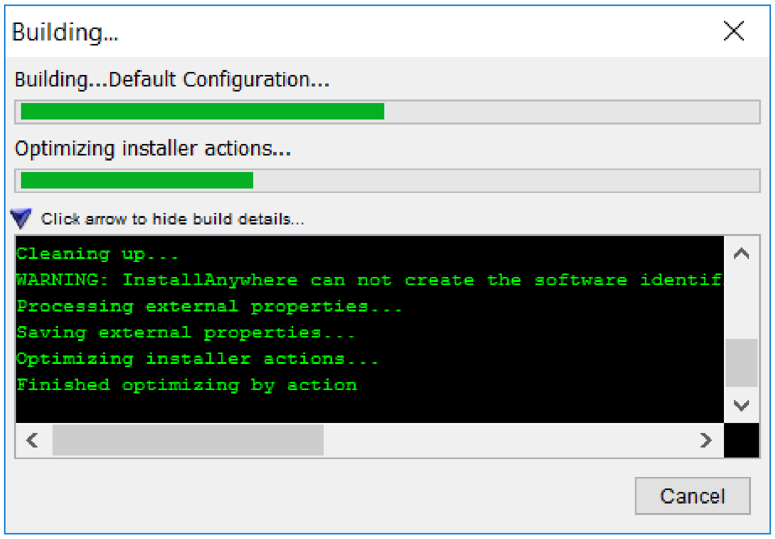
CD-ROM Installer Options
- [ ] Build CD-ROM Installers (Suitable for Distribution on Removable Media)
  - [ ] Optimize Installer Size by Platform and Tags
  - Installer will be split to fit 650 MB media

Merge Module/Template Options
- [ ] Build Merge Module/Template
  - [ ] Optimize Merge Module/Template Size by Platform and Tags
  - [✓] Read Only (May not be imported, but may be used in an 'Install Merge Module' action)

Build Output Location

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.

6. Click **Build Project** to build the OfficeSuite installer. The **Building** dialog box opens.
7. Click the blue arrow on the lower left of that dialog to view the build details.


Building...
Building...Default Configuration...
Optimizing installer actions...
Click arrow to hide build details...
Cancel

```
Cleaning up...
WARNING: InstallAnywhere can not create the software identif
Processing external properties...
Saving external properties...
Optimizing installer actions...
Finished optimizing by action
```

When the build is complete, the **Build Complete** dialog box opens. In this case, the build should take a minute or less.

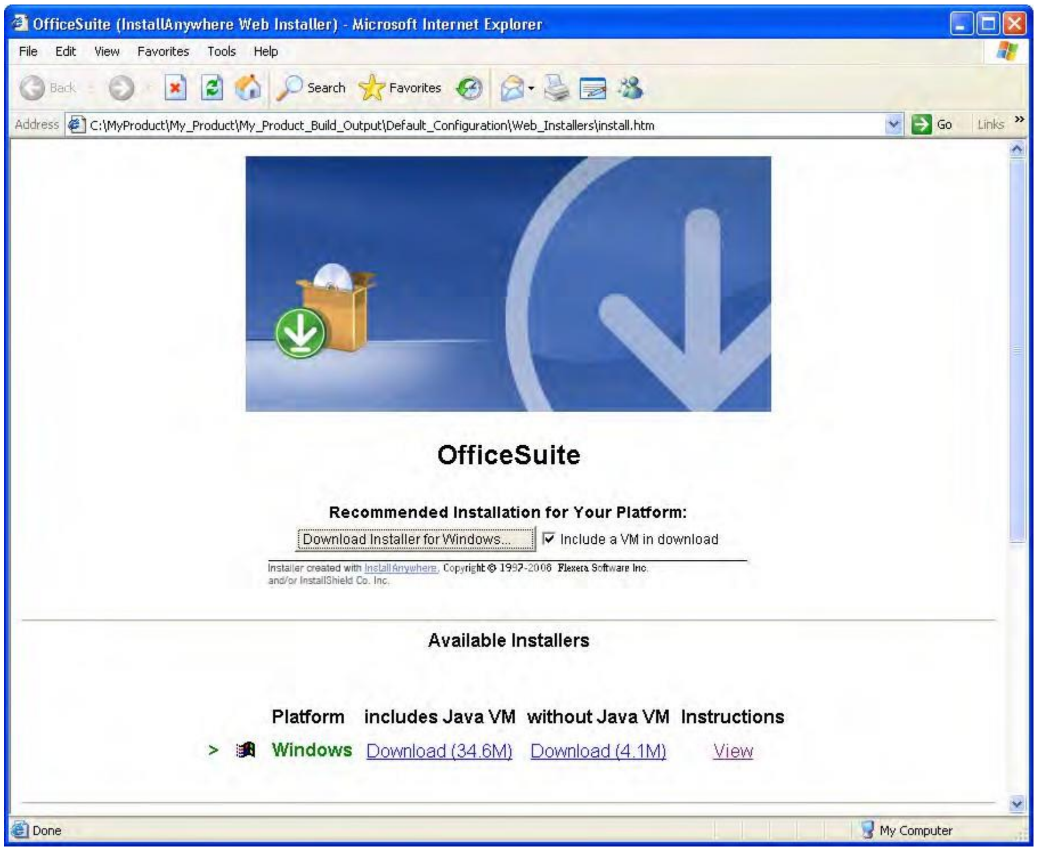
8. Click **OK** to close the **Build Complete** dialog box.
9. Continue with the steps in Testing the Installer.

Testing the Installer

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.

Task ***To test the installer:***

1. Perform the steps in Building the Installer.
2. On the **Build** task, click **Try Web Install**. The **Web Install Page** will load, and should request a security access.
3. 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.


OfficeSuite (InstallAnywhere Web Installer) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\MyProduct\My_Product\My_Product_Build_Output\Default_Configuration\Web_Installers\install.htm
Go Links
Done
My Computer

OfficeSuite

Recommended Installation for Your Platform:

Download Installer for Windows... [✓] Include a VM in download

Installer created with InstallAnywhere, Copyright © 1997-2008 Flexera Software Inc. and/or InstallShield Co. Inc.

Available Installers

Platform	includes Java VM	without Java VM	Instructions
Windows	Download (34.6M)	Download (4.1M)	View

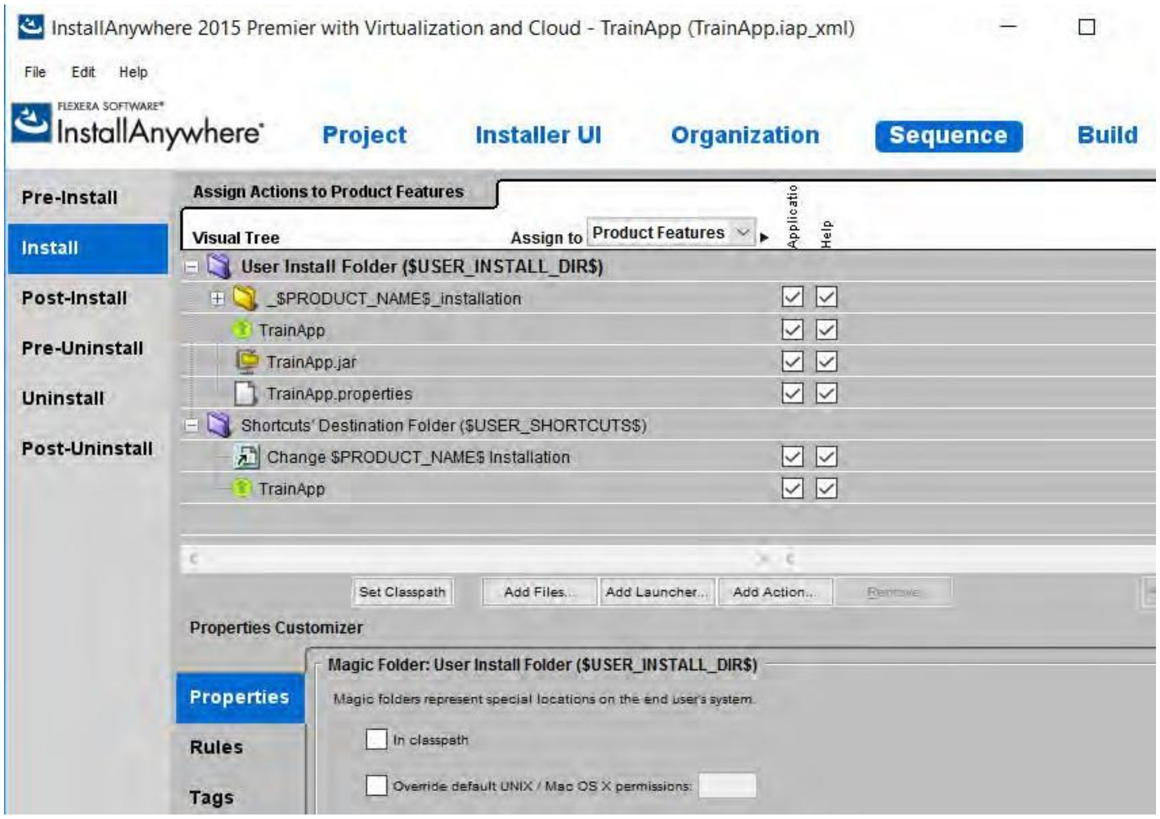
4. Click the **Download Installer for Windows** button below the image. The applet checks for sufficient disk space, downloads the installer, and executes the installer.
5. Run the installer. After the **Install Complete** panel, the installer should launch OfficeSuite. The OfficeSuite icon can now be selected from the Windows Start Menu to run the installed product.
6. Continue with the steps in Examining the TrainApp Project in the Advanced Designer.

Examining the TrainApp Project in the Advanced Designer

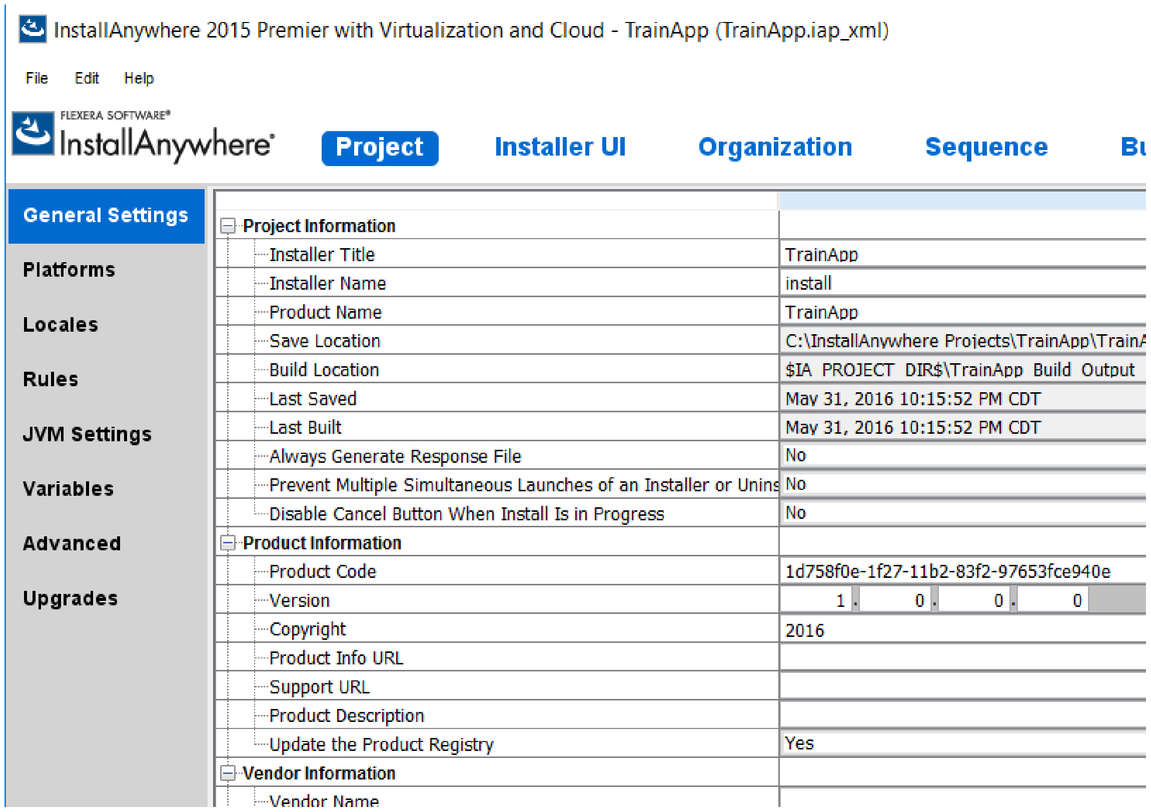
In this exercise, you will reopen the TrainApp project you created earlier and examine it in the Advanced Designer.

Task ***To examine the TrainApp project in the Advanced Designer.***

1. Perform the steps in Testing the Installer.
2. On the **File** menu, point to **Open Recent** and click **TrainApp.iap_xml** to open the project file you created in Creating the TrainApp Project.
3. Open the Sequence tab and select the **Install** task and examine the details of the actions created by the Project Wizard. For example, you can view the files and shortcuts created on the target system.


InstallAnywhere 2015 Premier with Virtualization and Cloud - TrainApp (TrainApp.iap_xml)
File Edit Help
FLEXERA SOFTWARE InstallAnywhere
Project
Installer UI
Organization
Sequence
Build
Pre-Install
Install
Post-Install
Pre-Uninstall
Uninstall
Post-Uninstall
Assign Actions to Product Features
Visual Tree
Assign to Product Features
Applicatio
Help
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
TrainApp
TrainApp.jar
TrainApp.properties
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
Change $PRODUCT_NAME$ Installation
TrainApp
Set Classpath
Add Files...
Add Launcher...
Add Action...
Remove
Properties Customizer
Properties
Rules
Tags
Magic Folder: User Install Folder ($USER_INSTALL_DIR$)
Magic folders represent special locations on the end user's system.
In classpath
Override default UNIX / Mac OS X permissions:

4. Open the **Project > Description** subtask and view the project properties filled in by the Project Wizard.



Project Information	
Installer Title	TrainApp
Installer Name	install
Product Name	TrainApp
Save Location	C:\InstallAnywhere Projects\TrainApp\TrainA
Build Location	$IA_PROJECT_DIR$\TrainApp_Build_Output
Last Saved	May 31, 2016 10:15:52 PM CDT
Last Built	May 31, 2016 10:15:52 PM CDT
Always Generate Response File	No
Prevent Multiple Simultaneous Launches of an Installer or Unins	No
Disable Cancel Button When Install Is in Progress	No
Product Information	
Product Code	1d758f0e-1f27-11b2-83f2-97653fce940e
Version	1 . 0 . 0 . 0
Copyright	2016
Product Info URL	
Support URL	
Product Description	
Update the Product Registry	Yes
Vendor Information	
Vendor Name	

5. Enter information in the **Product Info URL**, **Support URL**, and **Product Description** fields.

   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.

6. Continue with the steps in Building the TrainApp Project.

Chapter 4 Exercises: Building Releases

In this section, you will build the TrainApp project and specify VM requirements.

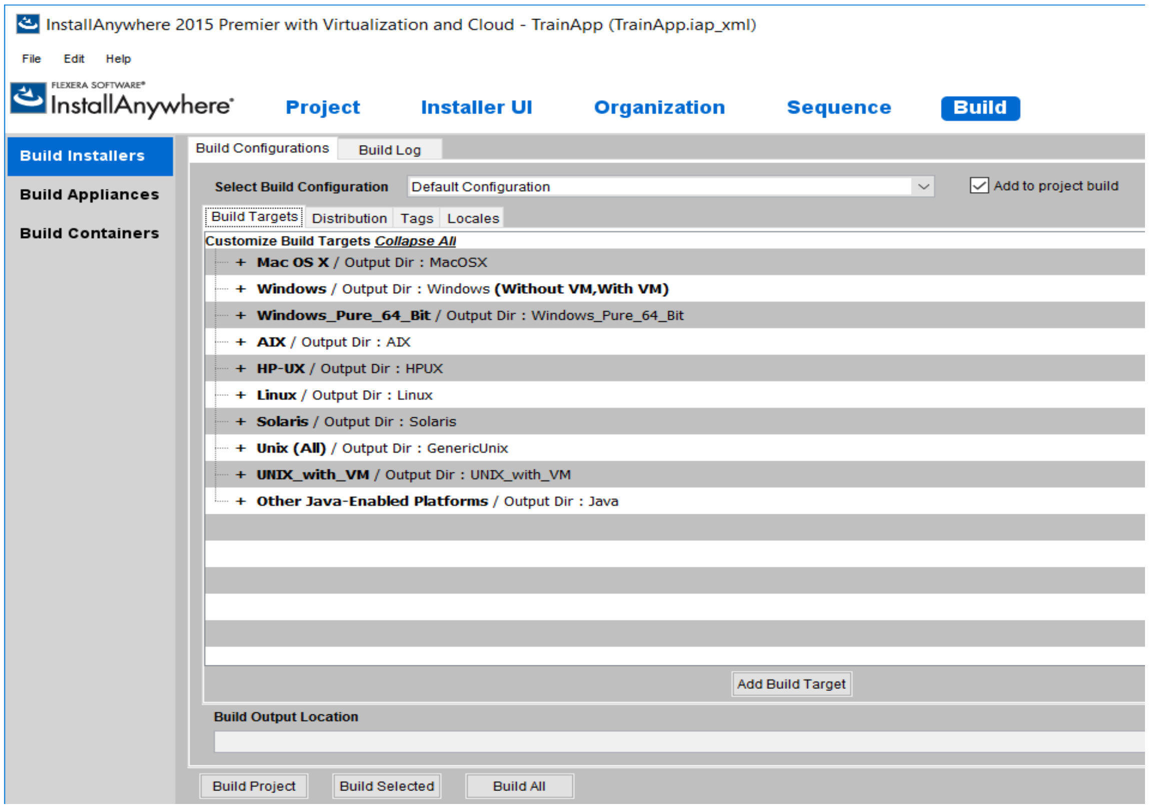
- Building the TrainApp Project
- Specifying VM Requirements

Building the TrainApp Project

In the **Build** task, you define the types of installation media to build for testing and eventual duplication and distribution. To build the TrainApp project, perform the following steps.

Task ***To build the TrainApp project:***

1. With the `TrainApp.iap_xml` project open in the Advanced Designer, open the **Build** task.


InstallAnywhere 2015 Premier with Virtualization and Cloud - TrainApp (TrainApp.iap_xml)

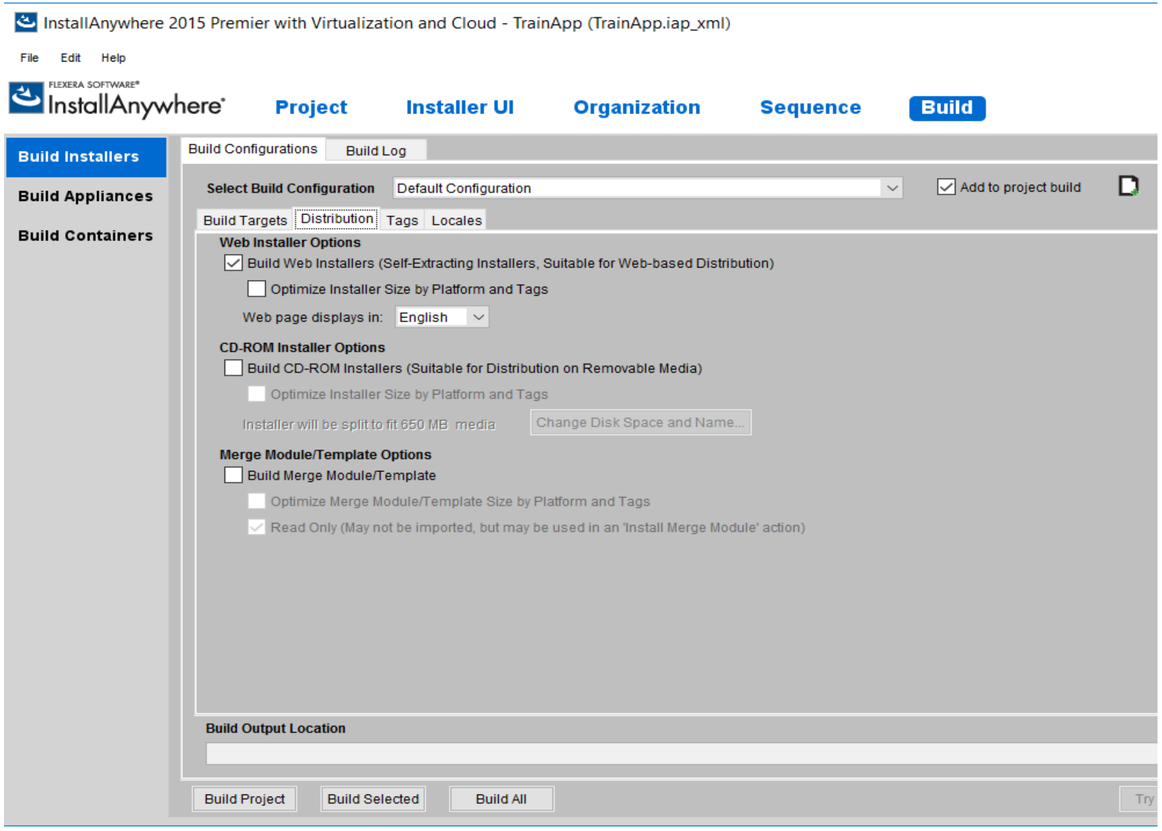
Select Build Configuration: Default Configuration ☑ Add to project build

Customize Build Targets *Collapse All*

- **Mac OS X** / Output Dir : MacOSX
- **Windows** / Output Dir : Windows **(Without VM,With VM)**
- **Windows_Pure_64_Bit** / Output Dir : Windows_Pure_64_Bit
- **AIX** / Output Dir : AIX
- **HP-UX** / Output Dir : HPUX
- **Linux** / Output Dir : Linux
- **Solaris** / Output Dir : Solaris
- **Unix (All)** / Output Dir : GenericUnix
- **UNIX_with_VM** / Output Dir : UNIX_with_VM
- **Other Java-Enabled Platforms** / Output Dir : Java

Build Output Location

2. On the **Build Targets** subtab of the **Build Configurations** tab, verify that at least the **Windows (Without VM)** setting is selected.
3. To delete the **Unix (All)** platform, which you will not use in this training class, click the **X** button to the left of the platform name. It is removed from the list.
4. Open the **Distribution** subtab of the **Build Configurations** tab.



Select Build Configuration: Default Configuration
☑ Add to project build

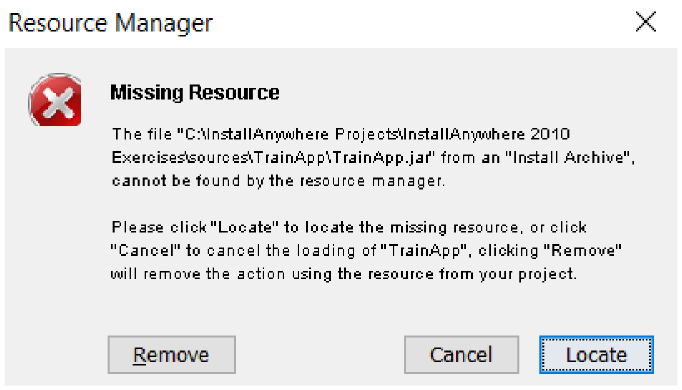
Web Installer Options
☑ Build Web Installers (Self-Extracting Installers, Suitable for Web-based Distribution)
☐ Optimize Installer Size by Platform and Tags
Web page displays in: English

CD-ROM Installer Options
☐ Build CD-ROM Installers (Suitable for Distribution on Removable Media)
☐ Optimize Installer Size by Platform and Tags
Installer will be split to fit 650 MB media

Merge Module/Template Options
☐ Build Merge Module/Template
☐ Optimize Merge Module/Template Size by Platform and Tags
☑ Read Only (May not be imported, but may be used in an 'Install Merge Module' action)

Build Output Location

5. For each project, you must select one at least one distribution option: Web Installer, CD-ROM, and/or Merge Module/Template. For this exercise, select only the **Build CD-ROM Installers** option.
6. To build the project, click **Build Project** at the bottom of the task.
7. When the build is complete, click **OK** to dismiss the small progress window and click the **Open in Explorer** button next to the **Build Output Location** field on the **Build** task. You can then explore the output directory to view the structure of the files to be burned to a CD-ROM.
8. If you have time, add the **Web Installer** option to your distribution settings, rebuild the installer, and examine the Web page used to install your project. If you run the entire installation, be sure to uninstall the product afterward.
9. 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:


Resource Manager

Missing Resource

The file "C:\InstallAnywhere Projects\InstallAnywhere 2010 Exercises\sources\TrainApp\TrainApp.jar" from an "Install Archive", cannot be found by the resource manager.

Please click "Locate" to locate the missing resource, or click "Cancel" to cancel the loading of "TrainApp", clicking "Remove" will remove the action using the resource from your project.

10. 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.

11. Continue with the steps in Specifying VM Requirements.

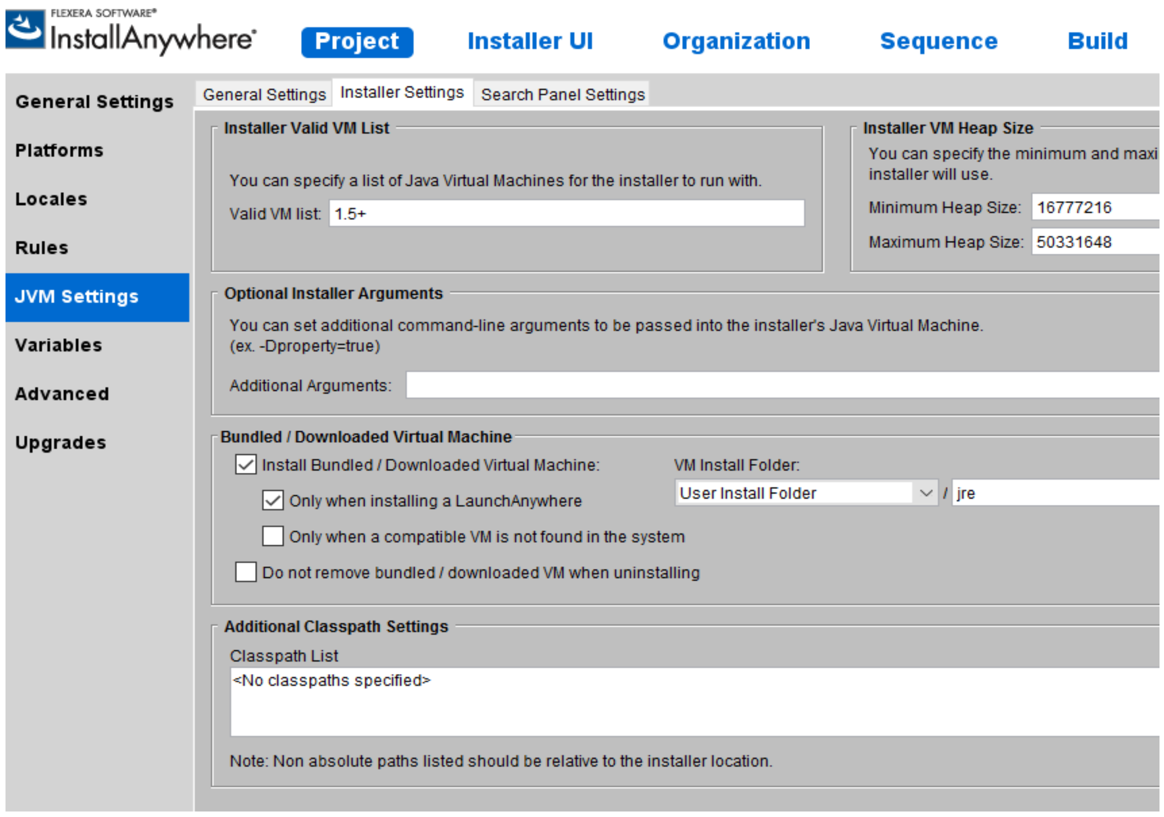
Specifying VM Requirements

To specify VM requirements, perform the following steps.

Task ***To specify VM requirements:***

1. Perform the steps in Building the TrainApp Project.

2. Open the **Project > JVM Settings, then select the Installer** Settings subtask. The **Project > Config** subtask opens.



Installer Valid VM List

You can specify a list of Java Virtual Machines for the installer to run with.

Valid VM list: 1.5+

Installer VM Heap Size

You can specify the minimum and maxi installer will use.

Minimum Heap Size: 16777216

Maximum Heap Size: 50331648

Optional Installer Arguments

You can set additional command-line arguments to be passed into the installer's Java Virtual Machine. (ex. -Dproperty=true)

Additional Arguments:

Bundled / Downloaded Virtual Machine

[✓] Install Bundled / Downloaded Virtual Machine:

[✓] Only when installing a LaunchAnywhere

[ ] Only when a compatible VM is not found in the system

[ ] Do not remove bundled / downloaded VM when uninstalling

VM Install Folder: User Install Folder / jre

Additional Classpath Settings

Classpath List

<No classpaths specified>

Note: Non absolute paths listed should be relative to the installer location.

3. In the **Valid VM list** field, enter a VM version greater than the largest version available on your system, such as `1.9+`.
4. Build the project.
5. Verify that running the project displays an error message that no VM meeting the requirement is present on the target system.
6. When finished, change the setting back to its original value.
7. Continue with the steps in Exploring the Installer UI Tasks.

Chapter 5 Exercises: Basic Installer Customization

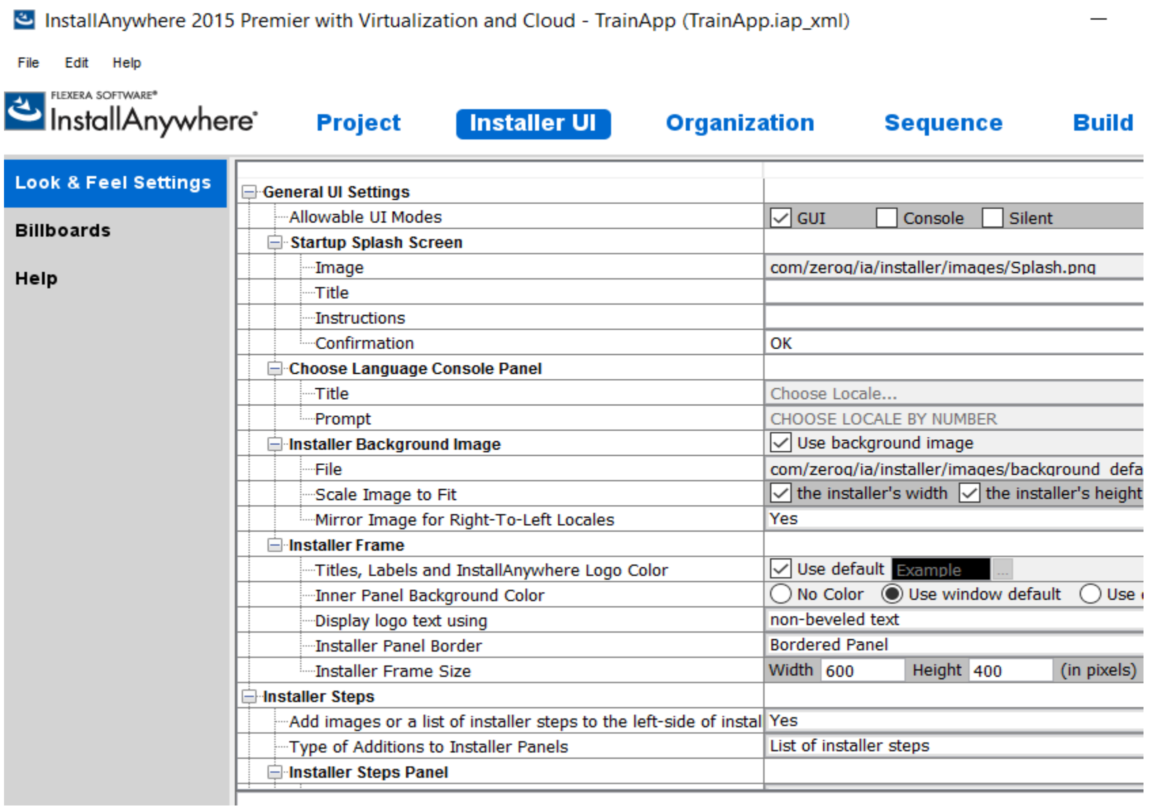
In this section, you will customize the installer by performing the following tasks:

- Exploring Look and Feel
- Using Installer Rules
- Using Rules to Control Visual Elements
- Managing Installer Flow Based on End-User Input
- Changing the Splash Screen
- Changing the Background Image
- Creating Installer Rules

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.



InstallAnywhere 2015 Premier with Virtualization and Cloud - TrainApp (TrainApp.iap_xml)

General UI Settings	
Allowable UI Modes	✓ GUI ☐ Console ☐ Silent
Startup Splash Screen	
Image	com/zerog/ia/installer/images/Splash.png
Title	
Instructions	
Confirmation	OK
Choose Language Console Panel	
Title	Choose Locale...
Prompt	CHOOSE LOCALE BY NUMBER
Installer Background Image	✓ Use background image
File	com/zerog/ia/installer/images/background_defa
Scale Image to Fit	✓ the installer's width ✓ the installer's height
Mirror Image for Right-To-Left Locales	Yes
Installer Frame	
Titles, Labels and InstallAnywhere Logo Color	✓ Use default Example
Inner Panel Background Color	○ No Color ● Use window default ○ Use
Display logo text using	non-beveled text
Installer Panel Border	Bordered Panel
Installer Frame Size	Width 600 Height 400 (in pixels)
Installer Steps	
Add images or a list of installer steps to the left-side of instal	Yes
Type of Additions to Installer Panels	List of installer steps
Installer Steps Panel	

Figure B-2: Installer UI > Look & Feel Subtask

Exploring the Installer UI Tasks

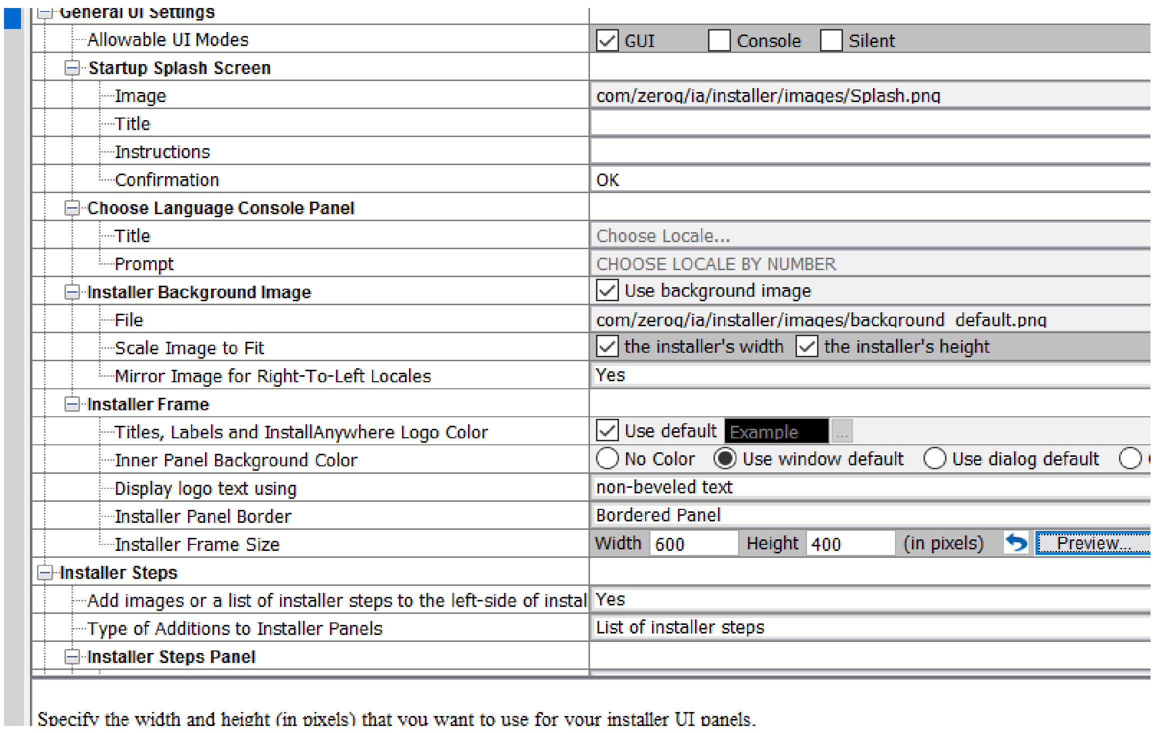
The **Installer UI > Look & Feel** subtask contains three tabs which enable you to configure the look and feel of the installer: **General UI Settings**, **Installer Steps**, and **Install Progress Panel**.

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.

To explore the **Installer UI** tasks, perform the following steps:

Task ***To explore the Installer UI tasks:***

1. With the `TrainApp.iap_xml` project open in the Advanced Designer, open the **General UI Settings** tab of the **Installer UI > Look & Feel** subtask. The **General UI Settings** tab sets general look and feel settings for the installer.



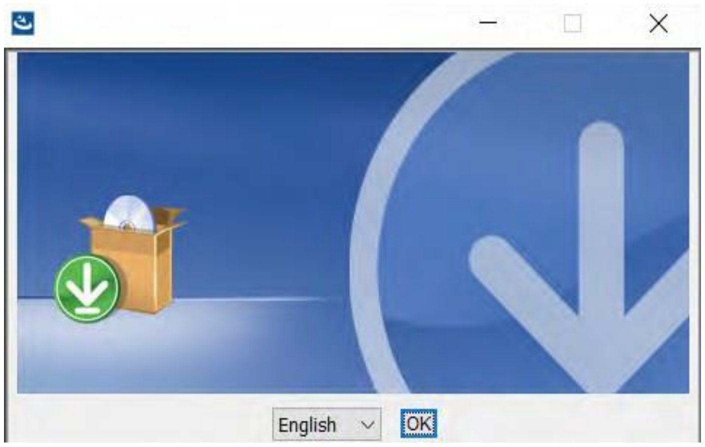
General UI Settings	
Allowable UI Modes	✓ GUI ☐ Console ☐ Silent
Startup Splash Screen	
Image	com/zerog/ia/installer/images/Splash.png
Title	
Instructions	
Confirmation	OK
Choose Language Console Panel	
Title	Choose Locale...
Prompt	CHOOSE LOCALE BY NUMBER
Installer Background Image	✓ Use background image
File	com/zerog/ia/installer/images/background_default.png
Scale Image to Fit	✓ the installer's width ✓ the installer's height
Mirror Image for Right-To-Left Locales	Yes
Installer Frame	
Titles, Labels and InstallAnywhere Logo Color	✓ Use default Example
Inner Panel Background Color	○ No Color ● Use window default ○ Use dialog default ○
Display logo text using	non-beveled text
Installer Panel Border	Bordered Panel
Installer Frame Size	Width 600 Height 400 (in pixels) Preview...
Installer Steps	
Add images or a list of installer steps to the left-side of instal	Yes
Type of Additions to Installer Panels	List of installer steps
Installer Steps Panel	

Specify the width and height (in pixels) that you want to use for your installer UI panels.

2. Under **Allowable UI Modes**, you can specify whether the installer will have a **GUI**, **Console**, or **Silent** mode. For this exercise, select **GUI**.

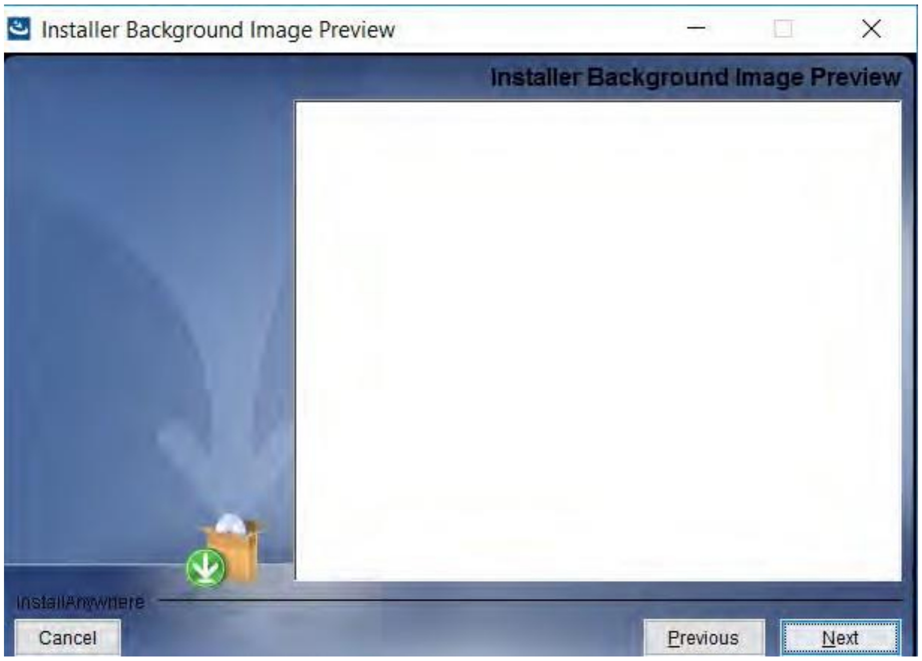
Note • *An installer may support all three modes at the same time. The* ***GUI*** *(Swing) mode offers the ability to have a background image, such as a company logo, to be displayed in the installer. The specific background image is selected under* ***Installer Background Image*** *on this tab.*

3. 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. In the **Startup Splash Screen** section, click the **Preview** button. The default splash screen opens:



4. Click **OK** to close the splash screen.

5. In the **Installer Background Image** section, click the **Preview** button. The default installer background image is displayed.


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

6. Click **Cancel** to close the image.

Note • *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-mode installers.

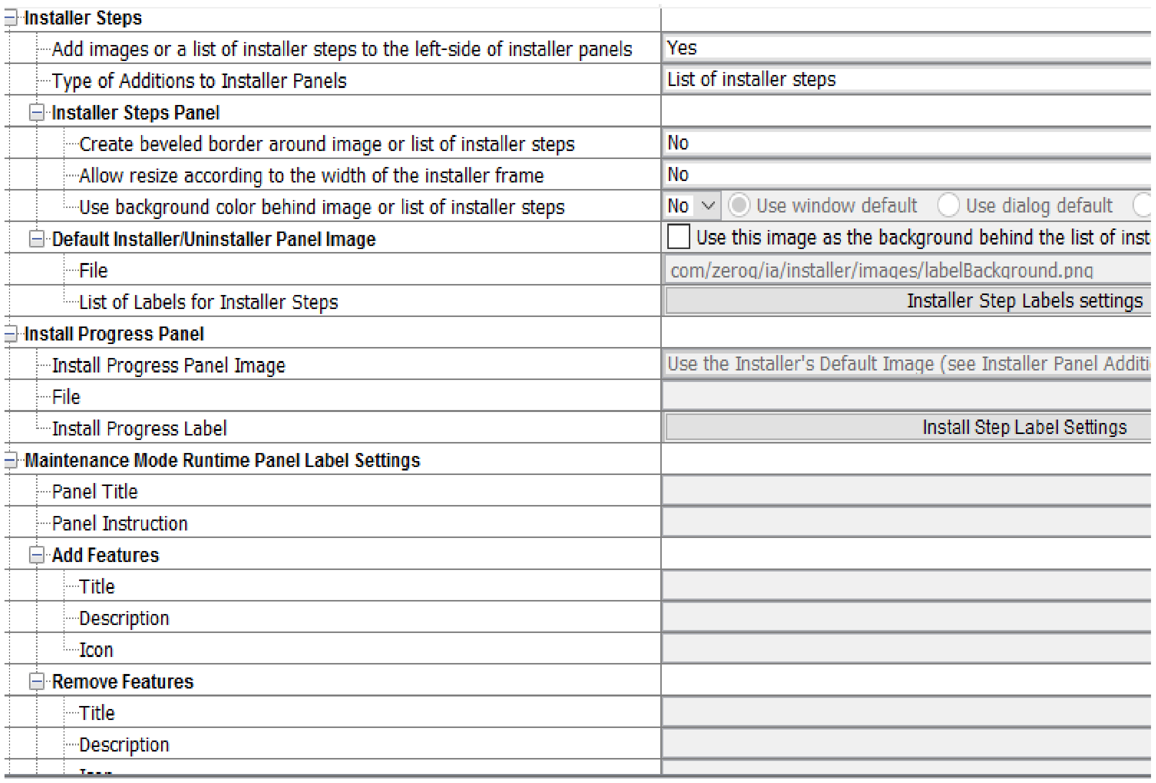
7. Continue with the steps in Making Additions to the GUI Installer Panels.

Making Additions to the GUI Installer Panels

To make additions to the GUI installer panels, perform the following steps.

Task ***To make additions to the GUI installer panels:***

1. Perform the steps in Exploring the Installer UI Tasks.
2. In the **Installer UI > Look and Feel** subtask, scroll down to the **Installer Steps** section, which allows you to customize installer panels for the left hand install progress steps rectangle. Click on the Installer Step Labels settings and the **Installer Steps** tab opens:



Installer Steps	
Add images or a list of installer steps to the left-side of installer panels	Yes
Type of Additions to Installer Panels	List of installer steps
Installer Steps Panel	
Create beveled border around image or list of installer steps	No
Allow resize according to the width of the installer frame	No
Use background color behind image or list of installer steps	No ◯ Use window default ◯ Use dialog default
Default Installer/Uninstaller Panel Image	☐ Use this image as the background behind the list of inst
File	com/zerog/ia/installer/images/labelBackground.png
List of Labels for Installer Steps	Installer Step Labels settings
Install Progress Panel	
Install Progress Panel Image	Use the Installer's Default Image (see Installer Panel Additi
File	
Install Progress Label	Install Step Label Settings
Maintenance Mode Runtime Panel Label Settings	
Panel Title	
Panel Instruction	
Add Features	
Title	
Description	
Icon	
Remove Features	
Title	
Description	

3. In the **Additions to GUI Installer Panels** section, select the **Add images or a list of installer steps to the left side of installer panels** option. When this option is selected, the rest of the fields on this tab are enabled, as is the **Install Progress Panel** tab.

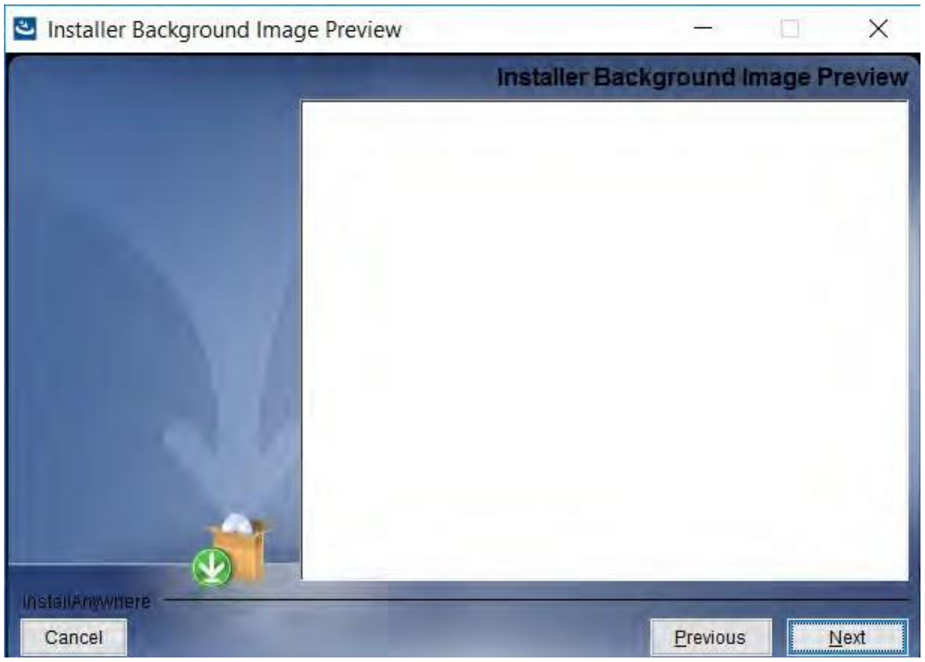
   Note • *If the* ***Add images or a list of installer steps to the left side of installer panels*** *option is not selected, the rest of the fields on the* ***Installer Steps*** *tab and the entire* ***Install Progress Panel*** *tab will be disabled because there will be no additional progress panels to modify. These additional panels are either images or labels.*

4. In the **Type of Additions to Installer Panels** section, select the **Images** option.
   - If **Images** is selected, no install labels will be displayed.
   - Selecting **Images** enables the fields in the **Default Installer/Uninstaller Panel Image** section of the **Installer Steps** tab, allowing you to select a default image.

- When **Images** is selected, the default image may be overridden by specifying an image on the **Install Progress Panel** tab of the **Installer UI > Look & Feel** subtask.
- You may also choose not to display an image, or select to use the same image as the previous panel.

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

5. In the **Default Installer/Uninstaller Panel Image** section, click **Preview**. The default panel image is displayed.


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

6. Click **Cancel** to close the image.
7. In the **Type of Additions to Installer Panels** section, select the **List of installer steps** option. When this option is selected:
   - The **List of Labels for Installer Steps** area is enabled, allowing you to enter custom labels for installer steps.
   - The **Installer Steps Background Image** area is enabled, allowing you to select a specific image to display in the rectangle on the left hand side where the installer step labels will be displayed.

Note • *Also when the **List of installer steps** option is selected along with the **GUI** option on the **General UI Settings** tab, you can specify an image with a transparent background to be displayed on top of the **Installer Background Image** selected on the **Look & Feel > General UI Settings** tab.*

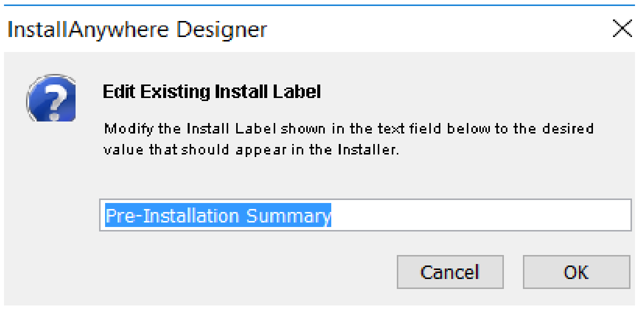
Note • *The size of the install progress pane is 380 by 270 pixels. Installer dimensions may change slightly by platform to better display text and different fonts.*

8. In the **List of Labels for Installer Steps** area of the **Installer Steps** tab, you can view and alter the installation steps labels.

- **Buttons**—The buttons to the right enable you to add or remove labels, or change the order of the labels.
- **Auto Populate button**—The **Auto Populate** button adds an installer step for every panel action added to the **Pre-Install** and **Post-Install** tasks.
- **Enabling text editing and altering the order**—If you want to edit the text of the installer steps or change the order of the installer steps, clear the selection of the **Auto populate labels when saving** option.
- **Altering the order**—To alter the order of the installer steps, use the arrows.
- **Altering the installer step icons**—To alter the small square graphics that are to the left of the text labels, click **Choose Icons**. 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.

Because you want to edit an installer step label, clear the selection of the **Auto populate labels when saving** option.

9. Select the **Pre-Installation Summary** label and click **Edit Label.** The **Edit Existing Install Label** dialog box opens.


InstallAnywhere Designer

Edit Existing Install Label

Modify the Install Label shown in the text field below to the desired value that should appear in the Installer.

Pre-Installation Summary

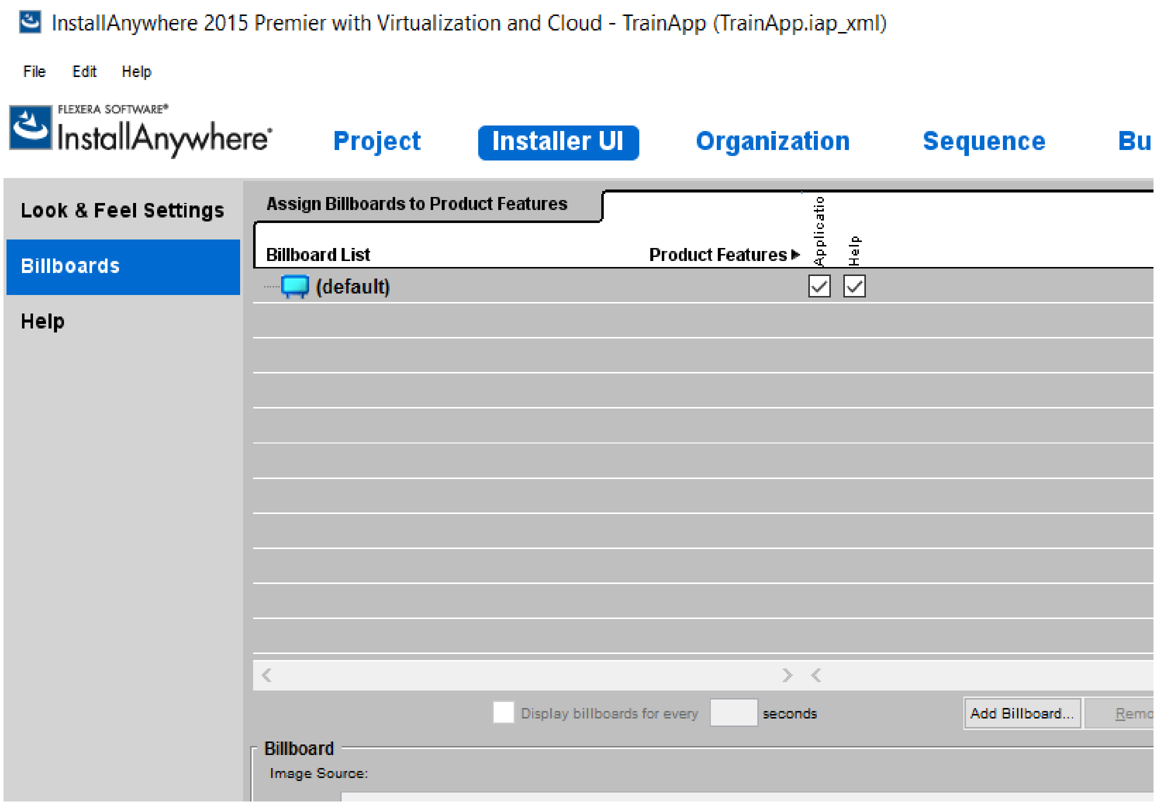
10. Change the label to `Summary` and click **OK**.
11. Continue with the steps in Selecting Billboards.

Selecting Billboards

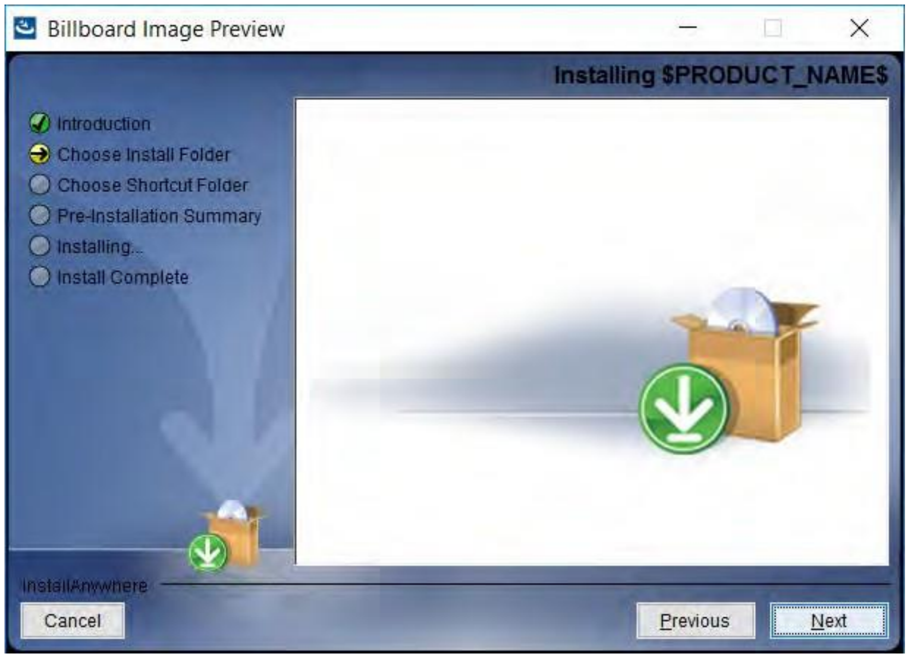
Billboards are images that appear in the large right hand pane of the installer while files are being installed. Billboards generally convey a marketing message, a description of the product, or simply something fun for the end user to see as the file installation is occurring.

Task ***To add a billboard:***

1. Perform the steps in Making Additions to the GUI Installer Panels.
2. Select the **Installer UI > Billboards** task.



3. In the **Billboard** section near the bottom of the window, click **Preview**. The default billboard image is displayed.



4. Click Cancel to close the preview.

5. Near the center of the screen, click the **Add Billboard** button. The **Choose an Image File** dialog box opens.

6. Select the `billboard1.gif` file from the `OfficeSuiteSourceFiles\ImagesAndDocs` directory.

***Note** • 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.*

There are several billboard graphics available in the `ImagesAndDocs` directory within the `OfficeSuiteSourceFiles` folder. For this exercise, it is advised that you add two billboards to the installation. If you add more than two, the appearance of the panels would be too short due to the small number of files in this installation.

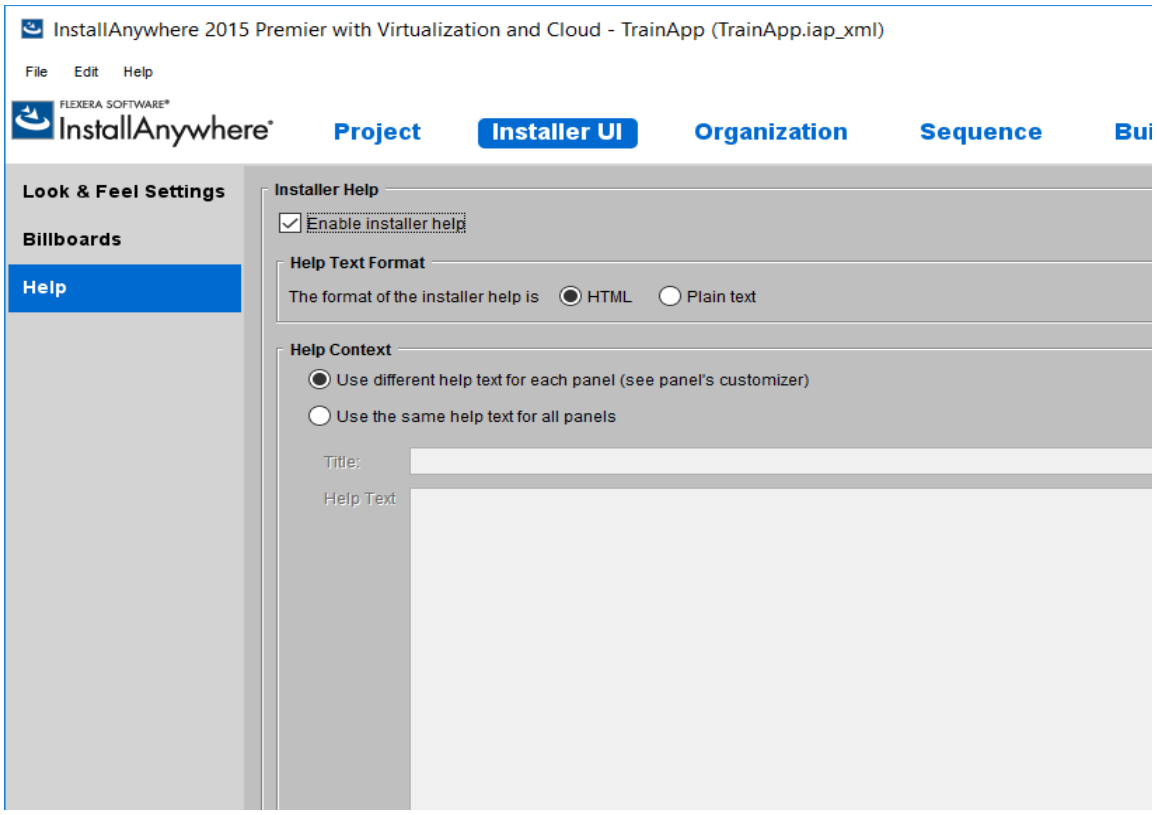
7. Click the **Add Billboard** button again and this time add the `billboard2.gif` file, which can be found in the same location as the file you previously added. Two images are now listed in the **Billboard List**.
8. Click **Preview**. Note the following regarding billboards:
    - **Amount of time each billboard is displayed**—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 be displayed for only a short time.
    - **Number of billboards that will be displayed**—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 can be assigned to features**—Billboards can also be assigned to features, and will only be displayed if the feature they are associated with installs.
    - **Order that billboards are displayed**—When adding multiple billboards, the billboards will display in the order they are shown in the **Installer UI > Billboards > Billboard List**.
9. Continue with the steps in Exploring the Help Options.

Exploring the Help Options

To specify an installer's help options, perform the following steps.

Task ***To explore the help options:***

1. Perform the steps in Selecting Billboards.
2. Open the **Installer UI > Help** subtask.
3. In the top section, select the **Enable installer help** option.


InstallAnywhere 2015 Premier with Virtualization and Cloud - TrainApp (TrainApp.iap_xml)

Installer Help

[✓] Enable installer help

Help Text Format

The format of the installer help is [● HTML] ○ Plain text

Help Context

[● Use different help text for each panel (see panel's customizer)]

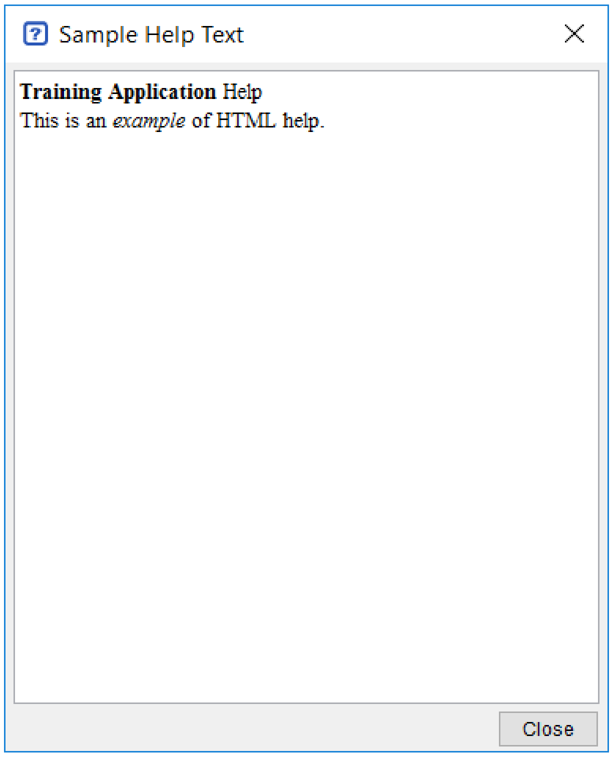
○ Use the same help text for all panels

Title:

Help Text

4. In **Help Text Format**, select **HTML**.
5. In **Help Context**, select **Use the same help text for all panels**.
6. In the **Title** text field, enter **`Sample Help Text`**.
7. In the **Help Text** text field, enter:

   `<b>Training Application</b> Help <br> This is an <i>example</i> of HTML help.`
8. Click **Preview**. The sample help topic is displayed.



Training Application Help
This is an *example* of HTML help.

9. Click **Close** to close the help topic.
10. Continue with the steps in Rebuilding the Project.

Rebuilding the Project

To rebuild this help project after making these user interface changes, perform the following steps.

Task ***To rebuild the project:***

1. Perform the steps in Exploring the Help Options.
2. Open the **Build** task.
3. Click **Build Project** to build the installer.
4. When the build is complete, click **Try Installer** to view the changes you have made to the installer user interface.
5. Continue with the steps in Using Installer Rules.

Using Installer Rules

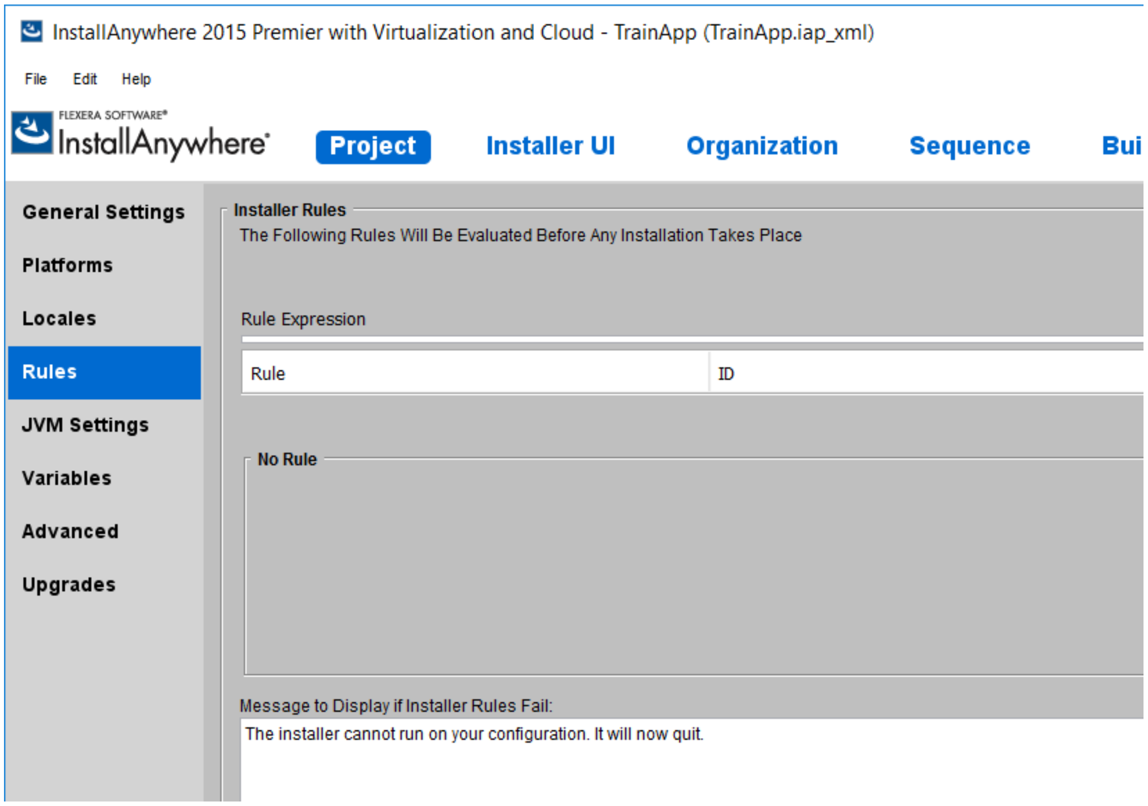
InstallAnywhere Rules can be implemented in a number of actions within the installer. However, the first set of rules evaluated in the installer is the Installer Rules. These rules, set in the **Project > Rules** task, allow you to control the complete installer based on rules.

For example, let's consider that the OfficeSuite product is designed to run only on Windows, Mac OS X, and Linux, but not on Solaris, SunOS, HP-UX, etc. To add this operating system condition to the OfficeSuite installer, perform the following steps:

To add a rule to a project, perform the following steps.

Task ***To add a rule to the project:***

1. On the **File** menu, point to **Open Recent** and click `My_Product.iap_xml` to open the OfficeSuite project.
2. Open the **Project > Rules** task.

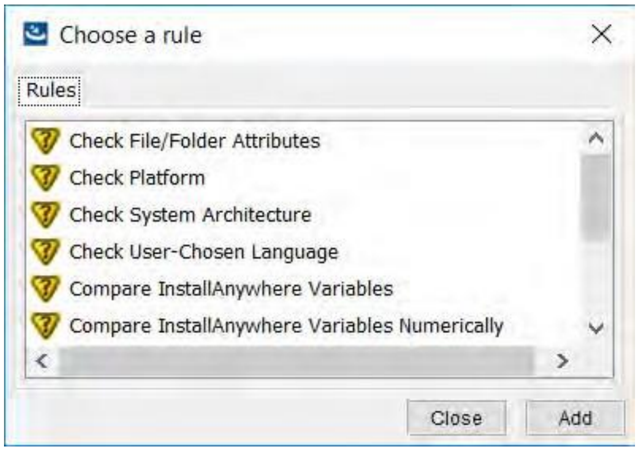


The Following Rules Will Be Evaluated Before Any Installation Takes Place

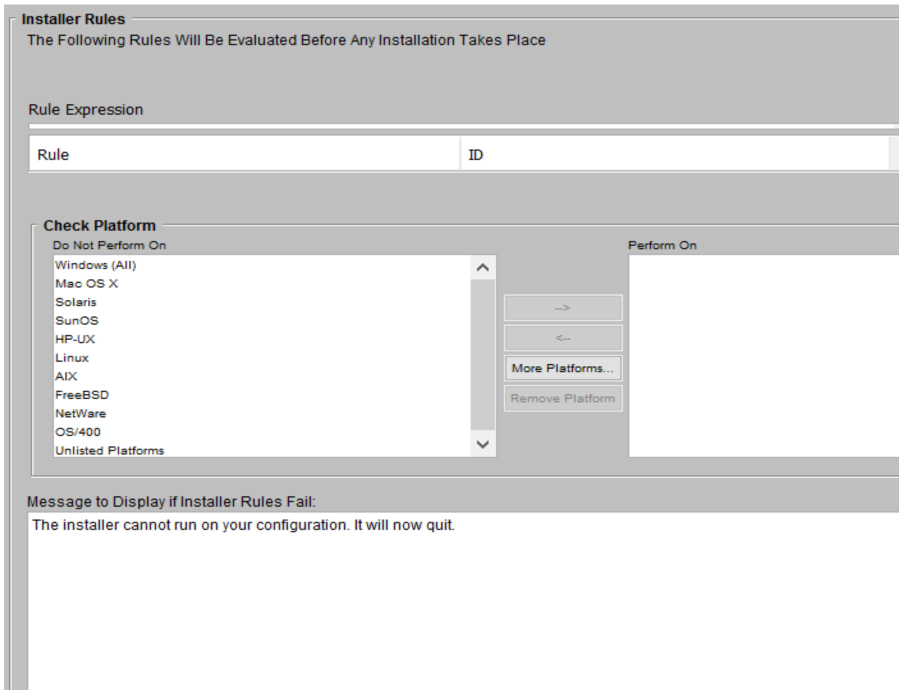
Message to Display if Installer Rules Fail:

The installer cannot run on your configuration. It will now quit.

3. Click the **Add Rule** button. The **Choose a rule** dialog box opens.


Choose a rule
Rules
Check File/Folder Attributes
Check Platform
Check System Architecture
Check User-Chosen Language
Compare InstallAnywhere Variables
Compare InstallAnywhere Variables Numerically
Close
Add

4. Select **Check Platform** and click **Add**. The Check Platform rule is now listed in the **Rules List** and the **Check Platform** customizer opens.



The installer cannot run on your configuration. It will now quit.

5. In the customizer for the **Check Platform** rule, select **Windows (All)**, **Mac OS X**, and **Linux** from the left-hand **Do Not Perform On** column.

Note • *You can hold down the Ctrl or Apple keys to select multiple items.*

6. Click the right arrow button to move the selected items to the **Perform On** column.

7. In the **Message to Display if Installer Rules Fail** box, modify the message that will appear if an end user attempts to run the installer on a platform other than those you've specified.

8. Open the **Build** task and click **Build Project** to build the installer.

9. When the build is complete, click **Try Installer** to run the installer. If you are running on a platform other than those that you've specified, the installer should run normally.

10. Return to the project, and remove the platform you are working on at the moment. Then rebuild and re-run the installer. You should see the message you entered indicating that the platform was disallowed.

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

11. Continue with the steps in Using Rules to Control Visual Elements.

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 errors on Windows should not appear on Mac OS X. Like the installer rules, you can use rules to control visual elements in the **Pre-Install** 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.

- Adding Panels to an Installer
- Adding a Rule to the Display Message Panel

Adding Panels to an Installer

In this exercise, you will add the following panels to the installer:

Table B-2 • Panels Used in this Exercise

Panel Type	Description
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. ● **Setting the default state**—You can set the default state of the radio buttons (**Accept** or **Decline**). ● **License agreement file**—You can choose a file to use for a license agreement. 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 display a text message to the end user during the installation. ● **Additional information**—This can be useful for conveying information about installation choices that the end user has made. ● **Helpful with debugging**—This panel is also particularly useful in debugging installer issue having to do with InstallAnywhere variables. ● **Can use variables**—You can add **Display Message** panels with variables resolved to test variable values you are using in rules.

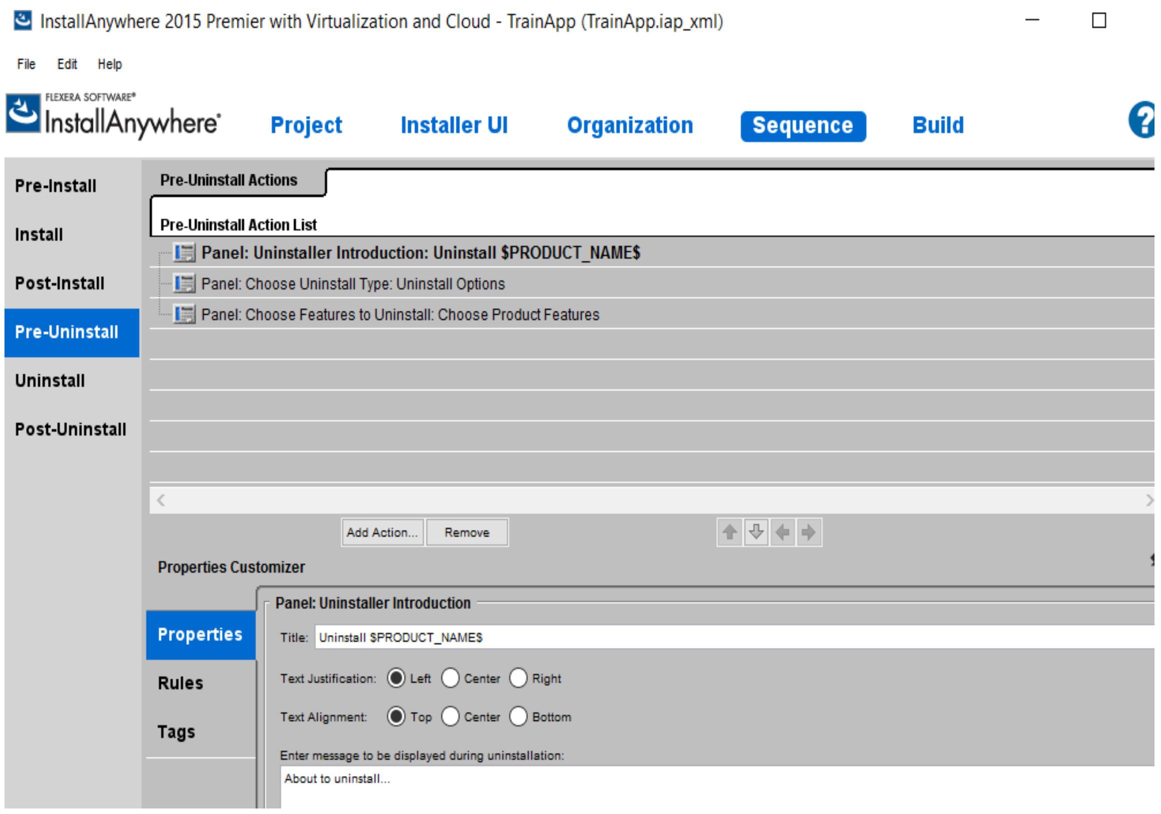
Table B-2 • Panels Used in this Exercise

Panel Type	Description
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 “read me” or errata type documents.

To add these panels to the installer, perform the following steps.

Task ***To add panels to an installer:***

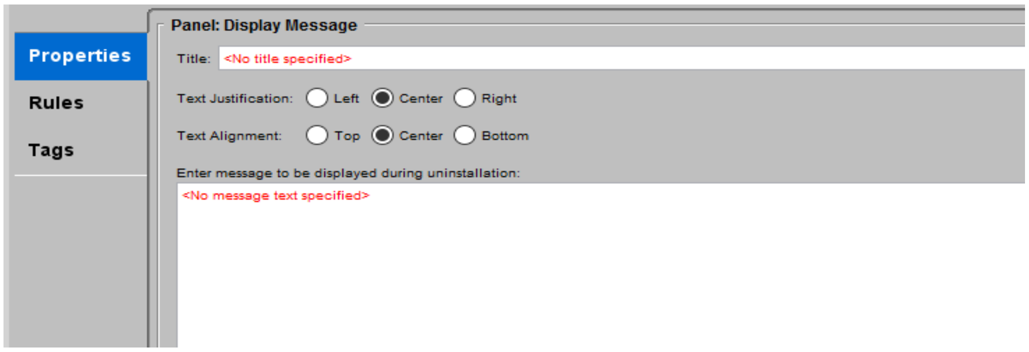
1. Perform the steps in Using Installer Rules.
2. Open the **Pre-Install Task**.



Panel: Uninstaller Introduction: Uninstall $PRODUCT_NAME$
Panel: Choose Uninstall Type: Uninstall Options
Panel: Choose Features to Uninstall: Choose Product Features

Panel: Uninstaller Introduction
Title: Uninstall $PRODUCT_NAME$
Text Justification: [✓ Left] Center Right
Text Alignment: [✓ Top] Center Bottom
Enter message to be displayed during uninstallation:
About to uninstall...

3. Click the **Add Action** button. The **Action** palette opens, displaying several tabs for differing types of actions.
4. Open the **Panels** tab.
5. To add a **Display Message** panel, perform the following steps:
   a. On the **Panels** tab, select **Panel: Display Message** and click **Add**. The **Panel: Display Message** action is added to the **Pre-Install Action List** and the **Panel: Display Message** customizer opens.



Panel: Display Message

Title: <No title specified>

Text Justification: Left [✓ Center] Right

Text Alignment: Top [✓ Center] Bottom

Enter message to be displayed during uninstallation:

<No message text specified>

b. In the **Title** box, enter **`Running on Windows`**.

c. In the **Enter message to be displayed during installation** box, enter the following text:

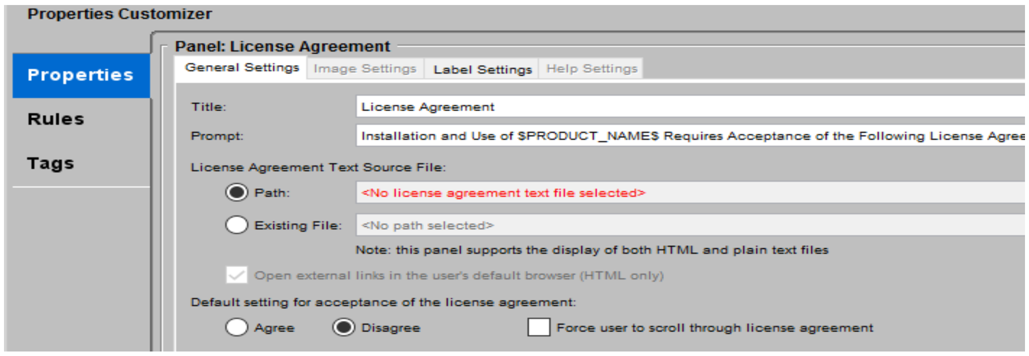
`To uninstall this application, use Add or Remove Programs from the Control Panel.`

Note • *You can use InstallAnywhere variables (using the* `$VARIABLE$` *notation) in the display message which will be resolved when the message is displayed. Here is an 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.
```

6. To add a **License Agreement** panel, perform the following steps:

a. On the **Panels** tab of the Actions palette, select **Panel: License Agreement** and click **Add**. The **Panel: License Agreement** action is added to the **Pre-Install Action List** and the **Panel: License Agreement** customizer opens.



Panel: License Agreement

Title: License Agreement

Prompt: Installation and Use of $PRODUCT_NAME$ Requires Acceptance of the Following License Agree

License Agreement Text Source File:

[✓ Path:] <No license agreement text file selected>

Existing File: <No path selected>

Note: this panel supports the display of both HTML and plain text files

✓ Open external links in the user's default browser (HTML only)

Default setting for acceptance of the license agreement:

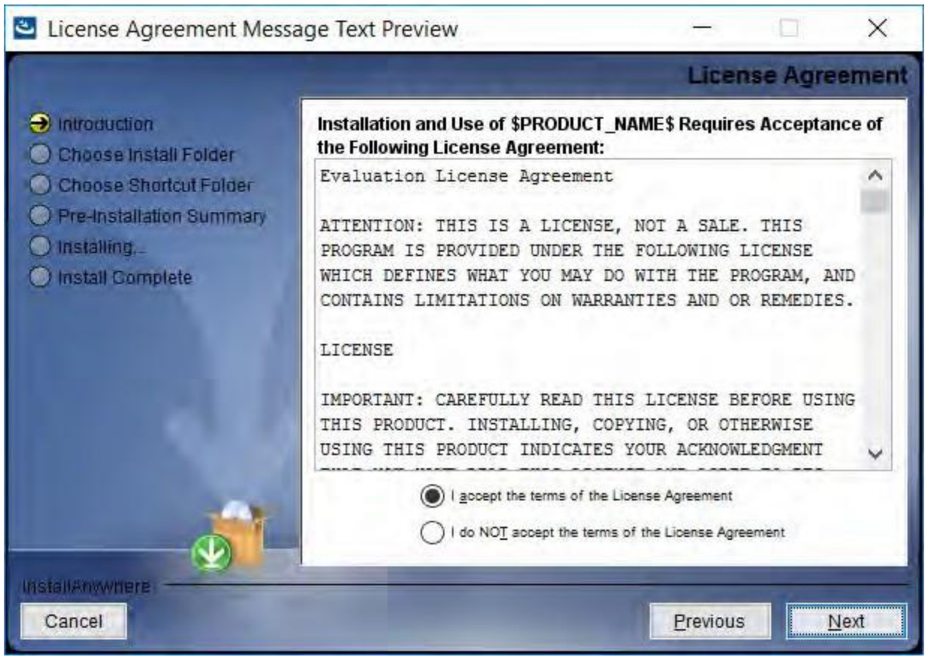
Agree [✓ Disagree] Force user to scroll through license agreement

b. Next to the **Path** option under **License Agreement Text Source File**, click **Choose File** and select the following license file:

`OfficeSuiteSourceFiles\ImagesAndDocs\License.txt`

c. Set the **Default setting for acceptance of this license agreement** option to **Agree**.

d. Click **Preview** to view the **License Agreement** panel.



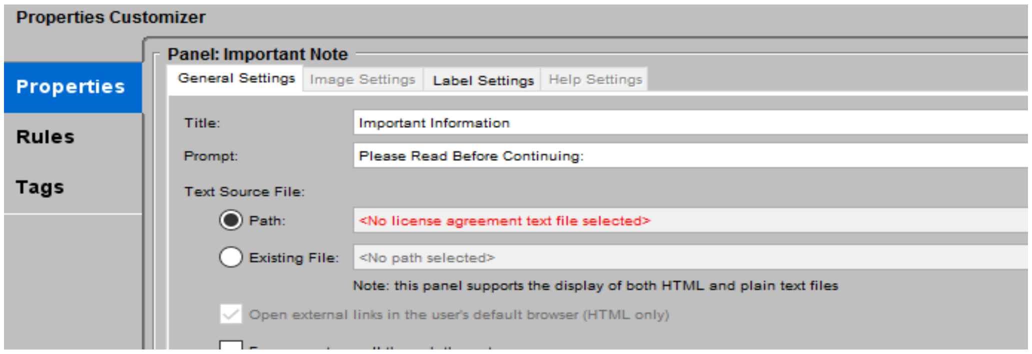
Installation and Use of $PRODUCT_NAME$ Requires Acceptance of the Following License Agreement:

```
Evaluation License Agreement

ATTENTION: THIS IS A LICENSE, NOT A SALE. THIS
PROGRAM IS PROVIDED UNDER THE FOLLOWING LICENSE
WHICH DEFINES WHAT YOU MAY DO WITH THE PROGRAM, AND
CONTAINS LIMITATIONS ON WARRANTIES AND OR REMEDIES.

LICENSE

IMPORTANT: CAREFULLY READ THIS LICENSE BEFORE USING
THIS PRODUCT. INSTALLING, COPYING, OR OTHERWISE
USING THIS PRODUCT INDICATES YOUR ACKNOWLEDGMENT
```

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

7. To add an **Important Note** panel, perform the following steps:

   a. On the **Panels** tab of the Actions palette, select **Panel: Important Note** and click **Add**. The **Panel: Important Note** action is added to the **Pre-Install Action List** and the **Panel: Important Note** customizer opens.

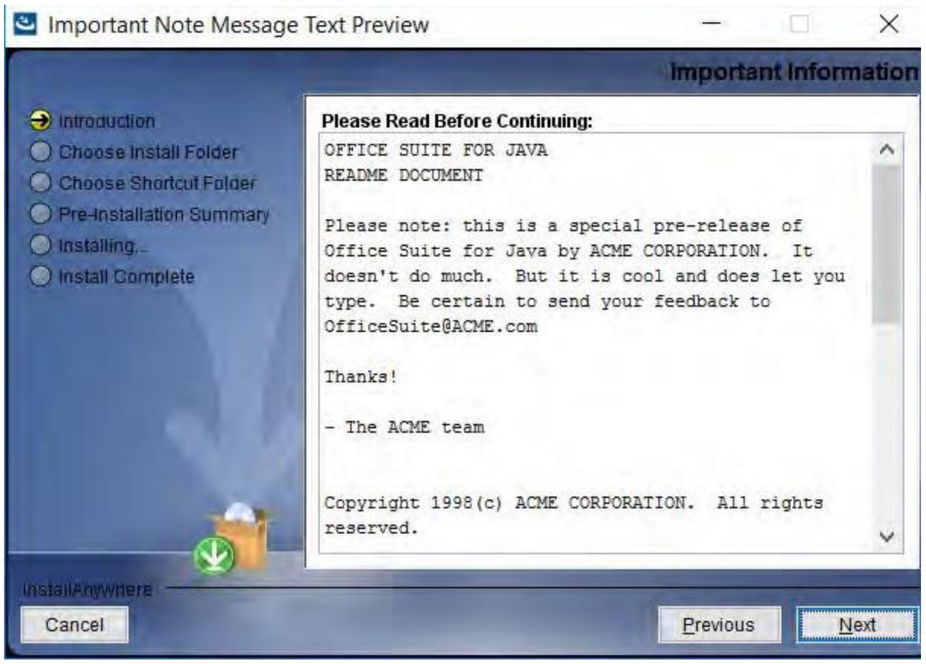


Title: Important Information
Prompt: Please Read Before Continuing:
Text Source File:
[✓ Path:] <No license agreement text file selected>
Existing File: <No path selected>
Note: this panel supports the display of both HTML and plain text files
[✓] Open external links in the user's default browser (HTML only)

   b. Next to the **Path** option under **Text Source File**, click **Choose File** and select the following text file:

      `OfficeSuiteSourceFiles\ImagesAndDocs\ReadMe.txt`

   c. Click **Preview** to view the **Important Note** panel.



Please Read Before Continuing:

```
OFFICE SUITE FOR JAVA
README DOCUMENT

Please note: this is a special pre-release of
Office Suite for Java by ACME CORPORATION.  It
doesn't do much.  But it is cool and does let you
type.  Be certain to send your feedback to
OfficeSuite@ACME.com

Thanks!

- The ACME team


Copyright 1998(c) ACME CORPORATION.  All rights
reserved.
```

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

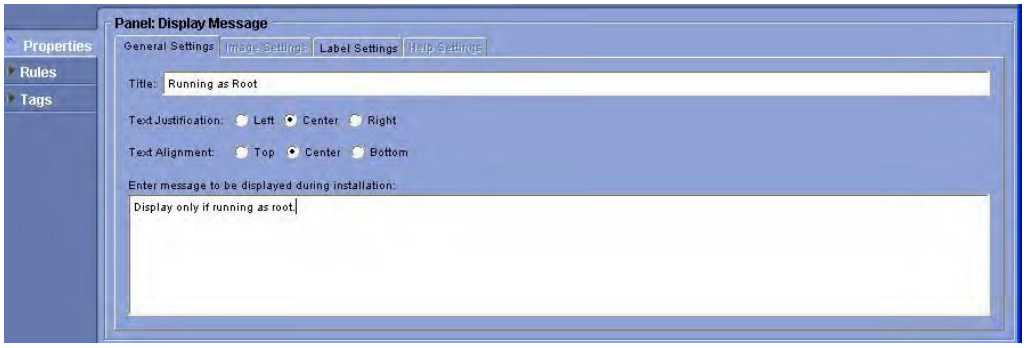
8. Continue with the steps in Adding a Rule to the Display Message Panel to add a rule to the **Display Message** panel.

Adding a Rule to the Display Message Panel

In this exercise, you will be adding a rule to determine whether the **Display Message** panel that you added in Adding Panels to an Installer is displayed.

Task ***To add a rule to determine whether the Display Message panel is displayed:***

1. Perform the steps in Adding Panels to an Installer.
2. Select **Panel: Display Message** in the **Pre-Install Action List**. The **Panel: Display Message** customizer is displayed.



Title: Running as Root

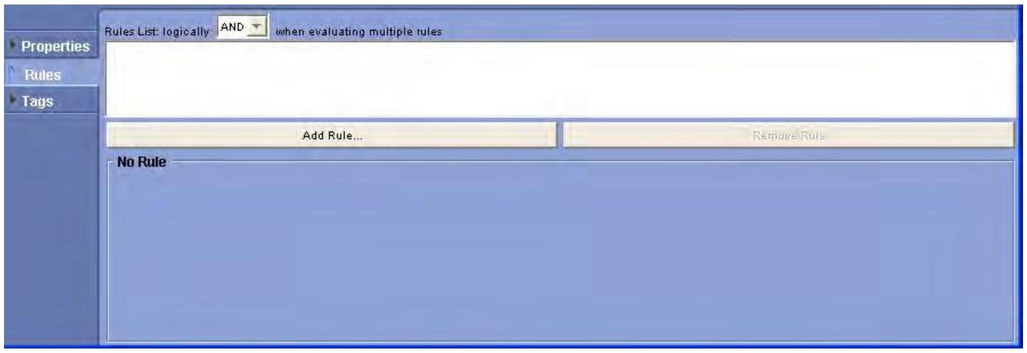
Text Justification: Left [✓ Center] Right

Text Alignment: Top [✓ Center] Bottom

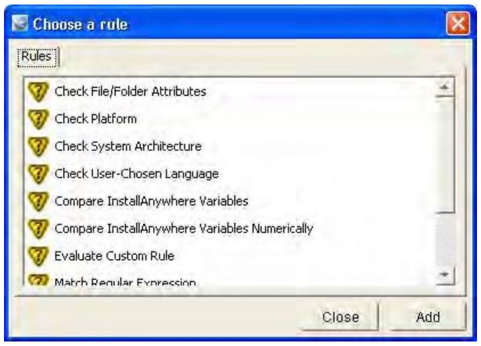
Enter message to be displayed during installation:

Display only if running as root.

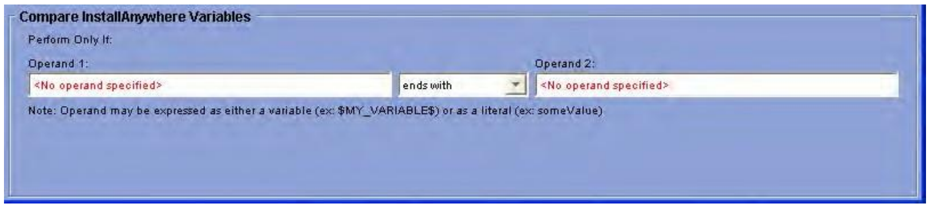
3. In the **Customizer**, open the **Rules** tab.



4. Click **Add Rule**. The **Choose a rule** dialog box opens.


Choose a rule
Rules
Check File/Folder Attributes
Check Platform
Check System Architecture
Check User-Chosen Language
Compare InstallAnywhere Variables
Compare InstallAnywhere Variables Numerically
Evaluate Custom Rule
Match Regular Expression
Close
Add

5. Select **Compare InstallAnywhere Variables**, then click **Add**. The **Compare InstallAnywhere Variables** rule is listed in the **Rules List** and the **Compare InstallAnywhere Variables** customizer opens:



Note: Operand may be expressed as either a variable (ex: $MY_VARIABLE$) or as a literal (ex: someValue)

6. Define this rule by doing the following:

   a. In the **Operand 1** field, enter **`$prop.os.name$`**.

   b. Set the operator to **contains**.

   c. In the **Operand 2** field, enter **`Windows`**.

   The panel will only display if the installation is being run on a Windows operating system.

   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.

7. Build and run the new installer project and notice the addition of the new panels. If you are running the installation on a Windows operating system, you will see the **Display Message** panel because it meets the condition of the rule.

***Note** • As a test, you could edit the rule on the **Display Message** panel by changing the operator to **does not contain**. The next time you build and run the installer project, the **Display Message** panel will not appear.*

To understand the basic behavior of InstallAnywhere functionality, you need to have a basic understanding of InstallAnywhere variables and a basic understanding of InstallAnywhere rules. In the next section, Managing Installer Flow Based on End-User Input, 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 Target File** 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.

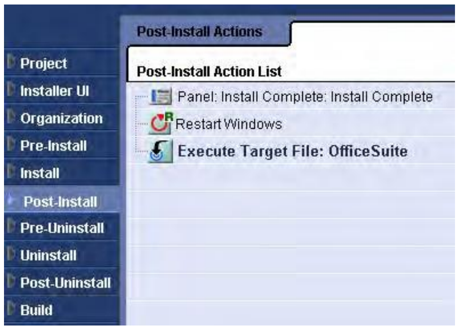
- Step One: Retrieving End-User Input
- Step Two: Applying the End User’s Choice

Step One: Retrieving End-User Input

To create an installer that retrieves end-user input, perform the following steps.

Task ***To retrieve end-user input:***

1. Perform the steps in Using Rules to Control Visual Elements.
2. Open the **Post-Install** task of the project. Three actions are listed in the **Post-Install Action List**:

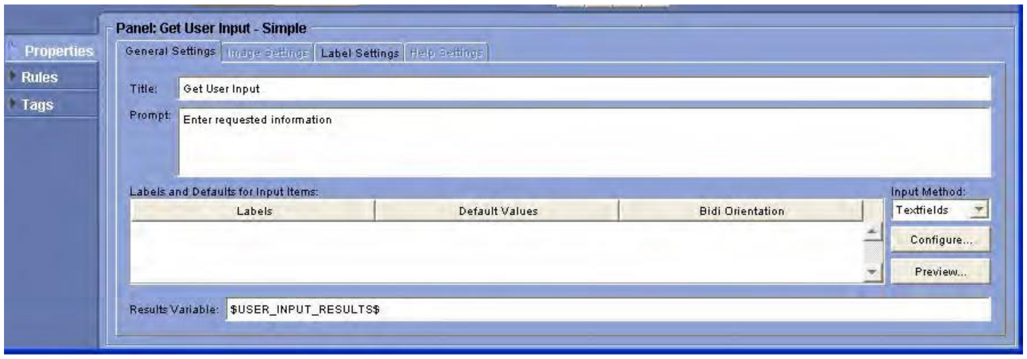


Post-Install Action List
Panel: Install Complete: Install Complete
Restart Windows
Execute Target File: OfficeSuite

Note • *Later you will create some new projects, but in this section you will continue to work with the OfficeSuite installer.*

3. Click **Add Action**. The **Actions** palette opens.

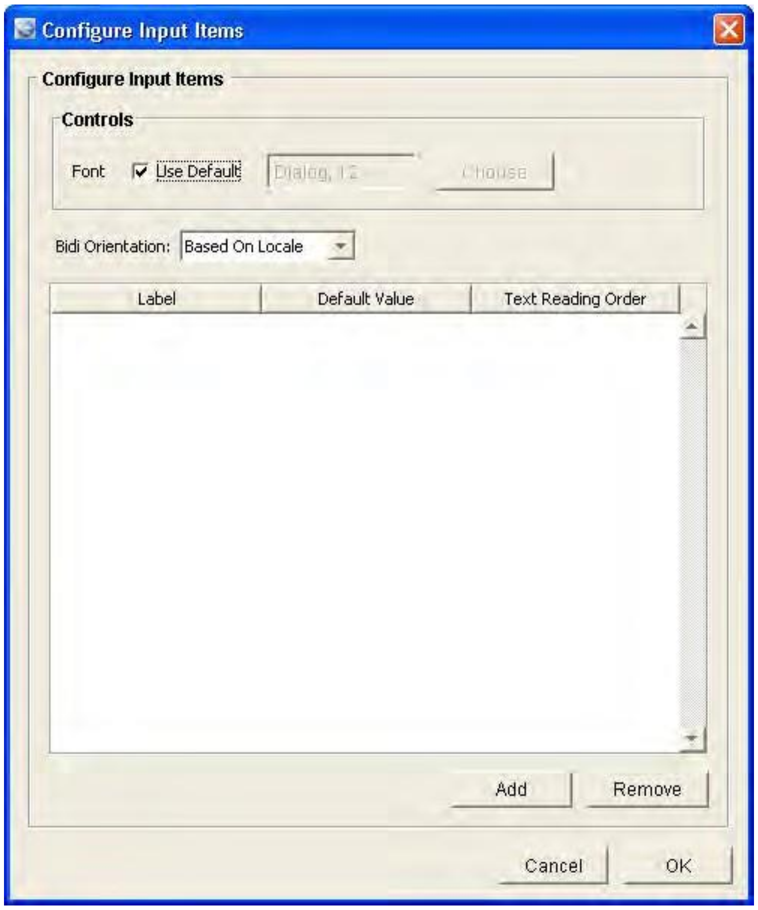
4. On the **Panels** tab, select **Panel: Get User Input - Simple** and click **Add**. The **Panel: Get User Input - Simple** action is added to the **Post-install Action List** and its customizer opens:



Title: Get User Input
Prompt: Enter requested information
Labels and Defaults for Input Items:
Input Method: Textfields
Results Variable: $USER_INPUT_RESULTS$

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.

5. In the **Title** field of the **Panel: Get User Input - Simple** customizer, enter **Launch**.
6. In the **Prompt** text field, enter **`Would you like to Launch OfficeSuite?`**
7. Because this is a Yes/No question, select **Radio Buttons** from the **Input Method** list.
8. Click the **Configure** button. The **Configure Input Items** dialog box opens, where you can add labels for the buttons and set their default states.



9. Click **Add**. An entry is added to the **Label** list.

10. Enter the following information to define the **Yes** label:

Property	Description
Label	Double-click in this field and enter the following text: `Yes, Launch OfficeSuite now.` ***Note*** • *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.*
Default Value	Selected
Text Reading Order	Based On Locale

11. Click **Add**. An second entry is added to the **Label** list.

12. Enter the following information to define the **No** label:

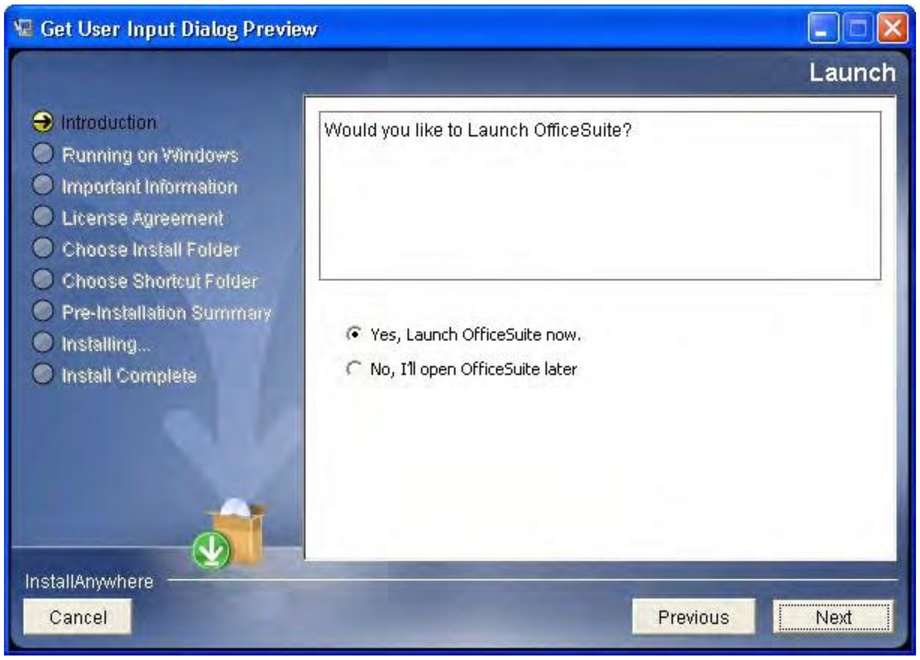
Property	Description
Label	Double-click in this field and enter the following text: `No, I'll open OfficeSuite later.`
Default Value	Not Selected
Text Reading Order	Based On Locale

13. Click **OK** to close the **Configure Input Items** dialog box.

14. Set the **Results Variable** to `$LAUNCH_APPLICATION$`.

Note • *The default results variable is* `$USER_INPUT_RESULTS$`. *However, you can change the variable to fit your needs or your naming scheme.*

15. Click the **Preview** button. You should see a preview of the `Get User Input` panel displaying two radio buttons.



Would you like to Launch OfficeSuite?

(●) Yes, Launch OfficeSuite now.
( ) No, I'll open OfficeSuite later

16. Click **Cancel** to close the preview panel.

17. In the **Post-Install Action List**, use the arrow keys to move the **Panel: Get User Input - Simple: Launch** action before the **Execute Target File** action.

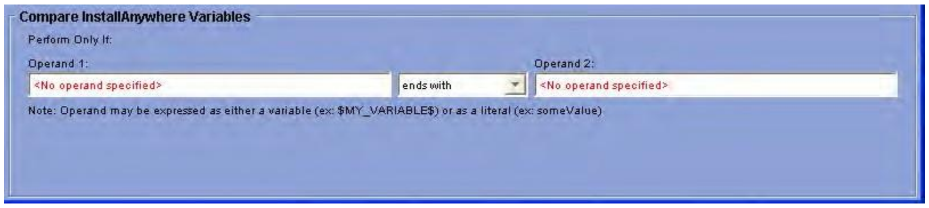
18. Continue with the steps in Step Two: Applying the End User's Choice.

Step Two: Applying the End User's Choice

In this exercise, to control the **Execute Target File** action added earlier, you are going to add a rule to that action. You want the **Execute Target File** 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 to accomplish this.

Task ***To apply the end user's choice:***

1. Perform the steps in Step One: Retrieving End-User Input.
2. Select the **Execute Target File** action in the **Post-Install Action List**. The **Execute Target File** customizer opens.
3. Open the **Rules** tab of the customizer.
4. Click **Add Rule** and select **Compare InstallAnywhere Variables** from the **Choose a rule** dialog box. The **Compare InstallAnywhere Variables** customizer opens.


Compare InstallAnywhere Variables
Perform Only If:
Operand 1: <No operand specified>
ends with
Operand 2: <No operand specified>
Note: Operand may be expressed as either a variable (ex: $MY_VARIABLE$) or as a literal (ex: someValue)

5. Define this rule by doing the following:

   a. In the **Operand 1** field, enter **`$LAUNCH_APPLICATION$`**.

   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.*

   b. Set the operator to **contains**.

   c. In the **Operand 2** field, enter **Yes**.

   The **Execute Target File** action will only be executed if the end user chooses the **Yes** option.

6. Rebuild and run 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, I'll open OfficeSuite later**, check your rule to ensure that you are correctly comparing the variable and that the case of the value is correct.

7. Continue with the steps in Changing the Splash Screen.

Changing the Splash Screen

In this exercise, you are going to change the splash screen for the TrainApp project.

Task ***To change the installer splash screen:***

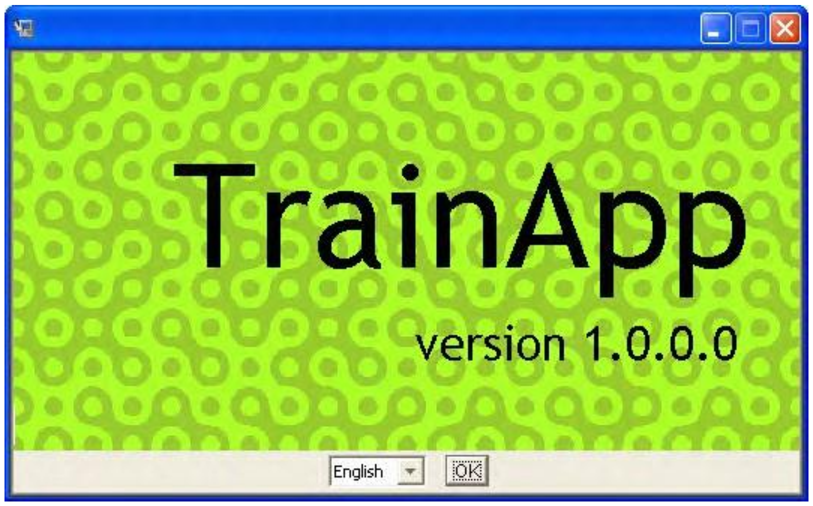
1. On the **File** menu, point to **Open Recent** and click **TrainApp.iap_xml.**

   Note • *If you are prompted to save the `My_Project` project, click **Save**.*

2. Open the **Installer UI > Look & Feel** task.
3. On the **General UI Settings** tab under **Startup Splash Screen**, click the **Choose** button and select the `TrainAppSplash.png` file in the `TrainApp` directory provided with your course files.

   Note • *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.*

4. In the **Startup Splash Screen** section, click **Preview** to view the new splash screen.



   After building and running your project, this splash screen will appear.

5. Click **OK** to close the splash screen.
6. Continue with the steps in Changing the Background Image.

Changing the Background Image

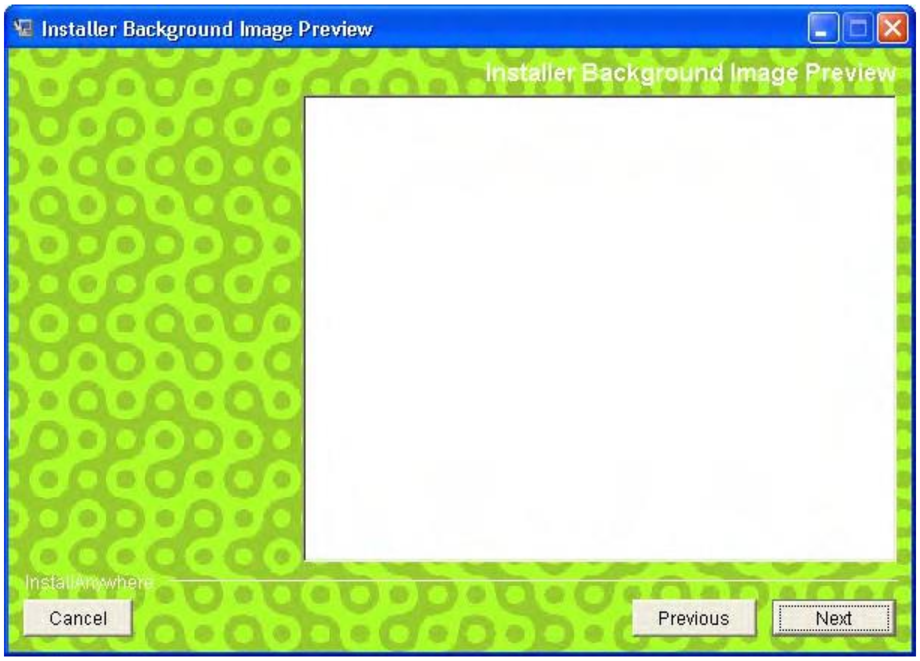
In this exercise, you are going to change the installer background image used in the TrainApp project.

Task ***To change the installer background image:***

1. Perform the steps in Changing the Splash Screen.
2. Open the **Installer UI > Look & Feel** task.

3. On the **General UI Settings** tab under **Installer Background Image**, click the **Choose** button and select the `TrainAppBackground.png` file in the `TrainApp` directory provided with your course files.

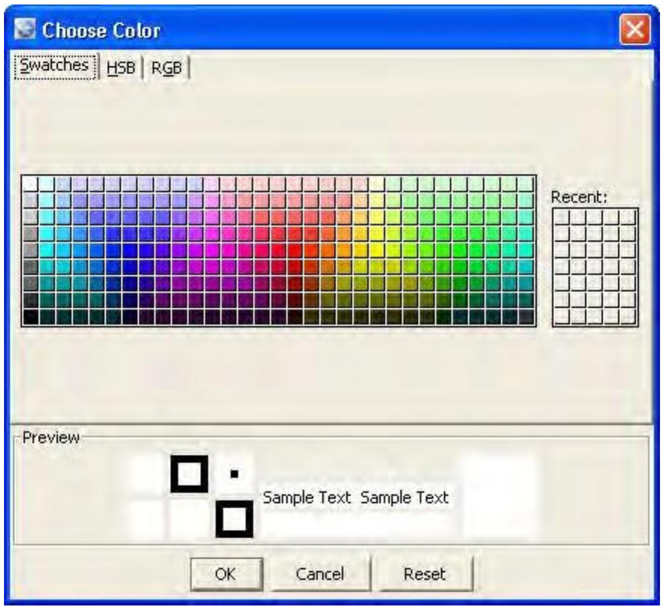
4. In the **Installer Background Image** section, click **Preview** to view the new background image.


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

5. To ensure the panel labels are legible on this light-colored background, you need to set the text color to black:

   a. Clear the **Use default** option next to **Titles, Labels, and InstallAnywhere Logo color**.

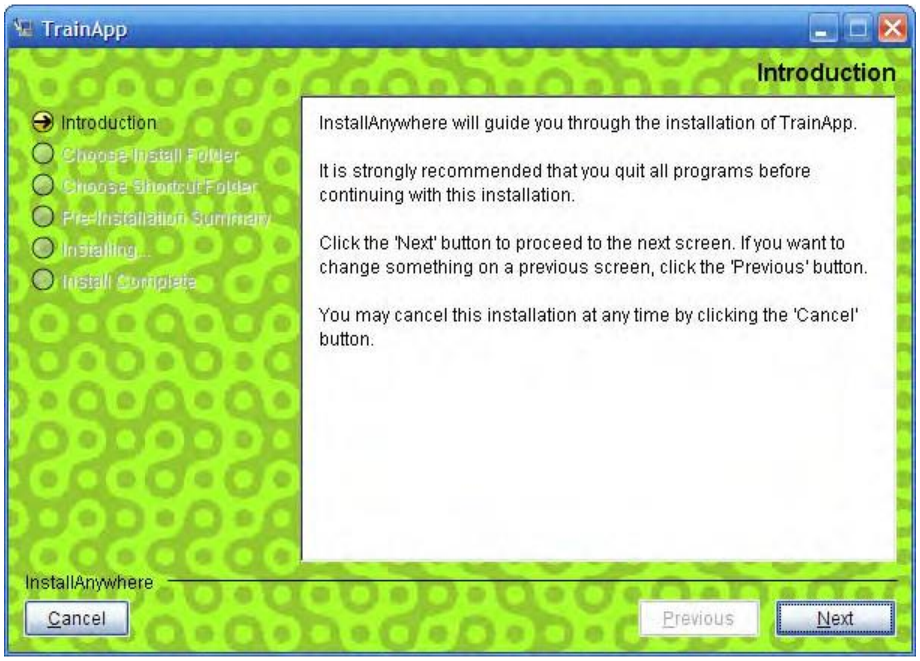
   b. Click the **Choose** button to open the **Choose Color** dialog box.



   c. Choose black (#000000) by clicking the black box in the bottom, left corner of the color grid.

   d. Click **OK**.

6. Build and run the project. The installer panels should appear similar to the following at run time:



InstallAnywhere will guide you through the installation of TrainApp.

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

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

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

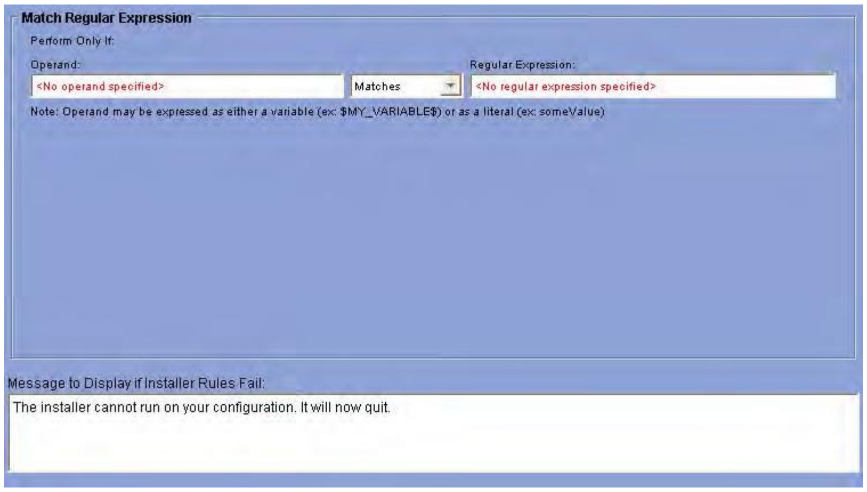
7. Continue with the steps in Creating Installer Rules.

Creating Installer Rules

In this exercise you will create an installer rule that exits the installation if a Sun VM (for example) is being used.

Task ***To create an installer rule:***

1. Perform the steps in Changing the Background Image.
2. Open the **Project > Rules** task.
3. Click **Add Rule**. The **Choose a rule** dialog box opens.
4. Select **Match Regular Expression** and click **Add**. The **Match Regular Expression** customizer is displayed.



<No operand specified>

<No regular expression specified>

Note: Operand may be expressed as either a variable (ex: $MY_VARIABLE$) or as a literal (ex: someValue)

Message to Display if Installer Rules Fail:

The installer cannot run on your configuration. It will now quit.

5. Set **Operand** to **`$prop.java.vendor$`** (which is the value of the `java.vendor` Java system property).

6. Set the operator to **Does not match**.

7. Set the **Regular Expression** to **^Sun** (strings that begin with "Sun").

8. Enter the following text in the **Message to Display if Installer Rules Fail** text box.

   ```
   This installer cannot run on a Sun VM.
   ```

9. Build and run your project to verify that the rule fails (because your machine is using a Sun VM) and that the installer cannot continue.

10. When finished, remove the **Match Regular Expression** rule to ensure future examples are not affected by it.

11. Continue with the exercises in Using the Basic Installer Organization.

Chapter 6 Exercises: Installer Organization

In this section, you will review basic installer organization, learn about Magic Folders and SpeedFolders, and learn how to modify the default uninstaller.

- Using the Basic Installer Organization
- Magic Folders
- Examining the TrainApp Project Organization
- Working with SpeedFolders
- Modifying the Default Uninstaller

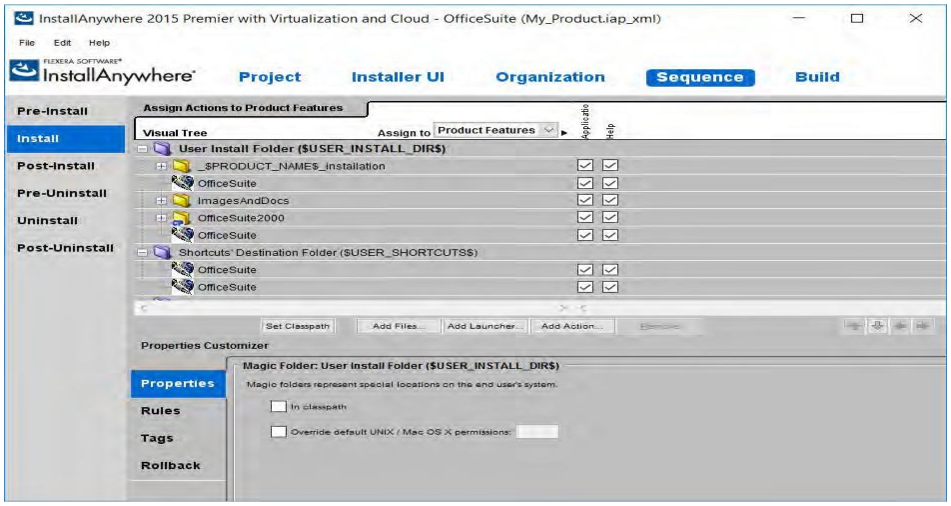
Using the Basic Installer Organization

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.

Task ***To modify the organization of the Install task:***

1. Open the `My_Product.iap_xml` project file (your OfficeSuite Project) in the InstallAnywhere Advanced Designer.
2. Open the **Install** task. You should have a two directories of files named `ImagesAndDocs` and `OfficeSuite2000`.


InstallAnywhere 2015 Premier with Virtualization and Cloud - OfficeSuite (My_Product.iap_xml)
File
Edit
Help
InstallAnywhere
Project
Installer UI
Organization
Sequence
Build
Pre-Install
Install
Post-Install
Pre-Uninstall
Uninstall
Post-Uninstall
Assign Actions to Product Features
Visual Tree
Assign to
Product Features
Application
Help
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
OfficeSuite
ImagesAndDocs
OfficeSuite2000
OfficeSuite
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
OfficeSuite
OfficeSuite
Set Classpath
Add Files...
Add Launcher...
Add Action...
Remove
Properties Customizer
Magic Folder: User Install Folder ($USER_INSTALL_DIR$)
Properties
Rules
Tags
Rollback
Magic folders represent special locations on the end user's system.
In classpath
Override default UNIX / Mac OS X permissions:

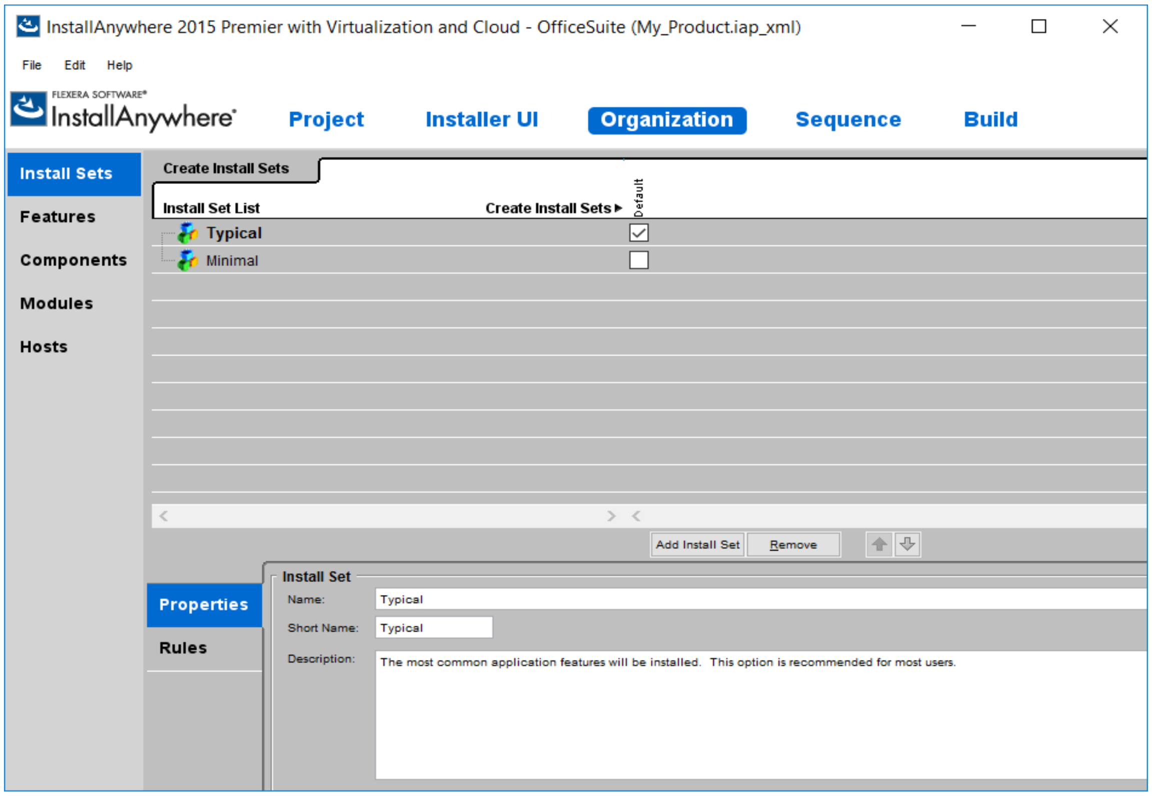
3. Using either the left and right arrows, or the drag and drop functionality, make sure that the `OfficeSuite2000` and `ImagesAndDocs` folders are at the root of your installation—the **User Install Folder ($USER_INSTALL_DIR$)**.

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

4. Rename the `OfficeSuite2000` directory to **`Program`** and rename the `ImagesAndDocs` directory to **`Data`**. You do this by selecting the directory, and then editing the value in the **Name** field in its customizer.

   Note • *InstallAnywhere allows you to rename resources independently of the source directories and files. 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.*

5. Open the **Organization > Install Sets** task.



Install Set List:

Install Set	Default
Typical	✓
Minimal	

Install Set

Name: Typical

Short Name: Typical

Description: The most common application features will be installed. This option is recommended for most users.

6. By default, an InstallAnywhere project contains two Install Sets: **Typical** and **Minimal**. To add a third Install Set, click **Add Install Set**. An untitled Install Set is added to the **Install Set List**.

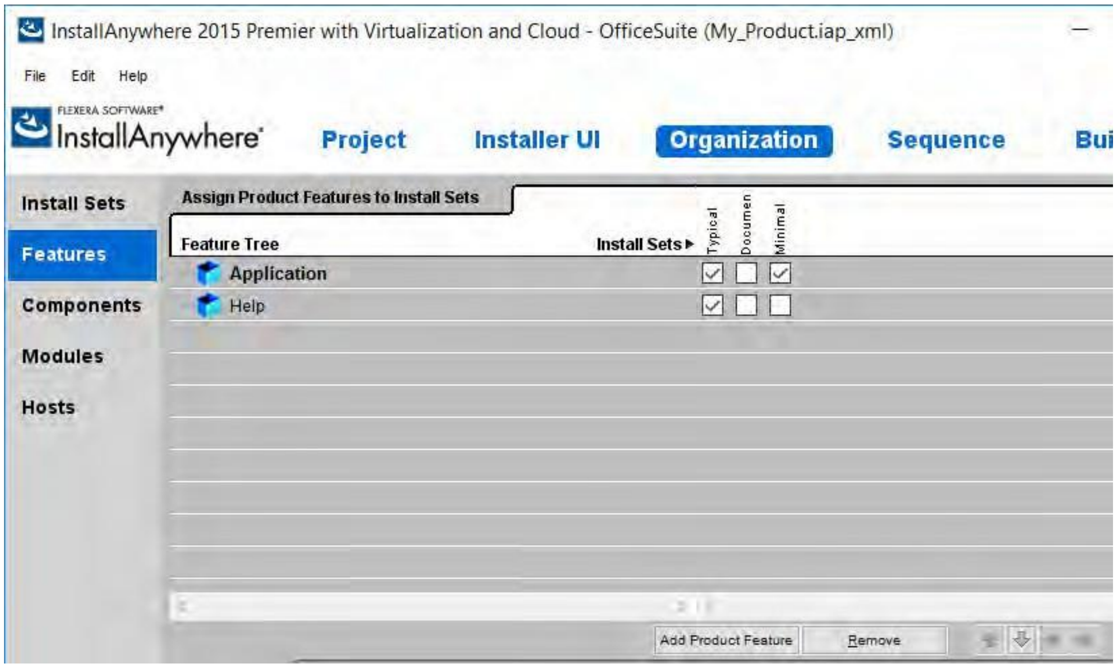
7. In the **Name** field for the new Install Set, enter `Documentation Only`.

8. In the **Description** field, enter `Only the user documentation will be installed`.

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 only need to enter the **Description**.*

9. Select **Typical** as the default by selecting the check box in the **Default** column.

10. Now you are ready to assign features to the **Install Sets**. Open the **Organization > Features** task. In this simple example, you have only two features: **Application** and **Help**.



Feature Tree	Typical	Documen	Minimal
Application	✓		✓
Help	✓		

***Note** • 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**.*

11. Now you will assign the features to the **Install Sets**, which are listed to the right of the **Feature Tree** on the **Organization > Features** task:

    a. For the **Application** feature, select the **Typical** and **Minimal** Install Sets.

    b. For the **Help** feature, select **Typical** and **Documentation Only** (making sure to not select **Minimal**).

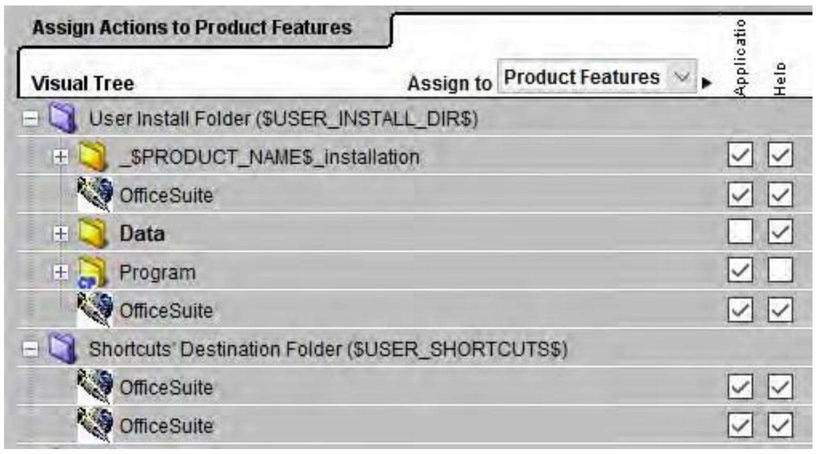
12. The check boxes along the right hand side of the **Visual Tree** in the **Install** task are used to assign files, folders, and actions to features within the task. To assign files to features in the **Install** task, perform the following steps:

    a. Open the **Install** task.

    b. Assign the entire contents of the `Program` folder to the **Application** feature.

    c. Assign the entire contents of the `Data` folder to the **Help** feature.

    The **Visual Tree** should now look like this:


Assign Actions to Product Features
Visual Tree
Assign to Product Features
Applicatio
Help

Visual Tree	Applicatio	Help
User Install Folder ($USER_INSTALL_DIR$)		
_$PRODUCT_NAME$_installation	✓	✓
OfficeSuite	✓	✓
Data		✓
Program	✓	
OfficeSuite	✓	✓
Shortcuts' Destination Folder ($USER_SHORTCUTS$)		
OfficeSuite	✓	✓
OfficeSuite	✓	✓

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.

13. In order to offer your user a choice, you need to add a panel to the **Pre-Install** that will allow them to select the Install Set.

    a. Open the **Pre-Install** task.

    b. Click **Add Action** and select **Panel: Choose Install Sets** from the **Panels** tab of the **Actions** palette.

    c. Use the arrows or the drag and drop functionality to move the **Choose Install Sets** panel immediately after the **Panel: Introduction**.

    d. In the **Panel: Choose Install Sets** customizer, select the **Choose Install Sets [followed by] Choose Product Features** option.

    Note • *If you select the **Choose Install Sets [followed by] Choose Product Features** option, the user will be able to choose individual features to install. While not strictly necessary in an installation as simple as the current example, it is suggested that you select this option so that you may see the results. The **Choose Product Features [only]** option allows you to present only the features to your user, creating a highly flexible installer. For this example, do not select this option.*

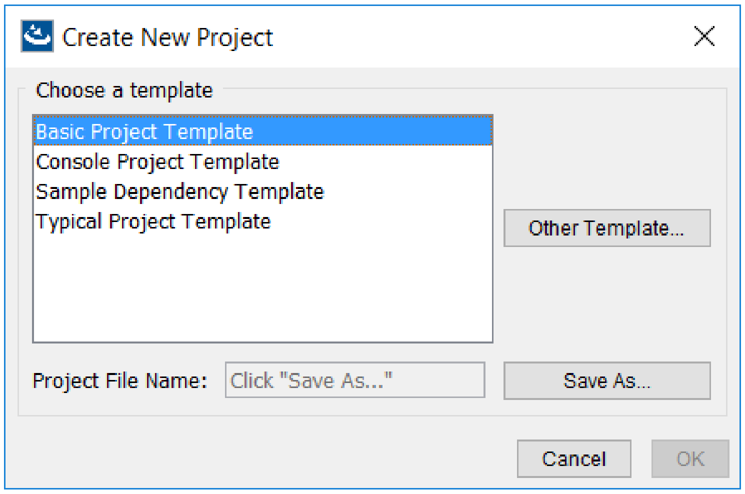
14. Build and run the installer. When the **Choose Install Sets** panel opens, select **Custom** so that you can see the results of selecting the **Choose Install Sets [followed by] Choose Product Features** option.

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.

Task ***To implement Magic Folders:***

1. On the **File** menu of the Advanced Designer, click **New**. The **Create New Project** dialog box opens.



2. With **Basic Project Template** selected, click **Save As**. The Save new project as dialog box opens.
3. Name the new project **`MagicFoldersProject`**.
4. 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
```

5. Back in the Advanced Designer, open the **Install** task.
6. Click **Add Files** and add all the text files you have just created to your `MagicFoldersProject` installer project.
7. You will now set Magic Folder destinations for each file. 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. Select each of the text files that you just added and set the **Path** property in its customizer to the following:

File	Path
`Desktop.txt`	Desktop Folder
`SystemDriveRoot.txt`	System Drive Root
`Home.txt`	Home Directory
`Temp.txt`	Temp Directory

File	Path
`Programs.txt`	Programs Folder
`System.txt`	System Folder
`MagicOne.txt`	$USER_MAGIC_FOLDER_1$
`MagicTwo.txt`	$USER_MAGIC_FOLDER_2$

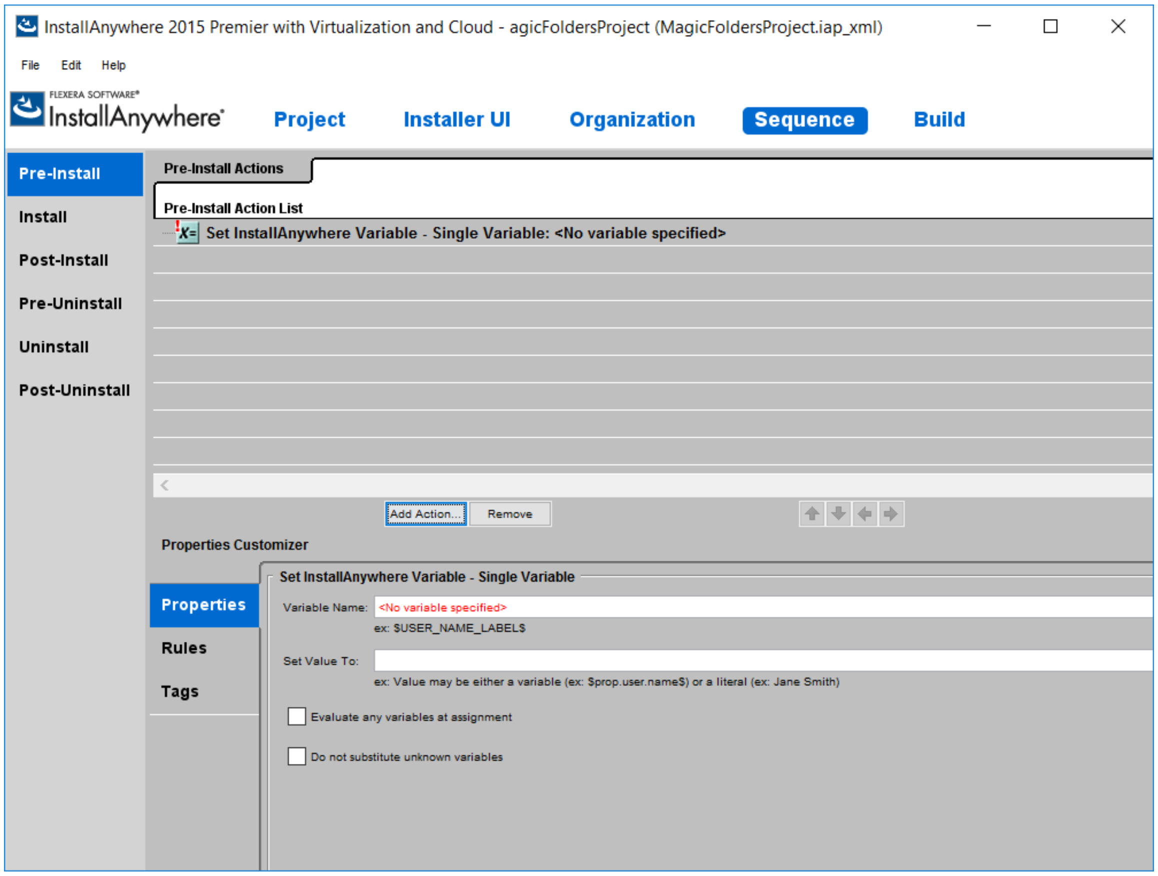
8. Because folders must be defined before they are installed, we need to define the User Magic Folders (`$USER_MAGIC_FOLDER_1$` and `$USER_MAGIC_FOLDER_2$`) that we have selected by performing the following steps:

a. Open the **Pre-Install** task.

b. Because we are only exploring Magic Folders in this installer, remove all the panels from the **Pre-Install** task by selecting them and clicking the **Remove** button.

Tip • *This will allow our installer to run more quickly, and without interaction.*

c. Click **Add Action** and select **Set InstallAnywhere Variable - Single Variable** from the **General** tab of the **Actions** palette. The action is added to the **Pre-Install Action List**.



Set InstallAnywhere Variable - Single Variable: <No variable specified>

Set InstallAnywhere Variable - Single Variable

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

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

☐ Evaluate any variables at assignment

☐ Do not substitute unknown variables

This action will define the `$USER_MAGIC_FOLDER_1$` variable, and therefore, define the location for the `MagicOne.txt` file.

d. In the **Variable Name** field in the **Set InstallAnywhere Variable - Single Variable** customizer, enter **`$USER_MAGIC_FOLDER_1$`**.

e. In the **Set Value To** field, enter the path to a location where you have write permissions, such as **`$INSTALL_DRIVE_ROOT$$/$MyFiles1`**.

***Note** • This value uses one of the Magic Folder variables to help define the path. The **$/$** will resolve to the proper **/** or \ or **:** depending on the system used.*

f. Click **Add Action** again and add another **Set InstallAnywhere Variable - Single Variable** action.

g. In the **Variable Name** field for the second action, enter **`$USER_MAGIC_FOLDER_2$`**.

h. In the **Set Value To** field, enter **`$SYSTEM_DRIVE_ROOT$$/$MyFiles2`**.

9. Build and run your installer.

10. After execution, check each file location that you have specified with a Magic Folder to see where, and in fact if, the file was correctly installed.

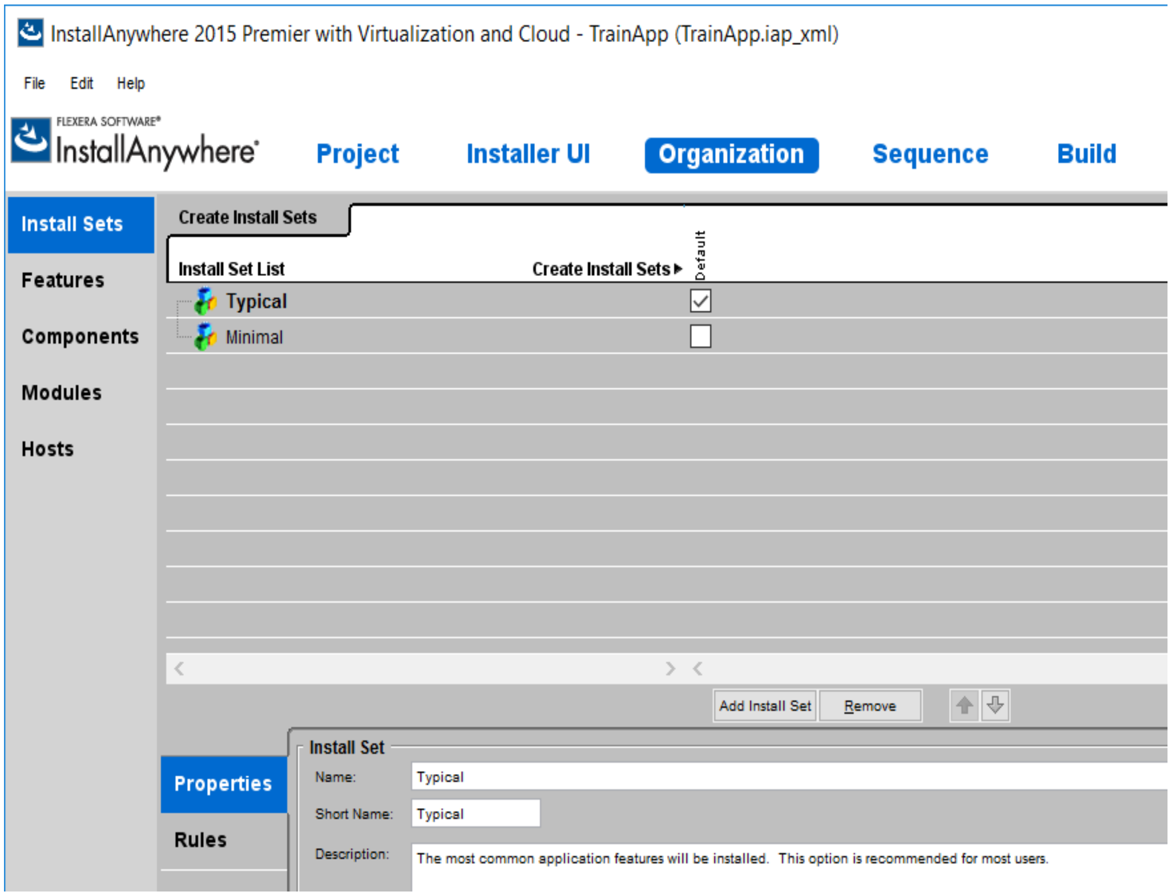
- The `Desktop.txt` file should appear on your desktop.
- The other files will appear in the locations that have been specified.
- The only tricky one here is the `Home.txt` file. This should appear in the user's home directory, which is highly variable. On most Windows systems, this will be in `C:\Users\USERNAME` and on Unix and derivative systems it will be `~$USER`.

Examining the TrainApp Project Organization

In this exercise, you will examine the organization of the TrainApp project. In the **Organization** task, you can modify properties of the install sets, features, and components used in your project.

Task ***To examine the organization of the TrainApp project:***

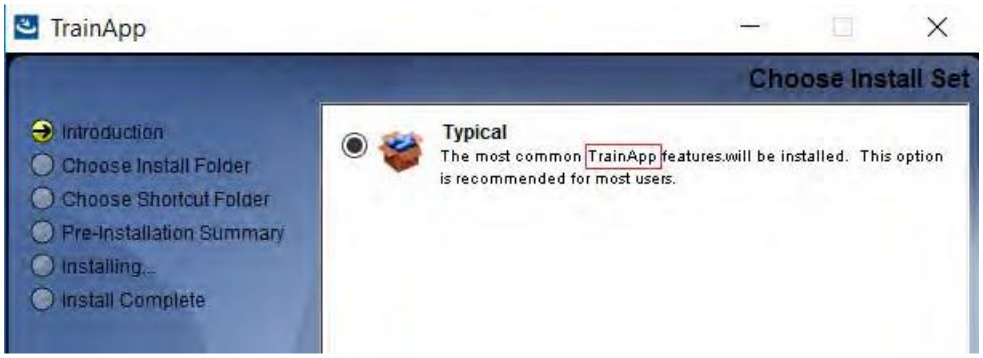
1. Open the `TrainApp.iap_xml` project file in the Advanced Designer.
2. Open the **Organization > Install Sets** task.



Install Set List	Default
Typical	✓
Minimal	

Install Set
Name: Typical
Short Name: Typical
Description: The most common application features will be installed. This option is recommended for most users.

3. Select **Typical** in the **Install Set List**.
4. Change the **Description** displayed to the end user from **`The most common product features`** to **`The most common $PRODUCT_NAME$ features`**. This will result in the product name appearing in the description at run time.
5. Open the **Pre-Install** task.
6. Click **Add Action** and add a **Panel: Choose Install Sets** action from the **Panels** tab of the **Actions** palette.
7. Build this project and run the installer. The **Choose Install Set** panel will display the edited description containing the product name:



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

8. Continue with the steps in Working with SpeedFolders.

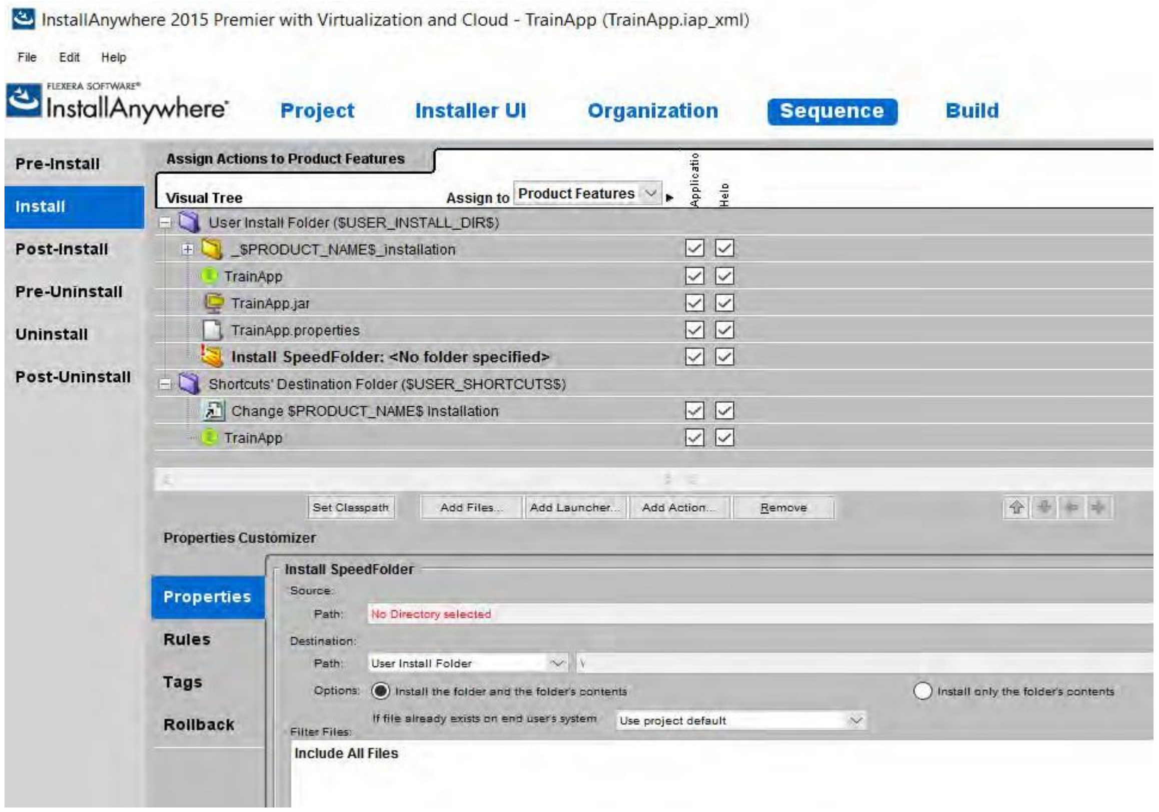
Working with SpeedFolders

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.

Task ***To create a SpeedFolder:***

1. Perform the steps in Examining the TrainApp Project Organization. The `TrainApp` project is open in the Advanced Designer.
2. Open the **Install** task.
3. Click **Add Action**, select the **Install SpeedFolder** action from the **Install** tab of the **Actions** palette, and click **Add**. The **Install SpeedFolder** action is now listed in the **Visual Tree**.


InstallAnywhere 2015 Premier with Virtualization and Cloud - TrainApp (TrainApp.iap_xml)
File Edit Help
FLEXERA SOFTWARE InstallAnywhere
Project
Installer UI
Organization
Sequence
Build
Pre-Install
Install
Post-Install
Pre-Uninstall
Uninstall
Post-Uninstall
Assign Actions to Product Features
Visual Tree
Assign to Product Features
Applicatio
Help
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
TrainApp
TrainApp.jar
TrainApp.properties
Install SpeedFolder: <No folder specified>
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
Change $PRODUCT_NAME$ Installation
TrainApp
Set Classpath
Add Files...
Add Launcher...
Add Action...
Remove
Properties Customizer
Properties
Rules
Tags
Rollback
Install SpeedFolder
Source:
Path: No Directory selected
Destination:
Path: User Install Folder
Options: Install the folder and the folder's contents
Install only the folder's contents
If file already exists on end user's system Use project default
Filter Files:
Include All Files

4. In the **Install SpeedFolder** customizer, click **Choose Folder** next to the **Source Path** field and select the `TrainApp\help` directory as the source.

Note • *As described in Adding Directories with SpeedFolders, you can click **Configure Filter** to open the **Filter Files** dialog box, where you can modify which files and folders are included and excluded, and preview the files that will be included.*

5. In the **Install** view, you can also control which files, shortcuts, and SpeedFolders are included with each feature. Select the **Install SpeedFolder: help** action in the **Visual Tree** and select the **Help** check box in the **Product Features** column, but clear the selection of the **Application** check box.

6. Continue with the steps in Modifying the Default Uninstaller.

Modifying the Default Uninstaller

In this exercise, you will modify the default uninstaller by creating an additional uninstall shortcut and by changing the installation location of the uninstaller.

- Adding an Additional Uninstall Shortcut
- Changing the Installation Location of the Uninstaller

Adding an Additional Uninstall Shortcut

To add an additional uninstall shortcut, perform the following steps.

Task ***To modify the default uninstaller:***

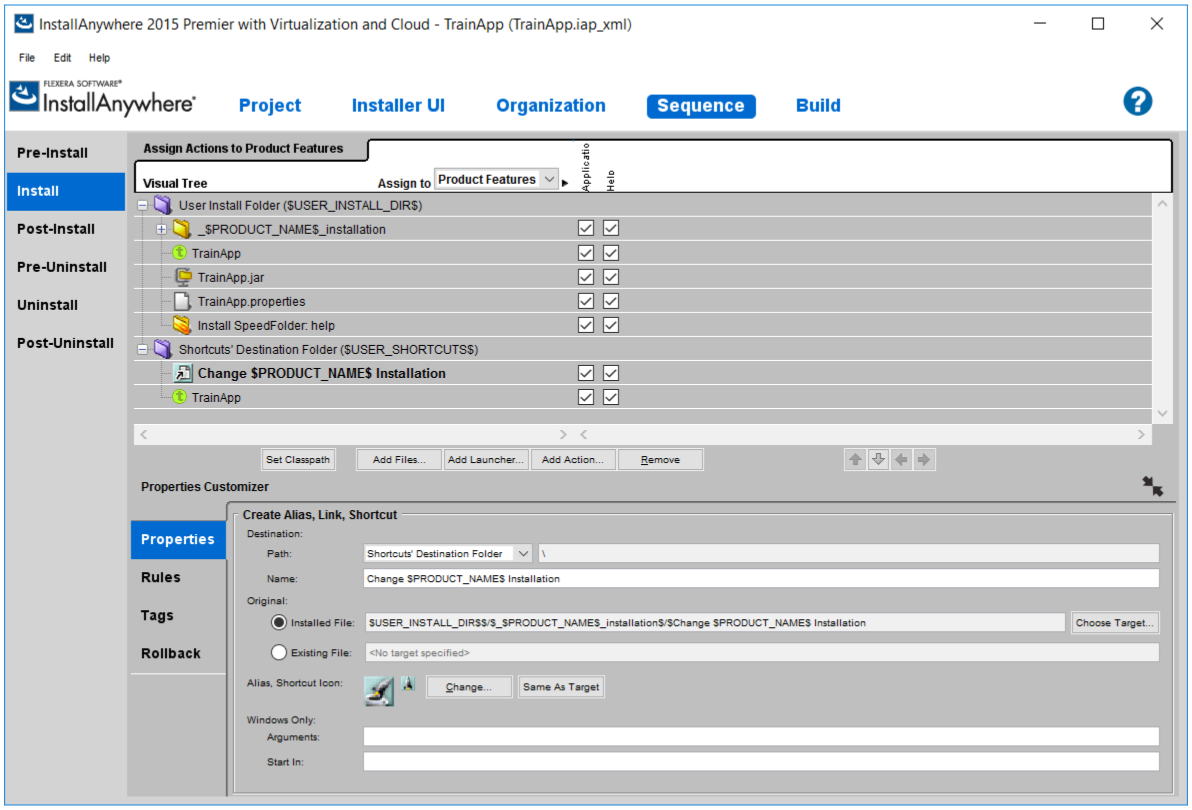
1. Perform the steps in Working with SpeedFolders. The `TrainApp` project is open in the Advanced Designer.

2. Open the **Install** task.

3. Select the **Shortcuts' Destination Folder ($USER_SHORTCUT$)** in the **Visual Tree**.

4. Click **Add Action** to open the **Actions** palette.

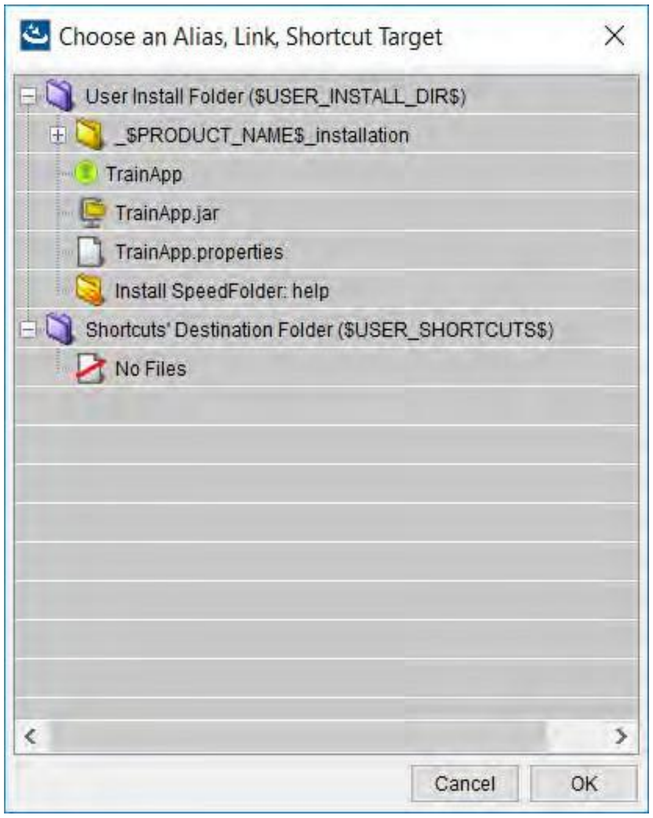
5. Select **Create Alias, Link, Shortcut** from the **Install** tab of the **Actions** palette and click **Add**. The new shortcut is listed in the **Visual Tree**.


InstallAnywhere 2015 Premier with Virtualization and Cloud - TrainApp (TrainApp.iap_xml)
File Edit Help
Project
Installer UI
Organization
Sequence
Build
Pre-Install
Install
Post-Install
Pre-Uninstall
Uninstall
Post-Uninstall
Assign Actions to Product Features
Visual Tree
Assign to Product Features
Applicatio
Help
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
TrainApp
TrainApp.jar
TrainApp.properties
Install SpeedFolder: help
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
Change $PRODUCT_NAME$ Installation
TrainApp
Set Classpath
Add Files...
Add Launcher...
Add Action...
Remove
Properties Customizer
Properties
Rules
Tags
Rollback
Create Alias, Link, Shortcut
Change...
Same As Target
Choose Target...

Destination:
Path: Shortcuts' Destination Folder \
Name: Change $PRODUCT_NAME$ Installation
Original:
● Installed File: $USER_INSTALL_DIR$$/$_$PRODUCT_NAME$_installation$/$Change $PRODUCT_NAME$ Installation
○ Existing File: <No target specified>
Alias, Shortcut Icon:
Windows Only:
Arguments:
Start In:

6. In the **Create Alias, LInk, Shortcut** customizer, verify that the **Installed file** option is selected.

7. Click **Choose Target**. The **Choose an Alias, Link, Shortcut Target** dialog box opens.


Choose an Alias, Link, Shortcut Target
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation
TrainApp
TrainApp.jar
TrainApp.properties
Install SpeedFolder: help
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
No Files
Cancel
OK

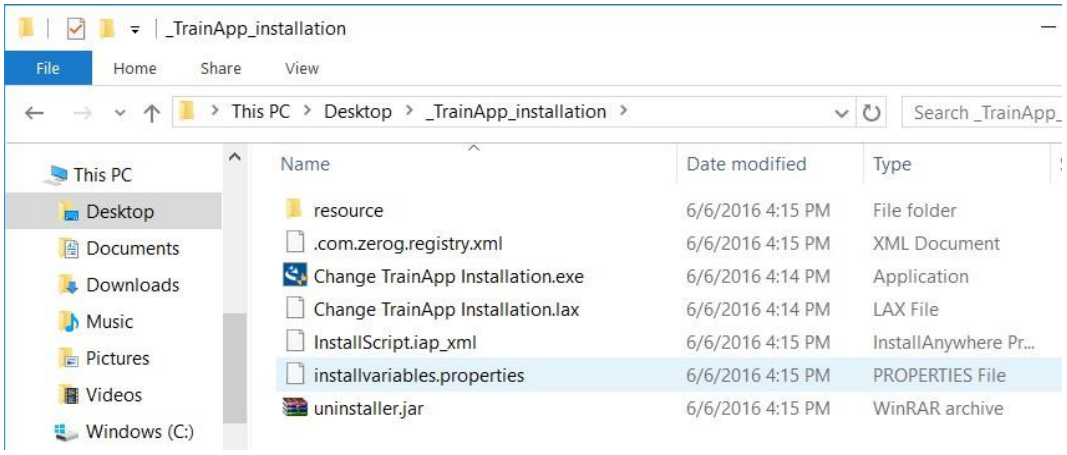
8. Select the **Change $PRODUCT_NAME$ Installation** executable and click **OK**.
9. Build and run the project, and verify that the uninstall shortcut (**Change TrainApp Installation**) is created in the **All Programs** menu.

Changing the Installation Location of the Uninstaller

To change the installation location of the uninstaller, perform the following steps.

Task ***To change the installation location of the uninstaller:***

1. In the **Install** task, right-click the `_$PRODUCT_NAME$_installation` directory icon and select **Move to MagicFolder**.
2. In the **Move to MagicFolder** submenu, select another destination such as **General** > **Desktop Folder**.
3. Build and run the project, and then verify that the uninstaller is created in the new location.



Name	Date modified	Type
resource	6/6/2016 4:15 PM	File folder
.com.zerog.registry.xml	6/6/2016 4:15 PM	XML Document
Change TrainApp Installation.exe	6/6/2016 4:14 PM	Application
Change TrainApp Installation.lax	6/6/2016 4:14 PM	LAX File
InstallScript.iap_xml	6/6/2016 4:15 PM	InstallAnywhere Pr...
installvariables.properties	6/6/2016 4:15 PM	PROPERTIES File
uninstaller.jar	6/6/2016 4:15 PM	WinRAR archive

4. Continue with the steps in Chapter 7 Exercises: Introduction to Advanced Actions and Panel Actions.

Chapter 7 Exercises: Introduction to Advanced Actions and Panel Actions

In this section, you will perform exercises to learn about some of InstallAnywhere’s more advanced and more useful installer actions. These actions represent operations performed by the installer or uninstaller.

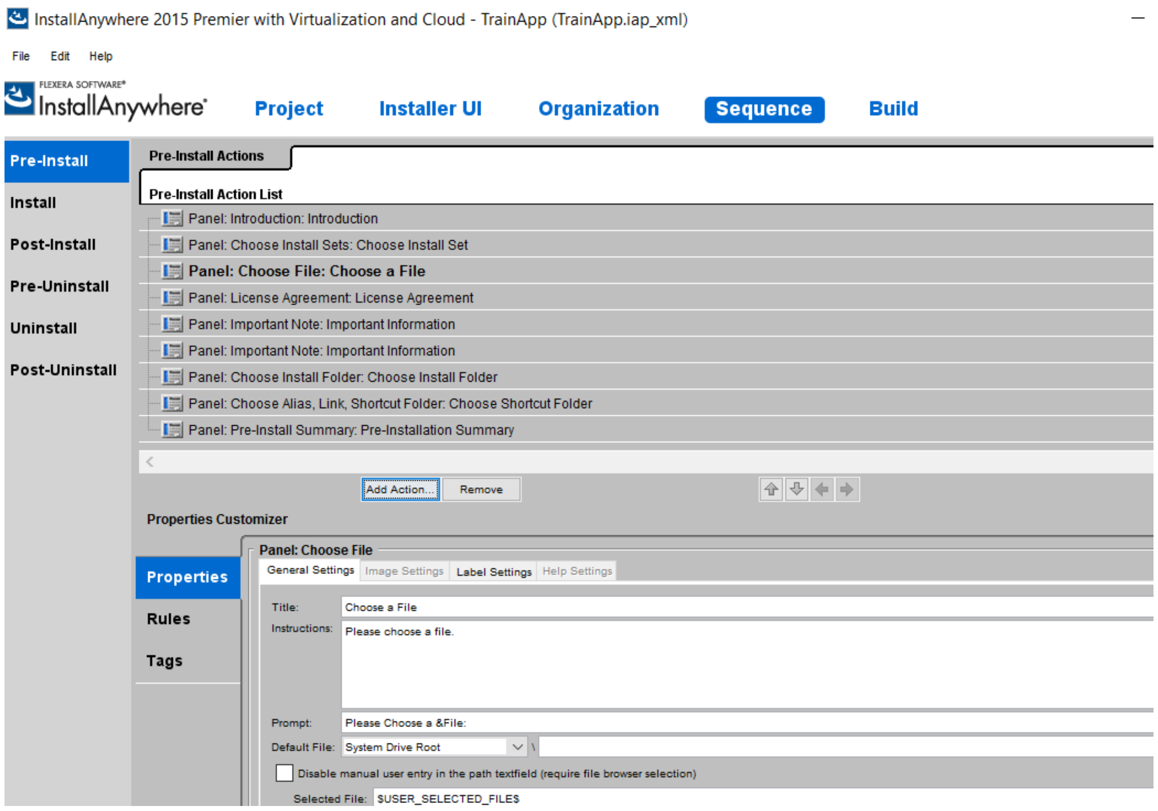
- Using Panels in Pre-Install
- Using Install Task Actions
- Creating Installer Logic Using Jump Labels and Actions
- Modifying TrainApp.properties File at Runtime

Using Panels in Pre-Install

In this exercise, you will add panels to the **Pre-Install** task of your installation and then rebuild it to investigate the appearance and configuration options of those new panels.

Task ***To add panels to the Pre-Install task:***

1. Open the `TrainApp` project in the Advanced Designer.
2. Open the **Pre-Install** task.
3. Add a **Choose File** panel by performing the following steps:
    a. Click **Add Action**, select **Panel: Choose File** from the **Panels** tab of the **Actions** palette, and click **Add**. The **Panel: Choose File** action is added to the **Pre-Install Action List**.



Pre-Install Actions

Pre-Install Action List

- Panel: Introduction: Introduction
- Panel: Choose Install Sets: Choose Install Set
- **Panel: Choose File: Choose a File**
- Panel: License Agreement: License Agreement
- Panel: Important Note: Important Information
- Panel: Important Note: Important Information
- Panel: Choose Install Folder: Choose Install Folder
- Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
- Panel: Pre-Install Summary: Pre-Installation Summary

Properties Customizer

Panel: Choose File

General Settings | Image Settings | Label Settings | Help Settings

Title: Choose a File
Instructions: Please choose a file.
Prompt: Please Choose a &File:
Default File: System Drive Root \
☐ Disable manual user entry in the path textfield (require file browser selection)
Selected File: $USER_SELECTED_FILE$

A **Choose File** panel in an installer prompts the end user to select a file.

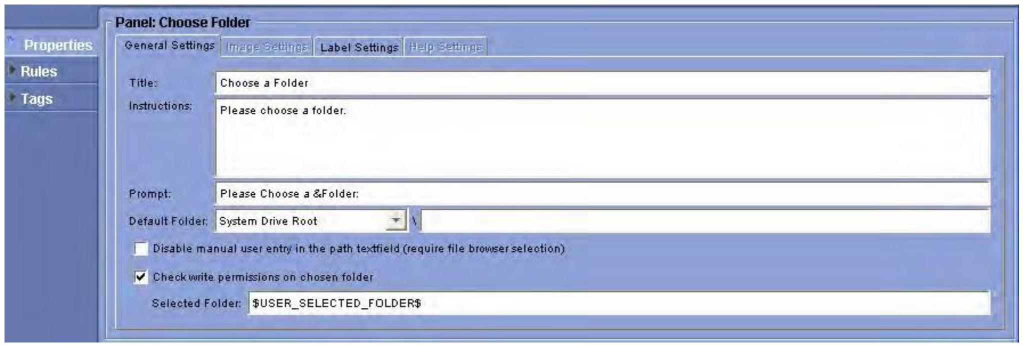
b. In the **Panel: Choose File** customizer, select the **Disable manual user entry in the path textfield (require file browser selection)** to require the user to browse to select the file.

c. In the **Selected File** field, you enter the InstallAnywhere variable that the **Choose File** panel will return to the installer to identify the file name of the selected file. By default, this value is `$USER_SELECTED_FILE$`. For this exercise, do not change this variable.

d. In the **Parent Folder** field, you enter the InstallAnywhere variable that the **Choose File** panel will return to the installer to identify the directory where the selected file is located. By default, this value is `$PATH_OF_SELECTED_FILE$`. For this exercise, do not change this variable.

4. Add a **Choose Folder** panel by performing the following steps:

a. Click **Add Action**, select **Panel: Choose Folder** from the **Panels** tab of the **Actions** palette, and click **Add**. The **Panel: Choose Folder** action is added to the **Pre-Install Action List** and the **Panel: Choose Folder** customizer opens.



Panel: Choose Folder

Title: Choose a Folder

Instructions: Please choose a folder.

Prompt: Please Choose a &Folder:

Default Folder: System Drive Root \

☐ Disable manual user entry in the path textfield (require file browser selection)

☑ Check write permissions on chosen folder

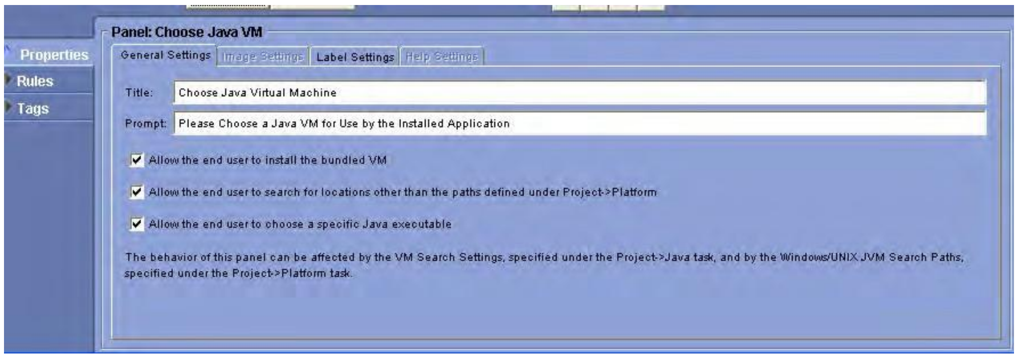
Selected Folder: $USER_SELECTED_FOLDER$

A **Choose Folder** panel requests that the user select a folder, much in the same way the **Choose File** panel works.

b. In the **Selected Folder** field, you enter the InstallAnywhere variable that the Choose Folder panel will return to the installer to identify the folder that was selected. By default, this variable is $USER_SELECTED_FOLDER$. For this exercise, do not change this variable.

c. Select the **Check write permissions on chosen folder** option to check to see if the installer has write permissions for the chosen folder.

5. Add a **Choose Java VM** panel by clicking **Add Action**, selecting **Panel: Choose Java VM** from the **Panels** tab of the **Actions** palette, and clicking **Add**. The **Panel: Choose Java VM** action is added to the **Pre-Install Action List** and the **Panel: Choose Java VM** customizer opens.



Panel: Choose Java VM

Title: Choose Java Virtual Machine

Prompt: Please Choose a Java VM for Use by the Installed Application

☑ Allow the end user to install the bundled VM

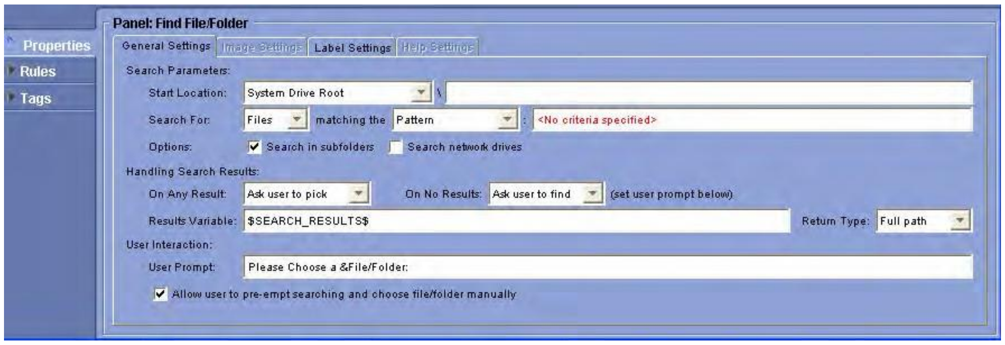
☑ Allow the end user to search for locations other than the paths defined under Project->Platform

☑ Allow the end user to choose a specific Java executable

The behavior of this panel can be affected by the VM Search Settings, specified under the Project->Java task, and by the Windows/UNIX JVM Search Paths, specified under the Project->Platform task.

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

6. Add a **Find File/Folder** panel by clicking **Add Action**, selecting **Panel: Find File/Folder** from the **Panels** tab of the **Actions** palette, and clicking **Add**. The **Panel: Find File/Folder** customizer opens.



Panel: Find File/Folder

Search Parameters:

Start Location: System Drive Root \

Search For: Files matching the Pattern : <No criteria specified>

Options: ☑ Search in subfolders ☐ Search network drives

Handling Search Results:

On Any Result: Ask user to pick On No Results: Ask user to find (set user prompt below)

Results Variable: $SEARCH_RESULTS$ Return Type: Full path

User Interaction:

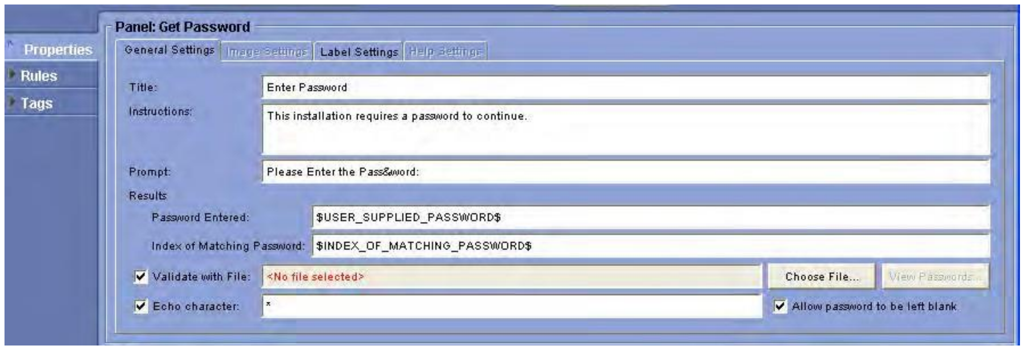
User Prompt: Please Choose a &File/Folder:

☑ Allow user to pre-empt searching and choose file/folder manually

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

7. Add a **Get Password** panel by performing the following steps:

   a. Click **Add Action**, select **Panel: Get Password** from the **Panels** tab of the **Actions** palette, and click **Add**. The **Panel: Get Password** customizer opens.



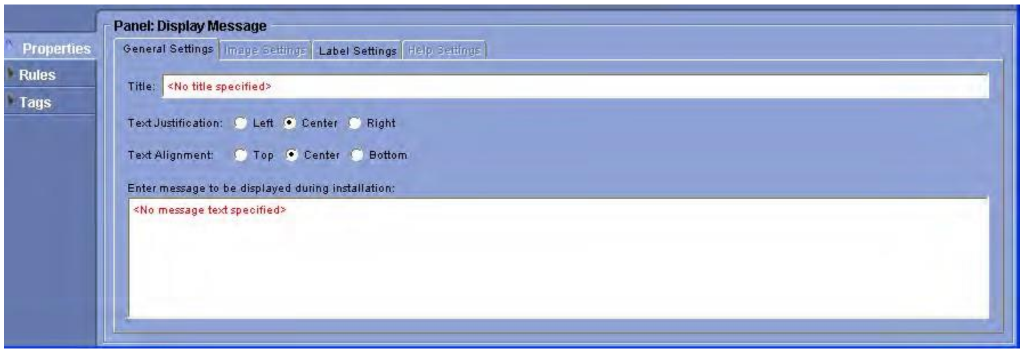
Title: Enter Password
Instructions: This installation requires a password to continue.
Prompt: Please Enter the Pass&word:
Results
Password Entered: $USER_SUPPLIED_PASSWORD$
Index of Matching Password: $INDEX_OF_MATCHING_PASSWORD$
☑ Validate with File: <No file selected>
☑ Echo character: *
☑ Allow password to be left blank

   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.

   b. The **Get Password** panel can be configured either to simply store a masked entry or to confirm against a file. To specify that you want to confirm the password against a file, select the **Validate with File** option.

   c. Next to the **Validate with File** field, click **Choose File** and select the `password.txt` file located in the `OfficeSuiteSourceFiles/ImagesAndDocs` directory.

8. Add a **Display Message** panel by clicking **Add Action**, selecting **Panel: Display Message** from the **Panels** tab of the **Actions** palette, and clicking **Add**. The **Panel: Display Message** customizer opens.



Title: <No title specified>
Text Justification: ○ Left ● Center ○ Right
Text Alignment: ○ Top ● Center ○ Bottom
Enter message to be displayed during installation:
<No message text specified>

   The **Display Message** panel exhibits the results variables from each of your previous panels.

9. Build and run your installer and view the panels that you just added.

10. Continue with the steps in Using Install Task Actions.

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

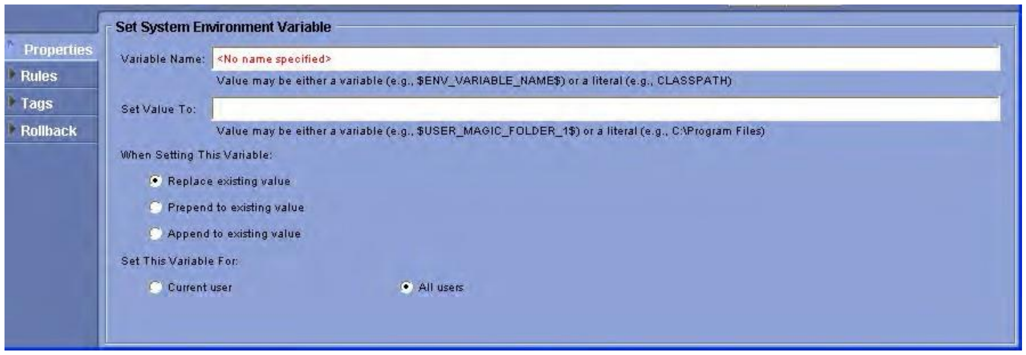
Using Install Task Actions

In this exercise, you will add panels to the **Install** task of your installation and then rebuild it to investigate the appearance and configuration of options of those new panels.

Task ***To add panels to the Install task:***

1. Perform the steps in Using Panels in Pre-Install. The TrainApp project is open in the Advanced Designer.
2. Open the **Install** task.
3. Add an **Install SpeedFolder** action by clicking **Add Action** to open the **Actions** palette, selecting **Install SpeedFolder** on the **Install** tab and clicking **Add**. The **SpeedFolder** is added to the **Visual Tree**.
4. Add a **Set System Environment Variable** action by performing the following steps:

   a. Click **Add Action** to open the **Actions** palette, select **Set System Environment Variable** on the **Install** tab, and click **Add**. The **Set System Environment Variable** action is added to the **Visual Tree** and its customizer opens.



Set System Environment Variable

Variable Name: <No name specified>

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

Set Value To:

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

When Setting This Variable:

- [✓ Replace existing value]
- Prepend to existing value
- Append to existing value

Set This Variable For:

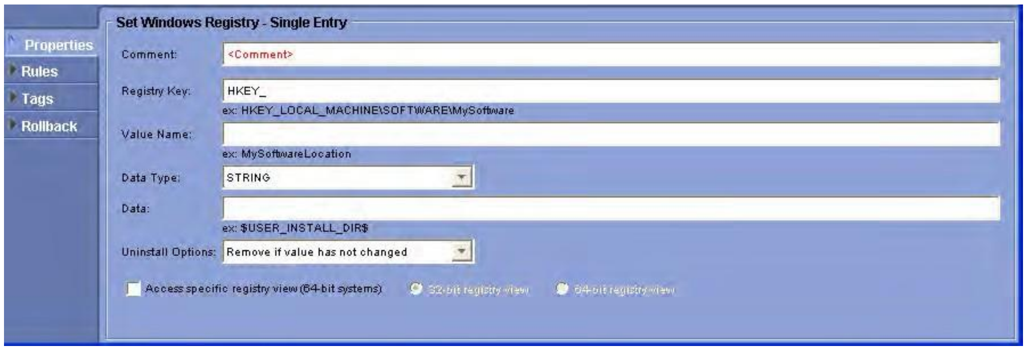
- Current user
- [✓ All users]

   b. In the **Set System Environment Variable** customizer, make the following selections:

Option	Value/Selection
Variable Name:	**`$PRODUCT_NAME$_DIR$`**
Set Value To:	**`$USER_INSTALL_DIR$`**
When Setting This Variable:	Append to existing value
Set This Variable For:	Current user

5. Add a **Set Windows Registry - Single Entry** action by performing the following steps:

   a. Click **Add Action** to open the **Actions** palette, select **Set Windows Registry - Single Entry** on the **General** tab and click **Add**. The **Set Windows Registry - Single Entry** action is added to the **Visual Tree** and its customizer opens.



Set Windows Registry - Single Entry

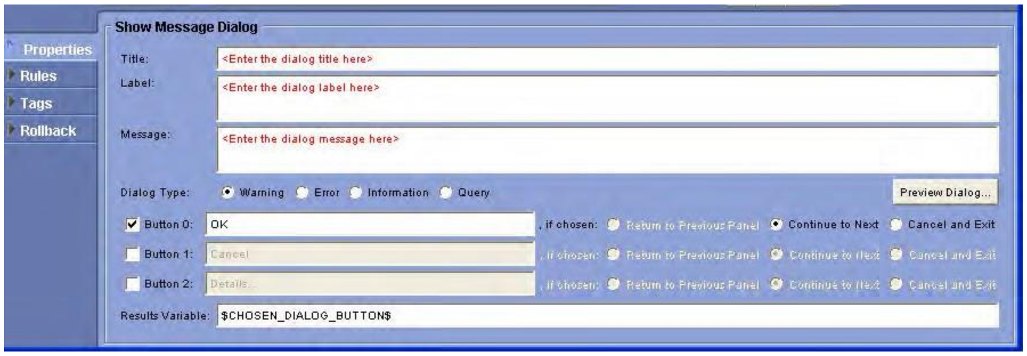
Comment: <Comment>
Registry Key: HKEY_
ex: HKEY_LOCAL_MACHINE\SOFTWARE\MySoftware
Value Name:
ex: MySoftwareLocation
Data Type: STRING
Data:
ex: $USER_INSTALL_DIR$
Uninstall Options: Remove if value has not changed
☐ Access specific registry view (64-bit systems) ○ 32-bit registry view ○ 64-bit registry view

b. In the **Set Windows Registry - Single Entry** customizer, make the following selections to define the registry entry:

Option	Value/Selection
Comment:	`Set Windows Registry Test`
Registry Key:	`HKEY_LOCAL_MACHINE\SOFTWARE\$PRODUCT_NAME$`
Value Name:	`InstallDirectory`
Data Type:	STRING
Data:	`$USER_INSTALL_DIR$`
Uninstall Options:	Remove if value has not changed

6. Add a **Show Message Dialog** action by performing the following steps:

a. Click **Add Action** to open the **Actions** palette, select **Show Message Dialog** on the **General** tab and click **Add**. The **Show Message Dialog** action is added to the **Visual Tree** and its customizer opens.



Show Message Dialog

Title: <Enter the dialog title here>
Label: <Enter the dialog label here>
Message: <Enter the dialog message here>
Dialog Type: [✓ Warning] Error Information Query
☑ Button 0: OK , if chosen: Return to Previous Panel [✓ Continue to Next] Cancel and Exit
☐ Button 1: Cancel , if chosen: Return to Previous Panel Continue to Next Cancel and Exit
☐ Button 2: Details... , if chosen: Return to Previous Panel Continue to Next Cancel and Exit
Results Variable: $CHOSEN_DIALOG_BUTTON$

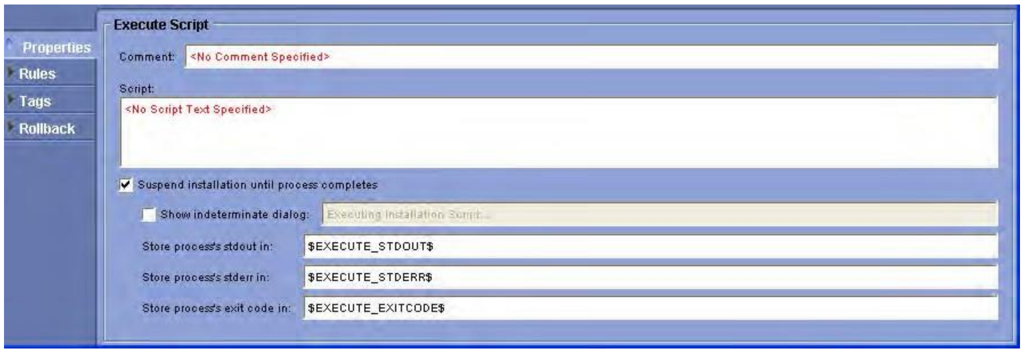
b. In the **Show Message Dialog** customizer, enter the following:

Property	Value
Title	`Variable Name`
Label	`Variable Name and Value`

Message	`Variable Name is $PRODUCT_NAME$_DIR`
	`Value is $USER_INSTALL_DIR$`

7. Add an **Execute Script/Batch File** action by performing the following steps:

a. Click **Add Action** to open the **Actions** palette, select **Execute Script/Batch file** on the **General** tab and click **Add**. The **Execute Script/Batch file** action is added to the **Visual Tree** and its customizer opens.



Execute Script

Comment: <No Comment Specified>

Script: <No Script Text Specified>

✓ Suspend installation until process completes

Show indeterminate dialog: Executing installation Script...

Store process's stdout in: $EXECUTE_STDOUT$

Store process's stderr in: $EXECUTE_STDERR$

Store process's exit code in: $EXECUTE_EXITCODE$

b. In the **Execute Script** customizer, enter the following:

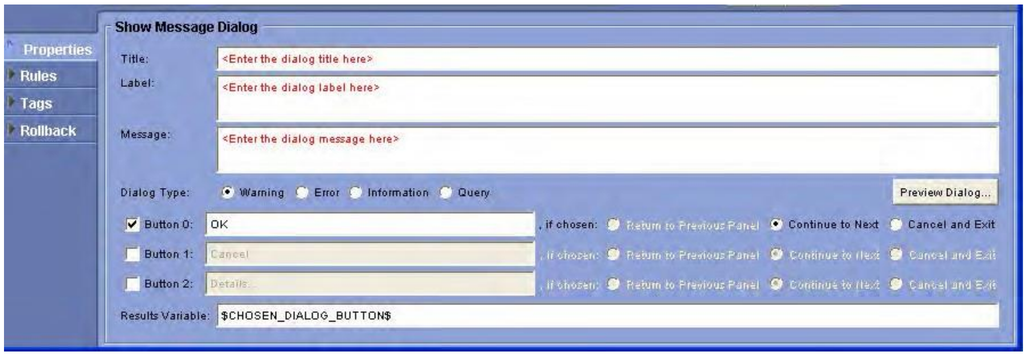
Property	Value
Comment	`Execute Script`

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

Suspend installation until process completes	Clear this selection.

8. Add another **Show Message Dialog** action by performing the following steps:

a. Click **Add Action** to open the **Actions** palette, select **Show Message Dialog** on the **General** tab and click **Add**. The second **Show Message Dialog** action is added to the **Visual Tree** and its customizer opens.



Show Message Dialog

Title: <Enter the dialog title here>

Label: <Enter the dialog label here>

Message: <Enter the dialog message here>

Dialog Type: ● Warning ○ Error ○ Information ○ Query

✓ Button 0: OK , if chosen: ○ Return to Previous Panel ● Continue to Next ○ Cancel and Exit

Button 1: Cancel , if chosen: ○ Return to Previous Panel ○ Continue to Next ○ Cancel and Exit

Button 2: Details , if chosen: ○ Return to Previous Panel ○ Continue to Next ○ Cancel and Exit

Results Variable: $CHOSEN_DIALOG_BUTTON$

b. In the **Show Message Dialog** customizer, enter the following:

Property	Value
Title	`Output`
Label	`Output Codes`

Property	Value

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

9. Rebuild and run your installer.

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

Creating Installer Logic Using Jump Labels and Actions

Advanced Note • *This is an advanced exercise.*

In this exercise, you will use the advanced features implemented so far in this section. You will create an installer that retrieves information from an end user, presents information to the end user for verification, and gives them the option to add more information.

Task ***To create an installer logic using Jump labels and actions:***

1. Create a new installer project.
2. To retrieve information from the end user, add an **Input** panel that contains at least three input types.
3. Modify the installer so that the following occurs:
   a. The installer will present the information to the end user for verification.
   b. If the user does not accept the information, they are returned to the input panel.
   c. The installer writes the collected information to a file.
   d. The installer offers the end user the option to enter more information.
   e. If the user chooses to enter more information, the installer returns to the initial input panel.
   f. The labels are modified so that the configuration process shares one label.

Modifying TrainApp.properties File at Runtime

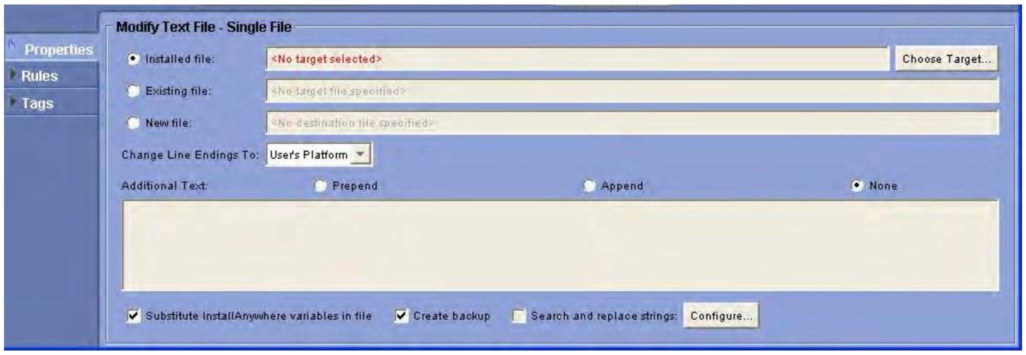
One of the files being installed with **TrainApp** is `TrainApp.properties`, which is a text file containing the following line:

```
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`.

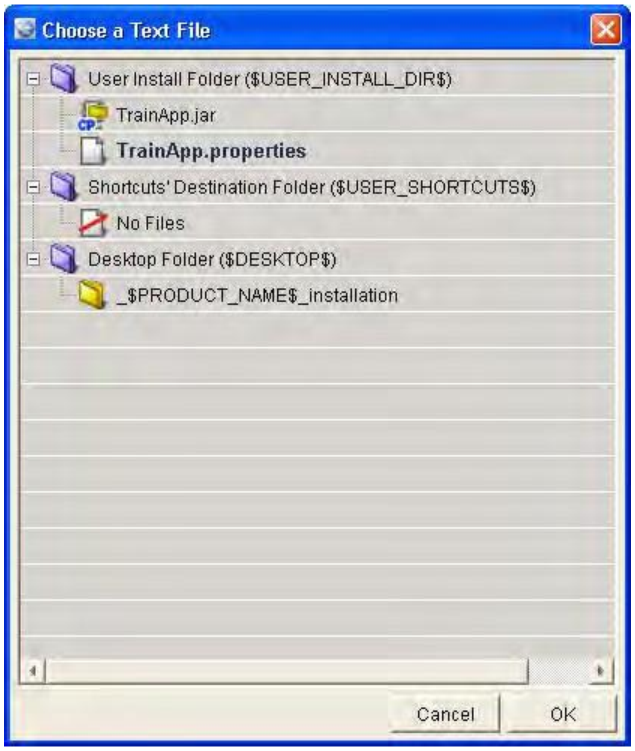
Task ***To modify the TrainApp.properties file at runtime:***

1. Open the `TrainApp` project in the Advanced Designer.
2. Open the **Post-Install** task.
3. Click the **Add Action** button to open the **Actions** palette, select **Modify Text File - Single File** on the **General** tab, and click **Add**. The **Modify Text File - Single File** action is added to the **Post-Install Action List** and its customizer opens.



- [✓ Installed file:] <No target selected>
- Existing file: <No target file specified>
- New file: <No destination file specified>
- Change Line Endings To: User's Platform
- Additional Text: Prepend / Append / [✓ None]
- [✓] Substitute InstallAnywhere variables in file
- [✓] Create backup
- [ ] Search and replace strings:

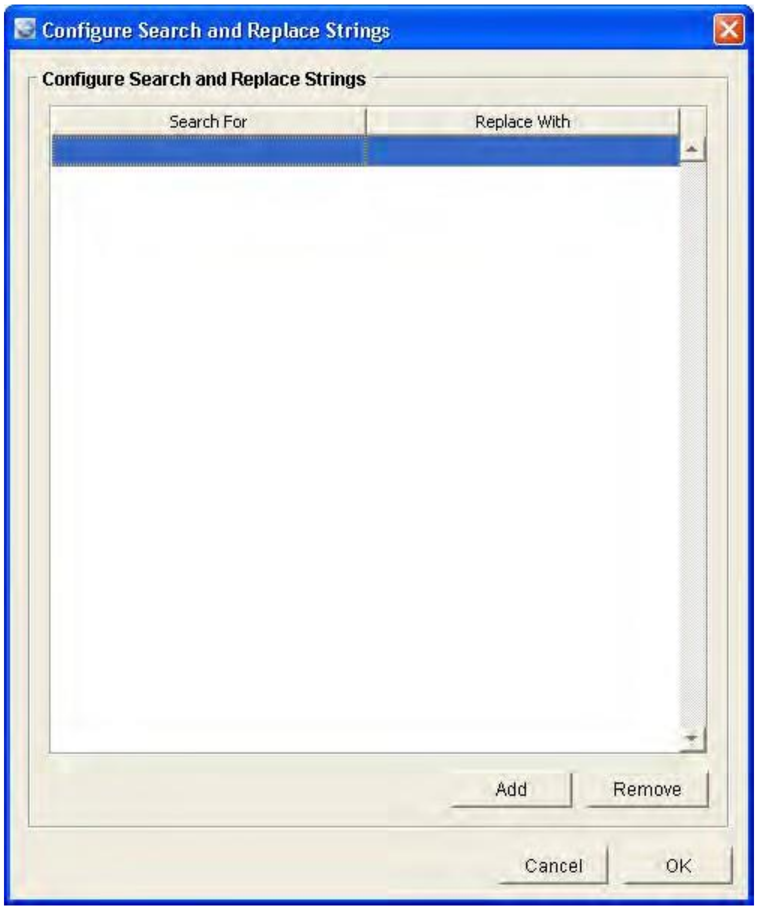
4. Select the **Installed file** option.
5. Click the **Choose Target** button. The **Choose a Text File** dialog box opens.



```
User Install Folder ($USER_INSTALL_DIR$)
    TrainApp.jar
    TrainApp.properties
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
    No Files
Desktop Folder ($DESKTOP$)
    _$PRODUCT_NAME$_installation
```

6. Select `TrainApp.properties` and click **OK**.
7. To search and replace contents of the file, select the **Search and replace strings** option at the bottom of the customizer and click **Configure**. The **Configure Search and Replace Strings** dialog box opens.

8. Click **Add** to add an entry to the list.


Configure Search and Replace Strings
Configure Search and Replace Strings
Search For
Replace With
Add
Remove
Cancel
OK

9. In the **Search For** field, enter **`%USERNAME%`**.

10. In the **Replace With** field, enter **`$prop.user.name$`**.

11. Click **OK**. InstallAnywhere will expand `$prop.user.name$` to the value of the Java system property `user.name`.

12. In the customizer, clear the **Create backup** option to prevent a copy of the original file called `TrainApp.properties.backup` from being created.

13. Build and run the installer.

14. Browse for the installation directory and open the `TrainApp.properties`. The file contents should now be similar to the following:

```
username=TrainingUser
```

15. Uninstall **TrainApp** after you have verified that the installer worked as expected.

Note • *A simpler technique would be to place* `$prop.user.name$` *directly in the* `.properties` *file, and ensure the **Substitute InstallAnywhere variables in file** option is selected on the **Modify Text File - Single File** customizer. Therefore, if you have time, modify the text file to use the property directly, and delete the "search and replace" instructions.*

Chapter 8 Exercises: Customizing the Uninstaller

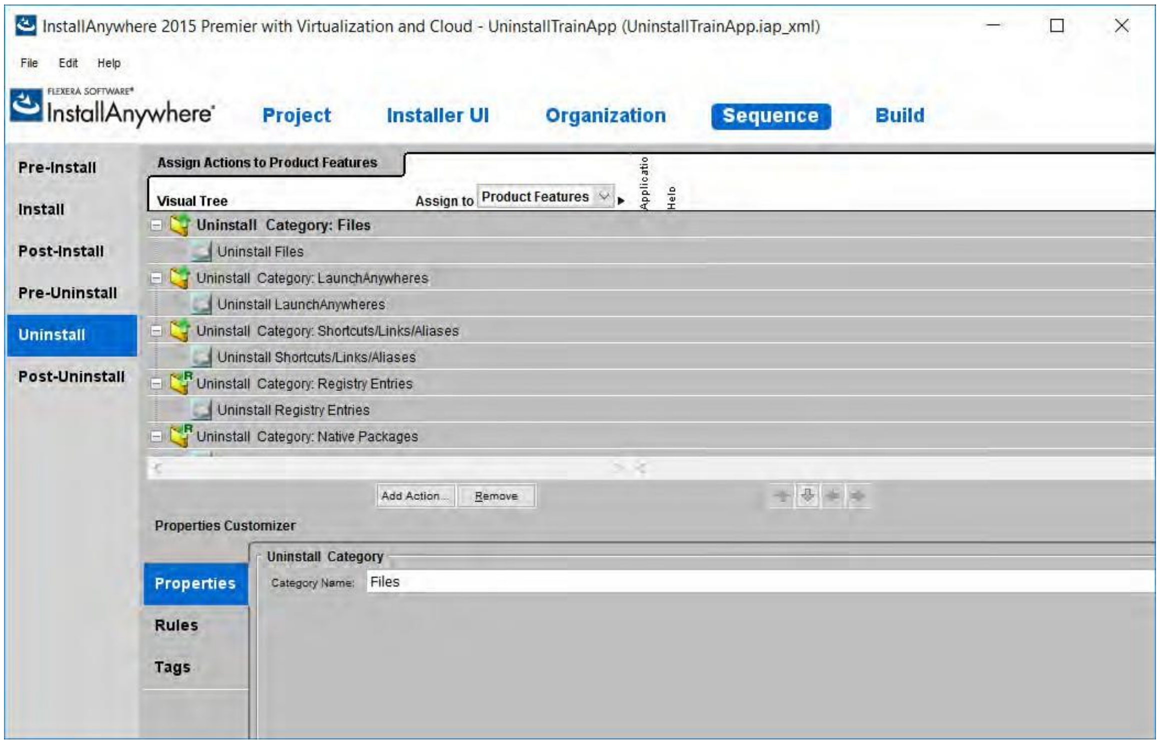
InstallAnywhere automatically creates an uninstaller for the project. The Uninstaller, much like the Installer, is a collection of panels, consoles, and actions. The standard Uninstaller uninstalls the application by executing each action's uninstallation procedure.

However, in some situations, you may want additional flexibility and have more control over how the uninstallation is performed. Therefore, you may want to use the **Uninstall** task in the Advanced Designer to customize the Uninstaller by adding, removing or changing some of the uninstall actions. For example, you may want to disable the uninstallation of an entire set of resources, rename files, copy and move files, display additional dialog messages, or execute some custom code at uninstall time.

In this exercise, you will customize the **Uninstall** task in a new project.

Task ***To customize the uninstaller:***

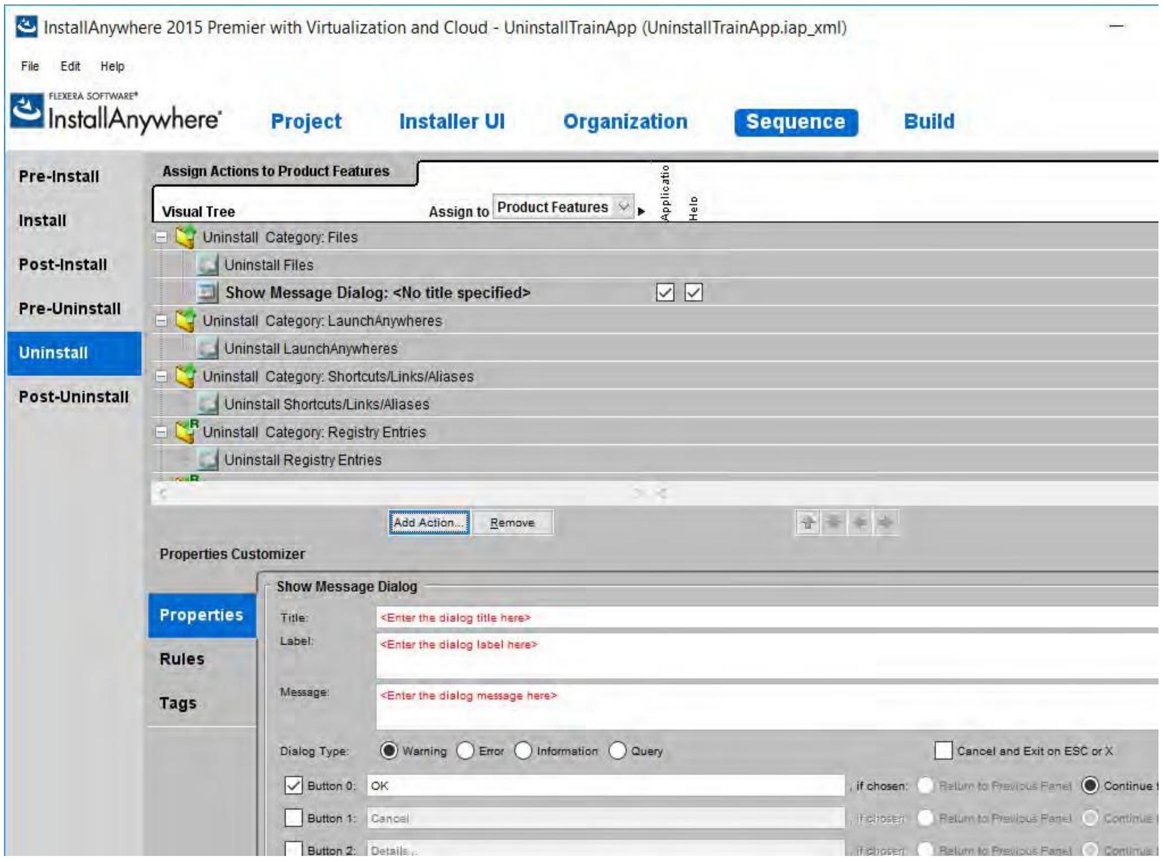
1. Open the Advanced Designer and create a new project named `UninstallTrainApp`.
2. Open the **Project > Uninstall** task.



Uninstall Category: Files
Uninstall Files
Uninstall Category: LaunchAnywheres
Uninstall LaunchAnywheres
Uninstall Category: Shortcuts/Links/Aliases
Uninstall Shortcuts/Links/Aliases
Uninstall Category: Registry Entries
Uninstall Registry Entries
Uninstall Category: Native Packages

Uninstall Category
Category Name: Files

3. Select **Uninstall Category: Files**.
4. Click **Add Action**, select **Show Message Dialog** on the **General** tab of the **Actions** palette, and click **Add**. The **Show Message Dialog** action is now listed under **Uninstall Category: Files**.


InstallAnywhere 2015 Premier with Virtualization and Cloud - UninstallTrainApp (UninstallTrainApp.iap_xml)
File Edit Help
InstallAnywhere
Project
Installer UI
Organization
Sequence
Build
Pre-Install
Install
Post-Install
Pre-Uninstall
Uninstall
Post-Uninstall
Assign Actions to Product Features
Visual Tree
Assign to Product Features
Uninstall Category: Files
Uninstall Files
Show Message Dialog: <No title specified>
Uninstall Category: LaunchAnywheres
Uninstall LaunchAnywheres
Uninstall Category: Shortcuts/Links/Aliases
Uninstall Shortcuts/Links/Aliases
Uninstall Category: Registry Entries
Uninstall Registry Entries
Add Action...
Remove
Properties Customizer
Show Message Dialog
Properties
Rules
Tags
Title: <Enter the dialog title here>
Label: <Enter the dialog label here>
Message: <Enter the dialog message here>
Dialog Type: Warning Error Information Query
Cancel and Exit on ESC or X
Button 0: OK , if chosen: Return to Previous Panel Continue
Button 1: Cancel
Button 2: Details...

5. In the **Show Message Dialog** customizer, enter the following information:

Property	Value
Title	`Confirmation`
Label	`Uninstallation Confirmation`
Message	`Are you sure that you want to uninstall this application?`

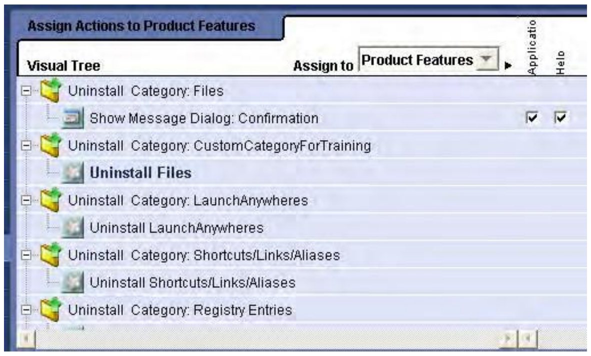
6. Click **Add Action**, select **Uninstall Category** on the **Uninstall** tab of the **Actions** palette, and click **Add**. A new, untitled **Uninstall Category** action is now listed in the **Visual Tree**.

7. In the **Uninstall Category** customizer, enter **`CustomCategoryForTraining`** in the **Category Name** field.

8. Using the arrow keys, move the **Uninstall Category: CustomCategoryForTraining** up so that it is the second category in the **Visual Tree**.

9. Under **Uninstall Category: Files**, delete the **Uninstall Files** action.

10. Add an **Uninstall Files** action to **Uninstall Category: CustomCategoryForTraining**. The Visual Tree should now look as follows:

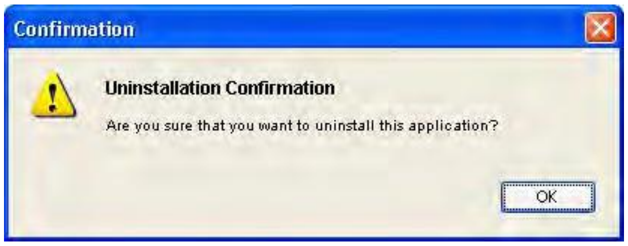


Uninstall Category: Files
Show Message Dialog: Confirmation
Uninstall Category: CustomCategoryForTraining
Uninstall Files
Uninstall Category: LaunchAnywheres
Uninstall LaunchAnywheres
Uninstall Category: Shortcuts/Links/Aliases
Uninstall Shortcuts/Links/Aliases
Uninstall Category: Registry Entries

11. Open the **Install** task and add the `TrainApp.jar` and `TrainApp.properties` files to the default installation location: **User Install Folder ($USER_INSTALL_DIR$)**.

12. Build and run the installer.

13. Launch the uninstaller by selecting **UninstallTrainApp > Change UninstallTrainApp Installation** on the Windows Start menu. The following message dialog box is displayed before the uninstaller will continue with the uninstallation:



Uninstallation Confirmation

Are you sure that you want to uninstall this application?

14. Press **OK** on the dialog box to proceed with uninstallation.

Chapter 9 Exercises: Implementing Maintenance Mode

You can choose to implement Maintenance Mode in an installer so that end users are able to add or remove features to previously installed products as well as repair broken installations.

InstallAnywhere's Maintenance Mode support also includes an Instance Management feature that enables you to specify whether multiple instances of a product can be installed on the same machine. If an installation includes both Maintenance Mode support and support for multiple instances of the product on the same machine, when an end user launches Maintenance Mode, they are able to specify which instance of the product they want to perform maintenance on.

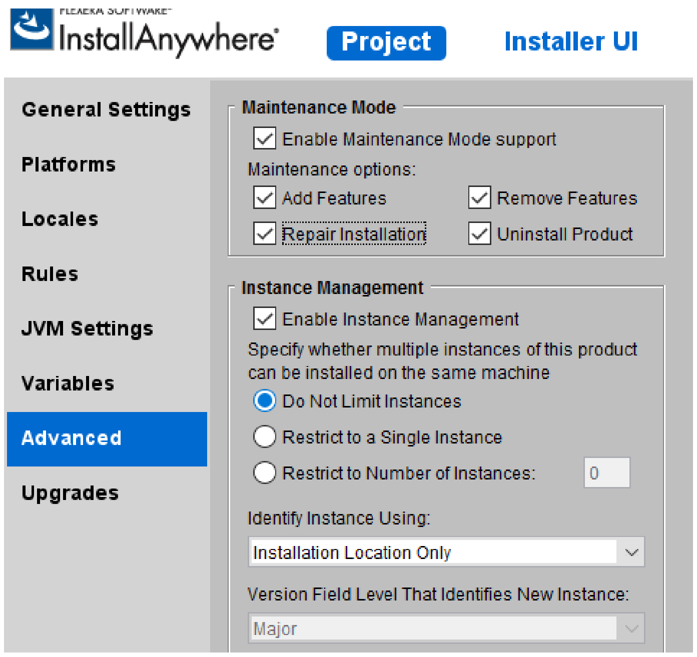
In this section, you will implement and test maintenance mode and set instance management options.

Enabling Maintenance Mode

In this exercise, you will enable and configure Maintenance Mode and then build and test the installer.

Task ***To enable Maintenance Mode:***

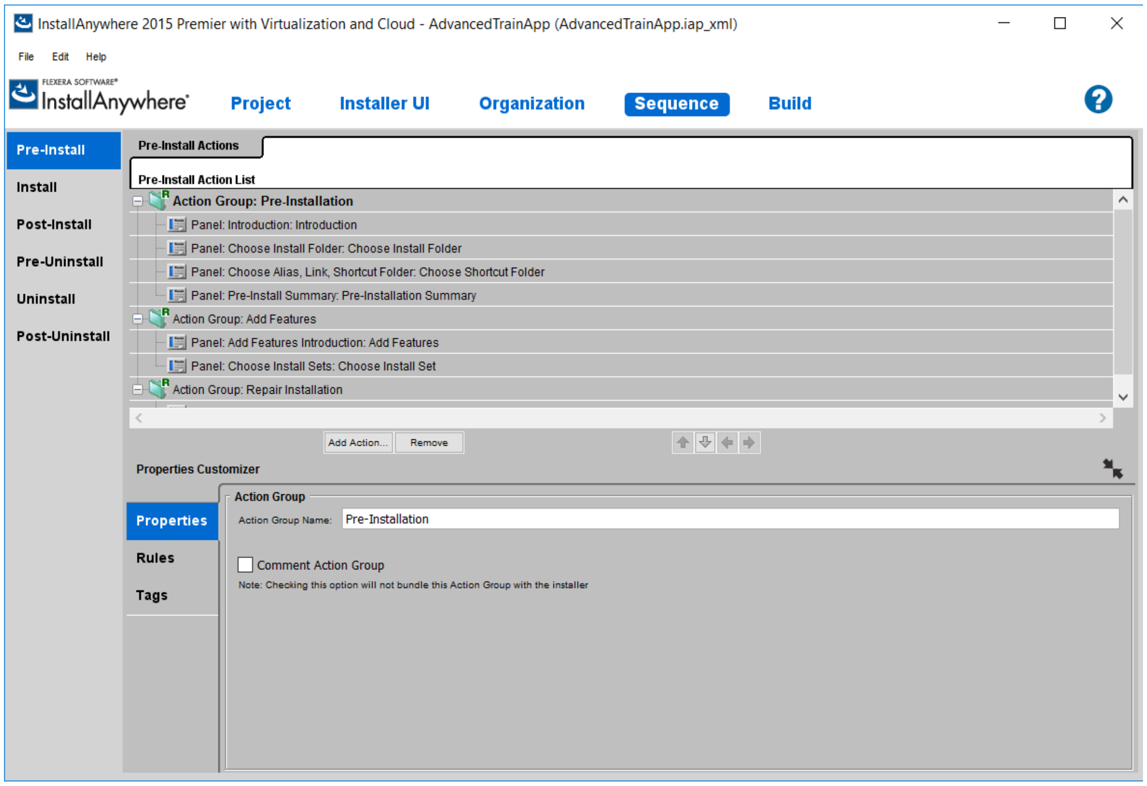
1. Open the Advanced Designer and create a new project named `AdvancedTrainApp`.
2. Open the **Project > Advanced** task.
3. In the **Maintenance Mode** area, select the **Enable Maintenance Mode support** option, and the **Add Features**, **Remove Features**, **Repair Installation**, and **Uninstall Product** options.



Maintenance Mode
[✓] Enable Maintenance Mode support
Maintenance options:
[✓] Add Features [✓] Remove Features
[✓] Repair Installation [✓] Uninstall Product

Instance Management
[✓] Enable Instance Management
Specify whether multiple instances of this product can be installed on the same machine
[● Do Not Limit Instances]
Restrict to a Single Instance
Restrict to Number of Instances: 0
Identify Instance Using: Installation Location Only
Version Field Level That Identifies New Instance: Major

4. Enabling the **Enable Maintenance Mode support** option will also enable the **Enable Instance Management** option. For this exercise, clear the **Enable Instance Management** option.
5. Open the **Pre-Install** task. Notice that all pre-existing actions in the project were added to an action group called **Pre-Installation**. Also notice that there are two other action groups listed: **Add Features** and **Repair Installation**.



Pre-Install Action List

- Action Group: Pre-Installation
  - Panel: Introduction: Introduction
  - Panel: Choose Install Folder: Choose Install Folder
  - Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
  - Panel: Pre-Install Summary: Pre-Installation Summary
- Action Group: Add Features
  - Panel: Add Features Introduction: Add Features
  - Panel: Choose Install Sets: Choose Install Set
- Action Group: Repair Installation

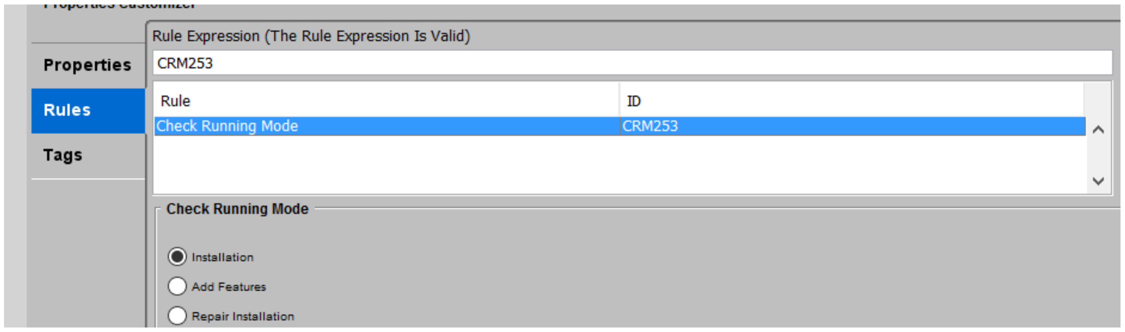
Action Group

Action Group Name: Pre-Installation

☐ Comment Action Group

Note: Checking this option will not bundle this Action Group with the installer

6. These three action groups (**Pre-Installation**, **Add Features**, and **Repair Installation**) are differentiated by a rule called **Check Running Mode** (CRM) rule.



Rule Expression (The Rule Expression Is Valid)

CRM253

Rule	ID
Check Running Mode	CRM253

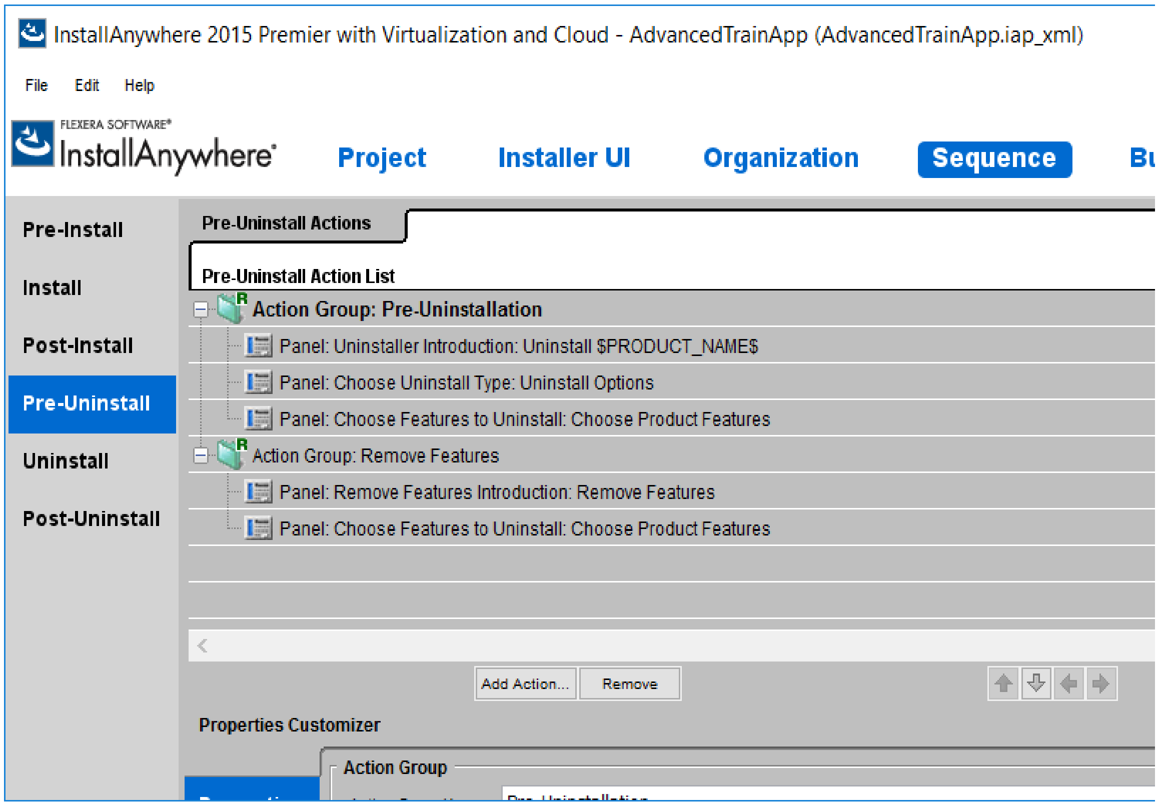
Check Running Mode

[✓ Installation]
Add Features
Repair Installation

All three action groups are automatically assigned a **Check Running Mode** rule:

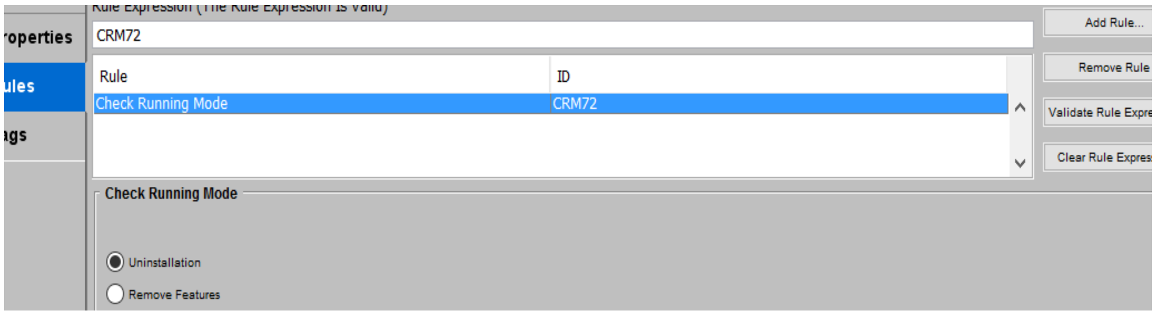
- The **Pre-Installation** action group has a CRM rule set to **Pre-Installation** mode.
- The **Add Features** action group has a CRM rule set to **Add Features**.
- The **Repair Installation** action group has a CRM rule set to **Repair Installation**.

7. Open the **Pre-Uninstall** task. Notice that there are two additional action groups listed: **Pre-Uninstallation** and **Remove Features**.



Pre-Uninstall Action List
- Action Group: Pre-Uninstallation
  - Panel: Uninstaller Introduction: Uninstall $PRODUCT_NAME$
  - Panel: Choose Uninstall Type: Uninstall Options
  - Panel: Choose Features to Uninstall: Choose Product Features
- Action Group: Remove Features
  - Panel: Remove Features Introduction: Remove Features
  - Panel: Choose Features to Uninstall: Choose Product Features

8. These action groups are also automatically assigned a **Check Running Mode** rule:



CRM72

Rule	ID
Check Running Mode	CRM72

Check Running Mode

[✓ Uninstallation] Remove Features

- The **Pre-Uninstallation** action group has a CRM rule set to **Uninstallation** mode.
- The **Add Features** action group has a CRM rule set to **Remove Features**.

9. Open the **Pre-Install** task.

10. Add a **Show Message Dialog** to the end of the **Add Features** action group and enter the following information in its customizer:

Property	Description
Title	Add 1
Label	Add 1

Property	Description
Message	`Do you want to add a feature?`

11. Add another **Show Message Dialog** action to the end of the **Repair Installation** action group and enter the following information in its customizer:

Property	Description
Title	`Repair 1`
Label	`Repair 1`
Message	`Do you want to remove a feature?`

12. Add another **Show Message Dialog** action to the very end of the **Pre-Install Action List** (outside of the action groups):

    a. Enter the following information in its customizer:

Property	Description
Title	`Add 2`
Label	`Add 2`
Message	`Do you want to add a feature?`

    b. To the **Show Message Dialog: Add 2** action, assign a **Check Running Mode** rule that is set to **Add Features**.

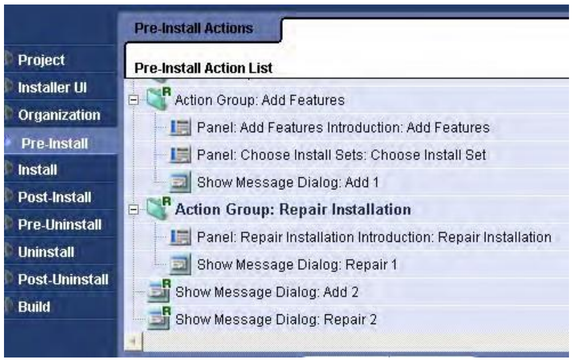
13. Add another **Show Message Dialog** action to the very end of the **Pre-Install Action List** (outside of the action groups and below the **Add 2** action):

    a. Enter the following information in its customizer:

Property	Description
Title	`Repair 2`
Label	`Repair 2`
Message	`Do you want to repair a feature?`

    b. To the **Show Message Dialog: Repair 2** action, assign a **Check Running Mode** rule that is set to **Repair Installation**.

The **Pre-Install Action List** should now look like this:



Pre-Install Action List
- Action Group: Add Features
  - Panel: Add Features Introduction: Add Features
  - Panel: Choose Install Sets: Choose Install Set
  - Show Message Dialog: Add 1
- Action Group: Repair Installation
  - Panel: Repair Installation Introduction: Repair Installation
  - Show Message Dialog: Repair 1
- Show Message Dialog: Add 2
- Show Message Dialog: Repair 2

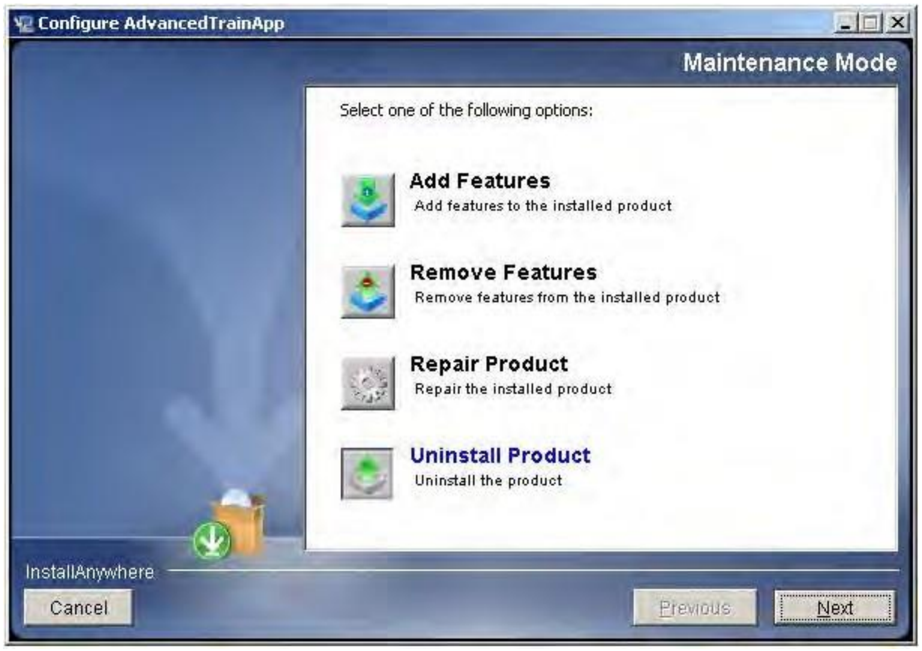
14. Open the **Pre-Uninstall** task.

15. Add a **Show Message Dialog** action to the end of the **Remove Features** action group and enter the following information in its customizer:

Property	Description
Title	`Remove Feature 1`
Label	`Remove Feature 1`
Message	`Do you want to remove a feature?`

16. Build and install the project.

17. Launch the Maintenance Mode launcher by selecting **AdvancedTrainApp > Change AdvancedTrainApp Installation** on the Windows Start menu or by running the `Change AdvancedTrainApp Installation.exe` file in the application installation directory.



Select one of the following options:

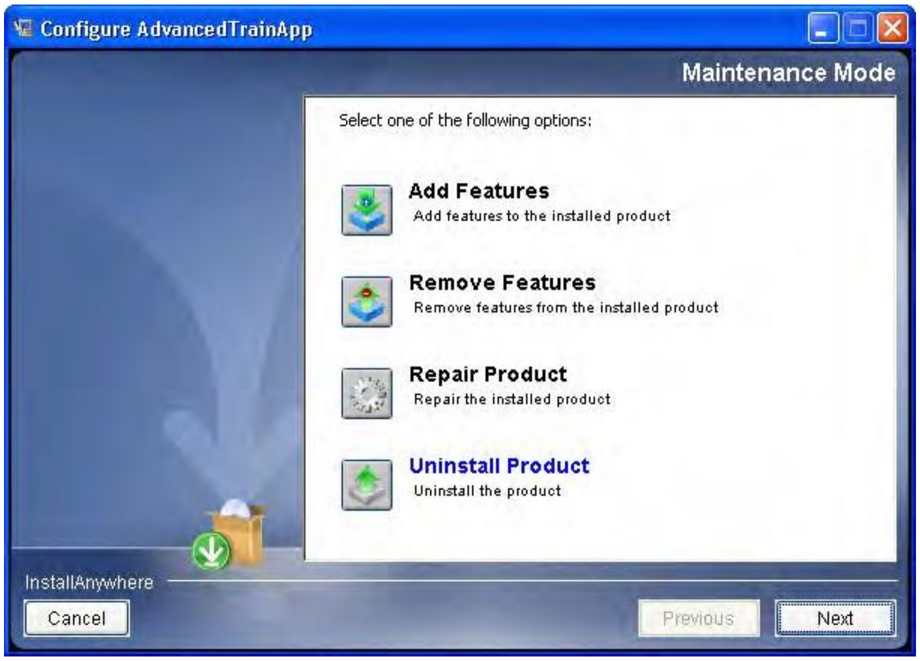
Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

The **Maintenance Mode** panel opens and displays four options: **Add Features**, **Remove Features**, **Repair Product**, and **Uninstall Product**.



Select one of the following options:

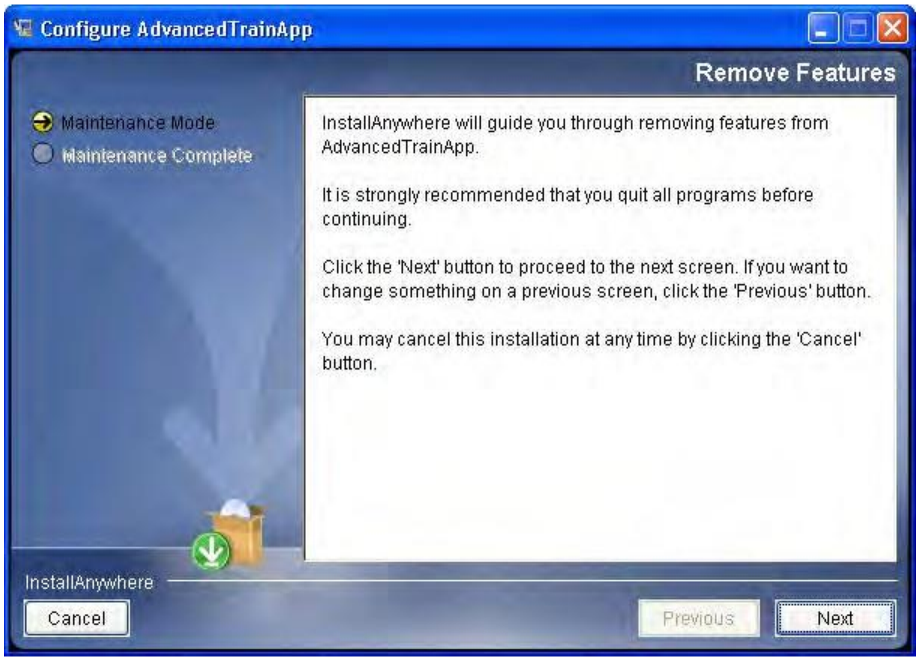
Add Features
Add features to the installed product

Remove Features
Remove features from the installed product

Repair Product
Repair the installed product

Uninstall Product
Uninstall the product

18. Select **Remove Features** and click **Next**. The **Remove Features** introduction panel opens.



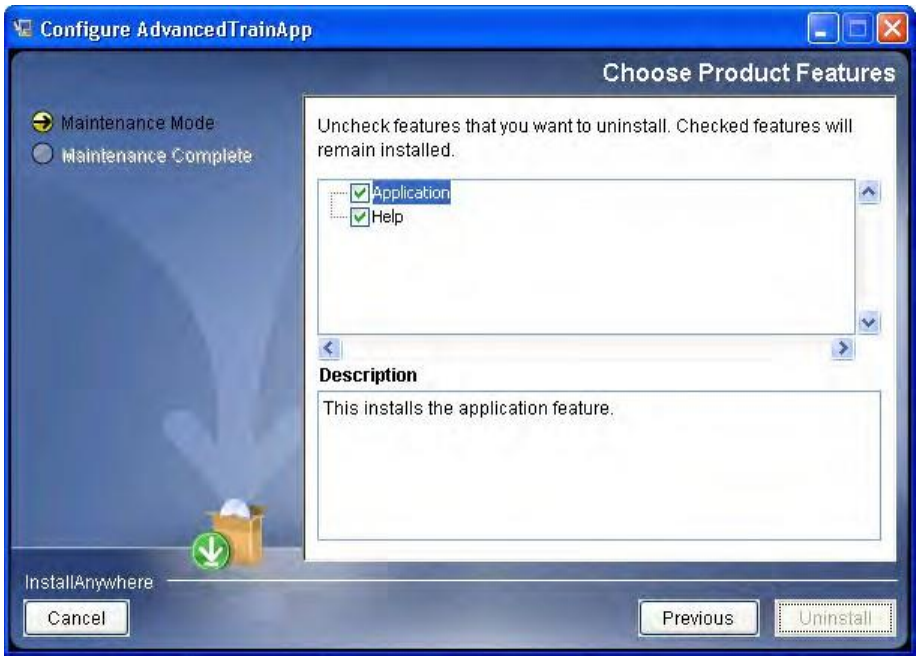
InstallAnywhere will guide you through removing features from AdvancedTrainApp.

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

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

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

19. Click **Next**. The **Choose Product Features** panel opens.



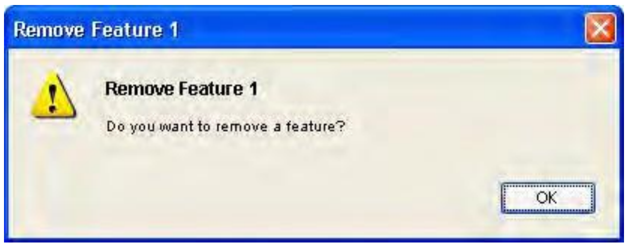
Uncheck features that you want to uninstall. Checked features will remain installed.

- [x] Application
- [x] Help

Description

This installs the application feature.

20. Clear the check box next to **Help** and click **Uninstall** to proceed. You are prompted to confirm the removal of the feature:



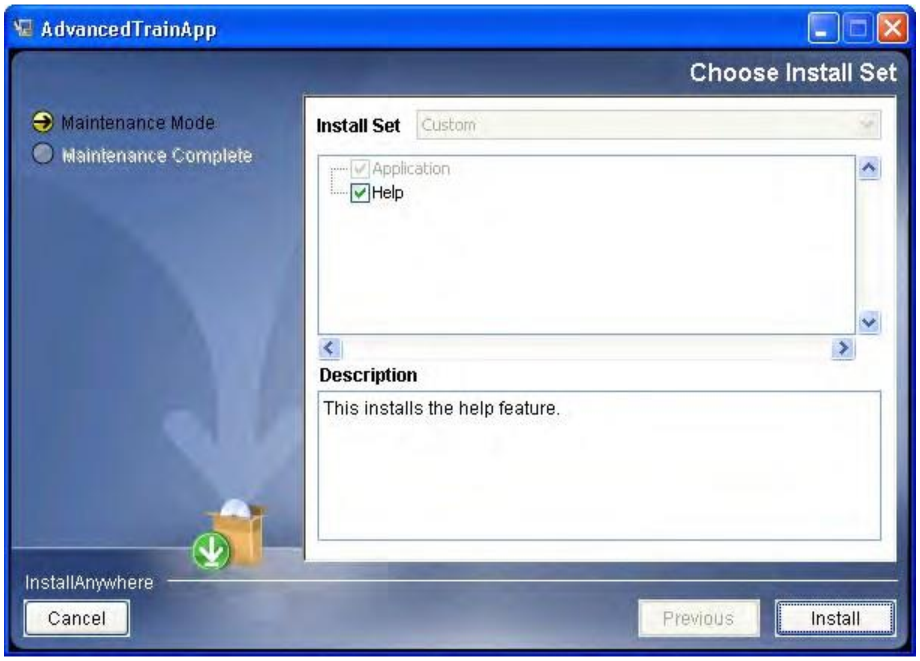
Remove Feature 1

Do you want to remove a feature?

21. Click **OK**. The **Help** feature will be removed.

22. Re-launch the Maintenance Mode launcher, and this time choose **Add Features** on the **Maintenance Mode** panel. The **Add Features** introduction panel opens.

23. Click **Next**. The **Choose Install Set** panel opens and the feature that is not currently installed is automatically selected.



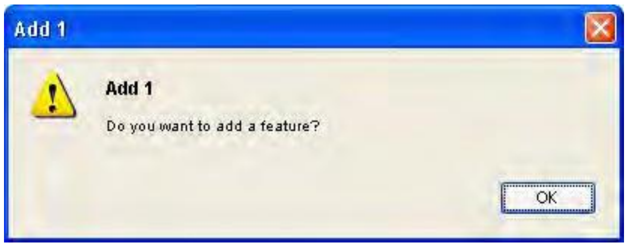
Install Set: Custom

- [✓] Application
- [✓] Help

Description

This installs the help feature.

24. Click **Install**. You are prompted to confirm that you want to add a feature.

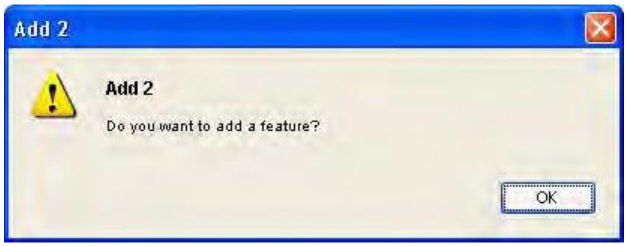


Add 1

Add 1

Do you want to add a feature?

25. Click **OK**. You are prompted a second time to confirm that you want to add a feature.



Add 2

Add 2

Do you want to add a feature?

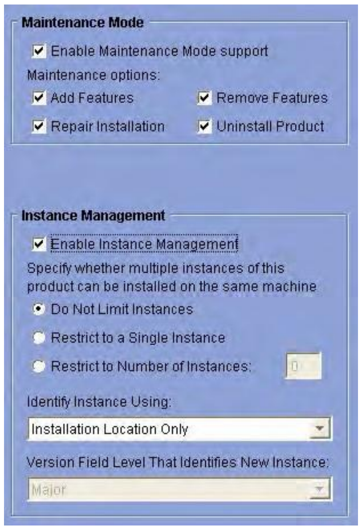
26. Click **OK**. The feature is added.
27. Re-launch the Maintenance Mode launcher, and this time choose **Uninstall Product** on the **Maintenance Mode** panel. The **Uninstall** introduction panel opens.
28. Click **Next**. The product is uninstalled.
29. Proceed with the steps in Setting Instance Management Options.

Setting Instance Management Options

InstallAnywhere's Maintenance Mode support also includes an Instance Management feature that enables you to specify whether multiple instances of a product can be installed on the same machine. If an installation includes both Maintenance Mode support and support for multiple instances of the product on the same machine, when an end user launches Maintenance Mode, they are able to specify which instance of the product they want to perform maintenance on.

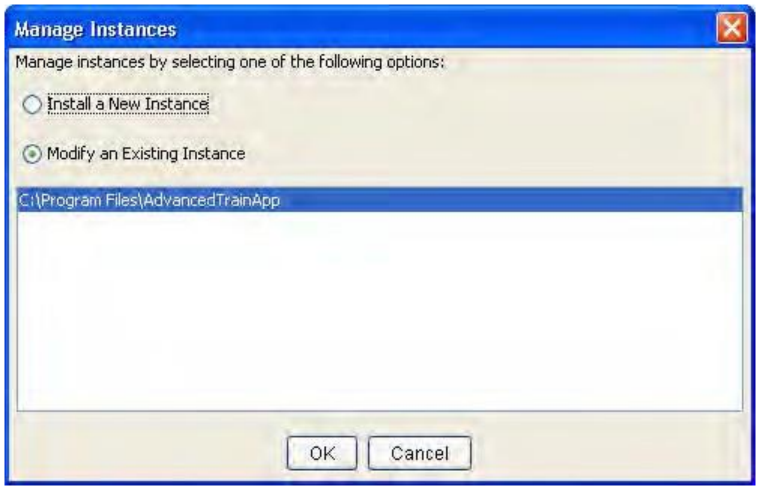
Task ***To set instance management options:***

1. Perform the steps in Enabling Maintenance Mode.
2. Open the **Project > Advanced** task.
3. Under **Instance Management**, select the **Enable Instance Management** option.

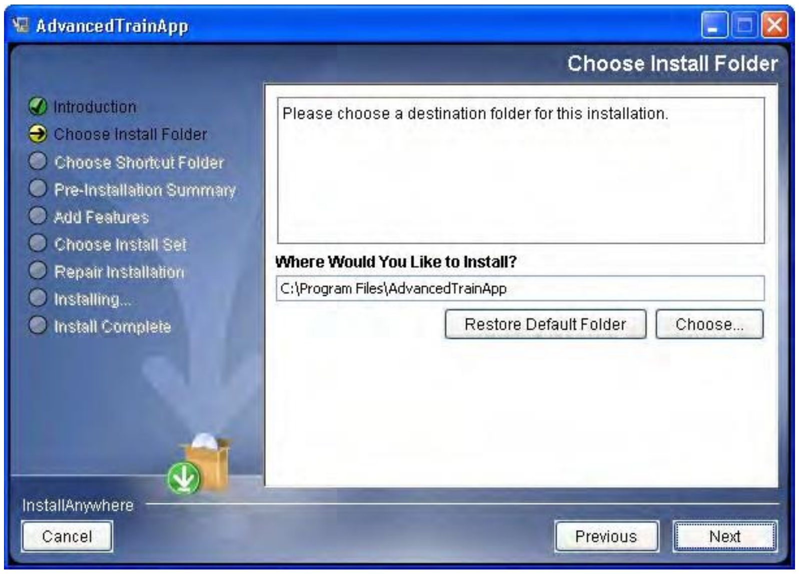

Maintenance Mode
☑ Enable Maintenance Mode support
Maintenance options:
☑ Add Features ☑ Remove Features
☑ Repair Installation ☑ Uninstall Product

Instance Management
☑ Enable Instance Management
Specify whether multiple instances of this product can be installed on the same machine
[✓ Do Not Limit Instances]
Restrict to a Single Instance
Restrict to Number of Instances: 0
Identify Instance Using:
Installation Location Only
Version Field Level That Identifies New Instance:
Major

4. Select the **Restrict to Number of Instances** option and enter **2** in the text box.
5. Build and install the product.
6. Try to install the product a second time. The **Manage Instances** dialog box opens.


Manage Instances
Manage instances by selecting one of the following options:
Install a New Instance
[✓ Modify an Existing Instance]
C:\Program Files\AdvancedTrainApp

7. Select the **Install a New Instance** option and click **OK** to proceed with the installation. The **Choose Install Folder** panel opens:



Please choose a destination folder for this installation.

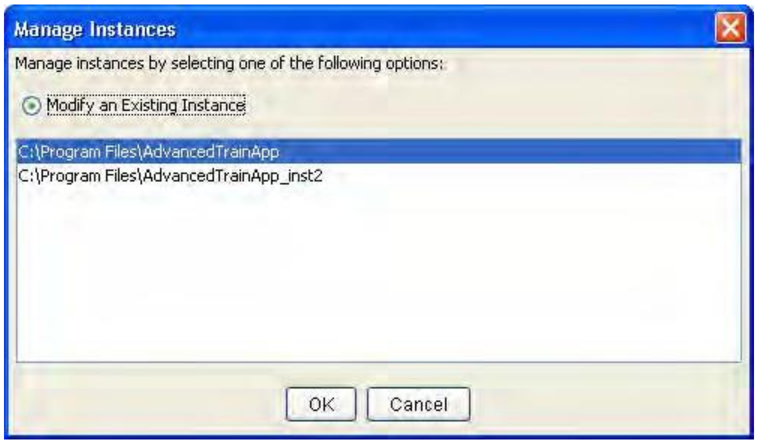
Where Would You Like to Install?

C:\Program Files\AdvancedTrainApp

8. In the Where would You Like to Install? field, enter `C:\Program Files\AdvancedTrainApp_inst2` and click **Next**.

9. Complete the installation.

10. Try to install the product a third time. The M**anage Instances** dialog box opens. Because it recognizes that there are already a maximum of 2 instances, only the **Modify an Existing Instance** option is available.



Manage instances by selecting one of the following options:

Modify an Existing Instance

C:\Program Files\AdvancedTrainApp
C:\Program Files\AdvancedTrainApp_inst2

11. Select one of the instances and click **OK** to proceed.

12. On the **Maintenance Mode** panel, choose **Uninstall Product** and proceed with uninstallation.

Chapter 11 Exercises: Source and Resource Management

These exercises cover how to effectively manage the availability of source files and resources when developing in a team environment.

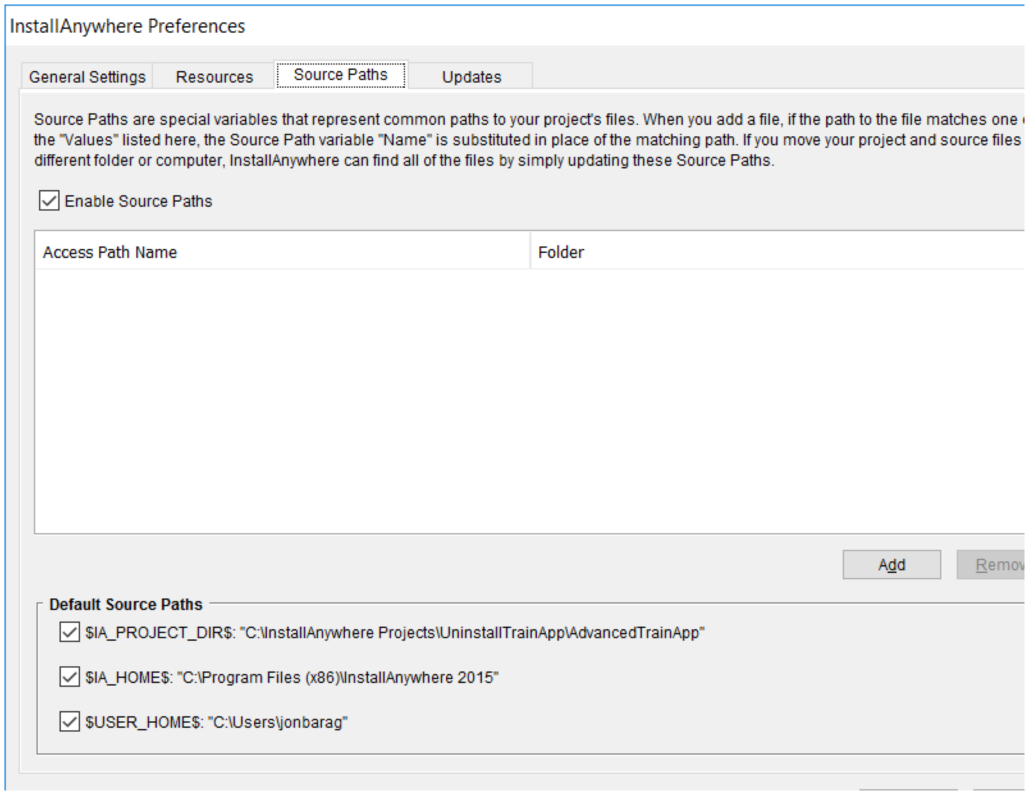
- Creating Source Paths
- Managing Resources in the InstallAnywhere Project File

Creating Source Paths

In this exercise, you will enable a default source path and add a user-defined source path.

Task ***To create source paths and enable a default source path:***

1. Open the `My_Product.iap_xml` project in the Advanced Designer.
2. Select **Preferences** on the **Edit** menu.
3. Open the **Source Paths** tab.



Source Paths are special variables that represent common paths to your project's files. When you add a file, if the path to the file matches one
the "Values" listed here, the Source Path variable "Name" is substituted in place of the matching path. If you move your project and source files
different folder or computer, InstallAnywhere can find all of the files by simply updating these Source Paths.

☑ Enable Source Paths

Default Source Paths

☑ $IA_PROJECT_DIR$: "C:\InstallAnywhere Projects\UninstallTrainApp\AdvancedTrainApp"

☑ IA_HOME: "C:\Program Files (x86)\InstallAnywhere 2015"

☑ $USER_HOME$: "C:\Users\jonbarag"

4. Under **Default Source Paths**, make sure that the `$IA_PROJECT_DIR$` option is selected.
5. Click **Add**. An entry is added to the **Source Paths** list.

6. Enter the following information:

Property	Value
Access Path Name	`OFFICE_SOURCE` ***Note*** • *Do not type dollar signs ($) around source paths when you add them into **Preferences** dialog box.*
Folder	Browse to the location of the OfficeSuite Source files, such as: `[InstallAnywhere Installation Location]\OfficeSuiteSourceFiles`

7. Click **OK** to close the **Preferences** dialog box.

Managing Resources in the InstallAnywhere Project File

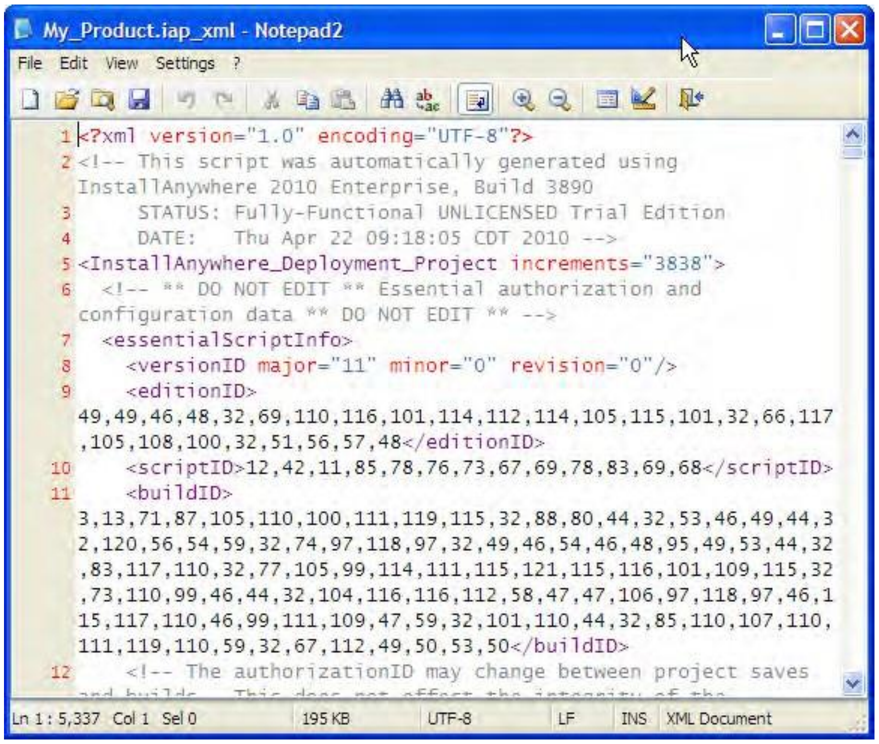
Advanced Note • *This is an advanced exercise.*

While not recommended as a best practice, it is possible to manage resources directly from the project file itself.

Because an InstallAnywhere project file (`.xml_iap`)is essentially a plain text file, it is possible to manipulate it directly. You can edit it using simple search and replace techniques, or using more advanced functions such as `sed` or awk scripts, or even an Extensible Style Language Transform.

Task ***To manage resources in an InstallAnywhere project file:***

1. Open an InstallAnywhere project file (.xml_iap) in a text editor.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This script was automatically generated using
InstallAnywhere 2010 Enterprise, Build 3890
     STATUS: Fully-Functional UNLICENSED Trial Edition
     DATE:   Thu Apr 22 09:18:05 CDT 2010 -->
<InstallAnywhere_Deployment_Project increments="3838">
  <!-- ** DO NOT EDIT ** Essential authorization and
configuration data ** DO NOT EDIT ** -->
  <essentialScriptInfo>
    <versionID major="11" minor="0" revision="0"/>
    <editionID>
49,49,46,48,32,69,110,116,101,114,112,114,105,115,101,32,66,117
,105,108,100,32,51,56,57,48</editionID>
    <scriptID>12,42,11,85,78,76,73,67,69,78,83,69,68</scriptID>
    <buildID>
3,13,71,87,105,110,100,111,119,115,32,88,80,44,32,53,46,49,44,3
2,120,56,54,59,32,74,97,118,97,32,49,46,54,46,48,95,49,53,44,32
,83,117,110,32,77,105,99,114,111,115,121,115,116,101,109,115,32
,73,110,99,46,44,32,104,116,116,112,58,47,47,106,97,118,97,46,1
15,117,110,46,99,111,109,47,59,32,101,110,44,32,85,110,107,110,
111,119,110,59,32,67,112,49,50,53,50</buildID>
    <!-- The authorization ID may change between project saves
```

2. Edit resource paths.

Chapter 12 Exercises: Advanced Installer Concepts

These exercises demonstrate additional installation-related concepts such as console-mode (for command-line-driven installations), silent installations (requiring limited or no user interaction), customized uninstallers (to control the removal of products and features), and setting installation rollback options.

- Building a Console-Enabled Installer Using OfficeSuite
- Building a Silent-Mode Installer
- Installing TrainApp in Silent Mode
- Hiding TrainApp from Add or Remove Programs
- Setting Installation Rollback Options

Building a Console-Enabled Installer Using OfficeSuite

Advanced Note • *This is an advanced exercise.*

In this exercise you will build a console-enabled version of the OfficeSuite installer.

Task ***To build a console-enabled installer:***

1. Open the InstallAnywhere Advanced Designer and create a new project named `OfficeSuiteConsole`.
2. Set up the project similar to the previous OfficeSuite installer that you created in earlier exercises:
    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.
3. 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.
4. 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.*
5. Browse to the **Post-Install Section** and insert the following consoles.
    - **Console: Install Complete**
    - **Console: Install Failed**
6. 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 `$INSTALL_SUCCESS$`.
7. Rebuild your installer, choosing Linux as one of your target platforms. The instructor will provide you with address log on information to a Linux system where you will test the installer.
8. 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

Advanced Note • *This is an advanced exercise.*

In this exercise, you will build a silent mode installer.

Task ***To build 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 to determine if the installer installed in the location that you expected.

Note • *For more information on silent installers, refer to **Silent and Console Installers** in the InstallAnywhere Help Library.*

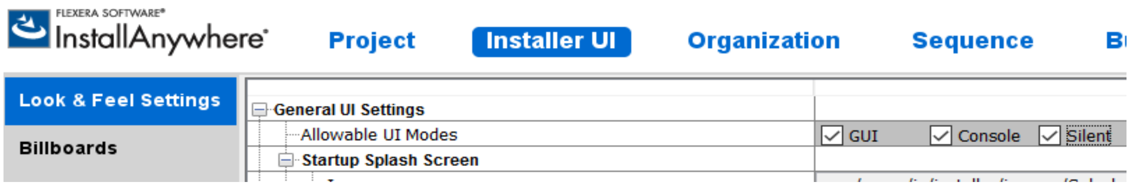
Installing TrainApp in Silent Mode

Advanced Note • *This is an advanced exercise.*

In this exercise, you will install the TrainApp application in silent mode.

Task ***To install TrainApp in silent mode:***

1. Open the `TrainApp` project in the Advanced Designer.
2. Open the **Installer UI > Look & Feel** task and select **Silent** is in the **Allowable UI Modes** setting.



Allowable UI Modes: ✓ GUI ✓ Console ✓ Silent

3. Rebuild the project using the **Build** task.
4. Open the build location by clicking **Open in Explorer**.
5. 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`.

6. When the installation has finished, open `installer.properties`. Its contents should be similar to the following:

```
# Fri Oct 31 25:00:00 CDT 2009
# 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
```

7. Next, uninstall **TrainApp**.
8. Then, perform the silent installation using the following command:

```
install -i silent -r installer.properties
```

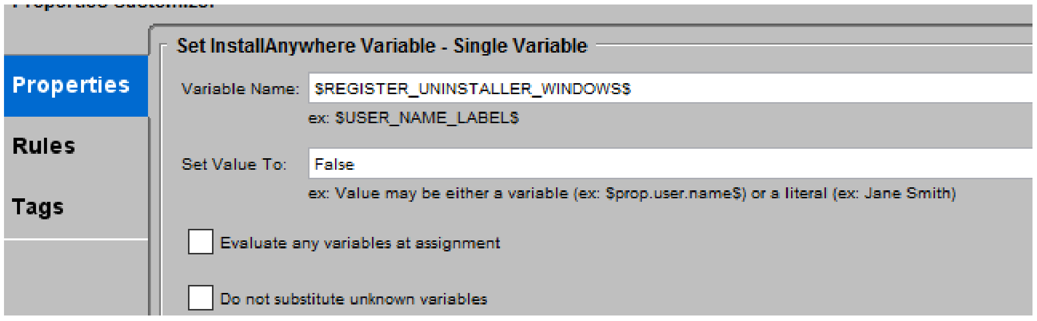
9. When the silent installation has finished, browse to the installation directory specified in the response file to verify that the product has been installed to the correct location.

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, you can prevent an application from being listed in the **Add or Remove Programs** panel on Windows.

Task ***To hide TrainApp from Add or Remove Programs panel:***

1. Open the `TrainApp` project in the Advanced Designer.
2. Open the **Pre-Install** task.
3. Click **Add Action**, select **Set InstallAnywhere Variable - Single Variable** action on the **General** tab of the **Actions** palette, and click **Add**. The **Set InstallAnywhere Variable - Single Variable** is added to the **Pre-Install Action List** and its customizer is displayed.



Properties
Rules
Tags

Set InstallAnywhere Variable - Single Variable

Variable Name: $REGISTER_UNINSTALLER_WINDOWS$
ex: $USER_NAME_LABEL$

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

☐ Evaluate any variables at assignment

☐ Do not substitute unknown variables

4. In the customizer, enter the following information:

Property	Value
Variable Name	`$REGISTER_UNINSTALLER_WINDOWS$`
Set Value To	`false`

5. Rebuild the project and run the installer. **TrainApp** will not be listed in **Add or Remove Programs**.

Note • *The uninstaller is still being created; to uninstall* **TrainApp**, *browse for the installation directory, open its* `_TrainApp_installation` *directory, and launch the uninstaller executable,* `Change TrainApp Installation.exe`.

6. 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`.

Note • *If you have other platforms available, you can specify integration with native registries on Linux and AIX using the* ***Project > Platforms > UNIX*** *task.*

Setting Installation Rollback Options

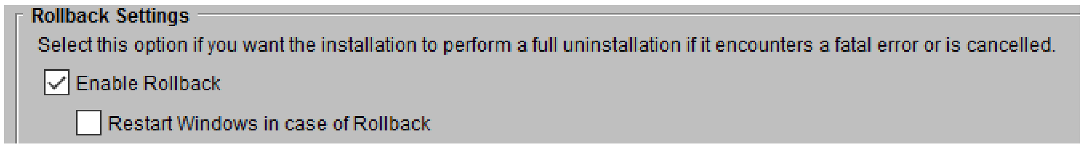
If an end user cancels an installation before it has completed, or if a fatal error occurs during the installation, the result can be an incomplete, corrupt application. To avoid this problem, you can enable the installation rollback option, so that when an end user cancels an installation or a fatal error occurs, the installer will automatically revert what has been altered or added to the system.

Also, on the **Rollback** subtab of action customizers in the **Install** task, you can specify rollback options on a file-by-file basis. You can fine tune how the installer treats individual project elements during a rollback and can specify which project elements would trigger a rollback if the installation of that element fails.

This exercise will demonstrate how to set installation rollback options.

Task ***To set installation rollback options:***

1. Open the InstallAnywhere Advanced Designer and create a new project named `RollbackTrainApp`.
2. Open the **Project > Advanced** task and verify that the **Enable Rollback** option is selected.


Rollback Settings
Select this option if you want the installation to perform a full uninstallation if it encounters a fatal error or is cancelled.
☑ Enable Rollback
☐ Restart Windows in case of Rollback

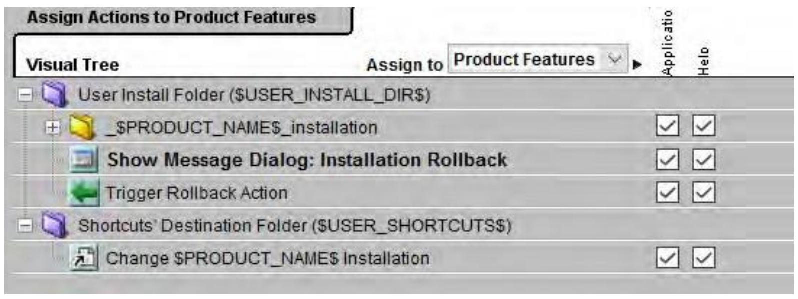
Note • *By default, the* ***Enable Rollback*** *option is selected for all new projects.*

3. Open the **Install** task.
4. Add a **Show Message Dialog** action and set the following values:

Property	Value
Title	`Installation Rollback`
Label	`Installation is rolling back...`
Message	`This installation has encountered a fatal error or was cancelled. The installation is now going to roll back.`

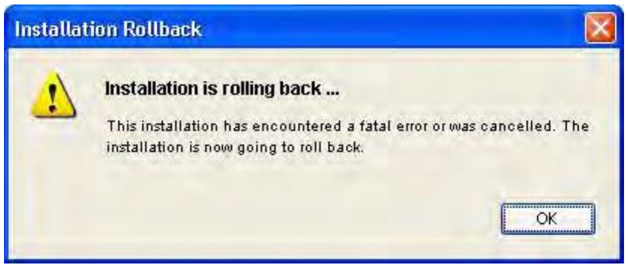
Note • *The purpose of this Show Message Dialog is to pause the installer just before the rollback occurs.*

5. Now, directly after the **Show Message Dialog** that you just added, add a **Trigger Rollback Action** to the **Install** task (which is on the **Install** tab of the **Actions** palette).


Assign Actions to Product Features
Visual Tree Assign to Product Features Applicatio Help
User Install Folder ($USER_INSTALL_DIR$)
_$PRODUCT_NAME$_installation ☑ ☑
Show Message Dialog: Installation Rollback ☑ ☑
Trigger Rollback Action ☑ ☑
Shortcuts' Destination Folder ($USER_SHORTCUTS$)
Change $PRODUCT_NAME$ Installation ☑ ☑

Note • *Other ways to force an installation to roll back would be to throw an instance of* `FatalInstallException` *in your Custom Code in the **Install** phase, or cancel the installation while the **Install** phase is in progress.*

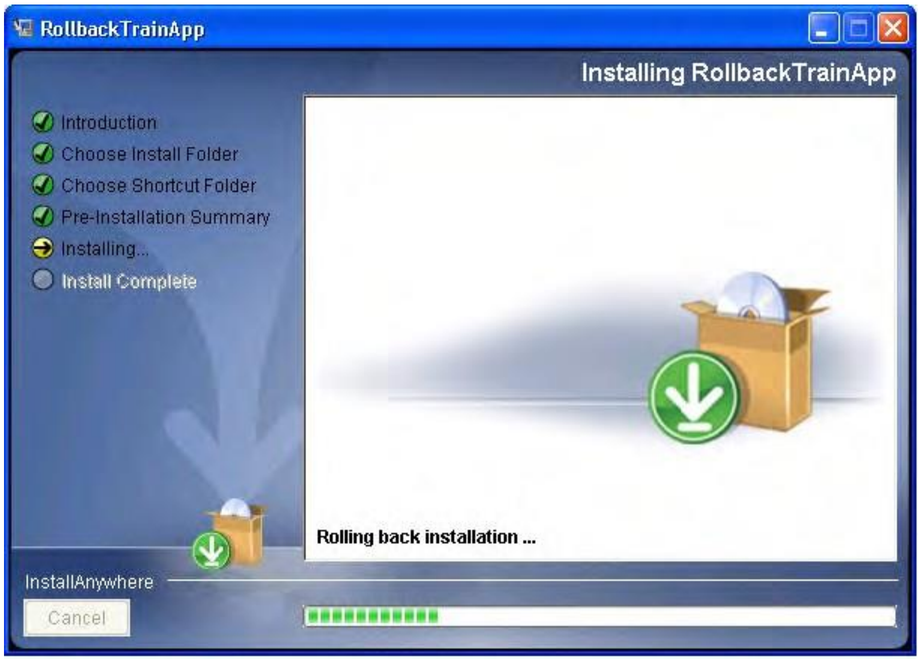
6. Build the project.
7. Click the **Try Installer** button to run the installation. When installation begins, the following message dialog box opens:


Installation Rollback

Installation is rolling back ...

This installation has encountered a fatal error or was cancelled. The installation is now going to roll back.

8. Click **OK**. Notice that the progress bar on the installer immediately starts to go in the reverse direction and the installation is rolled back.



Rolling back installation ...

The **Install Complete** panel is then displayed and states that:

```
The installation of RollbackTrainApp has been rolled back.
```

Chapter 13 Exercises: Creating and Editing Build Configurations

Each InstallAnywhere project can have multiple Build Configurations, each representing how the installer will be built for particular set of platforms, files, build distributions, JVMs, locales, and other settings. You can create and modify Build Configurations on the **Build Configurations** tab of the Advanced Designer's **Build** task.

In these exercises, you will create a new Build Configuration and

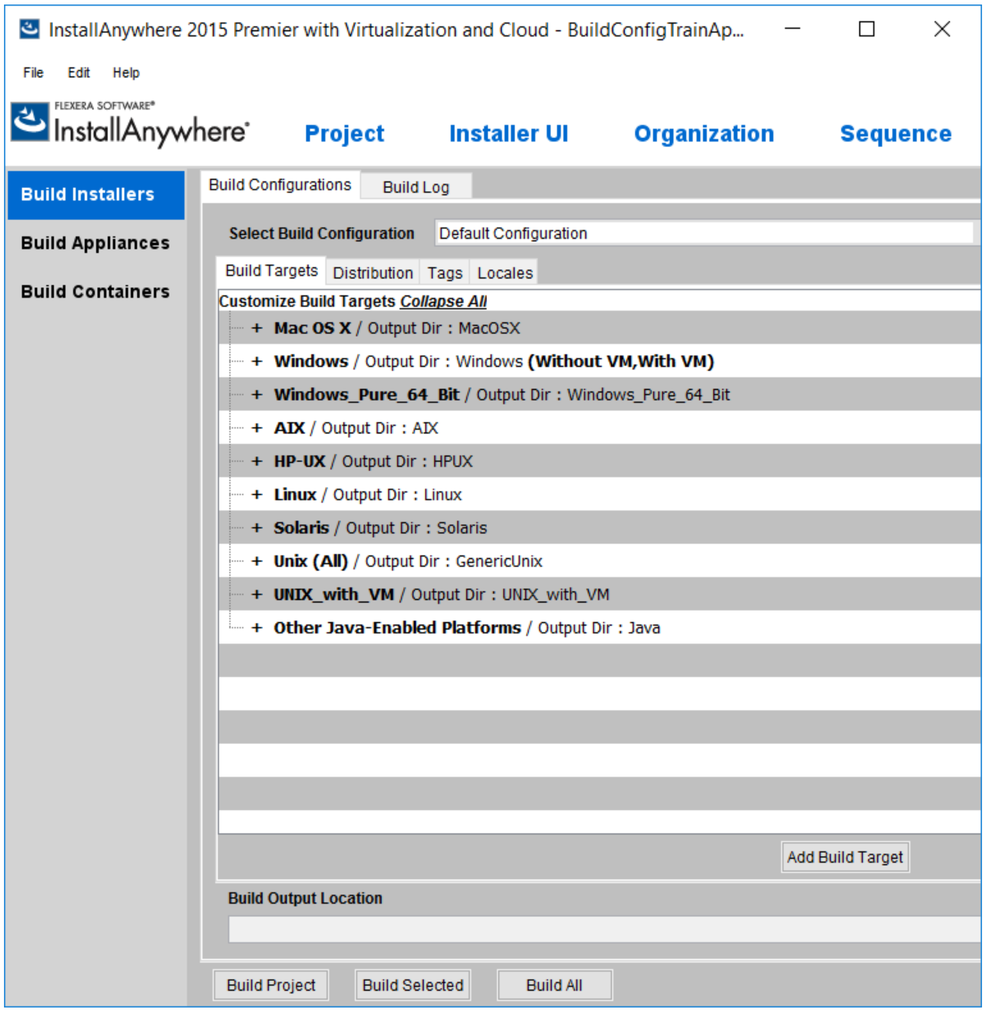
- Creating a New Build Configuration
- Using Build Tags to Customize Build Configurations

Creating a New Build Configuration

To create a new Build Configuration, perform the following steps.

Task ***To create a new Build Configuration:***

1. Open the InstallAnywhere Advanced Designer and create a new project named `BuildConfigTrainApp`.
2. Open the **Build** task. The **Build Configurations** tab is displayed and **Default Configuration** in selected in the **Select Build Configuration** list.


InstallAnywhere 2015 Premier with Virtualization and Cloud - BuildConfigTrainAp...
File Edit Help
Project Installer UI Organization Sequence
Build Installers
Build Appliances
Build Containers
Build Configurations Build Log
Build Targets Distribution Tags Locales
Add Build Target
Build Project Build Selected Build All

Select Build Configuration: Default Configuration

Customize Build Targets *Collapse All*

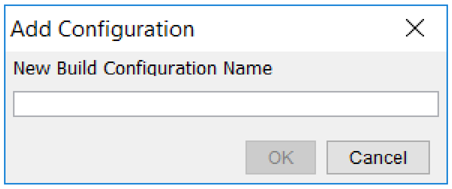
- **Mac OS X** / Output Dir : MacOSX
- **Windows** / Output Dir : Windows **(Without VM,With VM)**
- **Windows_Pure_64_Bit** / Output Dir : Windows_Pure_64_Bit
- **AIX** / Output Dir : AIX
- **HP-UX** / Output Dir : HPUX
- **Linux** / Output Dir : Linux
- **Solaris** / Output Dir : Solaris
- **Unix (All)** / Output Dir : GenericUnix
- **UNIX_with_VM** / Output Dir : UNIX_with_VM
- **Other Java-Enabled Platforms** / Output Dir : Java

Build Output Location

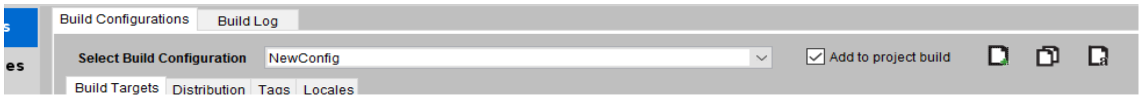
3. Click **Rename** and rename the existing **Default Configuration** to **`RenamedConfig`**.

4. With **RenamedConfig** selected, enable **Linux** and **Solaris** build targets by selecting the check boxes in the **Without VM** column of the **Build Targets** tab. Also, clear the **Windows** check boxes.

5. Click **Add**. The **Add Configuration** dialog box opens.



6. Enter **`NewConfig`** and click **OK**. This new Build Configuration is automatically selected in the **Select Build Configuration** list.


Build Configurations Build Log
Build Targets Distribution Tags Locales

Select Build Configuration: NewConfig

☑ Add to project build

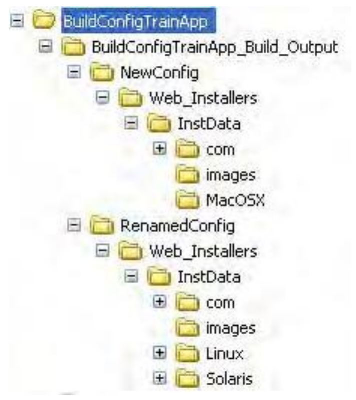
7. Enable the **Mac OS X** build target by selecting the check box in the **Without VM** column of the **Build Targets** tab. Also, clear the **Windows** check boxes.

8. Build both of the installer's Build Configurations by clicking the **Build All** button.

   Note • *When you click the **Build All** button, all Build Configurations in the project are built, not just the selected Build Configuration.*

9. When the build is complete, click the **Open in Explorer** button to open the **Build Output Location**.

10. Example the directory structure of the `BuildConfigTrainApp` folder:



```
BuildConfigTrainApp
  BuildConfigTrainApp_Build_Output
    NewConfig
      Web_Installers
        InstData
          com
          images
          MacOSX
    RenamedConfig
      Web_Installers
        InstData
          com
          images
          Linux
          Solaris
```

Note that the **NewConfig** Build Configuration includes a Mac OS X installer, while the **RenamedConfig** Build Configuration includes a Linux and Solaris installer.

11. Continue with the steps in Using Build Tags to Customize Build Configurations.

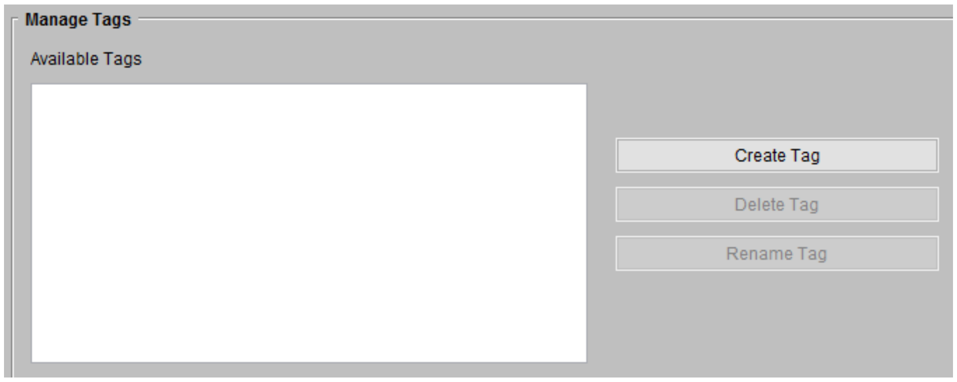
Using Build Tags to Customize Build Configurations

You can use Build Tags to tag resources and map them to Build Configurations. Using Tags in coordination with Build Configuration enables you to create super powerful installers.

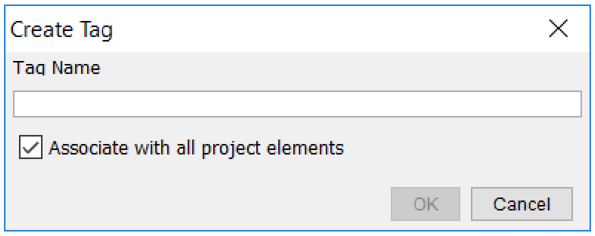
Task ***To use Build Tags to customize a Build Configuration:***

1. Perform the steps in Creating a New Build Configuration.

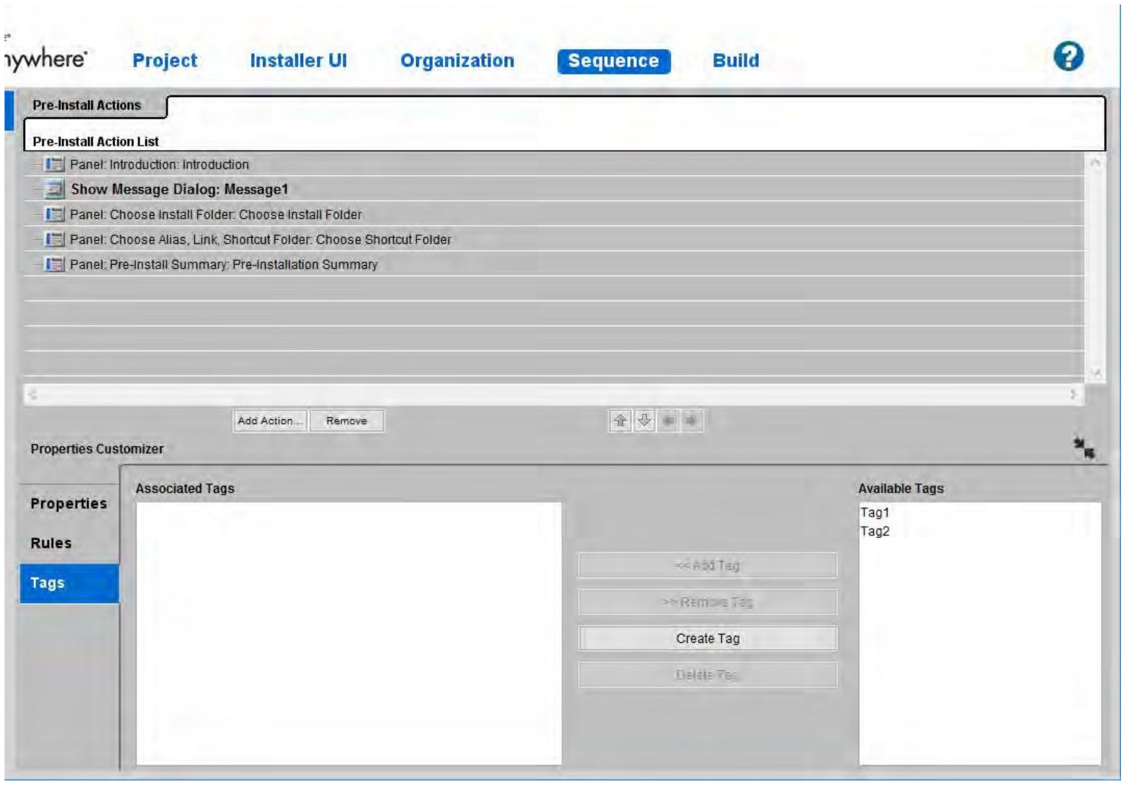
2. Open the **Project > Advanced** task and locate the **Manage Tags** section.



3. Click **Create Tag**. The **Create Tag** dialog box opens.



4. Enter **Tag1** and click **OK**. Make sure that the **Associate with all project elements option** remains selected.
5. Create another tag and name it **Tag2**.
6. Open the **Pre-Install** task.
7. Add a **Show Message Dialog** action and give it the **Title** of **`Message1`**.
8. Open the **Tags** tab of the Message1 **Show Message Dialog** action customizer.



Panel: Introduction: Introduction
Show Message Dialog: Message1
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary

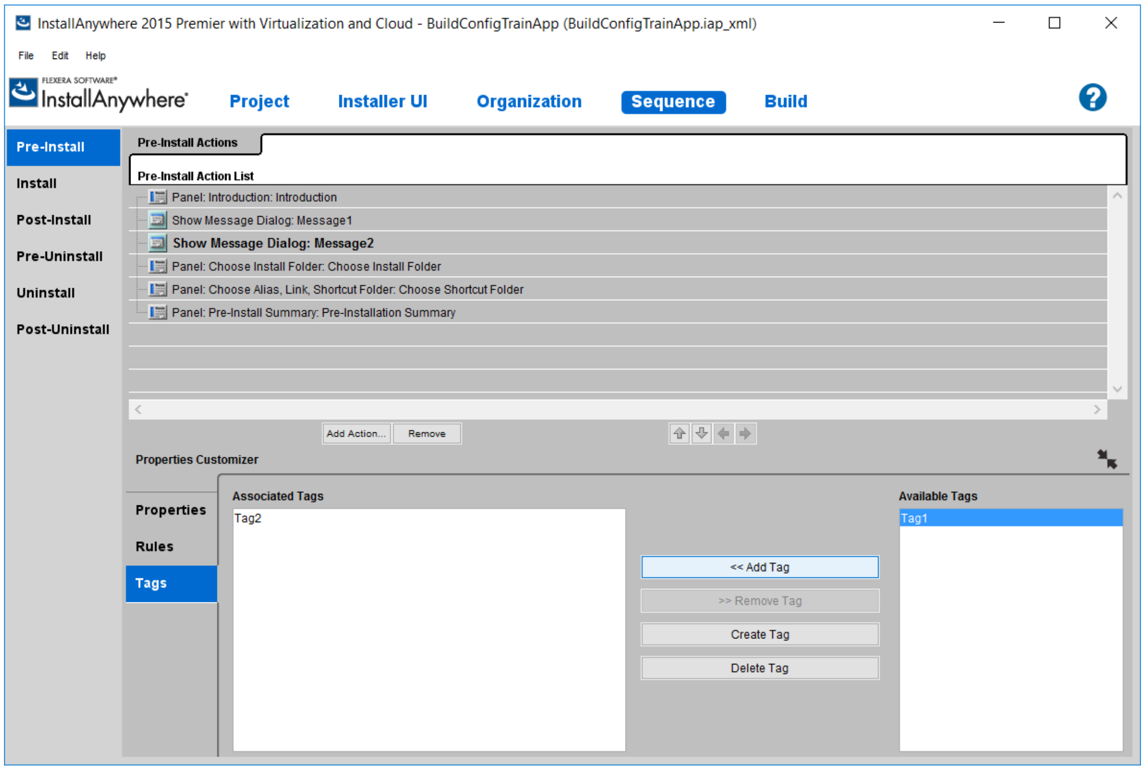
Tag1
Tag2

9. Move **Tag1** into the **Associated Tags** list and leave **Tag2** in the **Available Tags** list.

10. Add a second **Show Message Dialog** action and give it the **Title** of `Message2`.

11. Open the **Tags** tab of the Message2 **Show Message Dialog** action customizer.

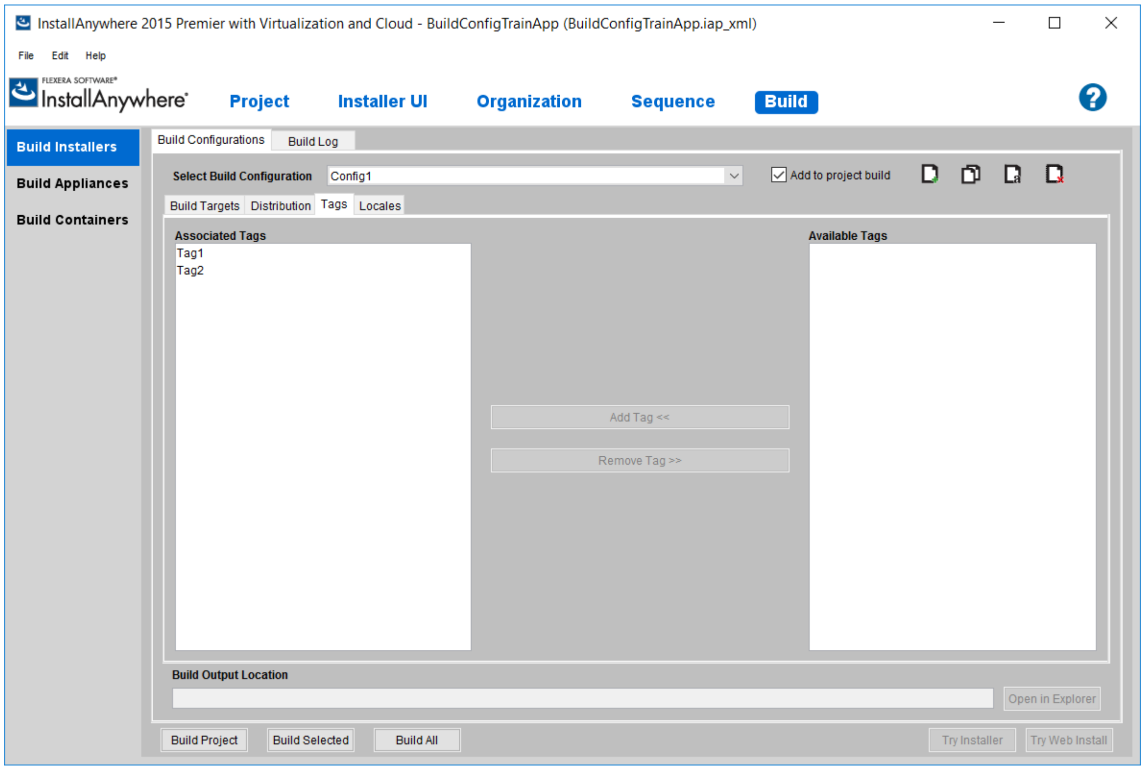
12. Move **Tag2** into the **Associated Tags** list and leave **Tag1** in the **Available Tags** list.


InstallAnywhere 2015 Premier with Virtualization and Cloud - BuildConfigTrainApp (BuildConfigTrainApp.iap_xml)
File
Edit
Help
Project
Installer UI
Organization
Sequence
Build
Pre-Install
Install
Post-Install
Pre-Uninstall
Uninstall
Post-Uninstall
Pre-Install Actions
Pre-Install Action List
Panel: Introduction: Introduction
Show Message Dialog: Message1
Show Message Dialog: Message2
Panel: Choose Install Folder: Choose Install Folder
Panel: Choose Alias, Link, Shortcut Folder: Choose Shortcut Folder
Panel: Pre-Install Summary: Pre-Installation Summary
Add Action...
Remove
Properties Customizer
Properties
Rules
Tags
Associated Tags
Tag2
Available Tags
Tag1
<< Add Tag
>> Remove Tag
Create Tag
Delete Tag

13. Open the **Build** task and create two new Build Configurations, named **`Config1`** and **`Config2`**.

14. Delete the **RenamedConfig** and **NewConfig** that you created in the previous exercise.

15. Select **Config1** from the **Select Build Configuration** list and open the **Tags** subtab.



Select Build Configuration: Config1
☑ Add to project build

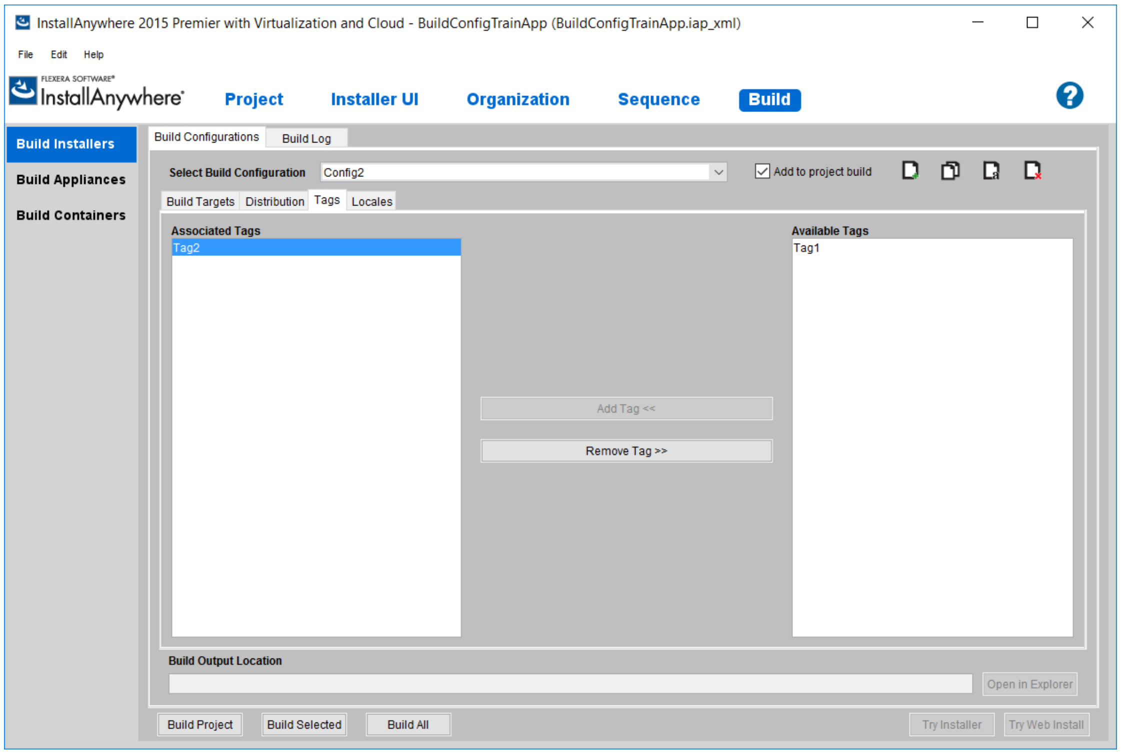
Associated Tags: Tag1, Tag2

Available Tags:

Build Output Location

16. Leave **Tag1** in the **Associated Tags** list and move **Tag2** to the **Available Tags** list.

17. Next, select **Config2** from the **Select Build Configuration** list and move **Tag1** to the **Available Tags** list, leaving **Tag2** in the **Associated Tags** list.



Select Build Configuration: Config2
☑ Add to project build

Associated Tags: Tag2

Available Tags: Tag1

Build Output Location

18. Click **Build All** to build both Build Configurations.

19. Select **Config1** from the **Select Build Configuration** list and click **Try Installer**. Notice that the **Message2** message dialog box does not appear in this installation; only **Message1** appears.

20. Next **Config2** from the **Select Build Configuration** list and click **Try Installer**. Notice that the **Message1** message dialog box does not appear in this installation; only **Message2** appears.

Chapter 14 Exercises: Advanced Organizational Concepts

In this exercise, you will create a merge module installer.

Creating Merge Modules

Advanced Note • *This is an advanced exercise.*

To create a merge module installer, perform the following steps.

Task ***To create a merge module installer:***

1. Create a merge module that can be used to install OfficeSuite as a part of a larger installation.
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 15 Exercises: Integrating InstallAnywhere with Automated Build Environments

In this section, you will learn how to build a project from the command line.

Building TrainApp from the Command Line

Advanced Note • *This is an advanced exercise.*

To build an InstallAnywhere project from the command line, perform the following steps:

Task ***To build a project from the command line:***

1. Close InstallAnywhere.
2. Open a command prompt and run a command similar to the following):

   ```
   "C:\Program Files\InstallAnywhere 2010\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.
3. If you have time, you can examine the arguments you can pass to the builder by entering the following:

   ```
   build.exe -?
   ```
4. Next, try adding and removing platforms or changing the output media type using the command-line arguments.

Chapter 16 Exercises: Custom Code

In this section, you will perform the following exercises:

- Adding FirstCustomCodeAction to TrainApp
- Setting Variables from Custom Code
- Using the InstallerResources Class
- Logging Errors from Custom Code
- Creating a Custom Rule
- Using Other ISMP Services
- Additional Exercises

Adding FirstCustomCodeAction to TrainApp

Advanced Note • *This is an advanced exercise.*

To add a FirstCustomCodeAction to a project, perform the following steps.

Task ***To add a FirstCustomCodeAction to a project:***

1. 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.
2. Verify that the message box appears at run time.
3. If you have time, package `FirstCustomCodeAction` as a plug-in by adding the `customcode.properties` file to the `.jar` file.
4. 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

Advanced Note • *This is an advanced exercise.*

To set variables from custom code, perform the following steps.

Task ***To set variables from custom code:***

1. Modify the `FirstCustomCodeAction` class to create a custom variable called `$NOW$`, whose value should be set to the current time.

   Note • *You can use* `java.util.Date` *to compute the current time, as done in the* `DisplayTime` *class.*

2. 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( ));
```

3. Recompile the class and repackage it into a `.jar` file.
4. Verify that the action works by (for example) using the expression `$NOW$` in a panel that follows the updated action.

Using the InstallerResources Class

Advanced Note • *This is an advanced exercise.*

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.

To use the InstallerResources class, perform the following steps.

Task ***To use the InstallerResources class:***

1. 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.
2. As described in Chapter 16, Custom Code, 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 "..."; }
}
```

3. 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
```

4. 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

Advanced Note • *This is an advanced exercise.*

The `CustomError` class provides methods for writing information to the installer log file. To

log errors from custom code, perform the following steps.

Task ***To log errors from custom code:***

1. Modify `FirstCustomCodeAction` to use the append`Message` and log methods to write a message to the log file.

   As described in Chapter 16, Custom Code, you obtain a `CustomError` object using code similar to the following:

   ```
   CustomError myErr = (CustomError)ip.getService(CustomError.class);
   ```

2. 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( );
   ```

3. To ensure the installer creates the installation log, open the **Project > Info** task and select the check box labeled **Generate** install log during installation.

4. After rebuilding and repackaging the class, run the installer and then browse for the installation log. It should contain the message specified in the append`Message` method, "`Calling FirstCustomCodeAction...`"

Creating a Custom Rule

Advanced Note • *This is an advanced exercise.*

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`.

To create a custom rule, perform the following steps.

Task ***To create a custom rule:***

1. 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");
}
}
```

2. 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.

3. 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.

Using Other ISMP Services

Advanced Note • *This is an advanced exercise.*

To use other ISMP services, perform the following steps:

Task ***To use other ISMP services:***

Following the examples in the manual, create example custom code actions or rules that call into other ISMP services.

- For example, use the `Win32RegistryService` interface to read the PATH value inside HKEY_CURRENT_USER\Environment, OR
- Use the `Win32Service` interface to determine whether the Event Log service (with internal name `Eventlog`) is present and running on the target system.

Additional Exercises

Advanced Note • *This is an advanced exercise.*

If you have extra time, the following are additional methods and classes that can be useful in custom code actions and rules. The javadoc documentation will provide further information about InstallAnywhere classes used in these exercises.

Table B-3 • Additional Exercises

Name	Description
Exit a running installation	In a custom code action, call *proxy*`.abortInstallation` to exit the running installation (*proxy* being the `InstallerProxy` object available to the action, often called `ip` in this manual).
Increment the installation progress bar while an action is taking place	Compile and run the SampleProgress example that ships with InstallAnywhere (in the `CustomCode/samples` directory). The example illustrates how to increment the installation progress bar while the action is taking place.
Save and retrieve encrypted variables in a response file	Extend an existing custom code action that uses InstallAnywhere variables so that it saves and retrieves encrypted variables in a response file using the `ReplayVariableService` class.

Chapter 17 Exercises: Custom Panels and Consoles

In this section, you will perform the following exercises:

- Adding a Get User Input Panel to TrainApp
- Interacting with a Custom Panel's Navigation Buttons

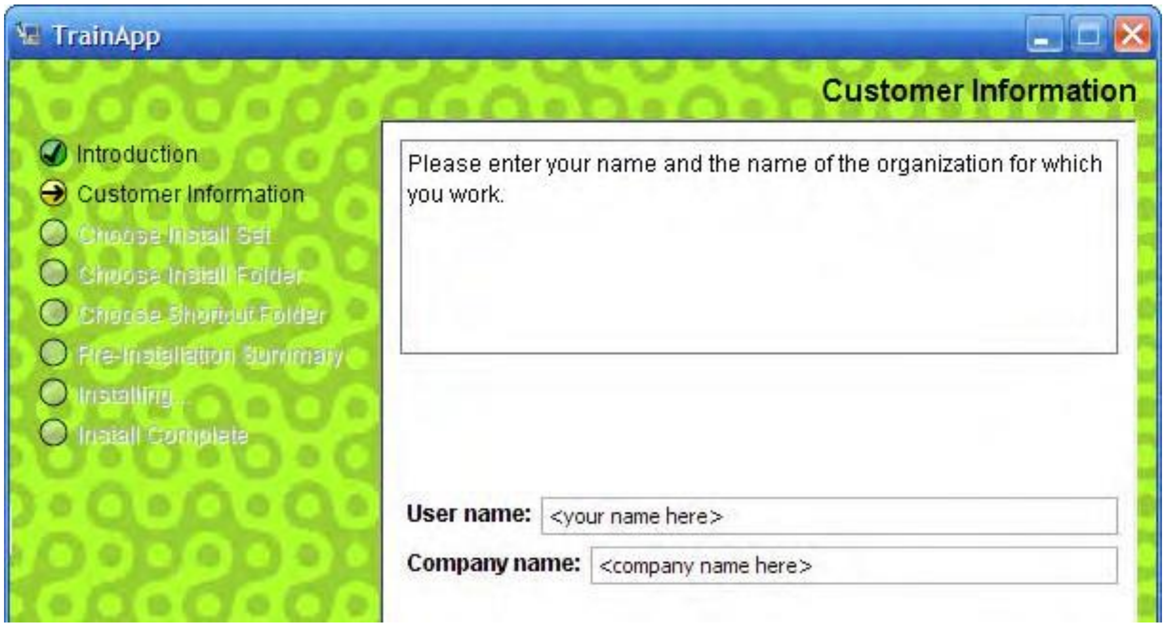
Adding a Get User Input Panel to TrainApp

Advanced Note • *This is an advanced exercise.*

To add a Get User Input panel to a project, perform the following steps:

Task ***To add a Get User Input panel:***

1. 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.



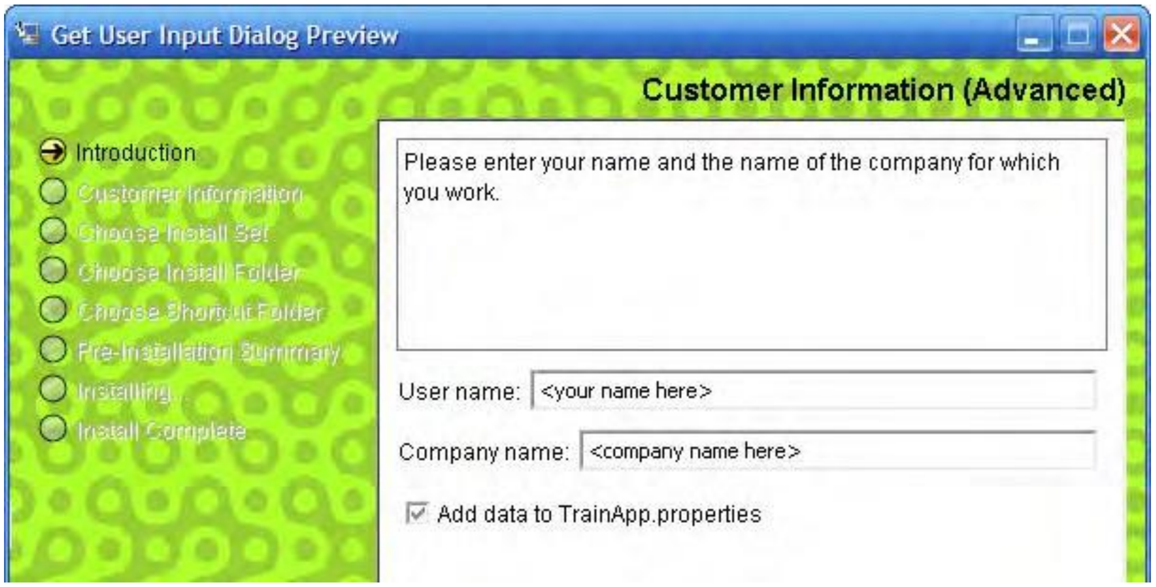
Please enter your name and the name of the organization for which you work.

User name: <your name here>

Company name: <company name here>

2. 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`.

3. 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`.



Please enter your name and the name of the company for which you work.

User name: <your name here>

Company name: <company name here>

☑ Add data to TrainApp.properties

4. 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

Advanced Note • *This is an advanced exercise.*

As described in Chapter 17, Custom Panels and Consoles, you can control the enabled state and visibility state of a custom panel's navigation buttons (its **Next**, **Previous**, and **Cancel** buttons) using methods in the GUIAccess class.

In this exercise, you will modify a `BlankCustomCodePanel` so that it uses the `GUIAccess` class to disable the panel's **Previous** button and hide the panel's **Cancel** button.

Task ***To interact with a custom panel's navigation buttons:***

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);
```

2. After recompiling and repackaging the custom code panel class, the panel should appear at run time with the specified buttons disabled and hidden.

Chapter 18 Exercises: Localizing and Internationalizing Installers

In this section, you will perform the following localization exercises:

- Localizing the TrainApp Installer
- Translating Custom Strings

Localizing the TrainApp Installer

Advanced Note • *This is an advanced exercise.*

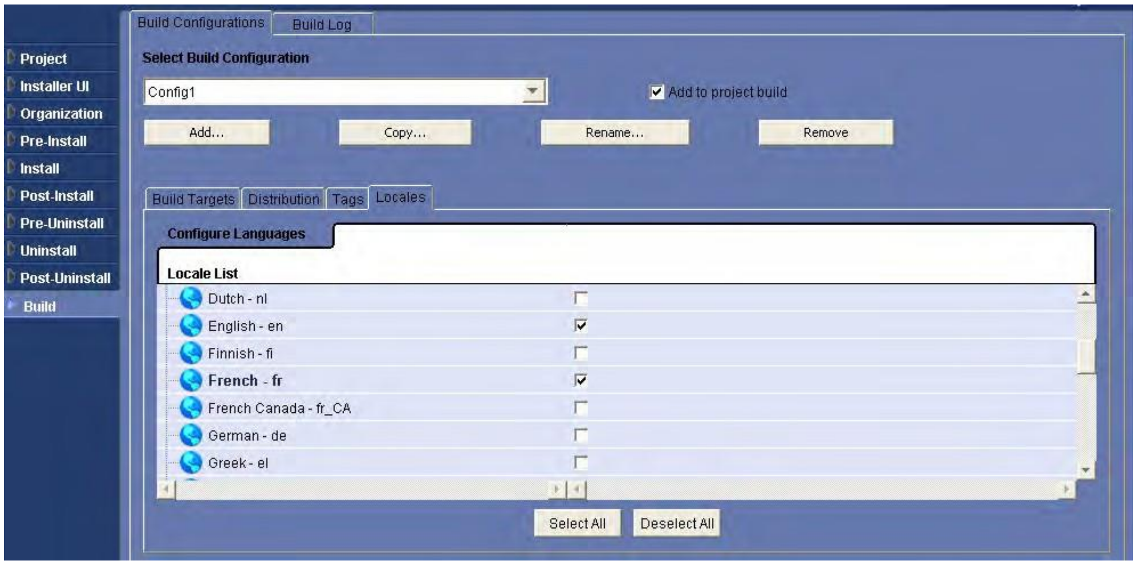
To localize the TrainApp project, perform the following steps:

Task ***To localize the TrainApp installer:***

1. In the **TrainApp** project, add support for one or two non-English locales using the **Locales** subtab on the **Build Configurations** tab of the **Build** task.

   For example, the following figure shows the French locale being added to this project's selected Build Configuration.

   Note • *The locale abbreviations are displayed after the locale names—**fr** for French, for example. These abbreviations will be needed when creating localized resource bundles.*



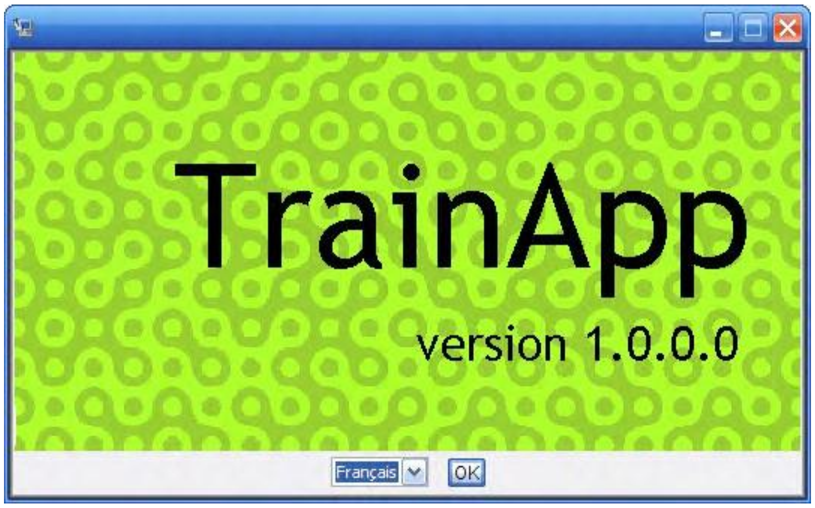
Select Build Configuration: Config1 — [✓] Add to project build

Configure Languages

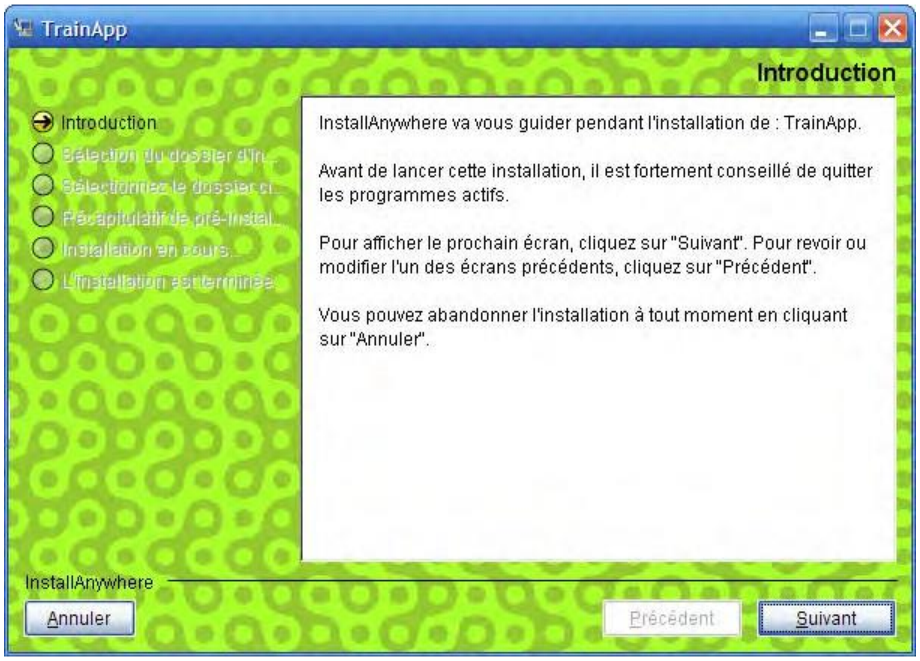
Locale List

Dutch - nl	☐
English - en	☑
Finnish - fi	☐
French - fr	☑
French Canada - fr_CA	☐
German - de	☐
Greek - el	☐

2. When you rebuild and run the installer as an end user, you will be prompted for the locale to use for installation.



At run time, the text on predefined panels will be displayed using the selected locale. The following is an example of the French locale:


TrainApp
Introduction
Introduction
Sélection du dossier d'in...
Sélectionnez le dossier ci...
Récapitulatif de pré-instal...
Installation en cours...
L'installation est terminée.
InstallAnywhere
Annuler
Précédent
Suivant

InstallAnywhere va vous guider pendant l'installation de : TrainApp.

Avant de lancer cette installation, il est fortement conseillé de quitter les programmes actifs.

Pour afficher le prochain écran, cliquez sur "Suivant". Pour revoir ou modifier l'un des écrans précédents, cliquez sur "Précédent".

Vous pouvez abandonner l'installation à tout moment en cliquant sur "Annuler".

Translating Custom Strings

Advanced Note • *This is an advanced exercise.*

The **Get User Input** panels you added to the TrainApp project contain hard-coded text for prompts and labels. To modify the project to use localized strings (for example, for French), perform the following steps:

Task ***To translate custom strings:***

1. Locate the `ProjectNamelocales` directory next to your project file.
2. Modify the appropriate values in the `custom_fr` (for French) text file.
3. If you have time, modify `FirstCustomCodeAction` to read a localized string at run time, using the `getValue` method of the `InstallerProxy` and other proxy classes.

Index

A

Action Groups 103, 107
action groups
    and Maintenance Mode 131
actions
    Trigger Rollback 179
Add Features 130
Add Files 37
Add or Remove Programs 176
Adding Directories with SpeedFolders 97
Adding Merge Modules 200
Advanced Action Methods 245
Advanced Designer 17, 319
Advertised Variables 196, 199, 200
AIX Software Vital Product Database 51
Ant 217, 220
Ant Build Integration 220
Apache Ant 217
Application Servers 115
Authoring Environment 17
Authoring Environment Requirements 17
Auto Populate 346

B

Background Images 70, 74
Best Practices for Components 93
Best Practices for Features 93 Best
Practices for Install Sets 93
bidirectional text 282
Billboards 29, 70, 78, 346
Build 332
Build Configurations
    about 181
    adding to project build 187
    assigning Tags to project elements 191
    associating Tags with 192
    benefits of using 182
    creating 181
    creating for migrated project 186
    creating new 186
    creating new Tags 190
    default for new and migrated projects 182
    editing 181
    Migrated Configuration 186
    options 183
    overview 181
    searching for Tags 192
    using Tags to customize 188
Build Configurations tab 183 Build
Log 68
Build Options 198
Build Targets 59
Build task
    Build Configurations tab 183
Building Releases 59
Build-properties File 219 Build-
time Merge Modules 197
Bundled Virtual Machine 62

C

CD-ROM Installers 66
Change Product_Name Installation.exe 139
Changing
    Localized Text 288
Check Running Mode rule 131
    about 135
    and Maintenance Mode 135

Choose Alias, Link Shortcut Folder 321
Choose Icons 346
Choose Install Folder 321
Classpath 31
Clean Components 90
Collaboration 204, 209
command-line
    build options 218
    builds 217
Component Dependencies 92
Components 85, 92, 93
Console actions 103
Console Installation 22
Contents view 206
Creating a DIM 205
Creating Debug Output 53
Custom Code 223, 287
Custom Errors 247
Customizer 327
    Common Properties 117

D

Database Servers 115
Debug information 154, 155
Debug output 53
Debugging 53, 151, 152, 153, 155
Debugging Custom Code 245
Default Translations 289
Dependencies 209
Dependency component 91
Dependency Failed Message 92
Deploy WAR/EAR Archive 117
Deployment 16
Design-time Merge Modules 197, 201
digital certificates 220 digitally
signed installers 220 DIM 204,
208, 210, 213
DIM References 211
Distribution 64, 333
dynamic build targets 60
Dynamic Merge Modules 198
Dynamic Text 283

E

Eclipse 204, 206, 210
Editions 22
Enable Rollback 178
Enterprise Edition 22, 155, 279
Execute Target File 329
Exit Codes 218

F

Features 89
File Service 240
Find Component in Registry 92
FLEXnet Connect 214

G

general actions 103
Graphics 29

H

headless systems 217
Help 77
hsqldb.jar 245

I

i5/OS 51
Image Settings 75
Install Actions 32
Install Merge Module 196
Install Progress Panel 345
Install Sets 83, 86, 88
Install task
    adding a Trigger Rollback action 179
InstallAnywhere 2008 Value Pack 1 features 51, 178, 218, 232,
    233, 271
InstallAnywhere Collaboration 204
InstallAnywhere installer 23 InstallAnywhere
Web Installer Applet 23 Installation 16
Installation Log 247
Installer Steps 75, 76
Installer UI 75
Installers
    Internationalizing 279
    Localizing 279
InstallShield 208
InstallShield Collaboration 204
Install-time Merge Modules 197
Instance Management 135
    Do not limit instances 137
    Identify Instance Using 137
    Restrict to a Single Instance 137
    Restrict to Number of Instances 137
    specifying options 136
    Version Field Level That Identifies New Instance 138
Internationalizing Installers 279
ISMP 240, 244
ISMP file service 240

ISMP Manifests 203

J

Jakarta 220
Java Virtual Machine
  International 289
Java virtual machines 54
JDBC drivers 116
JUnit tests 53

L

Label Settings 75, 76
Language Resource Properties 289
LaunchAnywhere 37, 38, 54, 118, 151, 153, 308, 310, 323
LaunchAnywhere executables 153
LAX
  LaunchAnywhere executable 153, 323
List of Installer Steps 75
locales 52, 155, 279, 280, 284
Locales subtask 52
Localizable Elements 289
Localization 52, 283, 284
Localized Text 288
Localizing
  Best Practices for, 288
  Custom Code 287
  Custom Installer Labels 286
  Installers 279
  Resources 284
  Splash Screen 284
logging 247

M

Mac OS X 118
  debugging 153
  troubleshooting 153
Mac OS X Installer 153
Mac OS X Support 50
Magic Folders 96
Maintenance Mode
  action groups 131
  Add Features user experience 142
  configuring 128
  creating new Tags on Project > Advanced subtask 190
  enabling 128, 129
  end user experience 141
  Instance Management options 135
  launcher 139
  Maintenance Mode launcher compared to Uninstaller 139
  Manage Instances dialog box 147
  options 129
  overview 127
  Pre-Install task 132
  Pre-Uninstall task 134
  Remove Features user experience 144
  Repair Product user experience 145
  Uninstall Product user experience 146 when multiple installed instances 147
Manifest File Format 100
Manifest Files 100
Merge Module Size 198
Merge Modules 194, 195, 197
Meta Information 212
Microsoft Visual Studio 204
Modifying
  Localized Text 288
multiplatform deployment 16
Multiplatform Installation 16

N

native2ascii 281

O

Optional Installer Arguments 54
Organization 85
Organizing Features and Components 92

P

Panel Actions 75, 103
  Settings 75
platform abbreviations 218
Platform Support 49
Platforms 49
plug-in actions 103
Plug-ins 106
Post-Install 108, 327, 331
Pre-Install 108, 321
pre-installation 29
ProductRegistryService 237
Project 32, 45, 46
Project Wizard 17, 31, 32
projects
  enabling installation rollback 178
Pure Java Installer 153
Pure Java task 52

Q

Querying the ISMP Registry 244

R

RAIR 51
Red Hat Package Management Support 50
Registered Application Information Repository 51
Remove Features 130
Repair Installation 131
requirements 17
response files 22
rollback options 178
  Enable Rollback 178
  Rollback Settings 178
  Rollback tab 179
  Trigger Rollback action 179
RPM Support 50
Rules 52, 78, 88, 90, 105, 327, 328
rules
  Check Running Mode 135

S

security 220
Security Service 242
Setting the Classpath 39, 310
Shared Components 91
silent installation 22 Source
Path variable 159 Source
Paths 157, 158, 159
SpeedFolders 97
Splash Screen 70
SQL 115
Standard Component 91
Standard Edition 23, 279
Static Text 283
stderr 53
stdout 53
SWVPD 51
System i (i5/OS) Support 51
System Utility Service 243

T

Tags
  assigning to project elements 191 associating
  with Build Configurations 192 creating 190
  searching for 192
  using to customize Build Configurations 188
Target Environments 17, 19
task
  Java 55
  Rules 52
Templates 195, 196
Timestamp 47
Translations 289
Trigger Rollback 179
Types of Components 91

U

UI Modes 70
UI Settings 78
Uninstall Categories 123
Uninstall Product 131
Uninstall task
  about customization 121
  adding Uninstall Categories and Actions 123
  assigning a rule to Uninstall Category 126 assigning
  actions to features and components 125 benefits of
  customizing 122
  customizing 121
  overview 121, 122
  preventing an Uninstall Category from being uninstalled 126
Uninstaller 75
  about 139
Unit Tests 210
Unix
  debugging 153
  LaunchAnywhere 118
  with VM 61
Unix installation 28
unsigned code 220
User Account Control 50

V

Variables 96, 155
Virtual Machine 54
  Bundled 62
Visual Studio 204
Visual Tree 100
VM Packs 61
VM Selection 62

W

WAR/EAR deployment 115
Web Installers 65
Win32 installer 152
Windows 118, 330
Windows NT 152
  troubleshooting 152
Windows Vista
  command line builder 218
Writing Custom Errors 247