

Symphony Marimba Client Automation Application Packager User Guide



Supporting

Symphony Marimba Client Automation Application Packager version 9.0.00

November 2015

© Copyright 2014 Symphony Teleca, Corporation or its subsidiaries. All rights reserved. All information contained in this document is confidential and proprietary to Symphony Teleca, Corporation and may not be disclosed, reproduced, used, modified, made available, used to create derivative works, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, by or to any person or entity without the express written authorization of Symphony Teleca, Corporation. In consideration for receipt of this document, the recipient agrees to treat this document and its contents as confidential and agrees to fully comply with this notice. This document refers to numerous products by their trade names. In most, if not all, cases their respective companies claim these designations as Trademarks or Registered Trademarks. This document and the related software described herein are supplied under license agreement or nondisclosure agreement and may be used or copied only in accordance with the terms of such agreement. The information in this document is subject to change without notice and does not represent a commitment on the part of Symphony Teleca, Corporation. Contact Symphony Teleca, Corporation Customer Support to verify the date of the latest version of this document. The names of companies and individuals used in the sample database and in examples in the manuals are fictitious and are intended to illustrate the use of the software. Any resemblance to actual companies or individuals, whether past or present, is purely coincidental. Symphony Teleca, Corporation reserves all copyrights, trademarks, patent rights, trade secrets and all other intellectual property rights in this document, its contents and the software described herein.

Contacting Symphony Teleca Customer Support

You can obtain technical support by contacting Customer Support by telephone or e-mail. We are available 24/7/365.

Please send an e-mail to: CustomerSupport@Symphonyteleca.com

You can also call us at +1 214 396 0493 or US Toll Free Number +1 855 394 1543.

Before contacting Symphony Teleca support

Please gather the following information and have it ready before contacting Symphony Teleca.

This will help us service your request immediately:

- Marimba channel version for each module being used
- Sequence of events leading to the issue
- Error messages received along with the time and date that you received them
- Environment details (number of transmitters, type of transmitters, number of endpoints, Operating System from servers and endpoints, Database version)
- Details about the problem
- Screenshots of errors
- Attachments of relevant logs and configuration files



Table of Contents

Preface	17
How this guide is organized.	18
Best Practice and other icons	19
Related documentation	19
Accessing product documentation.	22
Using the documentation channel.	22
Using the Channel Store	22
Chapter 1 Introduction	23
What is Application Packager?	24
Application Packager components.	24
Software distribution process	26
Application Packager interface	27
Package directory	28
Application Installer.	30
Scalability	31
Supported platforms.	32
Windows platform support	32
UNIX platform support	33
Reference information	34
Using help	34
Chapter 2 Using Packager for Shrinkwrap Windows Applications	37
Overview	38

Packaging shrink-wrapped Windows applications	38
Setting up the packaging environment	39
Creating a preinstall snapshot.	39
Selecting a snapshot source.	40
Selecting file systems	41
Selecting registry entries	45
Editing the Windows registry	46
Customizing snapshots	46
Selecting metabase entries	47
Generating the preinstall snapshot and saving the snapshot file.	49
Preparing to install the application	50
Installing the application and creating a postinstall snapshot	50
Packaging the application into a channel	51
Managing updates to the application.	52
What happens during a channel update	53
Taking over an application that already exists on the endpoints	54
Removing all files and registry keys associated with an application	55
Packaging the removal of an application	55
Working with Windows File Protection	57
Installing Office 2007 using the Packager for Shrinkwrap Windows Application	57
 Chapter 3	
Getting started	59
Before you get started	60
Preparing to package applications in UNIX	61
Starting Application Packager.	62
Working with channels.	62
Editing existing channels	62
Publishing channels	63
Automatic upgrade of packages during installation.	63
Importing channels	64
Removing channels from the list	64
 Chapter 4	
Using the Package Editor	67
Overview	69
Opening a channel for editing.	69
Editing the contents of a channel	70

Working with files	71
Working with INI files	78
Working with directories	82
Working with registry entries	83
Working with metabase entries	86
Renaming the items in a channel	92
Removing items from a channel	92
Marking items for removal from the endpoints	93
Searching for file system and registry entries	94
Using macros	95
Creating user-defined macros	96
Applying macros in INI files (Windows only)	97
Examples: Using macros in packaged applications	98
Editing user-defined macros	103
Removing user-defined macros	103
Setting channel startup options	104
Setting installation modes	106
Setting automatic rollback for failed installations	108
Preventing items from being stored for backup	109
Delaying the download of a channel's files	109
Ability to kill a process as a part of pre-post install	110
Enabling installer logging	111
Reading a channel's history-n.log files	112
Setting system reboot options (Windows only)	114
Detecting when reboots are needed	115
Using scripts to trigger reboots	116
Specifying the installation platform	117
Redirection behaviour	118
System32 and SysWOW64 folders in C:\Windows	118
Limitations	119
Program Files and Program Files (x86) folders in C:\	119
Registry hive inside WoW6432Node and outside WoW6432Node	120
Setting policies for installation, update, uninstallation, verification, and repair	121
Selecting policies for channels	121
Setting default policies for channels	123

Setting policies for individual items in a channel	124
Determining whether files are restored or removed during uninstallation	126
Removing all items from endpoints during uninstallation of a channel	127
Using environment variables	127
Adding an environment variable	128
Editing an environment variable	129
Removing an environment variable	129
Environment variables on different Windows platforms	130
Configuring system and channel dependencies	130
Configuring disk space requirements.	132
Configuring file requirements.	133
Configuring environment variables requirement.	134
Configuring registry entry requirements	136
Configuring channel requirements	138
Combining multiple requirements by using AND and OR dependencies. .	140
Customizing a channel by using scripts and classes	141
Customizing a channel using an executable script	141
Customizing a channel by using a Java class	145
Managing major and minor updates	150
Setting scripts and dependencies for major updates	151
Setting scripts and dependencies for minor updates	151
Managing services (Windows only)	152
Editing services	152
Adding services	157
Removing services	158
Editing the channel parameters and properties	158
Minimizing bandwidth usage	159
Preload and delayfiledownload parameters	160
Working with packages that contain user-specific files and registry entries . .	161
Using the multiple-user feature with Policy Management	163
Package behavior with the multiple-user feature	163
Verifying and repairing channels	165
Verifying channels before publishing and distributing	166
Scheduling verification and repair for channels	167
Manually running verify and repair for channels	168
Updating and repairing applications using shortcuts before startup (Windows	

only)	169
Associating packages with software library items in the Definitive Software Library (DSL)	171
Editing ASCII text files	173
Creating a text modifier group	174
Creating an insert line modifier	175
Creating a search and replace line modifier	176
Creating a search and replace text modifier	179
Setting installation and update policies for text modifiers	181
Verifying and repairing files modified by text modifiers	183
Troubleshooting text modifiers	183
Recording change history for channels	184
Saving changes to channels	186
Saving channels to a network drive	186
Reverting to the previously saved version of a channel	187
Saving a copy of a channel in a different directory	187
Recovering from channel corruption.	187
 Chapter 5 Using Windows Installer Packager	189
Overview	190
Packaging a Windows Installer or MSI application into a channel	191
Packaging an MSI database into a channel	192
Benefits and limitations of packaging files by reference	193
Packaging an MSI database with cabinet files	195
Packaging an MSI database with merge modules	196
Customizing Windows Installer packages	196
Opening a Windows Installer package in the Package Editor.	197
Setting the installation modes for Windows Installer packages	197
Advertising Windows Installer applications	199
Setting the user interface level for Windows Installer packages	200
Customizing MSI error codes and script error codes	201
Managing patches for Windows Installer packages	204
Managing transforms for Windows Installer packages	205
Setting the file download options for Windows Installer packages	207
Setting Windows Installer policies	209
Setting Windows Installer logging	212

Setting MSI logging options	213
Editing the contents of MSI packages	213
Configuring applications to use the Windows Installer command line	214
Repackaging Windows Installer packages.	215
Verify and repair behavior for Windows Installer packages	216
Using Windows Installer redirection.	216
 Chapter 6 Using File Packager	221
Overview	222
Packaging a set of files into a channel	222
Updating a channel	226
Installing Microsoft Office 2007 using File Packager	228
 Chapter 7 Using .NET Packager	231
Overview	232
Components of .NET applications	232
Packaging a .NET application into a channel	234
Editing a .NET Packager channel	236
The .NET Assembly properties	237
Properties page	238
Application Policy Configuration page	238
Adding virtual directories for Internet Information Services	244
Updating a channel	244
 Chapter 8 Using PDA Packager	247
Overview	248
Packaging files and PDA applications into a channel.	248
Using the PDA packager to update a channel	251
Deploying PDA packages to mobile devices.	252
 Chapter 9 Using Java Packager	253
Overview	254
About Java Virtual Machines	254
Packaging a Java application into a channel.	255
Passing arguments to the JVM	258
Editing a Java Packager channel	259

Updating a channel	259
Chapter 10 Using Virtual Packager	261
Overview	262
About virtualized applications	262
Requirements for packaging virtual applications	262
Packaging a virtual application into a channel.	263
Editing a Virtual Packager channel	264
Publishing a virtual package	264
Troubleshooting virtual packages	264
Chapter 11 Using Custom Application Packager	265
Overview	266
Packaging a custom application into a channel	266
Chapter 12 Using XML template files	269
Overview	270
Saving Package Editor settings to an XML template file	270
Configuring Application Packager to use an XML template file for default settings	271
Using XML template files with Packager for Shrinkwrap Windows Applications	272
Loading XML template files for configuration and filters	272
Saving and setting configuration and filters using XML template files	273
Applying an XML template file to existing channels	276
Chapter 13 Publishing channels	279
Overview	280
Publishing channels to a transmitter	280
Creating a backup copy of a published channel	283
Chapter 14 Agentless Deployment	285
Overview	286
Limitations	287
Creating a generic package in Application Packager	287
Application Packager Command-Line Interface (CLI) Commands for creating a generic package for agentless deployment	289

Using Content Replicator for creating a package for agentless deployment	290
Agentless Deployment	291
Logging	291
Agentless Discovery	292
Enabling Agentless Discovery in generic package using Application Packager	292
Enabling Agentless Discovery in generic package using Application Packager CLI	
292	
Enabling Agentless Discovery in generic package using Content Replicator CLI.	
293	
 Chapter 15 Troubleshooting	295
Overview	296
Issues with packages created using Application Packager	296
Looking at the log files	301
Application Packager's history logs	301
Endpoint logs	302
Turning on the debugging feature	302
When to use debugging	302
Turning on debugging	303
Specifying flags	304
 Appendix A Command-line reference	307
Overview	308
The runchannel program	308
Syntax	309
Packager for Shrinkwrap Windows Applications command-line options	309
Creating preinstall and postinstall snapshots	310
Comparing the snapshots and creating a channel	310
Windows Installer Packager command-line options	311
Repackaging the contents of an existing Windows Installer Packager channel	312
File Packager command-line options	313
Packaging a set of directories and files	313
Repackaging the contents of an existing File Packager channel	314
.NET Packager .NET Packager command-line options	315
Repackaging the contents of an existing .NET Packager channel	317
PDA Packager command-line options	319
Virtual Packager command-line options	323

Packaging a set of directories and files	323
Repackaging the contents of an existing Virtual Packager channel	324
Custom Application Packager command-line options	324
Applying an XML template file to channels	325
Upgrading channels	325
Checking the version of Application Packager and channels	326
Starting packaged applications from the command line	327
Staging MSI applications on endpoints.	329
Appendix B Policies for installation, update, uninstallation, verification, and repair	331
Overview	332
Policies for the entire channel	332
Installation policies	332
Update policies	334
Uninstallation policies.	336
Verification policies	338
Repair policies	340
Policies for directories and files	342
How update policies work when removing shortcuts	343
Installation policies	344
Update policies	346
Uninstallation policies.	348
Verification policies	349
Repair policies	351
Policies for INI files	352
Installation policies	353
Update policies	353
Uninstallation policies.	354
Verification policies	355
Repair policies	356
Policies for registry and metabase keys and values	356
Installation policies	357
Update policies	358
Uninstallation policies.	358
Verification policies	359
Repair policies	360

Policies for Windows NT services	361
Installation policies	362
Uninstallation policies	363
Update policies	364
Verification policies	364
Repair policies	365
Policies for text modifier groups and text modifiers	366
Installation policies	367
Update policies	367
Appendix C XML syntax	369
Overview	370
Generic tags	370
<DEPEND_CONNECTION_SPEED>	371
<PACKAGE>	372
<CUSTOMMACRO>	372
<MACRO>	373
<ENVIRONMENT>	374
<ENV_PROPERTY>	374
<SERVICES>	375
<SERVICE>	375
<SERV_GENERAL>	376
<SERV_SECURITY>	377
<SERV_ADVANCED>	378
<SERV_DEPEND>	378
<SERV_POLICIES>	379
<CONFIGURATION>	380
<PLATFORM>	380
<INSTALLER>	381
<REBOOT>	383
<POLICIES>	384
<STARTUP>	385
<VERIFYREPAIR>	387
<DEPENDENCIES>	388
<DEPEND_MEMORY>	389
<DEPEND_PROCESSOR>	389

<DEPEND_RESOLUTION>	390
<DEPEND_DISKSPACE>	390
<DEPEND_DRIVE>	391
<DEPEND_FILES>	391
<DEPEND_FILE>	391
<DEPEND_REGISTRY>	392
<DEPEND_REGKEY>	393
<DEPEND_CHANNELS>	394
<DEPEND_CHANNEL>	394
<DEPEND_ENVIRONMENT_VARS>	395
<DEPEND_ENVIRONMENT_VAR>	395
<CUSTOMIZATION>	396
<SCRIPTS>	396
<SCRIPT>	397
<PROPERTY>	398
<PARAMETER>	399
Non-MSI tags	400
<DIRECTORY>	400
<FILE>	402
<SHORTCUT>	403
<SYMLINK>	405
<REGKEY>	406
<REGVALUE>	407
<METABASEKEY>	408
<METABASEVALUE>	409
<MODIFIERGROUP>	410
<MODIFIER>	411
<INI>	412
MSI tags	414
<MSI_INSTALLATION>	414
<MSI_UILEVEL>	415
<MSI_PATCHES>	415
<MSI_PATCH>	415
<MSI_TRANSFORMS>	416
<MSI_TRANSFORM>	416
<MSI_DOWNLOAD>	416

<MSI_POLICIES>	417
<MSI_LOGGING>	417
Search and modify tags	418
<MODIFY_OBJECT>	418
<MODIFY_OBJECT_ATTRIBUTES>	420
Snapshot tags	428
<SNAPSHOT>	429
<FILESYSTEM>	429
<DIRECTORY>	430
<FILTER>	430
<REGISTRY>	431
<REGKEY>	431
<FILTER>	431
<METABASE>	431
<METKEY>	432
<FILTER>	432
Appendix D Windows system macros	433
Overview	434
System macros	434
Appendix E Choosing between Content Replicator and File Packager	439
Deciding which packager to use	439
Application Packager (Software Distribution module)	440
Content Replicator (Content Distribution module)	440
Comparison	441
	444
Glossary	445
Index	457

Preface

This guide contains configuration and administration information about Symphony Marimba Client Automation Application Packager. With Application Packager you can create binary software and data packages in a comprehensive variety of formats. The packages you create are in a proprietary format that Symphony Marimba Client Automation applications can easily and efficiently process. You can deploy the packages you create to any endpoints in your organization, ensuring consistency of software and data across the enterprise.

This guide is intended for software developers, system administrators, IT managers, and other personnel who package software for distribution to desktops or servers. This guide assumes that you are familiar with Symphony Marimba Client Automation components, such as tuners and transmitters. For a review of the basics of using these components, see *Symphony Marimba Client Automation Introduction* on the Customer Support website.

The following topics are provided:

- How this guide is organized (page 18)
- Best Practice and other icons (page 19)
- Related documentation (page 19)
- Accessing product documentation (page 22)

How this guide is organized

The chapters are grouped into the following categories:

- **Introduction**—This chapter provides an overview of how Application Packager works, basic concepts about packaging applications, and a description of this document’s organization and typographical conventions.
- **Using Application Packager**—The following chapters tell you how to package applications using the various components of Application Packager:
 - “Using Packager for Shrinkwrap Windows Applications” on page 37
 - “Using Windows Installer Packager” on page 189
 - “Using Custom Application Packager” on page 265
 - “Using File Packager” on page 221
 - “Using .NET Packager” on page 231
 - “Using PDA Packager” on page 247
 - “Overview” on page 254
- In addition, “Using the Package Editor” on page 67 tells you how to edit and customize applications, while “Using XML template files” on page 269 tells you how to use XML template files to configure packaged applications.
- **Publishing channels**—The chapter “Publishing channels” on page 279 tells you how to publish channels using the publishing wizard that appears when you click the Publish button anywhere in Application Packager.

Best Practice and other icons

Icons in the Configuration Management documentation have the following meaning.

Icon	Description
	The Best Practice icon highlights processes or approaches that Symphony Teleca has identified as the most effective way to leverage certain features.
	Other icons illustrate a step described in a procedure. For example: Make sure that Transmitter is selected.

Related documentation

Symphony Teleca provides Symphony Marimba Client Automation documents in PDF format. These documents are written for system administrators and are listed in the following table.

Guide	Description
<i>Symphony Marimba Client Automation Product Introduction</i>	Introduces you to Symphony Marimba Client Automation and its components and defines basic concepts about its core technology.
<i>Symphony Marimba Client Automation Installation Guide</i>	Provides: <ul style="list-style-type: none"> ■ information needed to design an infrastructure for your enterprise, which involves determining the machines you will use for the various components and whether you need to purchase additional hardware and software ■ instructions for a first-time installation of Symphony Marimba Client Automation and its associated components ■ instructions about upgrading to the current version ■ hardware requirements (such as processing speed, disk space, and RAM) and operating system requirements for supported platforms. This guide also lists the supported databases, directory services, and locales

Guide	Description
<i>Symphony Marimba Client Automation Installation Notes</i>	Lists supported platforms and system requirement.
<i>Symphony Marimba Client Automation Application Packager Guide</i>	Provides information about packaging software for distribution to desktops or servers. This guide also includes information about command-line usage, policies, XML templates, and Windows system macros.
<i>Symphony Marimba Client Automation CMS and Tuner Guide</i>	Provides information about the Common Management Services (CMS) and tuner infrastructure components. This guide also describes the tools and features you use to configure these components.
<i>Symphony Marimba Client Automation Configuration Discovery Integration for CMDB Implementation Guide</i>	Provides instructions about planning, installing, and configuring the Configuration Discovery integration. This guide also includes information about relationship classes and mappings, data exchanges, and reconciliation definitions.
<i>Database Schema Guide</i>	Provides reference information about database schema, such as table names, field names, indexes, and primary, foreign, and unique key constraints.
<i>Symphony Marimba Client Automation Deployment Manager Guide</i>	Describes how to use Deployment Management and Content Replicator to control and monitor the distribution of content and applications across heterogeneous server platforms and data centers. Deployment Manager extensions to Report Center and Application Packager are also described.
<i>Symphony Marimba Client Automation Device Management Guide</i>	Describes how to use Configuration Management products to manage your mobile devices. This includes Scanner Service to perform inventory scans on your mobile device endpoints; Report Center to run queries against your scanned data; Application Packager, using the PDA Packager, to package and publish files and applications to mobile devices; and Policy Service to define subscription policies for your mobile devices.
<i>Symphony Marimba Client Automation Package Deployment CLI Guide</i>	Provides syntax and usage information about the command-line options used with Content Replicator, Deployment Manager, and Application Packager. Using the SOAP interface feature is also described.
<i>Symphony Marimba Client Automation Patch Management Guide</i>	Helps you configure and administer Patch Management and the Patch Service plug-in. This guide also includes working with the patch repository, patches, patch groups, and custom patches, and deploying patches.
<i>Symphony Marimba Client Automation Policy Management Guide</i>	Helps you configure and administer Policy Management and the Policy Service plug-in. This guide also includes integration procedures for directory services, such as Active Directory, ADAM / AD LDS, and Sun ONE Directory.

Guide	Description
<i>Symphony Marimba Client Automation Reference Guide</i>	Provides reference information, such as command-line options, tuner properties, proxy properties, transmitter properties, channel properties, channel parameters, channel states, ports, and log IDs with associated log messages.
<i>Symphony Marimba Client Automation Report Center Guide</i>	Provides instructions about running queries of inventory information, configuring the Inventory and Logging plug-in, configuring endpoints, and integrating Report Center with other Configuration Management applications.
<i>Symphony Marimba Client Automation Transmitter and Proxy Guide</i>	Provides information about the transmitters and proxy infrastructure components. This guide also describes the tools and features you use to configure these components.
<i>Definitive Software Library Administrator's Guide</i>	Provides a description of the Definitive Software Library and explains how the DSL is useful to you, how to use the DSL console, and how to access the DSL using Configuration Management products, such as Report Center and Application Packager.

Accessing product documentation

You can access Symphony Marimba Client Automation documentation in the following ways:

- Using the documentation channel (page 22)
- Using the Channel Store (page 22)

The most recent information is located on the Customer Support website.

Using the documentation channel

If you are using a Windows platform, you can use Channel Store to subscribe to the Symphony Marimba Client Automation Product Documentation channel.

Using the Channel Store

The Symphony Marimba Client Automation documentation is located on the Marimba Channel Store.

Chapter

1 Introduction

This section explains how to use Application Packager and its components to package software for distribution to desktops or servers.

The following topics are provided:

- What is Application Packager? (page 24)
- Scalability (page 31)
- Supported platforms (page 32)
- Using help (page 34)

What is Application Packager?

You can use Application Packager to package a software application (and subsequent updates) into a format that you can then distribute to target endpoints, such as desktops or servers, using Symphony Marimba Client Automation products. After the packaged application is distributed and installed, the application can be started normally (using the Start menu, shortcuts, and so on) or it can be started from the tuner. A tuner is the interface through which Symphony Marimba Client Automation components interact. To address the variety of software applications you might want to distribute, Application Packager provides the following components:

- Packager for Shrinkwrap Windows Applications*
- Package Editor
- Windows Installer Packager*
- File Packager
- .NET Packager*
- PDA Packager*
- Java Packager
- Custom Application Packager

*Available in Windows only

Application Packager components

Application Packager contains the following packager components:

- **Packager for Shrinkwrap Windows Applications**—To package any commercial 32-bit Windows software application. Packager for Shrinkwrap Windows Applications is suited for repackaging commercial software products that you purchase and install using an installation program. It is useful when the Windows application you want to distribute comes in a self-extracting executable that installs files, registry keys, and shortcuts on the system. It is also useful for capturing system changes, for example, adding or removing a font, or changing the default font or wallpaper for a desktop.

Packager for Shrinkwrap Windows Applications packages an application by monitoring the state (taking a snapshot) of a machine's file system and registry before and after an application is installed on it. Using the snapshots, Packager for Shrinkwrap Windows Applications determines the files and registry entries needed to successfully install the application.

- **Package Editor**—To customize packaged applications. You can use the Package Editor's Package Contents page to add, edit, and remove files, directories, and registry entries in packaged applications. You can also use the Package Editor to set macros, customize environment settings, add installation scripts, and so on.
- **Windows Installer Packager**—To package Microsoft installations (MSI) that were created for the Microsoft Windows Installer. Windows Installer is an installation and configuration service that ships as part of the Microsoft Windows 2000 and later operating systems; it can also be installed in Windows 95/98 and Windows NT 4.0. Microsoft calls MSI installations "MSI packages."

An MSI package consists of at least one file with the .msi extension, which contains an installation database, a summary information stream, and data streams for various parts of the installation. An MSI package can also contain one or more transforms, internal source files, and external source files or cabinet files required by the installation.

Windows Installer Packager is useful if the application's vendor provided it in MSI format, and the application does not require frequent updates or changes. You might also want to use this packager if you do not want to violate Microsoft Logo compliance.

- **File Packager**—To package a collection of files. Unlike the other packagers, File Packager is not designed to package applications; it packages files and directories. Typically, you use File Packager with spreadsheets, HTML files, templates, and other file types that users want to view on their local system. This packager includes a repackaging feature that makes it useful for dynamic content that needs to be updated and refreshed often.
- **.NET Packager**—To package applications that have been created using Microsoft's .NET framework. The .NET framework is Microsoft's platform for building, deploying, and running Web services and applications.

- **PDA Packager**—To package either directories of files or Personal Digital Assistant (PDA) applications in the form of CAB files. Directories of files might include HTML files, ASCII files, or even .exe files. The CAB file format is a format often used for applications that run on mobile devices. The specific supported platform for CAB files is Strong Arm 1100.
- **Java Packager**—To package a Java application. It provides Java Virtual Machine (JVM) versioning and crash protection for the application that it packages. For example, you can run the application using an alternate JVM. Further, you can specify that the application use the same JVM as the tuner, another instance of the tuner’s JVM, or a JVM produced by another manufacturer.
- **Custom Application Packager**—To package any application, especially ones developed by companies for internal use. To use this packager, you need access to the application executable file, as well as knowledge about and access to all files and settings used by the application. This packager is useful when you know exactly which files, registry keys, or services you want to distribute to endpoints. It is also useful when you want to deliver some configuration data, such as a few registry keys, or a text modifier to search and replace some text in a certain file that exists on endpoints.

If you are packaging software for server endpoints, you might want to consider the Symphony Marimba Client Automation Content Replicator in certain situations. For more information, see “Choosing between Content Replicator and File Packager” on page 439.

Software distribution process

Packaging applications is only one step in the software distribution process. Follow the basic workflow described here.

► To distribute software

- 1 Package the software application.

Using the components of Application Packager, you package a software application and customize it using the Package Editor. The result of packaging is a channel that contains all the information necessary to install the application on target endpoints. A channel can be:

- An application of any type (Windows, Java, and so on).
- One or more content files, containing HTML or any data.

- A combination of the two previous bullets.

Using channels has two major benefits. First, they allow updates to be performed easily, efficiently, and even automatically. Second, during updates, using channels means that you do not need to transfer entire files. Channels allow for byte-level differencing, meaning that only the new or changed bytes are downloaded during updates.

2 Publish the channel to a transmitter.

Using Channel Copier, you publish the channel to a transmitter, where it is available to the target endpoints. A transmitter is essentially a server that delivers applications and content in the form of channels. In other words, it is similar to a Web server, but instead of serving Web pages, it serves channels.

3 Subscribe the endpoint tuners to the channel.

You can choose from a variety of methods to subscribe endpoint tuners to the channel:

- You can use the tuner to subscribe directly to the channel.
- You can use Policy Manager or Deployment Manager from the Symphony Marimba Client Automation console to subscribe groups of endpoints to channels based on various factors.

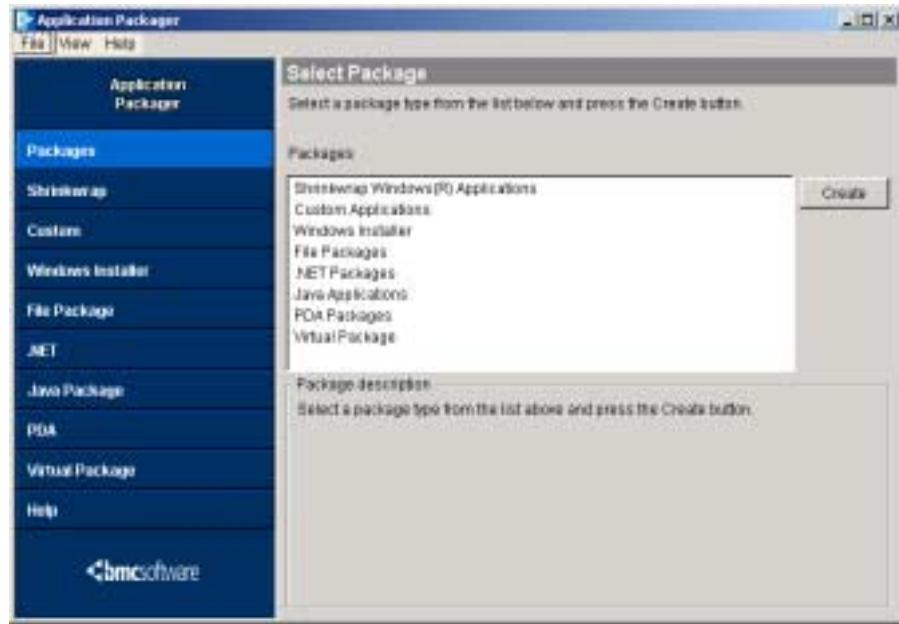
After you subscribe the endpoint tuners to the channel, you run the channel to installs the software application on the endpoints.

Application Packager interface

You can use Application Packager through the graphical user interface (GUI) or from the command-line interface.

When you start Application Packager from the GUI, it opens to the Select Package page. From this page, you can read a description of the packaging components that are available and begin packaging applications. The platform on which you run Application Packager determines which packaging components are available.

Figure 1-1: Select Package page



If you have an existing packaged application, you can click the appropriate packager from the button bar on the left, select the packaged application, and select a command at the bottom of the screen.

You can also use Application Packager to import existing packaged applications if they were created using one of the packaging tools (perhaps by a co-worker).

Package directory

After you package an application, the result is a package directory. The package directory is the directory in which Application Packager stores the files for a packaged application. A package directory is created when you use any of the packager components of Application Packager to package a software application. When you edit and publish a packaged application, you are actually editing and publishing the contents of the package directory. The package directory is also sometimes referred to as the channel directory or publish directory.

Usually, you do not need to manually edit the contents of the package directory; Application Packager edits the contents when you use the GUI or command line to edit the packaged application. However, it is useful to know what the package directory contains.

Figure 1-2: Contents of a package directory



Typically, the package directory contains the following files and directories:

- **parameters.txt** file—The file where Application Packager stores information for installing and launching a packaged application. The information in this file is used by the tuner when you run a channel to install and launch the packaged application. For a list of properties that you can use in a channel's **parameters.txt** file, see the *Symphony Marimba Client Automation Reference Guide* on the Channel Store.
- **properties.txt** file—The file where Application Packager stores information for publishing and running a channel. This file contains all the information necessary for publishing the channel, as well as instructions for the tuner on how to launch the channel. For a list of properties that you can use in a channel's **properties.txt** file, see the *Symphony Marimba Client Automation Reference Guide*.
- **files** directory—This directory contains the files that make up the packaged application. The files in this directory are stored by their checksum values instead of their real name.
- **plugin** directory—This directory contains information about any plug-ins included with the channel. These plug-ins are usually designed to work with the transmitter to which the channel is published.

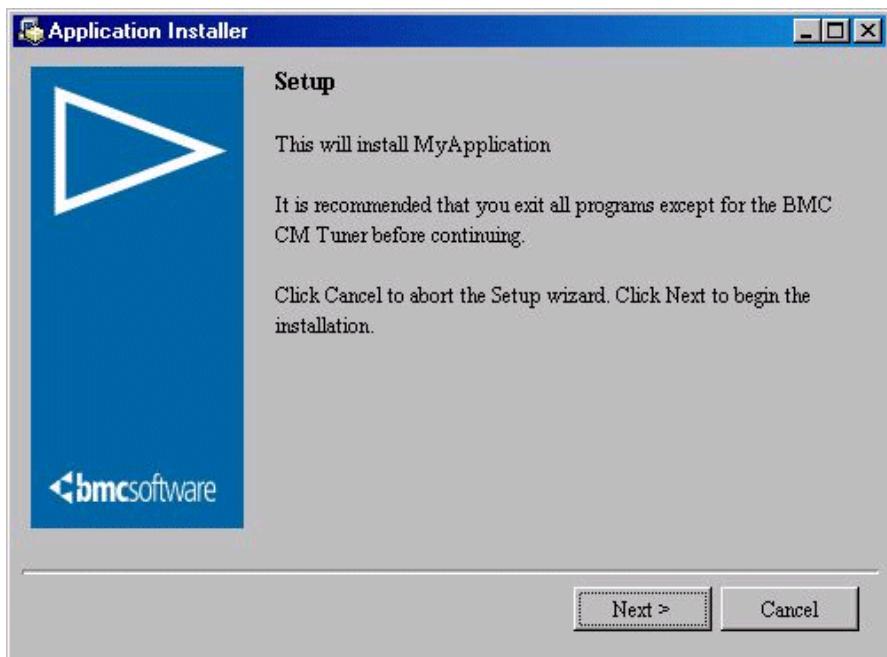
- signed directory—Each application that is packaged with Application Packager has its own signed directory. If the packaged application is signed with a code-signing certificate, the contents of this directory are used as the basis for a channel-signing signature that is used to guarantee the integrity of the software application. The signed directory also contains the .ncp directory, which contains the manifest file (manifest.ncp). The manifest file contains the code and instructions required to install and update the application.

The package directory created by the PDA Packager is somewhat different. For more information about the package directory for PDA packages, see the *Device Management Administrator's Guide*.

Application Installer

Application Installer installs the packaged application on an endpoint. If you have configured a channel to install in full GUI mode, Application Installer usually appears when a user at the endpoint subscribes to and runs, for the first time, a channel created using Application Packager.

Figure 1-3: Application Installer window



You can also configure Application Installer to run without interacting with the user:

- In silent mode, Application Installer does not display any dialog boxes or progress bars. It uses the default channel settings.
- In semi-silent mode, Application Installer displays a progress bar only. It uses the default channel settings.

For more information about configuring a channel to use silent and semi-silent modes, see “Setting installation modes” on page 106.

Scalability

The following table describes the outer limits of scalability for Application Packager.

Table 1-1: Application Packager scalability

Benchmark	Limit
Maximum number of files you can package	300,000
Maximum number of files you can publish (in a single directory)	65,000
Maximum file size you can package	34 GB
Maximum channel size	10 GB
Maximum filename length	255 characters
Maximum number of subdirectories	50,000 File sizes can decrease this number.
Maximum number of files and subdirectories under one directory	100,000
Maximum directory depth	120 levels (if all directory names have one character) The complete path (including drive name and directory separators) cannot exceed 246 characters in Windows. The same limit is 255 characters in Unix.

Supported platforms

You can use Application Packager to package software for distribution on both Windows and UNIX platforms. Remember that you must package software on the platforms to which you plan to distribute the software. That is, your packaging platforms must match the platforms of your endpoints.

This section provides general information for packaging on various platforms. For detailed information about the operating systems and versions supported for Application Packager, see the release notes, available on the Marimba Channel Store.

Windows platform support

The following packager components are available in Windows:

- Packager for Shrinkwrap Windows Applications
- Windows Installer Packager
- Custom Application Packager
- File Packager
- .NET Packager
- Java Packager
- Virtual Packager
- PDA Packager

Note: Windows 2008 Core has minimum user interface support. Package creation using the Application Packager user interface is not recommended on this OS. Instead use the Application Packager CLI commands to create all supported packages.

The Windows Terminal Services (WTS) product provides clients remote access to applications that run on a server. Applications are installed on the server only, and clients access the applications through WTS client software. WTS transmits only the user interface of the application to the client. The client then returns keyboard and mouse clicks to be processed by the server. Most of the processing and resources are used on the server, and very few client resources are necessary. Each user logs on and sees only that individual session, which is managed transparently by the server operating system and is independent of any other client session. WTS servers require one of these:

- Microsoft Windows NT® Server 4.0, Terminal Server Edition (TSE), an extension to the Windows NT Server product line that delivers the Windows operating system experience through terminal emulation.
- Microsoft Windows 2000 Server, which includes Terminal Services as a component and provides remote computers access to Windows-based programs running on the server through terminal emulation.

Sections in this book that apply only to Windows Terminal Services machines are marked “WTS only.”

UNIX platform support

The following packager components are available in UNIX:

- Custom Application Packager
- File Packager
- Java Packager

In UNIX, the Custom Application Packager and File Packager products have similar functionality, but they differ in presentation. When you use Custom Application Packager, you create an empty package and then use the Package Editor to add directories and files. When you use File Packager, you specify the directories and files you want to include in the package when creating the package.

If you are packaging from a UNIX tar file, cpio archive, or SVr4 package, see the instructions in “Preparing to package applications in UNIX” on page 61.

When you are packaging for UNIX platforms, the following sections are useful:

- “Editing file properties” on page 73
- “Editing directory properties” on page 82

- “Customizing a channel by using scripts and classes” on page 141
- “Using macros” on page 95
- “Setting policies for installation, update, uninstallation, verification, and repair” on page 121
- “Editing ASCII text files” on page 173
- “Verifying and repairing channels” on page 165

If you are packaging software for server endpoints, you might also want to consider using the Symphony Marimba Client Automation Content Replicator in certain situations. For more information, see “Choosing between Content Replicator and File Packager” on page 439.

Reference information

This book contains the following reference information.

Reference information	See
Command-line options you can use to package applications, as an alternative to using the Application Packager GUI.	“Command-line reference” on page 307
Policies you can set for the entire channel, as well as policies that apply to specific items (such as directories, files, registry entries, metabase entries, and services) in a channel.	“Policies for installation, update, uninstallation, verification, and repair” on page 331
Syntax for the XML tags you can use when packaging applications.	“XML syntax” on page 369
Macros that are available in Windows.	“Windows system macros” on page 433

Using help

The first time you click the Help link in an application, you see a security warning. It prompts you to install and run an applet that has been code-signed by Quadralay Corporation. This applet provides the dynamic table of contents and a topic search feature for help. If you do not install the applet, the help system still works, but some functionality is not available.

Help is distributed with the application you are using and is updated automatically when the product channel is updated. Because help is part of a channel, the help files are stored on your workstation and are always available, even in the absence of a network connection.

Help is context-sensitive. When you are on a particular page of the browser-based interface for an application and you click the Help link in the upper-right corner of the page, the help that appears corresponds to that page of the interface. Some topics contain a hyperlinked list of related topics at the end of the topic (indicated by a “See Also” heading).

To look for help topics, you have the following options:

- **Contents button**—Click the Contents button in the help window to see the table of contents for all the help topics.
- **Index button**—Click the Index button in the help window and scroll through the alphabetical list of entries to find the entry you want. Click the index entry to display the corresponding help topic.
- **Search button**—Click the Search button, type one or more words in the text box, and click Go. The page contains a list of all the topics in the help system that contain the word or phrase you entered. If you entered more than one word, the search finds help topics that contain all the words you entered. The topics found by searching are ranked in order of relevance.
- **Returning to a previous help topic**—To go back to a previous help topic, use standard navigation shortcuts. For some browsers, you can press your keyboard’s Backspace key to go back, or you can right-click and choose Back. In the help window, there is no Back button.

2 Using Packager for Shrinkwrap Windows Applications

This section shows you how to use the Packager for Shrinkwrap Windows Applications to package any commercial 32-bit Windows software application. This packager is ideally suited for repackaging commercial software products that you purchase and install using an installation program.

The following topics are provided:

- Overview (page 38)
- Packaging shrink-wrapped Windows applications (page 38)
- Setting up the packaging environment (page 39)
- Creating a preinstall snapshot (page 39)
- Preparing to install the application (page 50)
- Installing the application and creating a postinstall snapshot (page 50)
- Packaging the application into a channel (page 51)
- Managing updates to the application (page 52)
- Taking over an application that already exists on the endpoints (page 54)
- Removing all files and registry keys associated with an application (page 55)
- Packaging the removal of an application (page 55)
- Working with Windows File Protection (page 57)

Overview

Packager for Shrinkwrap Windows Applications packages an application by monitoring the state (taking a snapshot) of a machine's file system and registry before and after an application is installed on it. Using the snapshots, the packager determines the files and registry entries needed to successfully install the application. This packager is available in Windows only.

Note: You cannot use Packager for Shrinkwrap Windows Applications to directly distribute Windows service packs or operating system patches.

You can, however, create a channel using this packager that contains the setup.exe patch installer. After the channel is installed, you can run the file manually or you can use the Package Editor to configure the channel to run the patch automatically when the channel starts.

This section contains instructions for using Packager for Shrinkwrap Windows Applications through the GUI. You can also use the command-line interface. For more information, see “Packager for Shrinkwrap Windows Applications command-line options” on page 309website.

Packaging shrink-wrapped Windows applications

Follow the basic workflow described here to package a shrink-wrapped Windows application into a channel.

- ▶ **To package a shrink-wrapped Windows application into a channel**
 - 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
 - 2 Select Packager for Shrinkwrap Windows Applications.
 - 3 To create a preinstall snapshot, perform the following steps:
 - a Select the file systems to monitor.
 - b Select the registry entries to monitor.
 - 4 Install the application.
 - 5 Create a postinstall snapshot.
 - 6 Package the files and settings into a channel.

- 7 (Optional) Use the Package Editor to review or edit the channel.
- 8 Use the channel publishing wizard to publish the channel to a transmitter.

Setting up the packaging environment

This section describes how to set up an environment in which you can run Application Packager and accurately capture an application installation for deployment on your target endpoints.

Capturing the changes made during an application installation depends on how clean the packaging machine is and how closely it matches the target endpoints to which the application will be deployed. Ideally, you create your snapshots on a clean machine (that is, a machine with the operating system and little else installed). Using a clean machine produces quick snapshots that are easy to set up and less prone to error. After each channel is created, you could uninstall the application, returning the packaging machine to its previous state. The cleaner the machine you use for packaging, the more independent the channels you package on that machine.

Symphony Teleca recommends that you create a packaging machine with two operating systems, and use one of them as a backup. You can then use the backup operating system to copy over the one you used for packaging applications.

Note: You must package software on a WTS machine if you are going to deploy it to WTS machines.

Creating a preinstall snapshot

The first stage in creating a new channel from an application is to create the preinstall snapshot. The packager compares the state of a machine before and after installing an application to determine how to create the channel for the application. The two most important steps in creating a preinstall snapshot are described in “Selecting file systems” on page 41 and “Selecting registry entries” on page 45.

During the preinstall snapshot, the packager inspects all the files and registry entries in the areas you specify and stores the information for a postinstall comparison.

Ideally, you create your snapshots on a clean machine (that is, a machine with the operating system and little else installed). Using a clean machine produces quick snapshots that are easy to set up and less prone to error. After each channel is created, you could uninstall the application, returning the machine to its previous state. If you do not have a machine to dedicate to this purpose, it is not a problem; you can create your snapshots on any machine running an operating system that matches that of your target endpoints.

Selecting a snapshot source

In most cases, you probably want to create a new snapshot to create a new channel. Alternatively, you can use an existing snapshot file (snapshot files have the .msp extension) instead of creating a new one.

► To select the source of your preinstall snapshot

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 From the list of package types in Application Packager, select Shrinkwrap Windows Applications and click Create.

The packager appears, opening to an introduction page.
- 3 To proceed to the Preinstall Snapshot page, click Next twice.
- 4 If you want the packager to inspect the file systems and registries that you can specify later, perform the following steps:
 - a Select “Generate a new preinstall snapshot now.”
 - b Click Next.
 - c On the Generate Preinstall Snapshot page, load any configuration and user-defined filters you want to use when taking the snapshot.
 - d Click Next.
 - e On the File System Selector page (and the following pages), identify the file systems and registries to use for the new snapshot. For more information, see “Selecting file systems” on page 41.
- 5 To use a snapshot file you created earlier, perform the following steps:
 - a Select Load the preinstall snapshot from disk.
 - b Click Next.

- c On the Load Preinstall Snapshot page, select the snapshot file to use.
Snapshot files use the .msp extension.

Selecting file systems

The next step in creating a preinstall snapshot is selecting the drives and directories that the packager monitors.

You can also load an XML template file that contains the configuration and filters you want the packager to use. For more information, see “Loading XML template files for configuration and filters” on page 272.

You can also exclude all file systems from the snapshot you are creating.

► To exclude all file systems from your snapshot

- On the File System Selector page, clear the Include file system in snapshot check box.

Adding drives and directories to the snapshot

You can select the drives and directories that are monitored when the packager creates the preinstall snapshot.

► To add drives and directories to a snapshot

- 1 On the File System Selector page, click Add.
- 2 To add an entire drive to your snapshot, specify a drive. For example, to include everything on drive E, type E:.
- 3 To add a directory to your snapshot, specify a directory. For example, to include the Program Files directory on drive C, type C:\Program Files.
- 4 If you want the snapshots to include the removal of files and directories, select the “Capture removal of files and directories” check box.

Removed files and directories appear on the Package Contents page of the Package Editor with an X over them. For more information, see “Packaging the removal of an application” on page 55.

When the snapshot is taken, the drives and directories you added are included in the snapshot.

Removing drives and directories from the snapshot

You can remove drives and directories that appear in the file system list. If you remove a drive or directory, it is excluded during the snapshot process.

► To remove drives and directories from a snapshot

- 1 To remove drive and directories, perform one of the following steps:

-On the File System Selector page, select the drive or directory you want to remove, and click Remove.

-To remove all drive or directories from the list, click Remove All.

When the snapshot is taken, the drives and directories you removed are excluded from the snapshot.

Ignoring file system items during a snapshot

You can use Packager for Shrinkwrap Windows Applications to exclude (ignore) directories, files, and other items from your snapshots. For example, you might want to take a snapshot of your entire C drive, except for a few large directories that you know are unrelated to the application channel you are creating.

Building a filter list (list of items to ignore)

You can specify a file or directory that you want to ignore, or you can set up filters to build a list of exclusions for you. You can use a filter to specify the criteria, and then it searches for the files or directories that satisfy your criteria. For example, you might want to exclude all directories containing the word *finance* in its label. The filter would exclude directories labeled finance, my_finance, 98finance, and so on.

Whether you want to set up a filter or identify an item directly, you use the same process.

► To add a file or directory to your filters

- 1 On the File System Selector page, click Filters.

If the Filters button is not active, make sure that “Include file system in snapshot” is selected.

- 2 Click Add.

- 3 For Filter Description, specify a name that describes the items you want to ignore.

The name is optional, but it can provide a more descriptive title than the one the packager creates for you if you leave the text box blank.

- 4 Specify the criteria for the filter using the Ignore boxes and drop-down lists. You can specify the following information for a filter:

Type—Select the type of item (directory, file, or key) for which you are creating a filter.

Noun—Select the type of information (name, parent, or path) you are using to specify a filter.

Verb—Select the comparison (is, isn't, contains, doesn't contain, begins with, and ends with) you want to make when using the filter.

Adverb—Specify the information you are using to identify a filter, for example, a directory or file name. You can also use macros; for more information, see “Macros you can use in filters” on page 44. You can also browse the local machine to specify a particular name, parent, or path.

Enabled—Specify whether to enable the filter when taking the snapshot.

For example, to exclude all directories with the name `temp` (no matter what the parent directory is), the Add File System Filter Entry dialog box might look like the following illustration:

Figure 2-1: Add File System Filter Entry dialog box



To exclude a specific directory or file, you can choose to use a path instead of a name. Perform one of the following steps:

- To broaden the criteria so that the filter finds multiple directories, experiment with the drop-down list options.

-To narrow the scope of your filter even more, click More and add another layer of criteria that must be satisfied. To remove a layer of criteria, click Fewer.

5 Click OK.

Your filter appears in the Ignore list and is enabled.

Macros you can use in filters

This section lists the macros you can use in filters. For more information about these macros, see “Windows system macros” on page 433website.

Table 2-1: Macros used in filters

Macro type	Description
System Macros	\$SYS.BOOT\$ \$SYS.COMPUTER\$ \$SYS.IP\$ \$SYS.SYSTEM\$
User Profile Macros	\$SYS.DESKTOP\$ \$SYS.PROGRAMS\$ \$SYS.STARTUP\$ \$SYS.STARTMENU\$
System Profile Macros	\$SYS.COMMONDESKTOP\$ \$SYS.COMMONPROGRAMS\$ \$SYS.COMMONSTARTUP\$ \$SYS.COMMONSTARTMENU\$

Editing the filter list

After you have created a filter, you can edit it or remove it from the list.

► To edit the filter

- 1 Double-click the filter name in the Ignore list.
- 2 Make your changes on the page that appears.

► To remove a filter

- To remove filters, perform one of the following steps:
 - From the Ignore list, select the filter you want to remove, and click Remove.

-To remove all filters from the list, click Remove All.

Editing the default filter settings

Packager for Shrinkwrap Windows Applications is set to ignore a large set of files and directories for you automatically. From the list of items that are ignored by default, you can disable a filter so that an item is included in your snapshot. You cannot change the filters that appear in the default filter list, but you can choose whether a filter is enabled or disabled.

The default filter list is different for the various Windows operating systems. The packager might need to ignore different directories and registry entries depending on the specific version of Windows on which you are creating snapshots.

► To edit the default filter settings

- 1 On the File System Selector page, click Filters.

If the Filters button is not active, make sure that Include file system in snapshot is checked.

- 2 Click Defaults.
- 3 Clear the Enabled check boxes for any items you do not want to ignore when the snapshot is taken.
- 4 Click OK.

The default filters do not appear in the Ignore list.

- 5 To see the current default ignore settings, return to the File System Ignore - Default Ignore page.

Selecting registry entries

Selecting registries to include and exclude from your snapshot is similar to selecting file systems. You can add, remove, and ignore registry entries using the same techniques that are described in “Selecting file systems” on page 41. On the File System Selector page, click Next to get to the Registry Selector page.

Editing the Windows registry

Installing a 32-bit Windows application involves changes to the local file system and (usually) the Windows registry. The registry is a repository for configuration information about the hardware, system software, and applications running on a 32-bit Windows machine.

The structure, contents, and files associated with Windows registries vary according to the version of the operating system you are running. As a result, you need a different version of your channel for each operating system (Windows 95/98, Windows NT 4, Windows 2000, Windows XP) that you intend to support. Also, create snapshots on the same operating system as your target endpoints.

Normally, the packager manages all the registry settings in the channels it creates for you. Its before-and-after snapshot process captures all the changes to the registry. For advanced users, the packager provides an editor that you can use to edit the registry changes it captures and change them before they are packaged into the channel.

You can also use the Package Editor to edit the file list, create installation macros, and set system variables for your channel. For more information, see “Using the Package Editor” on page 67.

It is assumed that you already understand the registry. It is beyond the scope of this document to teach registry fundamentals. See Microsoft’s website and publications for information about the registry.

Customizing snapshots

After selecting the registries to include, click Next to get to the Snapshot Customization Options page. Use this page to select the options you want to use while taking snapshots to package an application. The following options are available:

- **Enable Microsoft IIS Metabase Support**—Select this check box to capture any changes made to Microsoft’s Internet Information Server (IIS) metabase when installing an application. Select this check box if you are packaging a server application that interacts with the IIS metabase. If this check box is not available, the machine you are using for packaging does not have a supported version of IIS installed.

Note: IIS is a group of internet servers that includes programs for building and administering websites, a search engine, and support for developing Web-based applications.

- **Enable parsing INI files (KEY, VALUE pairs)**—Select this check box if you want INI files to be treated as special files. The packager parses the INI files and captures any changes made to the key and value pairs in them. Any changes found by the packager are merged with the existing file found at the endpoint. By default, this check box is selected. If you want the packager to process INI files as ordinary files and not merge changes within the files, clear this check box. For more information, see “Working with INI files” on page 78.
- **Enable processing of .NET features**—Select this check box to enable .NET-specific features, such as parsing of .NET assembly policy configuration files.

In addition, you can also save XML template files that contain the configuration and filters you want the packager to use. For more information, see “Loading XML template files for configuration and filters” on page 272.

Selecting metabase entries

Applications built specifically for use in Windows server environments require and interact closely with Microsoft’s Internet Information Server (IIS). IIS stores information in a metabase, which resembles the registry but is more sophisticated and has optimized information retrieval capabilities. Like the registry, the IIS metabase contains key nodes and value nodes. When you install a server application, the application makes changes to the IIS metabase by adding or editing key nodes and value nodes.

You can use the packager to include, exclude, and ignore metabase entries, such as key nodes and value nodes, in your snapshots. Include metabase entries in your snapshots if you are packaging a server application that interacts with and makes changes to IIS. The packager captures any changes that the server application makes to the metabase so that it can be distributed to other machines.

► **To enable capturing IIS metabase changes in your snapshot**

- 1 On the Snapshot Customization Options page, select the Enable Microsoft IIS Metabase Support check box.

Note: If this check box is not available, the machine you are using for packaging does not have a supported version of IIS installed.

- 2 Click Next.

The Metabase Selector page appears. You can use this page to select the metabase entries to include, exclude, or ignore in your snapshot. By default, the \LM and \SCHEMA keys are included in your snapshot.

► **To add a metabase key to a snapshot**

- 1 On the Metabase Selector page, click Add.
- 2 In the text box, specify the name and path of the metabase key you want to add.
- 3 Click OK.

The metabase key you added appears on the Metabase Selector page.

► **To remove a metabase key from a snapshot list**

- To remove metabase keys, perform one of the following steps:
 - On the Metabase Selector page, select the metabase key you want to remove, and click Remove.
 - To remove all metabase keys from the list, click Remove All.

► **To exclude a metabase key from the snapshot**

- 1 On the Metabase Selector page, click Filters.

The Metabase Selector - Ignore page appears.

- 2 Add to the list any metabase entries you want to ignore.

You can specify the name, parent, path, or a combination of these three to filter metabase entries. No metabase entries are ignored by default. This is similar to selecting file system items to ignore. For more information, see “Ignoring file system items during a snapshot” on page 42.

-
- 3 Click OK.

Generating the preinstall snapshot and saving the snapshot file

After selecting the file, directories, registry entries, and metabase entries that are to be included and excluded from your snapshot, you are ready to generate the preinstall snapshot. The snapshot can be saved to a file or just kept in memory to be compared to the postinstall snapshot. Saving the snapshot file takes a little more time and disk space, but you can use the snapshot at a later time.

You should save the snapshot file if the application requires restarting the machine after installation. After installing the application and restarting the machine, you can load the preinstall snapshot in the packager.

Note: If you are taking preinstall and postinstall snapshots of an application that is being installed on one disk drive, save the preinstall (and postinstall) snapshots on another drive, so that you do not need to worry about filtering the saved snapshot file. For example, if you are installing the application on the C drive, save the preinstall snapshot file on the D drive.

► To generate the preinstall snapshot and save the snapshot as a file

- 1 On the Registry Selector or Metabase Selector page (depending on whether the machine has a metabase installed), click Next.
- 2 To start taking the preinstall snapshot, click Generate.
- 3 When Snapshot complete appears, click Next.
- 4 Click Save.
- 5 In the dialog box that appears, specify a name and path for the snapshot file. Do not change the default file extension (.msp).

The Postinstall Snapshot page appears.

Preparing to install the application

Between the preinstall snapshot and the postinstall snapshot, you must install your application on the machine on which you are running Packager for Shrinkwrap Windows Applications. After the installation, the packager takes a postinstall snapshot, compares the two snapshots, and determines exactly what is required to install your application.

Some applications restart the machine after an installation. If the application you are packaging restarts the machine after installation, be sure to save your preinstall snapshot before exiting Application Packager and starting the installation. After the installation is complete, restart Application Packager and Packager for Shrinkwrap Windows Applications. Then load the preinstall snapshot as described in “Selecting a snapshot source” on page 40.

Installing the application and creating a postinstall snapshot

After taking the preinstall snapshot and preparing the installation, you must install the application you want to package. Then you must take a postinstall snapshot of the machine. Normally, take a snapshot using exactly the same file system and registry settings you used during the preinstall snapshot.

You can save your postinstall snapshot as a file and use it at a later time to create the channel.

► To install the application and take the postinstall snapshot

- 1 On the Postinstall Snapshot page, confirm that the specified preinstall snapshot is correct.
 - You probably want to install the application and then generate a postinstall snapshot. This procedure is described in the following steps.
 - However, to use a snapshot file that you created earlier, click Load postinstall snapshot from disk. Then click Next and select the post-snapshot file. You can then skip to “Packaging the application into a channel” on page 51.
- 2 To create a postinstall snapshot, select Install application and generate a postinstall snapshot.
- 3 Click Next.

The Install Application page appears.

- 4 Install your application.
- 5 Click Next.
- 6 On the Postinstall Snapshot page, select one of the following options:

-Edit the settings for the postinstall snapshot by selecting Modify postinstall snapshot settings, and click Next.

Usually, you do not want to edit these settings—you want to use the same snapshot settings that was used for the preinstall. If you do edit the postinstall snapshot settings, you must select files, directories, registry entries, and metabase entries that you want to include in the snapshot, just as you did for the preinstall snapshot. For more information, see “Selecting file systems” on page 41, “Selecting registry entries” on page 45, and “Selecting metabase entries” on page 47.

-To use the same snapshot settings that were used for the preinstall, click Next.

The Application Name page appears.

Packaging the application into a channel

After completing the preinstall and postinstall snapshots, the packager knows what file systems and registry entries are needed to create the channel. Now provide a name for the channel and specify a directory in which to save the channel.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► To package the application into a channel

- 1 On the Application Name page, specify a name for the channel you are creating.

The name you select is used when creating and publishing the channel. It also appears in the channel list when you browse for a transmitter.

- 2 Click Next.

- 3 On the Channel Directory page, specify the path to identify the directory where you want to save the packaged application. For more information, see “Package directory” on page 28.

If the directory you specify does not exist, the packager creates it for you.

- 4 Click Next.

- 5 Click Generate.

The packager compares the snapshots and creates the channel in the directory you specified.

Now that the channel has been created, you can publish the channel, as described in “Publishing channels” on page 279. Or you can first edit the channel and make changes before you publish it, as described in “Using the Package Editor” on page 67.

Managing updates to the application

When a shrink-wrapped Windows application that you have packaged is updated by the software developer, you can provide an update to the endpoints where that application is installed. For example, if the commercial word-processing application used throughout your enterprise is updated from version 7.0 to version 8.0, you could make version 8.0 available as an update to the channel you previously distributed.

When you publish a channel update to the transmitter, the update overwrites the previous version of the channel on the transmitter. Before you package and publish a new version of the channel to the same channel URL, it is recommended that you create a backup copy of a previous version of a channel. For more information about using Channel Copier to create a backup copy of the channel on the transmitter, see “Creating a backup copy of a published channel” on page 283.

► To package and publish an update channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Start Packager for Shrinkwrap Windows Applications on a clean machine (that is, a machine with the operating system and little else installed).
- 3 If the previous version of the application that you are packaging is installed on your machine, uninstall it.

- 4 Create a preinstall snapshot. For more information, see “Creating a preinstall snapshot” on page 39.
- 5 Prepare to install the new version of the application:
 - If the update amends the installation for the previous version, install the previous version before running the update.
 - If the update does not depend on the files in the previous version, install the updated version directly.
- 6 Install the application and create a postinstall snapshot. For more information, see “Installing the application and creating a postinstall snapshot” on page 50.
- 7 Publish the channel to the same URL as the original version of the channel. If you previously included the version number in the channel title, you can change it at publish time.

Note: You should create a backup copy of the channel before you publish a new version to the same channel URL. If you keep a backup copy, you can deploy and re-install the previous version if necessary.

What happens during a channel update

During an update, Application Packager always tries to make the channel on the endpoint match what was packaged (what you would see if you opened the channel in the Package Editor). Example of a simple update:

- The endpoint has version 1 of the package. Version 1 contains files A and B.
- You create version 2 of the package. Version 2 contains file C.

When you deploy version 2 of the channel to the endpoints, the endpoints ultimately have the files included in version 2. During the update, Application Packager calculates the difference between version 1 and version 2. Then it executes the following set of instructions (generated from the difference between version 1 and version 2):

- Delete file A.
- Delete file B.
- Add file C.

Taking over an application that already exists on the endpoints

If you already have an application installed on your endpoints, but you want to manage the application using Symphony Marimba Client Automation products, you can do so.

Use Packager for Shrinkwrap Windows Applications to replicate the state of an endpoint after installation. The shrinkwrap functionality uses preinstallation and postinstallation snapshots to package an application, thereby propagating file and registry key paths that match the needed state on the endpoint. When you package an application already present on an endpoint, you must install the same version of the existing application to take it over for Configuration Management.

► To take over an application that already exists on your endpoints

- 1 In Packager for Shrinkwrap Applications, set the global policies for the channel using the Configuration – Policies tab. Set the Install policies as you normally would for a first-time installation. Usually, select Install if object is newer – compare objects based on versions.
- 2 For the Uninstallation policies, select Always remove existing object.
- 3 For the Verify Installed policies, select:
 - Verify contents of the object – If contents differ, allow Windows versions to increase
 - Verify the attributes of the object
 - Verify the existence of the object
- 4 For the Verify Not Installed policies, select Verify the existence of the object.
- 5 For the Repair policies, select:
 - Repair the contents of the object – Do not repair content if versions increased
 - Repair the attributes of the object
- 6 On the Configuration – Installer tab, select the Don't back up any objects check box. When you selection this option, you save space by ensuring that no backups are made for existing files on the endpoints.

- 7 Locate the parameters.txt file in the package directory, and add the property preload=true. This property decreases the bandwidth used when downloading the channel to endpoints.

Note: You can also edit the parameters.txt file through the graphical interface. For more information, see “To edit channel properties and parameters” on page 159.

- 8 Deploy the channel to the endpoints.

The application is now under the control of Application Packager. If you later uninstall the channel, the application is uninstalled and no longer exists on the endpoints.

Removing all files and registry keys associated with an application

Follow the steps in “Taking over an application that already exists on the endpoints” on page 54 and then remove or uninstall the channel from your endpoints.

Packaging the removal of an application

Although you usually use Application Packager to install applications on endpoints, you might also want to use it to remove applications from endpoints. You can use the packager to capture the removal of the directories, files, registry entries, and metabase entries that make up the application.

The process of packaging the removal of an application is very similar to that of packaging an installation. The difference is that you install the application before you take the preinstall snapshot and you uninstall the application before you take the postinstall snapshot. If you edit the channel using the Package Editor, items marked for removal appear on the Package Contents page with an X over them.

Note: In Windows, Application Packager currently does not support capturing the removal of environment variables. However, Application Packager does remove environment variables only if the name, value, and case of the environment variable (with the captured removal) in the package matches exactly an environment variable on the endpoint. For example, if the environment variable is “TEST=testvalue” in both the package and the endpoint, Application Packager removes it from the endpoint. However, if the environment variable is “TEST=testvalue” in the package and “Test=testvalue” on the endpoint, Application Packager does not remove it from the endpoint.

► To package the removal of an application

- 1 Install the application you want to remove from the endpoints.
- 2 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 3 Select Packager for Shrinkwrap Windows Applications.
- 4 To create a preinstall snapshot, perform the following steps:
 - Select the file systems to monitor. Be sure to select the Capture removal of files and directories check box.
 - Select the registry entries to monitor. Be sure to select the Capture removal of registry entries check box.
- 5 Uninstall the application.
- 6 Create a postinstall snapshot.
- 7 Package the files and settings into a channel.
- 8 (Optional) Use the Package Editor to review or edit the channel.
- 9 Use the channel publishing wizard to publish the channel to a transmitter.

When the channel (containing the items to be removed) is installed at the endpoint, the specified items are uninstalled. Depending on how the channel is configured, the uninstalled items might be stored in a backup directory (the `unfiles` directory of the channel in the tuner’s workspace directory), so that they can be restored if you uninstall the channel. For more information, see also “Preventing items from being stored for backup” on page 109.

Working with Windows File Protection

Windows File Protection (WFP) is a feature introduced by Microsoft for the Windows 2000 product family. This feature is a way for Microsoft to protect system files to make sure that only the owners and vendors of those files can edit them. If files under WFP are overwritten by anyone else, they are restored to the last certified version kept in the DLL cache.

By default, the files under WFP are the system files included in WINNT\SYSTEM32. These include core kernel DLLs and libraries generally touched only by operating system service packs or deeply integrated applications such as Internet Explorer.

WFP does not allow Symphony Marimba Client Automation CM applications, such as Application Packager, to overwrite the protected files. Thus, when it comes to distributing applications that make changes to these protected files, Application Packager must keep the original installation package intact. As a result, you cannot use Packager for Shrinkwrap Windows Applications to package these types of applications. If the original package is in a file with the .msi extension, use Windows Installer Packager. If it is in a file with the .exe extension, follow the vendor's instructions, use Custom Application Packager or File Packager to create the channel, and then start the executable using a postinstall script.

Installing Office 2007 using the Packager for Shrinkwrap Windows Application

The following procedure describes how to use Packager for Shrinkwrap Windows Applications to package and install Office 2007.

Before you begin

You can configure Office 2007 installation options such as, silent installation, reboot control, installation directory, etc., in the config.xml file in the <PackageID>.WW folder. <PackageID> can be Enterprise, Prop, etc. For more information on editing the config.xml file, see the following Microsoft TechNet article: <http://technet2.microsoft.com/Office/en-us/library/41f07f9b-f0d0-489d-a185-d7b96f21f5611033.mspx>

► **To install Office 2007 using the Packager for Shrinkwrap Windows Applications**

- 1 Create a pre-install snapshot using the Packager for Shrinkwrap Windows Applications.
- 2 Modify the config.xml file if needed.
- 3 Install Office 2007.
- 4 Create a post-install snapshot.
- 5 Generate the difference between the pre- and post-install snapshot.
- 6 Configure reboot options in the Package Editor if needed.
- 7 Publish the package as a channel.
- 8 Install Office 2007 from the transmitter.

Chapter

3 Getting started

This section shows you how to start the Application Packager and introduces you to the Application Packager interface.

The following topics are provided:

- Before you get started (page 60)
- Starting Application Packager (page 62)
- Working with channels (page 62)

Before you get started

Distributing software works best when the machine on which you are packaging software closely matches the endpoint machine on which you will install the packaged software. Depending on the variety and types of endpoints to which you will be distributing software, you might need more than one packaging machine. To distribute software to different platforms, you must have a packaging machine for each of those platforms.

Application Packager is a flexible tool, but there are a few restrictions and recommendations that might make you more successful:

- On the machine you will use to package software, install a tuner (including Channel Manager), and subscribe to Application Packager and Channel Copier. Optionally, to sign packages using certificates, install Certificate Manager.
- If you package MSI packages, you also need to have Microsoft Windows Installer on the packaging machine.
- Depending on the application you are packaging, you might need to package separate channels for Windows 95 and 98, Windows NT, Windows 2000, and Windows XP. Software applications that interact more closely with the operating system, such as Internet Explorer, are more likely to require packaging as separate channels.
- Symphony Teleca recommends that you package applications on a machine that has only one operating system installed. You might encounter problems if you package applications that have two or more operating systems installed, even if they are on different partitions, for example, Windows on one partition and Linux on another.
- To guarantee that channels have not been tampered with or corrupted before they arrive at the target endpoints, you can use a channel-signing certificate. To sign a channel, you need a channel-signing certificate from a certificate authority (such as VeriSign) when you publish the channel. If you do not have a certificate, you can use Certificate Manager to request and install one.
- For packaging most applications, install the tuner and run it as an application. For packaging device drivers, install and run the tuner as a service.
- When naming packages, use alphanumeric characters only. Using special characters in a package name might result in problems with packages.

Preparing to package applications in UNIX

First, analyze and extract the contents of the application you want to package. Follow these instructions depending on the format of the application you are packaging.

UNIX tar file

► To extract the contents of a tar file

- 1 Create a new directory, for example, /tmp/newdir.
- 2 Go to the directory you just created, for example, cd /tmp/newdir.
- 3 To extract the contents of the archive to /tmp/newdir, type the command:
`tar -xf <path to the tar file>`

UNIX cpio archive

► To extract the contents of a cpio archive

- 1 Create a new directory, for example, /tmp/newdir.
- 2 Go to the directory you just created, for example, cd /tmp/newdir.
- 3 To extract the contents of the archive to /tmp/newdir, type the command:
`cpio -idf <path to the archive>`

UNIX SVr4 package

► To prepare an SVr4 (pkgadd) package for Symphony Marimba Client Automation packaging

- 1 Create a new directory, for example, /tmp/newdir.
- 2 If the SVr4 package is in the stream format (a single file), convert it to the filesystem format using the command:
`pkgtrans [path to package] /tmp/newdir`
- 3 If the package is already in the filesystem format, note its location for subsequent steps. These steps assume its location is /tmp/newdir.
- 4 To create a response file for your package, type the command:
`pkgask -d /tmp/newdir -r /tmp/newdir/response`

This command performs a “dummy” installation of your package. It asks you each of the installation questions that it would ask during a normal installation, and saves the answers in the response file.

Starting Application Packager

Before you start Application Packager, install and start the tuner. You use the tuner’s GUI (also called the Channel Manager) to start, subscribe, and update the Application Packager channel.

► To start Application Packager

- 1 On the tuner’s GUI (Channel Manager), right-click the Application Packager channel and choose Start.

The Application Packager window appears and the Select Package page appears.

- 2 To create a new channel from an application, click a button on the left side for the type of channel you want to create and click Create. For example, to create a channel using the Custom Application Packager, click Custom.

The packager starts and you are ready to create a new channel from an application.

Working with channels

You can use the Package Editor to edit, publish, import, or remove an existing channel.

Editing existing channels

To change the contents, configuration, and properties of a channel, you can use the Package Editor. You can access the editor from within the various packagers or from the list of packaged applications.

Note: If you import and edit a channel that was packaged using a previous version of Application Packager, you are asked if you want to upgrade the channel. You must upgrade the channel before you can edit it with the current version of Application Packager.

► To edit a channel

- 1 On the Application Packager window, click a button on the left side for the type of channel you want to edit.

A list of the channels you have packaged using that particular packager appears. If the channel does not appear in this list, import the channel. For more information, see “Importing channels” on page 64.

- 2 Select the channel you want to edit and click Edit.

The Package Editor appears in a separate window. The options available in the Package Editor are different for each type of channel.

- 3 Edit the channel.

For more information, see “Using the Package Editor” on page 67.

- 4 Save your changes, and exit the Package Editor.

Publishing channels

Each packaging component of Application Packager creates a complete set of files for the application you want to distribute as a channel. On the packager or the Package Editor, you can click the Publish button to launch a wizard that guides you through the process of publishing the channel to a transmitter. Publishing a channel to a transmitter makes it available for distribution to endpoints.

To sign the channel for security, use a channel-signing certificate from a certificate authority (such as VeriSign). The appropriate trust settings should be preset in the channel for you by the packager. For more information, see “Publishing channels” on page 279.

Automatic upgrade of packages during installation

Prior to BBCA 8.3.00, you had to repackage all the packages every time a new hotfix or feature is released for Application Packager. To overcome this task, from BBCA 8.3.00, the packages are automatically upgraded to 8.3.00 version during installation. Defects resolved in Application Packager until 8.3.00 version will be available for older version packages without the need for re-packaging.

If auto-upgrade of packages to 8.3.00 version is not needed, you can disable the auto-upgrade feature by setting the `marimba.packages.autoupgrade` tuner property to false.

Hence forward from BBCA 8.3.00, if any new hotfix is released for Application Packager 8.3.00, then any package created with that hotfix version will run with the same version of Application Packager.

This feature ensures that the tuner bundles the Application Packager binaries of corresponding tuner version. Package installation will use binaries from the tuner only if the tuner version is latest than the package version.

Importing channels

You can import a channel that was previously packaged by you or someone else using Application Packager. Determine the location of the package directory. After it is imported, you can edit the channel using the Package Editor. For more information, see “Using the Package Editor” on page 67.

Note: If you copy a package from a CD, the files and directories in the package have the “read-only” attribute set. Importing such a package into Application Packager causes an error with an “access denied” exception in the history log. Make sure you remove the “read-only” attribute from the contents of a package before importing it into Application Packager.

► To import an existing channel

- 1 On the main Application Packager window, choose File > Import.
- 2 From the Directory Finder window, select the package directory to import.
The channel is added to the list of packaged applications.
- 3 You can now edit the channel.
For more information, see “Editing existing channels” on page 62.

Removing channels from the list

For each packaging component, Application Packager maintains a list of channels that were previously packaged. This section describes how you can remove a channel from that list.

► To remove an existing channel from the list

- 1 On the main Application Packager window, click a button on the left side that reflects the type of channel you want to remove.

A list of the channels you have packaged using that particular packager appears.

- 2 Select the channel you want to remove and click Remove.

The channel is removed from the list of packaged applications. However, the package directory is not removed from the file system or from the transmitter to which you might have published the channel previously.

Chapter

4 Using the Package Editor

This section shows you how to use the Package Editor to review and edit channels created using Application Packager before you publish them.

The following topics are provided:

- Overview (page 69)
- Opening a channel for editing (page 69)
- Editing the contents of a channel (page 70)
- Using macros (page 95)
- Setting channel startup options (page 104)
- Setting installation modes (page 106)
- Enabling installer logging (page 111)
- Setting system reboot options (Windows only) (page 114)
- Specifying the installation platform (page 117)
- Redirection behaviour (page 118)
- Setting policies for installation, update, uninstallation, verification, and repair (page 121)
- Using environment variables (page 127)
- Configuring system and channel dependencies (page 130)
- Customizing a channel by using scripts and classes (page 141)
- Managing major and minor updates (page 150)
- Managing services (Windows only) (page 152)
- Editing the channel parameters and properties (page 158)

- Minimizing bandwidth usage (page 159)
- Working with packages that contain user-specific files and registry entries (page 161)
- Verifying and repairing channels (page 165)
- Updating and repairing applications using shortcuts before startup (Windows only) (page 169)
- Associating packages with software library items in the Definitive Software Library (DSL) (page 171)
- Editing ASCII text files (page 173)
- Recording change history for channels (page 184)
- Saving changes to channels (page 186)
- Recovering from channel corruption (page 187)

Overview

You can use the Package Editor to review and edit channels created using Application Packager before you publish them. You can make the following changes:

- View, add, remove, and edit directories, files, and registry entries in the channel.
- Use macros for making global installation substitutions.
- Configure startup and installation options for the channel and the application it installs.
- Set installation and uninstallation policies for the channel and for individual files in the channel.
- Configure the verify and repair options for the channel.
- Set package dependencies to configure the memory, disk space, or files required to install and run a channel.



To get help using the Package Editor, click Help in the toolbar. The help topic associated with the area of the editor you are using appears. To open help to an introductory topic about the Package Editor, choose Help > Contents.

Opening a channel for editing

You can use the Package Editor to edit channels that you have packaged using the following packagers:

- Packager for Shrinkwrap Windows Applications
- Custom Application Packager
- File Packager
- Windows Installer Packager
- .NET Packager

You can open a channel for editing from the main Application Packager window or from the individual packagers. This topic tells you how to open a channel from the main Application Packager window.

Note: If you import and edit a channel that was packaged using a previous version of Application Packager, you are asked if you want to upgrade the channel. You must upgrade the channel before you can edit it with the current version of Application Packager. You can also upgrade several channels at one time by using the command-line option `-upgrade`. For more information, see “Upgrading channels” on page 325 website.

► To open a channel for editing

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 On the main Application Packager window, click one of the following packagers:
 - Shrinkwrap
 - Custom
 - Windows Installer
 - File Packager
 - .NET

Depending on the packager you select, a list of previously packaged channels appears.

- 3 Select a channel from the list and click Edit.

To edit a channel that someone else prepared using Application Packager, you must import it first. For more information, see “Importing channels” on page 64.

The Package Editor opens the channel and displays the Package Contents page where you can edit the contents and properties. You can select other pages in the editor by clicking their tabs.

Editing the contents of a channel

The Package Contents page appears when you start the Package Editor. You can use the Package Contents page to:

- Add, delete, rename, and edit files, directories, registry entries, and other contents of your channel.

- Search the channel for files, directories, and registry keys.

When you edit a Packager for Shrinkwrap Windows Applications channel, the Package Contents page shows you the difference between the pre- and postinstall snapshot. Do not expect to see every file in a directory you selected during the snapshot process. Only the files, directories, and other contents needed to create the channel appear here.

When you edit a Packager for Shrinkwrap Windows Applications channel using the Package Editor, you are overriding the settings that the snapshot gathered from your system. For example, perhaps the application's installation program put the application files in C:\Program Files\ACME, but you want the application to be installed at C:\ACME on the endpoint. The Package Editor can help you make this kind of change.

If you know how to use Microsoft's Registry Editor (regedit), you will find this page very familiar. The next few topics provide basic instruction on the most common commands that are available from this page. For more information, see your Registry Editor documentation.

Depending on what you select (file, directory, registry key, or other content), you can right-click to see a pop-up menu of available commands for the selected item. (The same commands are available using the Edit menu in the Package Editor's main menu bar.)

Working with files

This section describes how to use the Package Contents page in the Package Editor to add, remove, and edit the files in a channel. It also describes how to edit a file's properties.

Adding files to a channel

You can use the Package Contents page to add files to a channel. When you add files to a channel, those files are installed on users' machines when users subscribe to and install that channel.

You also have the option of adding files by reference. For more information, see "Adding files by reference" on page 72.

► To add a file

- 1 On the left pane, select the parent directory in which you want to add a file.
- 2 Right-click the directory name and choose Add > File.

3 Select the file you want to add.

4 Click Open.

The file you added appears on the Package Contents page.

Adding files by reference

You can use the Package Contents page to add a file reference to a channel. When you use a file reference, the contents of the file are published directly to the transmitter.

Adding files by reference has the following benefits:

- File references are useful when you are packaging an application that includes large files. When you add files to a channel, their contents are copied to the package directory. As a result, the storage required by the package directory is at least the total size of the original files. The additional storage can be significant, especially for large files. However, if you add files by reference, you do not need the additional storage because the contents of the files are not stored in the package directory.
- File references are also useful when you have a channel with files that are, for the most part, static except for one or two files that are occasionally updated. When updating the channel, you do not need to replace the original files in the channel with the newer versions. If the channel includes references to the updated files in the file system, the contents of the updated files are used when you republish the channel.
- You can also take advantage of file references if you have multiple channels that use the exact same file. You save space because you do not need multiple copies of the file in each package directory.

Adding files by reference has the following limitations:

- Channels that contain files added by reference are not portable because they point to an absolute path on the local machine. Therefore, the channel cannot be transferred to another machine without some changes to the channel.
- If a channel contains any files that are added by reference, the `delayfiledownload` parameter does not take effect during updates. For more information, see “Delaying the download of a channel’s files” on page 109.

- File attributes, such as timestamps, might not be accurate for files that are added by reference. Because files might have changed from when they were added by reference, the file attributes might be outdated.
- You cannot add INI files by reference because of the way Application Packager handles them. For more information about INI files, see “Working with INI files” on page 78.

► To add a file reference

- 1 On the Package Contents page, locate the directory to which you want to add a file reference.
- 2 Right-click the directory and choose Add > File by reference.
- 3 Select the file to which you want to refer.
- 4 Click Open.



A reference to the file appears on the Package Contents page.

► To change an included file to a file reference

- 1 On the Package Contents page, locate the file you want to change to a file reference.
- 2 Right-click the file and choose Properties.
- 3 To specify properties, perform the following steps:
 - a Select the See file content here check box.
 - b In the text box, specify the name and path of the file to which you want to refer.



To browse the machine and find the file to which you want to refer, click Browse.

- 4 Click OK.



A reference to the file appears on the Package Contents page.

Editing file properties

You can edit the properties of files that are part of a channel you are editing. Depending on the type of file you select, you see a different set of properties to edit.

► To edit the properties of a file

- 1 Select the file you want to edit.



You might need to expand directories or use the find feature.

- 2 Right-click the file and choose Properties.
- 3 Set the properties for the file.

The properties you can set depend on the type of file. For example, in Windows, if you are setting properties for a dynamic-link library (DLL) file, there is a Register DLL check box that you can select so that the file self-registers (adds itself to the registry) at installation time. In UNIX, you can set read, write, and execute (RWX) permissions for users and groups, as well as user and group IDs.

- 4 You can also use this dialog box to set installation, uninstallation, and verification policies. For more information, refer “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.
- 5 Click OK.

Marking DLL files for registration (Windows only)

When you edit a channel, you can mark all or individual DLL files for registration. When you mark a DLL file for registration, it attempts to self-register (add itself to the registry) during installation time.

Registration of DLL files is part of the Windows Component Object Model (COM). Self-registering DLL files implement the methods `DLLRegisterServer` and `DLLUnregisterServer` to allow third-party applications to register the DLL files in the registry (\HKCR\Classes).

Use the DLL registration feature when you know that:

- The DLL file can be registered and implements `DLLRegisterServer`.
- All the dependencies needed for the DLL file to register are satisfied.

This feature is most helpful for channels created using File Packager and Custom Application Packager, where the registry keys are not captured. Channels created using Packager for Shrinkwrap Windows Applications might already include changes to the registry keys in HKCR, so the DLL registration feature is generally not needed.

► To mark all DLL files for registration

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 Choose Package > Register DLLs.
- 4 If you do not want DLL files to be marked for registration, choose Package > Unregister DLLs.

Note: If you set this attribute to true, all files that have the .dll and .ocx extensions attempt to self-register at installation time. To specify self-registration for individual files that have the .dll and .ocx extensions, use the File Properties dialog box for the file (right-click the file on the Package Contents page of the Package Editor and choose Properties).

► To mark individual DLL files for registration

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 Select the DLL file you want to mark for registration.



You might need to expand directories or use the find feature.

- 4 Right-click the file and choose Properties.

The File Properties dialog box appears.

- 5 Select the Register DLL check box.

If you do not want the DLL file to be marked for registration, clear the Register DLL check box.

- 6 Click OK.

Registration of DLL files

In Windows, Application Packager registers and deregisters registerable DLL files by loading the library into the process address space and invoking the exported methods in the DLL files. Before version 4.7.2.1, Application Packager loaded the DLL files into the process address space of the tuner, which made the tuner vulnerable to any defects that might exist in the registration and deregistration methods of the DLL files. Potentially, defects in third-party DLL files might cause problems with the tuner. To avoid these problems, beginning in version 4.7.2.1, Application Packager uses separate non-console Windows applications to register or deregister DLL files.

Application Packager uses one of the following Windows applications:

Table 4-1: Registration of DLL files

Application	Description
regsvr32.exe	If the file path <i><Windows_system_directory>\regsvr32.exe</i> exists on the endpoint, Application Packager uses it to silently register and deregister DLL files. For registration, it uses the following command-line syntax: <i><Windows_system_directory>\regsvr32.exe /s</i> <i><DLL_file_path></i> For deregistration, it uses the following command-line syntax: <i><Windows_system_directory>\regsvr32.exe /s /u</i> <i><DLL_file_path></i>
regsvrw.exe	If regsvr32.exe does not exist on a Windows NT or above endpoint, Application Packager uses the internal utility regsvrw.exe.
regsvr.exe	If regsvr32.exe does not exist on a Windows 95/98/ ME endpoint, Application Packager uses the internal utility regsvr.exe.

Viewing the contents of a file

You can use the Package Contents page to view the contents of files, such as INI files and ASCII text files. After viewing the contents of an ASCII text file, you might want to change the text contents by using text modifiers. For more information, see “Editing ASCII text files” on page 173.

► To view the contents of a file

- 1 Locate the file you want to view.



You might need to expand directories or use the find feature.

- 2 Right-click the file and choose View Contents.

The File Contents dialog box appears and shows the file's contents.

- 3 Click OK.

Adding shortcuts to a channel (Windows)

You can use the Package Contents page to add shortcuts to a channel. When you add shortcuts to a channel, those shortcuts are installed on users' machines when users subscribe to and install that channel.

► To add a shortcut

- 1 On the left pane, select the parent directory in which you want to add the new shortcut.
- 2 Right-click the directory name and choose New > Shortcut.
- 3 To set the shortcut's properties, perform the following steps:
 - a Specify a name for the shortcut. The name must include the .lnk extension.
 - b For Path, specify the complete path to the item to which you want the shortcut to point, such as a file or directory.

Note: When creating shortcuts using Package Editor, you can specify file system paths, but not URLs.

- c Optionally, you can also set other properties for the shortcut, including arguments and icons.
- d Click OK.



The shortcut you added appears on the Package Contents page.

You can also configure shortcuts so that when an application is started using the shortcut, the packaged application is automatically updated or repaired. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.

Note: When you add a shortcut file (a file with a .lnk extension in Windows) to a package, the actual file to which the shortcut refers is added to the package. To add a real shortcut, follow the steps in this procedure.

Editing symbolic links (UNIX only)



You can edit symbolic links to files that are part of a channel you are editing.

► **To edit the symbolic link to a file**

- 1 Select the symbolic link you want to edit.
- 2 Right-click the symbolic link and choose Properties.
- 3 In the dialog box that appears, edit the link information for the file.
- 4 To change the installation and uninstallation properties for the file, click the Install Policy and Uninstall Policy tabs. For more information, see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.

Working with INI files

For Application Packager to work properly with INI files, INI files must be in the standard format. INI files should contain one or more sections, each indicated by a section heading enclosed in brackets. The entries in each section must consist of the key name, an equals sign (=), and the key value. A typical section might look like:

```
[section1]
key1=value1
key2=value2
```

How Application Packager handles INI files

When you create a package using Packager for Shrinkwrap Windows Applications, the packager identifies only the items added or changed in an INI file. For example, in a pre-snapshot, the file config.ini contains the following items:

```
[section1]
key1=value1
key2=value2
```

and in the post-snapshot, the file config.ini contains the following items:

```
[section1]
key1=value1
key3=value3
```

The packager identifies the following items for installation:

```
[section1]
key3=value3
```

At installation time, Application Packager tries to add [section1], key3=value3 to the file config.ini. In cases where key3 exists in the endpoint's config.ini, Application Packager overwrites its value with value3. In cases where config.ini is missing all together, Application Packager creates config.ini with section1 and key3=value3.

At uninstallation time, Application Packager finds the file config.ini and removes the key name key3. It removes section1 only if it no longer contains any entries.

Note: This behavior applies with the default installation and uninstallation policies. It no longer applies if you have changed the installation or uninstallation policies.

INI files that contain duplicate sections

Application Packager merges duplicate INI file sections into a single section when the following events occur:

- When adding an INI file to a package
- When writing the INI file to disk at installation time

For example, the INI file contains the following duplicate sections:

```
[section]
key1=value1
[section]
key2=value2
```

When the INI file is added to a package, or written to disk at installation time, Application Packager merges the duplicate sections into a single section:

```
[section]
key1=value1
key2=value2
```

INI files that contain duplicate entries

The INI file format does not allow duplicate key-value pairs in a section. Duplicate key-value pairs have the same key name and appear under the same section in the INI file. However, due to new policies in Application Packager version 4.7.2.1, you can maintain duplicate key-value pairs.

- **New policies**—If the application you are packaging requires duplicate key-value pairs, you can use two new policies for INI files (available in Application Packager version 4.7.2.1 and higher):

- New installation policy—Allow duplicate values.
- New update policy—Allow duplicate values.

If you set these policies, a key-value pair does not overwrite another key-value pair with the same key name but a different value. The following examples show how Application Packager resolves duplicate key-value pairs with the installation and update policies Allow duplicate values set:

Example 1

The package contains:

```
[section1]
key1=value1
```

The endpoint contains:

```
[section1]
key1=value2
```

The result is:

```
[section1]
key1=value1
key1=value2
```

Example 2

The package contains:

```
[section1]
key1=value1
```

The endpoint contains:

```
[section1]
key1=value1
```

The result is:

```
[section1]
key1=value1
```

You cannot have duplicate entries with the same key name and value, as in the following example:

```
[section1]
key1=value1
key1=value1
```

Example 3

The package contains:

```
[section1]
key1=value1
key1=value2
```

The endpoint contains:

```
[section1]
key1=value1
```

The result is:

```
[section1]
key1=value1
key1=value2
```

- **Repair operation**—If an INI file has its installation or update policies set to Allow duplicate values, during a repair operation Application Packager appends a key-value pair to the section instead of overwriting an existing key-value pair with the same key name. However, if a key-value pair with the same key name and value already exists in the INI file at the endpoint, Application Packager does not append anything.
- **Uninstall operation**—If an INI file has its installation or update policies set, Application Packager keeps track of entries that were created or merged so that they can be removed properly during uninstallation. During an uninstall operation, Application Packager checks which entries were created or merged with the Allow duplicate values policy and removes an entry from the INI file at the endpoint only if both the key name and key value matches. If only the key name matches, Application Packager does not remove the entry from the endpoint.

Working with directories

This section describes how to use the Package Contents page in the Package Editor to add, remove, and edit the directories in a channel. It also describes how to edit a directory's properties.

Adding directories to a channel

You can use the Package Contents page to add directories to a channel. The directory you add to a channel can be a directory that already exists on the packaging machine or a new directory. When you add directories to a channel, those directories are created and their contents (including subdirectories and files) are installed on users' machines when users subscribe to and install that channel.

► To add an existing directory

- 1 On the left pane, select the parent directory in which you want to add an existing directory.
- 2 Right-click the directory name and choose Add > Directory.
- 3 Specify the name of the directory you want to add to the channel.
- 4 Click OK.

The directory you added appears on the Package Contents page.

► To add a new directory

- 1 On the left pane, select the parent directory in which you want to add the new directory.
- 2 Right-click the directory name and choose New > Directory.
- 3 Specify a name for the directory you are adding to the channel.
- 4 Click OK.

The directory you added appears on the Package Contents page.

Editing directory properties

You can edit the properties of directories that are part of a channel you are editing.

► To edit the properties of a directory

- 1 Select the directory you want to edit.



You might need to expand directories or use the find feature.

- 2 Right-click the directory and choose Properties.
- 3 Set the properties for the directory.

For example, in Windows, you can set read-only, system, hidden, archive, and compressed (RSHAC) attributes for the directory. In UNIX, you can set read, write, and execute (RWX) permissions for users and groups, as well as user and group IDs.

You can also use this dialog box to set installation, uninstallation, and verification policies. For more information, see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.

Note: You cannot edit directory properties by double-clicking a directory or selecting the directory and pressing ENTER. Doing either expands the directory to show its contents or collapses the directory to hide its contents.

Working with registry entries

This section describes how to use the Package Contents page in the Package Editor to add, delete, and edit the registry entries in a channel. It also describes how to edit a registry entry’s properties.

Note: Registry entries are present only on 32-bit Windows platforms.

WARNING: Use extreme care when editing the registry entries that are part of the channels you create. It is possible to make changes that could make the target endpoint inoperable after your channel is installed. Use the same care when editing registry entries with the Package Editor that you use with Microsoft’s Registry Editor. For more information, see “Editing the Windows registry” on page 46.

Adding registry entries to a channel

You can use the Package Contents page to add registry entries to a channel. When you add registry entries to a channel, those registry entries are installed on users' machines when users subscribe to and install that channel.

► To add a registry entry

- 1 On the left pane, select the parent registry key directory in which you want to add the new registry entry.
- 2 Right-click the key directory name, and choose New.
- 3 Select the type of registry entry you want to add:
 - Subkey—A directory for other keys
 - String Value
 - Binary Value
 - Dword Value
- 4 In the dialog box that appears, perform the following steps:
 - a Specify a name for the registry entry you are adding.
 - b If necessary, specify value data and other information for the registry entry.
 - c Click OK.

The registry entry you added appears on the Package Contents page.

Importing registry entries from a registry file

In Windows, you can use the Registry Editor (`regedit.exe`) to export parts of the registry to text files. You can use the Package Editor to import these registry files (saved with the `.reg` extension) and add the registry entries to your packaged application.

Registry files with the following headers are supported:

- Windows Registry Editor Version 5.00 (for Windows 2000 and higher).
- REGEDIT4 (for Windows 95, Windows 98, and Windows NT).

These headers should appear at the beginning of the registry file.

► To import registry entries from a registry file

- 1 On the Package Editor window, choose Package > Import .reg file.
- 2 Select the registry file you want to import.
- 3 Click OK.

A dialog box indicates whether the editor successfully imported the entries from the registry file. The registry entries in the file you imported now appear on the Package Contents page.

Note: If a key or key-value pair already exists in the packaged application, importing a registry file containing that same key or key-value pair results in a dialog box that gives you the option of overwriting registry keys and registry key values or keeping the existing ones. Overwriting a registry key removes any existing values and subkeys. Overwriting a registry value only occurs for those key-value pairs that have different values.

Editing registry value data

A registry consists of multiple top-level keys. Each top-level key can have nested subkeys, each of which can have multiple value entries. A value entry contains a value name and the data assigned to it. You can edit the value data. Editing a value name is covered in “Renaming the items in a channel” on page 92.

► To edit a value data entry

- 1 Select the registry entry you want to edit.
- 2 Right-click the registry entry and choose Edit.
- 3 In the dialog box that appears, specify the new value data.
- 4 Click OK.

Editing registry entry properties

You can edit the properties of registry entries in a channel you are editing. Depending on the type of registry entry you select, you see a different set of properties to edit.

► To edit the properties of a registry entry

- 1 Select the registry entry you want to edit.



You might need to expand directories or use the find feature.

- 2 Right-click the registry entry and choose Properties.
- 3 Set the properties you want for the registry entry. For more information, see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.

Working with metabase entries

If the channel you are editing consists of an application built specifically for use in Windows server environments, it might include metabase entries. The metabase is a binary file (similar to the registry) in which Microsoft’s Internet Information Server (IIS) stores information. Like the registry, the metabase contains key nodes and value nodes. When you install a server application, the application makes changes to the metabase by adding or editing key nodes and value nodes. Packager for Shrinkwrap Windows Applications captures any changes that the application has made to the metabase and stores them in the channel for distribution to other machines.

This section describes how to use the Package Contents page in the Package Editor to add, delete, and edit the metabase entries, such as key nodes and value nodes, in a channel. It also describes how to edit a metabase entry’s properties and use macros in metabase entries.

Note: The metabase is present in Windows only if Microsoft’s Internet Information Server (IIS) is installed.

WARNING: Use extreme care when editing the metabase entries that are part of the channels you create. It is possible to make changes that could make IIS (and consequently your web server or FTP server) inoperable after your channel is installed. Use the same care when editing metabase entries with the Package Editor that you use with Microsoft’s MetaEdit.

Adding metabase entries to a channel

You can use the Package Contents page to add metabase entries to a channel. When you add metabase entries to a channel, those metabase entries are added to the metabase on machines that subscribe to and install that channel.

► To add a new metabase key

- 1 On the Package Contents page, right-click IIS Metabase and choose New Key.
- 2 For Root of new key, select the root of the new key you are adding.
- 3 For Key name, specify the name of the new key you are adding.
- 4 Click OK.

The metabase key you added appears on the Package Contents page.

► To add a new metabase subkey

- 1 On the Package Contents page, right-click the IIS metabase key to which you want to add a subkey, and choose New > Subkey.
- 2 Specify the name of the new subkey you are adding.
- 3 Click OK.

The metabase subkey you added appears on the Package Contents page.

► To add a new metabase entry

- 1 On the Package Contents page (left pane), right-click the IIS metabase key or subkey to which you want to add a metabase entry and choose New.
- 2 Select the type of metabase entry you want to add:

Subkey—A directory for other keys.

String Value—A sequence of characters.

Dword Value—A numeric value, often used for numeric properties or true-false conditions.

Binary Value—Raw binary data; usually a string of byte characters.

Expand String—A string of text characters; usually consists of text, but also contains a variable that is replaced when it is called by an application.

Multi-String—A collection of strings containing text characters.

- 3 In the dialog box that appears, specify the information for the new metabase entry you are adding. For more information about the different metabase entries and their attributes, see your IIS documentation or Microsoft's website (www.microsoft.com).

Note: For multi-string values, use single quotation marks (‘ ’) around each string and delimit the strings in the list using semicolons (;).

- 4 Click OK.

The metabase entry you added appears on the Package Contents page.

Editing metabase entry properties

You can edit the properties of metabase entries in a channel. Depending on the type of metabase entry you select, you see different properties to edit.

► To edit the properties of a metabase entry

- 1 Select the metabase entry.

You might need to expand metabase keys to find a metabase entry.



Note: You cannot use the find feature to search for metabase entries.

- 2 Right-click the metabase entry and choose Properties.
- 3 Set the properties for the metabase entry. The installation, uninstallation, and verification policies you can set are the same as those for registry entries. For more information, see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.
- 4 Click OK.

Editing metabase value data

The metabase consists of keys and nested subkeys, each of which can have multiple value entries. A value entry contains a value name and the data assigned to it. When you select a metabase key or subkey in the left pane of the Package Contents page, the right-hand pane shows the value entry or entries for that key or subkey. You can use the Package Editor to edit the value data as well as edit an entry's user type and attributes. (Editing a value name is covered in “Renaming the items in a channel” on page 92.)

► To edit a metabase value entry’s data

- 1 Select the metabase entry you want to edit.
- 2 Right-click the metabase entry and choose Edit.
- 3 In the dialog box that appears, edit the value’s data. Depending on the type of entry you are editing, you can edit strings, binary data, or dword data. You can also change a metabase entry’s user type and attributes.

Note: For multi-string values, use single quotation marks ('') around each string and delimit the strings in the list using semicolons (;).

A metabase entry’s user type indicates what type of object uses the entry. IIS has four user type values: (Each user type is represented by a number; you use this number when specifying a metabase entry’s user type in the Package Editor.)

Server (1)—Contain information that corresponds to the server.

File (2)—Indicates that a file, directory, or virtual directory uses the entry.

Web Application Management (WAM) object (100)—Indicates that the WAM object is using the entry when it is managing the resources for a particular application.

Application (101)—Used for Active Server Pages (ASP) application entries that apply to the ASP engine itself.

Each attribute is represented by a check box in the Package Editor. Each metabase entry has the following attributes:

Inherit—Indicates whether a metabase entry inherits a value from its parent key or subkey.

Secure—Indicates whether a metabase entry's data should be stored and transported securely.

Reference—Indicates that a metabase entry's value data is received by reference.

Volatile—Specifies whether a metabase entry can be overwritten every time an application or the operating system starts.

Insert Path—Returns the full path describing the location of the entry when the metabase retrieves the value of the property. If you do not select this check box, the metabase returns only the value.

- 4 Click OK.

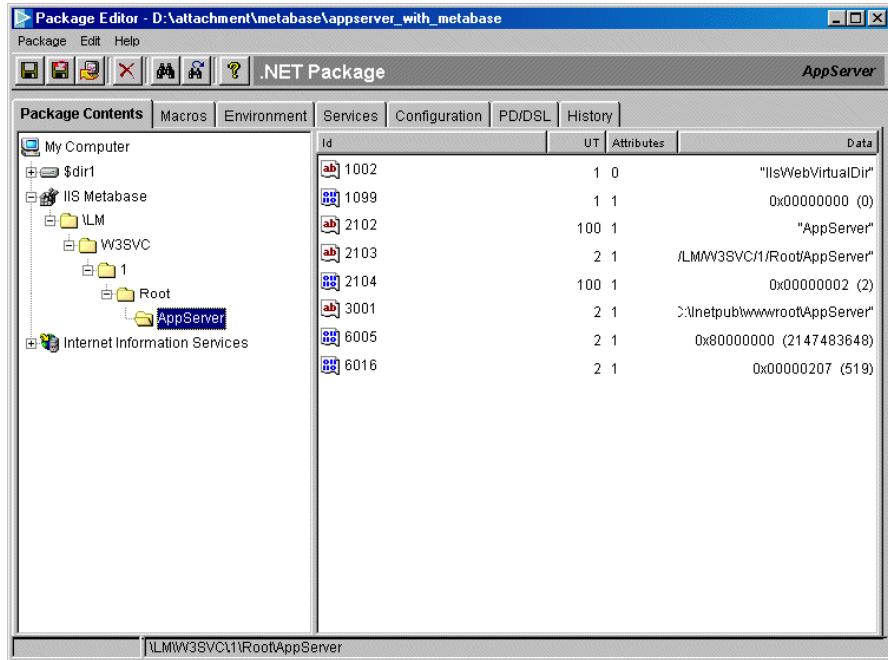
Setting metabase properties for an IIS application

When you install an IIS application, it installs a metabase with default values. You can set your own custom values using the Package Contents tab. In the following example, you create a metabase property for the attribute AppFriendlyName, related to the IIS Virtual Directory name AppServer. The procedure is similar when you set other attributes, such as directory browsing flags or access flags. For more information about IDs and configurable locations, visit the Microsoft Windows Server section of the Microsoft website.

► To set a metabase property for a package or application

- 1 On the Package Content tab, create the structures shown in the following figure using the procedures described in the previous sections.

Figure 4-1: Package Content tab



- 2 The ID for AppFriendlyName is 2102. In this example, to set the AppFriendlyName to Appserver, type the value AppServer for ID 2102.

Using macros in metabase keys and values

You can create macros using the Package Editor and then use those macros in metabase keys and values.

► To use a macro in a metabase key

- 1 Create a macro, as described in “Creating user-defined macros” on page 96.
- 2 To add a new metabase key, right-click the metabase and choose New Key.
- 3 For Key name, specify the name of the new key you are adding and include the name of the macro you created. For example, if you created a macro name \$KEY_NAME and want to use it in a metabase key with the path \LM\W3SVC\1\Root\, type this key name: \LM\W3SVC\1\Root\\$KEY_NAME\$.
- 4 Click OK.

The metabase key you added (including the macro name) appears on the Package Contents page.

► **To use a macro in metabase value data**

- 1 Create a macro, as described in “Creating user-defined macros” on page 96.
- 2 Perform one of the following steps:
 - To add a new metabase entry, right-click the metabase and choose New > String Value, Expand String, or Multi-String.
 - To edit an existing metabase entry, right-click an entry and choose Edit.
- 3 In the dialog box that appears, specify the macro name in String Data. You can use the macro name in combination with other data or other macros.
- 4 Click OK.

Renaming the items in a channel

You can rename directories, files, registry keys (except the top-level keys), registry value names (except default values), and other contents of a channel. If you rename an item, be sure that the application you are packaging uses the new name too; the Package Editor does not interact with calls inside the application, so you must make sure that renaming items does not break the application.

► **To rename an item in a channel**

- 1 Select the item you want to rename.
- 2 Right-click the item and choose Rename.
- 3 In the dialog box that appears, specify the new name.

Tip: In Windows, you can rename an item by selecting it and pressing F2.

Removing items from a channel

You can use the Package Contents page to remove directories, files, registry keys, and other items from a channel. For the registry, you can delete default value entry data, but the empty default value still appears on the Package Contents page.

► To remove an item from a channel

- 1 Select the item you want to remove.



You might need to expand directories or use the find feature.

- 2 Right-click the item and choose Delete.

The item you removed no longer appears on the Package Contents page.

Note: The Delete command is not available for items that cannot be removed.

Marking items for removal from the endpoints

Although you usually use Application Packager to install directories, files, registry entries, and other items on endpoints, you might also want to use it to remove items from endpoints. You can mark items for removal by using the Package Editor to invert the directory, file, or registry entry. Inverting the item changes the operation associated with the item from installation to removal. Items to be removed from the endpoint appear on the Package Contents page of the Package Editor with an X over them.

When the channel (containing the items to be removed) is installed at the endpoint, the specified items are uninstalled. Depending on how the channel is configured, the uninstalled items might be stored in a backup directory (the `unfiles` directory of the channel in the tuner's workspace directory), so that they can be restored if you uninstall the channel. For more information, see also "Preventing items from being stored for backup" on page 109.

If you use Packager for Shrinkwrap Windows Applications to create the channel, you can also configure the snapshots to include the removal of items by selecting the Capture removal of files and directories check box and the Capture removal of registry entries check box. Items marked for removal also appear on the Package Contents page of the Package Editor with an X over them. For more information, see "Packaging the removal of an application" on page 55.

► To mark an item for removal from the endpoint

- 1 Start Application Packager, as described in "Starting Application Packager" on page 62.
- 2 Open a channel for editing, as described in "Opening a channel for editing" on page 69.

- 3 On the Package Contents page, select the item you want to mark for removal.
- 4 Right-click and choose Invert.



The icon beside the item's name appears with an X over it. For example, a directory marked for removal has this icon beside it.

If you mark a container object (such as a directory or registry key) for removal, its child objects (subdirectories/files or registry subkeys/value pairs) are also marked for removal.

Note: If a directory is inverted (marked for removal), its contents should not be inverted again (marked for installation instead of removal). Otherwise, the package installation fails at the endpoint because the contents are being installed in a directory that does not exist.

Searching for file system and registry entries

To make it easier to locate directories, files, subkeys, and value entries, the Package Editor provides a Find command. You can refine your searches by setting optional search parameters.

► To search for file system and registry entries

- 1 From the left pane of the Package Contents page, select the type of item you want to search for:
 - To search for a file, click a directory object.
 - To search for a registry entry, click a registry key.
 - To search for both files and registry entries, click My Computer.
- 2 Right-click the item and choose Find.
- 3 Specify the name of the item you want to find.
If you do not know the whole name, clear the Match whole string check box.
- 4 Select any appropriate file or registry selection options.
- 5 Click Find.
- 6 To search for another occurrence of the item, choose Edit > Find Next.

Using macros

This section describes how to use the Package Editor to add, edit, and remove macros from a channel.

You can use macros to customize channels. You can use macros in place of the following items:

- Directory path
- Current user name
- System name
- Boot drive
- System directory (for example, Windows, System, Favorites, and so on)
- Environment variable value
- Another macro
- Current date or time
- Channel or package properties
- Registry value
- IP address

You can use macros in the following places:

- Directory name
- File name
- Registry key name
- Registry value name
- Registry value data
- Environment variable name
- Environment variable definition
- INI file section name
- INI file key name
- INI file value
- Shortcut parameter

■ Windows service parameter

For example, you can create macros for your channel that allow users to replace the installation directory for an application with a directory of their choice. At installation time, the macro can prompt users for the needed path. If the user does not provide a path, or your macro does not use the prompting option, the default path (the path shown on the Package Contents page) is used. You can create as many macros as you need for a channel.

Packager for Shrinkwrap Windows Applications also uses a set of predefined macros that you can use but cannot change. These are listed on the Macros page under Available Macros. All macros appear on the Macros page under Available Macros and User-defined Macros.

When you use a macro, make sure the macro name is preceded by a dollar sign (\$). When used within a string, the macro name should also be followed by a dollar sign, for example, `home_dir\$custom_directory$\desktop`. Macros are case-sensitive.

Creating user-defined macros

You can create a macro in two ways. You can use the Define Macro command from the Package Contents page, or you can choose the Add command from the Macros page. The former method is described in the following procedure.

► To create a macro

- 1 On the Package Editor window, click the Package Contents tab.
- 2 Select the directory or file system for which you want to create a macro.
- 3 Right-click the directory or file system and choose Define Macro.
- 4 Select a macro type from the Type list:

User Defined—Has the default user namespace `$name`. Macros in this namespace cannot automatically resolve themselves. They either have default values, or they must be resolved by the user at runtime.

Registry Value—(Windows only) Has the namespace `$REG.name`. It expects a second macro called `$name` (in the default namespace) to exist. This macro should contain a complete path to a registry value.

Environment Variable—Has the namespace `$ENV.name`. It resolves to the value of the environment variable `name`.

Package Property—This macro has the namespace \$APP.name. It resolves to the value of the application property name.

Tuner Property—Has the namespace \$TUNER.name. It resolves to the value of the tuner property name.

Windows Installer Property—(Windows only) Has the namespace \$MSI.name. It resolves to the value of the Windows Installer property name. Microsoft Windows Installer (MSI) macros cannot be used in preinstall scripts.

- 5 For Macro Name, specify a name for your macro. For Environment Variable, Package Property, and Tuner Property macros, make sure that the macro name matches the corresponding property or variable name.

This name (preceded by \$) appears on the Package Contents page but is not seen by the people using your channel.

The directory you selected in step 2 appears automatically in the Default Macro Definition box. The default value is used if the registry value, property, or environment variable does not exist on the user's machine.

- 6 If the macro type you selected is Registry Value, specify a registry key and value in the corresponding Registry Key and Value boxes.

You can see another macro within your macro. When using a macro within a string, make sure that the macro has a terminating \$ sign, for example, C:\Windows\\$user\$\my_app.

- 7 To prompt the user for the path or value to use at installation time, select the Query user for definition check box and in the text box, specify the message you want the user to see.

- 8 Click OK.

Your macro appears in the list of user-defined macros.

Applying macros in INI files (Windows only)

You can apply macros to INI files using this procedure.

► To apply macros in an INI file

- 1 Locate the INI file in which you want to apply macros.



You might need to expand directories or use the find feature.

- 2 Right-click the file and choose Properties.

- 3 Select the Apply macros to the contents of this INI file check box.
- 4 Click OK.

Examples: Using macros in packaged applications

This section provides examples of how you can use macros with applications that you have packaged with Application Packager. It includes the following examples:

- “Using a registry key in a macro” on page 98
- “Using an environment variable in a macro” on page 100
- “Using a tuner property in a macro” on page 101
- “Creating a macro that allows users to select the installation directory” on page 102

Note: These examples are based on machines using Windows.

Using a registry key in a macro

In this example, you create a macro that reads a registry key to determine the installation directory. This example assumes that there is a registry key on the endpoint machine that gives the path to the installation directory (step 1).

► To use a registry key in a macro

- 1 Create a new registry key under `HKEY_LOCAL_MACHINE\SYSTEM\reginstall` with the following name and value:

Name—`installdir`

Value—`C:\channels\reginstall`

- 2 Use Application Packager to package an application.
- 3 To edit the packaged application, open it with the Package Editor.
- 4 In the editor, select the installation directory for the application you packaged. In this example, the directory is `C:\channels\install`.
- 5 Right-click the directory and choose Define Macro.
- 6 In the New Macro dialog box, perform the following steps:

- a For Type, select User Defined.
- b For Macro Name, specify the name you want, for example, `dir1`.

For Default Macro Definition, the directory you selected in step 4 (`C:\channels\install`) appears automatically.

7 Click OK.

On the Package Contents page, the name of the new macro you created (`$dir`) appears in place of the installation directory (`C:\channels\install`).

8 Click the Macros tab.

Under User-defined Macros is the macro you created, `$dir1`, which has the default path for the installation directory.

9 Click Add.

10 In the New Macro dialog box, perform the following steps:

- a For Type, select Registry Value.
- b For Macro Name, specify the name you want, for example, `reginstall`.
- c For Default Macro Definition, specify the default path you want to use for the installation directory, for example, `C:\channels\reginstall`.
- d For Registry Key, type `HKEY_LOCAL_MACHINE\SYSTEM\reginstall`.
- e For the corresponding Value, type `installdir`.

Note: The registry key and value that you specify should be the same as those that you used when creating the registry key in step 1. You use the same value so that at installation time the packaged application reads the registry key value that you defined. However, if the registry key does not exist, the macro uses the default macro definition.

11 Click OK.

The Macros page appears. The new macro you created appears under User-defined Macros with the name `$REG.reginstall`.

- 12 Under User-defined Macros, select `$dir1` and edit it so that the Default Macro Definition is `$REG.reginstall`.
- 13 Save the packaged application and publish it.

- 14 From the endpoint, subscribe to the packaged application, and check where the application got installed.

It should be installed under C:\channels\reginstall.

Using an environment variable in a macro

In this example, you create a macro that reads an environment variable to determine the installation directory.

► To use an environment variable in a macro

- 1 On the endpoint machines, create an environment variable with the following name and value:

Name—installdir

Value—C:\channels\install

- 2 Use Application Packager to package an application.
- 3 To edit the packaged application, open it with the Package Editor.
- 4 In the editor, select the installation directory for the application you packaged. In this example, the directory is C:\channels\install.
- 5 Right-click the directory and choose Define Macro.
- 6 In the New Macro dialog box, perform the following steps:
 - a For Type, select User Defined.
 - b For Macro Name, specify the name you want, for example, installdir.

For Default Macro Definition, the directory you selected in step 4 (C:\channels\install) appears automatically.

- 7 Click OK.

On the Package Contents page, the name of the new macro you created (\$dir) appears in place of the installation directory (C:\channels\install).

- 8 Click the Macros tab.
- 9 Click Add.
- 10 For Type, select Environment Variable.
- 11 For Environment Variable Name, specify the name you want, for example, installdir.

- 12 For Default Macro Definition, specify the default path you want to use for the installation directory, for example, C:\channels\install. This path is used if the environment variable is not defined at the endpoint.

- 13 Click OK.

The Macros page appears. The new macro you created appears under User-defined Macros with the name \$ENV.installdir.

- 14 Under User-defined Macros, select \$dir and edit it so that the Default Macro Definition is \$ENV.installdir.

- 15 Save the packaged application and publish it.

- 16 From the endpoint, subscribe to the packaged application, and check where the application got installed.

It should be installed under C:\channels\install.

Using a tuner property in a macro

In this example, you create a macro that takes a tuner property (in this case, the tuner ID) and writes its value to a registry key.

► To write a tuner property value to a registry key

- 1 Use Application Packager to package an application.
- 2 To edit the packaged application, open it with the Package Editor.
- 3 In the editor, click the Macros tab.
- 4 Click Add.
- 5 For Type, select Tuner Property.
- 6 For Tuner Property Name, specify the name you want, for example, marimba.license.identifier.
- 7 For Default Macro Definition, specify the default value you want to use if the tuner property is not defined at the endpoint, for example, 12345.
- 8 Click OK.

The Macros page appears. The new macro you created appears under User-defined Macros with the name \$TUNER.marimba.license.identifier.

- 9 Click the Package Contents tab.

- 10 Add a registry key under HKEY_LOCAL_MACHINE\Software\Castanet Info with the following name and value:
Name—TunerID
Value—\$TUNER.marimba.license.identifier
- 11 Save the packaged application and publish it.
- 12 From the endpoint, subscribe to the packaged application, and check that the registry key TunerID appears under HKEY_LOCAL_MACHINE\Software\Castanet Info and has the correct value.

Creating a macro that allows users to select the installation directory

In this example, you create a macro that allows users to select the installation directory when a packaged application is installed. You can use the Define Macro command from the Package Contents page, or you can choose the Add command from the Macros page. The former method is described in the following example.

- **To create a macro that enables users to select the installation directory**
- 1 Use Application Packager to package an application.
 - 2 To edit the packaged application, open it with the Package Editor.
 - 3 On the editor's Package Contents tab, select the installation directory for which you want to create a macro. In this example, the directory is C:\channels\install.
 - 4 Right-click the directory and choose Define Macro.
 - 5 In the New Macro dialog box, perform the following steps:
 - a For Type, select User Defined.
 - b For Macro Name, specify the name you want, for example, install. For Default Macro Definition, the directory you selected in step 3 (C:\channels\install) appears automatically.
 - c Select the Query user for definition check box, and in the text box, specify the message you want the user to see. For this example, type this message in the text box:

In what directory should the MyApplication files be installed?

6 Click OK.

On the Package Contents page, the name of the new macro you created (\$install) appears in place of the installation directory (C:\channels\install).

This name (\$install) appears on the Package Contents page but is not seen by the people using your channel.

7 In the editor, click the Macros tab.

The new macro you created appears under User-defined Macros with the name \$install, the default definition C:\channels\install, and with the query check box selected.

8 Save the packaged application and publish it.

9 From the endpoint, subscribe to the packaged application, and check that the installer prompts you for the installation directory when installing the application. By default, it is installed under C:\channels\install.

Editing user-defined macros

You can edit the user-defined macros in a channel. You can change any property of the macro except the macro name. To change the macro name, delete the macro and recreate it with the new name.

► To edit a user-defined macro

1 On the Package Editor window, click the Macros tab.

2 Select the user-defined macro you want to edit.

You cannot edit the predefined macros displayed in the Available Macros list.

3 Click Edit.

4 Edit the macro properties.

5 Click OK.

Removing user-defined macros

You can remove user-defined macros from a channel.

► To remove a user-defined macro

1 On the Package Editor window, click the Macros tab.

- 2 To remove user-defined macros, perform one of the following steps:
 - Select the user-defined macro you want to remove, and click Remove.
 - To remove all user-defined macros from the list, click Remove All.

You cannot remove the predefined macros that appear in the Available Macros list.

 - 3 Click Close or Add/Close.

Setting channel startup options

Channels that you create using Application Packager are installed from the tuner. After installation, if you want users to be able to start the channel from the tuner (and from the list of channels in the Windows Start menu), you must set that option in the Package Editor. By default, this option is not enabled.

► **To allow the application to be started from the tuner**

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Startup tab.
- 3 Select the Launch application from Tuner check box.

Note: You can configure the process creation associated with this feature by setting the `processCreationFlags` in the `parameters.txt` file of the package. For more information, see the *Symphony Marimba Client Automation Guide*, available on the Marimba Channel Store.

- 4 For Path, specify the name and path of the executable file that starts the application.

You can include a macro name in the path if necessary, for example, `$macro_name$`.

You can also click the Package Contents tab, right-click the executable file (such as an .exe file or a .cmd file), and choose Set as Launch Path. When you do so, the path of the specified file is automatically added to the Path text box on the Configuration - Startup page.

Note: Packaged applications cannot launch `java.exe`, even if you specify its path in this text box. As a workaround, you can include a batch file in your packaged application that launches `java.exe`.

- 5 To launch the application with any special arguments (for example, starting in a minimized state), for Arguments, specify the arguments. Use a space to separate multiple arguments.
- 6 If the application requires a working directory to be set, for Working dir, specify the name and path of the directory.

You can also use a macro for the working directory.

- 7 If you want the application to run independently from the tuner after it is launched, select Detach application from the Tuner after launching it. If you use this option, the application can continue to run even if you exit the tuner.
- 8 If you want the tuner to manage exiting the application, select Allow Tuner to manage application termination.
- 9 If the application does not respond to an exit request and you want to force the application to exit after a short delay, select the Force application termination after a short delay check box and specify the number of seconds for the delay.

Specifying a delay is useful when you want to use the tuner to exit the application, but you want to allow applications to exit gracefully, or to allow users to save their work before the application is forced to exit.

- 10 Set the following options depending on how you want to run the application:

Allow “-runexe” option to execute any program from the command line—To allow users to execute *any* application using the command-line option `-runexe <application_name>`, where `<application_name>` includes the full path of the application executable or batch file. If you do not select this check box, you can use the command-line option `-runexe` without any arguments to start only the application with the executable file specified in the Path text box.

Run the application in the user's context instead of the tuner's context—(Windows only) To run the application in this channel in the user's context instead of the tuner's context. This option might be useful when running the tuner as a service. By default, applications started using the tuner inherit the tuner's context. Inheriting the tuner's context might cause problems when the tuner is running as a service because the application is not running in the logged-on user's context (and is missing the user's security and identity settings). If you select this option, the application has the rights and permissions of the logged-on user only.

- 11 Select Package->Save to save the channel.
- 12 In the package directory, open the parameters.txt file using a text editor and add one or both of these properties:

```
run.install=true  
run.update=true
```

Note: You can also edit the parameters.txt file through the graphical interface. For more information, see “To edit channel properties and parameters” on page 159.

These properties control whether the specified application should be launched after the initial installation or after updates (both major and minor). For more information, see the *Symphony Marimba Client Automation Reference Guide*, available on the Marimba Channel Store.

Setting installation modes

This section describes the installation modes that you can set for a packaged application. These installation modes determine what appears on the endpoint when a packaged application is installed. For example, you might want to use Full mode for *desktop* endpoints, so that users can respond to dialog boxes to customize the installation. On the other hand, you might want to use Silent mode for *server* endpoints because you do not want any dialog boxes to appear during installation.

You can use the Package Editor to set the following installation modes:

- **Full mode**—Specifies an interactive installation that prompts the user for responses. In this mode, the installer displays a number of dialog boxes and progress bars to the user during installation.

- **Semi-silent mode**—Displays progress bars, but *not* dialog boxes, to the user. Instead of prompting the user for responses, the installer uses the default channel settings.
- **Silent mode**—Displays no dialog boxes or progress bars to the user. The installer uses all of the default channel settings during installation.

Note: Semi-silent or Silent mode overrides any other Application Packager settings that require user interaction. To give users the option to configure applications during installation (for example, to select an installation directory), select Full mode.

- **Preview mode**—Creates a channel that simulates the installation process without actually installing any content. This is useful when you want to perform checks without downloading any data to the endpoint, such as checking for available disk space or operating system.

When you install a packaged channel with preview mode selected, any preinstall scripts and dependency checks are run; however, directories, files, registry entries, metabase entries, and services are not installed.

► **To set the installation modes**

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 Select one of the installation modes: Full mode, Semi-silent mode, or Silent mode. Each mode is described on “Setting installation modes” on page 106.
- 4 Optionally, you can also select the Preview mode check box.
- 5 For information about the other options on this tab, see the following topics:
 - “Enabling installer logging” on page 111
 - “Setting automatic rollback for failed installations” on page 108
 - “Preventing items from being stored for backup” on page 109
- 6 You can also set policies for installation and other operations. For more information, see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.

Setting automatic rollback for failed installations

This section describes how you can set a channel to automatically roll back to its previous state if installation fails. If installation fails when a channel is being installed for the first time, it automatically rolls back the channel to the uninstalled state. Files and other items in the channel that were installed are uninstalled, while items that were removed or modified during installation are restored. If an update to a channel fails, it automatically rolls back the channel to its state before the update.

► To set automatic rollback for a failed installation

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 Select the Auto-rollback on failed installs and updates check box.

Details about automatic rollback

There are two kinds of rollbacks: rollback install and rollback updates. This section describes how each one works and when each one is triggered.

Table 4-2: Rollbacks

Type	When	Result
Install	Triggered when an installation fails, causing the channel to be in a failed state	The channel ends up in an uninstalled state. All items are removed or restored to their previous states regardless of uninstallation policies and properties (such as nobackup). However, no scripts are executed and no dependencies are checked. A channel does not roll back if the postinstall script fails but installation of all other items succeed. If a text modifier fails to install (probably because the file it is trying to edit is read-only), it causes a rollback.
Update	Triggered when a <i>major update</i> fails causing the channel to be in failed state. For more information about major updates, see “Managing major and minor updates” on page 150.	The channel ends up in a rollback state. This state indicates that the channel update has failed and has been brought back to its previous state. However, this state cannot successfully verify, repair, or install. It can only be “updated” with the correct version of the channel to bring it back into an “installed” state. During the rollback of an update, all files overwritten are temporarily backed up so they can be restored. Like rollback install, no scripts are executed and no dependencies are checked during this phase.

Preventing items from being stored for backup

This section describes how you can prevent items from being stored for backup when you install a channel. Usually, when a channel is installed, existing directories, files, registry entries, and metabase entries that are removed or overwritten are stored in a backup directory (the `unfiles` directory of the channel in the tuner's workspace directory). These are used to restore the system to its previous state when the channel is uninstalled. If you select the check box described in this section, those directories, files, registry entries, and metabase entries are not restored if the channel is uninstalled.

This option might be useful to make sure that an application's directories, files, registry entries, and metabase entries do not reside on endpoints after the channel is uninstalled. It also saves some disk space.

WARNING: Selecting this option might cause the removal of files and registry entries that are critical to other applications.

► To prevent items from being stored for backup

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 Select the Don't backup any objects check box.

Delaying the download of a channel's files

This section describes how you can delay the download of a channel's files to the endpoints until any pre-scripts have run. Usually, when subscribing an endpoint to a channel, all the files in the channel are downloaded to the endpoint before running any scripts and starting installation. In some cases, you might want to delay the download of a channel's files to the endpoints. For example, if you use a preinstall script to run some commands or check that some conditions are met before installing an application, you might want to delay the download so that time and bandwidth are not wasted if the commands fail or if the conditions are not met.

You might also want to use this option with the `preload` parameter. You can use these together to prevent files that are already on the endpoint from being unnecessarily downloaded again (see "Minimizing bandwidth usage" on page 159).

If you select the check box described in this section, the channel download occurs in two parts. The initial download includes only the installer and the manifest file (`manifest.ncp`), which contains the code and instructions required to install and update the application. After any pre- scripts have run, the second download brings over the files necessary to install or update the application.

► **To delay the download of a channel's files**

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 Select the Delay download of files until pre scripts have run check box.

Ability to kill a process as a part of pre-post install

During packaging, you can specify one or more processes to kill before or after any of the installation phases.

► **To specify a process to kill:**

- 1 In the Package Editor, click the Configuration=>Process tab.
- 2 Click the Add button to display a dialog.
- 3 Enter the name of the process that you want to kill.
- 4 Select whether to kill the process before or after selected installation phases.
- 5 Choose the installation phases before or after which you want to kill the process.
- 6 To ignore the exit/return code, click the supplied option.
- 7 Click OK.

If only one instance of the process is running on the target system, the process is killed without notification.

If more than one instance of the process is running on the target system, a notification message asks you if you want to kill all instances of the process.

If the specified process is not running on the target system, a warning message displays in the channel log.

Enabling installer logging

You can enable logging when the application you packaged is installed. You can also configure when the log rolls over by setting a schedule or by specifying a file size.

Each packaged application, or channel, that the tuner has subscribed to has history log files. These log files record channel events, such as when a channel is started or updated. They also record information about the installation of the packaged application, including the failure to install objects. These files are located in each channel's directory inside the tuner's workspace directory. The channels' directories are numbered by the order in which the tuner subscribed to the channels. (The first channel subscribed to would have the `ch.1` directory, the second one the `ch.2` directory, and so on.)

The log files are named by the order in which they are created. For example, the channel's first log file is named `history-1.log`, the second one `history-2.log`, and so on.

Note: Enabling logging in the Package Editor overrides the tuner's default logging options for individual channels.

► To enable installation logging and configure when the log rolls over

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 Select the Enable installer logging check box.

You cannot configure log rollover options if installer logging is not enabled.

- 4 To set a regular schedule for rolling over the installer log, select the Roll over option and choose one of the following from the drop-down list:

- Hourly
- Daily
- Weekly
- Monthly
- Never

- 5 To set a file size that the installer log should reach before rolling over, select the Roll over at option and specify a file size in kilobytes (KB).

Reading a channel's history-n.log files

The log files begin with a header that includes the following information:

- **Host**—Name of the host machine on which the tuner is installed.
- **Created**—Date the log was created. You can ignore the long code number at the beginning of the line.
- **Version**—Version of the Symphony Marimba Client Automation technology used to generate the log. (This entry is not Inventory-specific and can be ignored in most cases.)
- **Roll**—Log-rolling policy for this log file, including the frequency of rolling, the maximum size of the log file, and the number of previous log files that can exist.

Consider this example:

```
Host: acme
Created: 1030568030617; Wed Aug 28 13:53:50 PDT 2002
Version: 0
Roll: bysize,32;1
```

The actual log entries vary depending on the event recorded. Usually, they include the following information about each entry:

- The date and time
- The severity level
- The user ID
- The log ID
- The log message
- Additional information (usually preceded by #)

For more information about the log IDs and severity levels, see the logging codes chapter in the *Symphony Marimba Client Automation Reference Guide*, which is available on the Marimba Channel Store.

Here are some examples of log entries:

```
[28/Aug/2002:13:53:50 -0700] - audit jeff 1100 Channel created
[28/Aug/2002:13:53:50 -0700] - audit jeff 1101 Channel update started
```

```
[28/Aug/2002:13:53:51 -0700] - audit jeff 1107 Channel index  
installed: <index id="urn:idx:odr+YbY/zG/8NMpZNLiyWw=="  
size="1635186">  
[28/Aug/2002:13:53:51 -0700] - audit jeff 1110 Channel checkpoint  
created: undoo  
[28/Aug/2002:13:53:51 -0700] - audit jeff 1102 Channel update  
finished: 172.16.4.90:5282: [Files changed=36, downloaded=13, bytes  
received=24050/190425]  
[28/Aug/2002:13:53:51 -0700] - audit jeff 1150 Channel instance  
started  
#installing rollback  
[28/Aug/2002:13:53:52 -0700] - audit jeff 9026 Adapter started  
[28/Aug/2002:13:53:52 -0700] - audit jeff 9227 The adapter will  
execute in the specified mode: 0, INSTALL  
[28/Aug/2002:13:53:52 -0700] - audit jeff 9020 Begin Install: http://  
acme:5282/MyApplication  
#[28/Aug/2002:13:53:55 -0700]  
#mode is: INSTALL  
[28/Aug/2002:13:53:57 -0700] - audit jeff 9002 Preparing to install  
[28/Aug/2002:13:53:57 -0700] - audit jeff 9008 Backup phase  
[28/Aug/2002:13:53:57 -0700] - audit jeff 9009 Install phase: http://  
acme:5282/MyApplication  
#preinstall object: F:\Install Directory op=2, ADD succeeded  
#preinstall object: F:\Install Directory\New Folder op=2, ADD  
succeeded  
[28/Aug/2002:13:53:58 -0700] - audit jeff 9231 Object operation to be  
performed: F:\Install Directory\New Folder\New Worksheet.xls, op: 2,  
ADD  
#install object: F:\Install Directory\New Folder\New Worksheet.xls  
op=2, ADD succeeded  
[28/Aug/2002:13:53:58 -0700] - audit jeff 9012 Object installed:  
F:\Install Directory\New Folder\New Worksheet.xls  
[28/Aug/2002:13:53:58 -0700] - audit jeff 9231 Object operation to be  
performed: F:\Install Directory\New Folder\New Document.doc, op: 2,  
ADD  
#install object: F:\Install Directory\New Folder\New Document.doc  
op=2, ADD succeeded  
[28/Aug/2002:13:53:58 -0700] - audit jeff 9012 Object installed:  
F:\Install Directory\New Folder\New Document.doc  
#postinstall object: F:\Install Directory op=2 succeeded  
#postinstall object: will op=0 succeeded  
[28/Aug/2002:13:53:58 -0700] - audit jeff 1120 File map updated  
[28/Aug/2002:13:53:58 -0700] - major jeff 9055 Install succeeded  
[28/Aug/2002:13:53:58 -0700] - audit jeff 9000 Operation complete:  
http://acme:5282/MyApplication, mode: 0, INSTALL  
[28/Aug/2002:13:53:58 -0700] - audit jeff 9030 Adapter finished:  
http://acme:5282/MyApplication  
#[28/Aug/2002:13:53:59 -0700]  
###adapter stopped###
```

Setting system reboot options (Windows only)

Some applications require you to reboot the system after installation. For example, in Windows, a reboot is needed to overwrite locked files. Locked files might be DLL files that have been locked into memory by processes that have loaded them or files that have been locked by applications that are using them.

For tuner version prior to version 8.0, Application Packager prompts for reboots on the endpoints based on the settings made in the Package Editor Window.

For tuners of version 8.0 or later, the tuner prompts for reboots based on the reboot settings in Application Packager's Package Editor Window as well as the Common Reboot Service settings on the endpoint.

Note: The reboot dialog box might appear on a user's machine only if reboot and snooze interaction is enabled on the endpoint tuner.

► To set system reboot options

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 Select one of the system reboot options:

Allow installer to determine if reboot is required—Determines if a reboot is required using the indicators described in “Detecting when reboots are needed” on page 115. This option is the default setting.

Always reboot after successful installation—Always reboots the system after installation.

- 4 If you selected one of these two options, you can control the user interaction for performing reboots by using the following options:

■ **Show button on dialog to reboot the machine**—Informs users of the required system reboot. The Restart Required dialog box displays with Restart, Info, and Snooze buttons, allowing users to reboot immediately or postpone the reboot.

If the Reboot dialog box cannot be displayed because reboot interaction is disabled at the endpoint, the reboot is postponed.

- **Show button on dialog allowing the user to cancel the reboot**—Informs users of the required system reboot without actually rebooting the machine for them. Users see a dialog box that indicates a reboot is required, but the reboot does not take place automatically; it waits for users to click OK. (The dialog box does not contain a Cancel button.)
If the Reboot dialog box cannot be displayed because the installation is in Silent mode, no reboot is performed.
- If both check boxes are selected, the Restart Required dialog box displays with the Restart, Info, Snooze, and Cancel buttons. These options allow users to reboot immediately, postpone the reboot, or cancel the reboot.
If the Reboot dialog box cannot be displayed because reboot interaction is disabled at the endpoint, the reboot is postponed.
- If neither check box is selected, the reboot is performed automatically. The Restart Required dialog box displays a restart countdown that allows users to save their work in any open applications.
If the Reboot dialog box cannot be displayed because reboot interaction is disabled at the endpoint, the reboot occurs automatically.
- **Don't reboot after successful installation even if a reboot is necessary**—Prevents any reboot from taking place, even if it is necessary for completing the installation.

Notes:

- If you snooze a reboot for a channel and then delete the channel before the scheduled snooze time, the reboot dialog will still pop up.
- If the installation of a channel that requires a reboot fails and you uninstall the failed channel, you may still get the reboot dialog.

Detecting when reboots are needed

If a script in a packaged application happens to copy over locked files or delete locked files with a Windows API call, the adapter (part of the Application Installer) makes sure that it detects this situation by searching the registry keys, values, and INI files described in Table 4-3 on page 116.

The adapter stores the values of each item that it uses to determine if a reboot is necessary. Later, it compares those values with the values it retrieves after the scripts (and install, update, repair, or rollback) have completed. If differences are detected, the adapter indicates that a reboot is needed. From there, the reboot options specified take effect and determine if a reboot should occur.

Table 4-3: Reboot indicators

Reboot indicator	Description	Platform
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	This registry key indicates what is executed on the next reboot of Windows.	Windows
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run	This registry key indicates what is executed on the next reboot of Windows and on the next login of the current user.	Windows
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce	This registry key indicates what is executed on the next reboot of Windows. Unlike the Run key, the entries are a one-time event.	Windows
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce	This registry key indicates what is executed on the next reboot of Windows and on the next login of the current user. Unlike the Run key, the entries are a one-time event.	Windows
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations	This registry value is modified by the Windows API, MoveFileEx. The API adds entries to this multi-string entry every time a locked file needs to be copied over or removed on the next reboot.	Windows NT or later
<Windows directory>\wininit.ini, [rename] section	To copy over locked files or to remove them on the next reboot, an entry of the form: DestinationFileName=Source FileName needs to be added to the INI file section named rename.	Windows 9x

Using scripts to trigger reboots

To trigger a reboot using scripts, the scripts must change the registry keys and values described in “Detecting when reboots are needed” on page 115. Batch files or executables, on the other hand, must change the specified registry keys and values programmatically using the following methods:

- Directly editing the keys and values.

- Calling 32-bit Windows APIs, such as MoveFile(...), that add entries to these registry keys and values.

For Windows 9x, the script must programmatically add entries to the rename section of the wininit.ini file.

If you are using Java and IScript, you must change the same registry keys and values as the batch files and executables. In addition, the script can grab the IAdapterContext interface from the IScriptContext. The following sample code shows what needs to be added to the Invoke() method:

```
IScriptContext scriptContext;  
IAdapter adapter = (IAdapter) scriptContext.getFeature("adapter");  
adapter.scheduleReboot(true);
```

For more information about using IScript, see “Customizing a channel by using scripts and classes” on page 141.

Specifying the installation platform

You can use the Package Editor to specify the platforms or operating systems on which you will install channels. By default, the operating system on which you packaged the channel is selected.

► To select the platform on which you will install a channel

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Platform page, select the operating system on which you will install the channel. In Windows, select a specific Windows platform.
- 3 Select the Allow installation of this package on newer releases of the selected platform option, if you want to install the channel on all the higher versions of the selected operating system.

For example, if you specify Windows XP as the operating system and if this option is selected, then the channel is installed on Windows XP OS and all higher versions of the Windows XP OS such as Windows 2008, Windows Vista, and Windows 7. If you want to implement a flat hierarchy model, clear the Allow installation of this package on newer releases of the selected platform option.

Note: By default, the **Allow installation of this package on newer releases of the selected platform** option is checked. The packages are installed on the specified operating system and all the higher versions of the specified OS. When this option is not selected, the segmentation hierarchy is set to flat for this specific segment of the package.

When pre-8.2.01 packages are upgraded to 8.2.01 version, the packages by default follow the default segmentation hierarchy.

For non-Windows platforms, select the appropriate architecture.

Redirection behaviour

The Application Packager handles redirection of file deployments on 64-bit endpoints while installation is performed on any of the following directories:

- System32 and SysWOW64 folders in C:\Windows
- Program Files and Program Files (x86) folders in C:\
- Registry Hive inside WoW6432Node and outside WoW6432Node

System32 and SysWOW64 folders in C:\Windows

The redirection behaviour of files by Application Packager on System32 and SysWoW64 folders in the C:\Windows directory is described as follows:

- Application Packager deploys the 64-bit packages to the System32 folder.
- Application Packager deploys the 32-bit packages to the SysWOW64 folder.

Note: Even though the files in packages are specified to install inside the System32 folder, the operating system redirects the files and installs them in the SysWOW64 folder.

However, you can control the redirection behavior by making the required configuration settings in Application Packager.

Note: By default, the redirection feature is enabled for a 64-bit package, and disabled for a 32-bit package.

To control the redirection behaviour, in Application Packager, navigate to Package Editor > Configuration > Platform, and make the required changes. If you do not select the redirection option, the operating system deploys the files by redirecting them from the System32 folder to the SysWOW64 folder. If you select the redirection option, the operating system prevents the files from being redirected to the SysWOW64 folder and instead deploys the files in the System32 folder.

Limitations

- If redirection to System32 folder option is not selected, and if a 32 or 64 bit package is installed on a 64-bit tuner, the Application Packager installs the packages in System32 folder. Redirection is controlled only for 32-bit applications which means that files can be redirected from the SysWoW64 to the System32 folder. On a 64-bit operating system having the 64-bit tuner, you cannot redirect files from a System 32 folder to the SysWOW64 folder.
- Redirection from SysWow64 to System 32 fails in Windows 8 and Windows 2012 because of a Windows defect. You can find more information about this defect at the following location:
[http://msdn.microsoft.com/en-us/library/windows/desktop/aa365743\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa365743(v=vs.85).aspx)

Program Files and Program Files (x86) folders in C:\

The redirection behaviour of files deployed by Application Packager on ProgramFiles and Program Files (x86) folders in the C:\ drive is described as follows:

- Application Packager deploys the files of 64-bit packages to the Program Files folder.
- Application Packager deploys the files of 32-bit packages to the Program Files (x86) folder.

You can control the redirection behavior by making the required configuration settings in Application Packager. To control the redirection behaviour, in Application Packager, navigate to Package Editor > Configuration > Platform and make the required configuration changes.

Note:

- By default, the redirection feature is enabled for a 64-bit package, and disabled for a 32-bit package, which means that a 64-bit package is deployed to the Program Files folder and a 32-bit package is deployed to the Program Files (x86) folder.
- Application Packager handles the redirection between Program Files and Program Files (x86) folders only when the target location is specified with the \$SYS.PROGRAMFILES macro which is available from 8.2.02.001.

Registry hive inside WoW6432Node and outside WoW6432Node

The redirection behaviour of files by Application Packager on registry hive inside and outside WoW6432Node is described as follows:

- Application Packager deploys the 64-bit packages outside the WoW6432Node registry hive.
- Application Packager deploys the 32-bit packages files inside the WoW6432Node registry hive.

You can control the redirection behavior by making the required configuration settings in Application Packager. To control the redirection behaviour, in Application Packager, navigate to Package Editor > Configuration > Platform and make the required configuration changes.

Note: By default, the redirection feature is enabled for a 64-bit package, and disabled for a 32-bit package, which means that a 64-bit package is deployed outside the WoW6432Node registry hive and a 32-bit package is deployed inside the WoW6432Node registry hive.

Setting policies for installation, update, uninstallation, verification, and repair

This section describes how to set policies that determine whether files and the other contents of a channel are installed, updated, uninstalled, verified, or repaired. Using the Package Editor, you can set policies at two levels:

- You can set policies as the *default for all the contents* of a channel.
- You can also override the default policies and set policies for *individual items* in a channel.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

Selecting policies for channels

Selecting install, update, and uninstall policies for your channels depends on the following factors:

- Whether you want to overwrite existing and shared files at the endpoints.
- Whether you want to uninstall existing and shared files at the endpoints.
- How much control over channels you want to give users at the endpoints.
- Whether you want to prompt users at the endpoints when installing or updating channels.
- Whether you want to allow the users to postpone installation to a later period of time

You can use verify and repair features to check and, if necessary, re-install objects in a channel. The verify function gathers information, but takes no action. The repair function updates corrupted files.

You can use the `to customize, update, or change` objects after installation—but verify and repair can overwrite those changes. To prevent this, you can set policies for files and other objects.

Determine whether you want to verify policies for channels, based on your answers to the following questions:

- If you customize or change files, directories, and other objects during or after installation, do you want repair operations to overwrite your customizations or changes with the original versions?

- Do you want users to edit selected files, but still be able to use the verify and repair feature on other files?

In general, you want to set default policies for the entire channel. Then, if necessary, you can override those defaults and set policies for individual items in the channel.

In addition, remember the following guidelines when setting policies:

- Different items in the channel have different policies available. For example, you can set uninstallation and verification policies for files, but not for text modifiers.
- Policies do not apply to environment variables.
- Any changes to the policies, whether global or object-level, are considered major updates. See “Managing major and minor updates” on page 150 for more information.
- For policies, attributes the application checks include the RHSA (read-only, hidden, system, and archive) attribute flags, the last modified time, and the creation time.
- Editing a file modifies its attributes (specifically, the last modified time). Therefore, verifying a file’s attributes fails if the file’s contents have been modified.
- When you set policies for container objects (such as directories, registry keys, and metabase keys) you are actually setting the policies for the child objects in them (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs). Container objects do not follow the policies, but the child objects can inherit the policies. For example, when you set the installation policies for a directory, the name of the Install Policy tab is “Install policy for this directory’s contents.”

Specifying whether user is allowed to postpone installation

You can also specify whether the user is allowed to postpone the installation of the package to a later period of time. With this the users can decide appropriate time for installation of packages based on their convenience.

► To specify whether user is allowed to postpone installation

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Policies tab.

- 3 Click the Install tab.
- 4 Use the Allow User to postpone Installation option to decide the postpone behavior. Select this option if the user is allowed to postpone installation of the package.
 - a You can configure the maximum postpone time based on:
 - Until a specified time – This will allow users to postpone installation any number of times until the specified time exceeds (format: mm/dd/yyyy hh:mm a).
 - Until specified number of attempts – This will allow users to postpone installation only upto specified number of times
 - b You can also add a custom message to be displayed to the user, in the postpone dialog.

Note:

- When package is configured to allow users to postpone installation, the postpone dialog is displayed irrespective of the package UI mode(fully interactive, semi-interactive or silent)
- The postpone option is applicable only during installation of the package(for packages in “install-pending” or “uninstalled” states).

Setting default policies for channels

You can use the Package Editor to set default policies for all items in the channel.

► To set default policies for a channel

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Policies tab.

The six tabs (Install, Update, Uninstallation, Verify Installed, Verify Not Installed, and Repair) show the different types of policies you can set for the channel.

- 3 On each of the available tabs, select a policy.

Policies for verifying and repairing channels

You can use the Package Editor to set the verify and repair policies for a channel. The policies you select apply to all objects (including files and registry entries) in a channel. However, you can override them by setting policies for individual objects.

You can set three types of policies:

- **Verify Installed**—Take effect during the verify and repair phases for objects that were installed by a channel (created using Application Packager) at one point. An object can be installed during installation, update, or repair of a channel.
- **Verify Not Installed**—Take effect during the verify and repair phases for objects that were never installed by a channel (created using Application Packager). For example, if a newer version of the file is already installed on the endpoint and there is an older version in the channel, the newer version is not overwritten by the older one if the installation policy Install if object is newer—Compare objects based on versions is selected.
- **Repair**—Take effect during the repair phase for objects.



Note: Make sure the relationship between the verify and repair policies makes sense when you set them. The verify policies determine whether a repair takes place. As a best practice, your repair policies should match your verify policies. Otherwise, you might see some unintended behavior. Repairing an object does not guarantee that verifying the object succeeds the next time it runs.

Setting policies for individual items in a channel

When you set policies for individual items in a channel, you override the default policies set for the entire channel.

When you set policies for container objects (such as directories, registry keys, and metabase keys) you are actually setting the policies for the child objects in them (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs). For example, when you set the installation policies for a directory, the name of the Install Policy tab is “Install policy for this directory’s contents.” Container objects get these policies only:

- Always install

- Always update
- Uninstall if possible (Container objects are not uninstalled unless they are empty of all child objects.)
- Always verify

► To set a policy for an individual item in a channel

- 1 On the Package Editor window, click the Package Contents tab.
- 2 Select the item for which you want to set a policy.

You might need to expand directories, registry keys, metabase keys, or text modifier groups to find a file, registry entry, metabase entry, or text modifier.

- 3 Right-click the item and choose Properties.

The item's Properties dialog box appears. Depending on the item you selected, you see some of the following tabs:

- Install
- Update
- Uninstall
- Verify Installed
- Verify Not Installed
- Repair

These tabs show the types of policies you can set for the item.

- 4 Click each tab and select a policy for the item.
- 5 Click OK.

Determining whether files are restored or removed during uninstallation

To determine what happens during uninstallation, check the global uninstallation policies (select the Configuration – Policies – Uninstallation tab) or the uninstallation policies for each individual object (right-click the object, choose Properties, and select the Uninstall Policy tab), depending on how policy inheritance is set up. This section describes what happens during uninstallation for the various policies:

- **Uninstall object if it was installed, prompt user before removing shared objects (available in Windows only), or Uninstall object if it was installed**

If one of these policies is selected, an uninstall restores the file to the state it was in when it was first installed by the channel. (A file can be installed by the channel for the first time on an install or an update.) However, if the property `nobackup=true` is set, an uninstall always removes the file from the endpoint (if it was installed previously).

Example 1

File A is not on the endpoint. An install adds file A version 1. An update updates file A to version 2. An uninstall removes file A.

Example 2

File A version 1 is on the endpoint. An install updates file A to version 2. An update updates file A to version 3. An uninstall restores file A back to version 1. However, if the property `nobackup=true` is set, an uninstall removes file A.

- **Always uninstall existing object**

If this policy is selected, an uninstall restores the file to the state it was in when the channel was first deployed, whether the file was actually installed by the channel or not. However, if the property `nobackup=true` is set, an uninstall always removes the file from the endpoint.

Example 1

File A version 1 is on the endpoint. An install does not install file A version 2. An update does not install file A version 3. An uninstall leaves file A version 1 on the endpoint.

Example 2

File A version 1 is on the endpoint. An install does not install file A version 2. A user manually overwrites file A version 1 with file A version 3. An uninstall causes file A version 3 to revert back to file A version 1. However, if the property nobackup=true is set, an uninstall always removes the file from the endpoint.

Example 3

File A version 1 is not on the endpoint. An install does not add file A version 1. A user manually adds file A version 2. An uninstall removes file A version 2.

- **Always remove existing object**

If this policy is selected, an uninstall always removes the file from the endpoint if it exists.

Example

File A version 1 is on the endpoint. An install does not install file A version 2. A user manually overwrites file A version 1 with file A version 3. An uninstall removes file A version 3.

- **Don't uninstall object**

If this policy is selected, an uninstall never removes the file or restores the file on the endpoint to its previous state.

Removing all items from endpoints during uninstallation of a channel

If you want all files, directories, registry keys, and registry key values to be removed during uninstallation of a channel, set the global uninstallation policy to Always remove existing object. One exception is that directories and registry keys are not removed if they previously existed prior to their initial installation.

Using environment variables

You can include environment variables in your channel by specifying them in the Package Editor. You can:

- Create new environment variables in your channel and add them to the target system during installation.

- Edit standard environment variables, such as appending a new directory to the PATH variable.
- Include macros you created using the Application Packager.

Note: The policies do not apply to environment variables. Default policies cannot be edited for environment variables.

Adding an environment variable

You can add environment variables to your channel, and they are added to the target system when the channel is installed. You have access to these variables immediately for use in your scripts and for your application. Other applications have access to the variables after you restart the system.

► To create a new environment variable

- 1 On the Package Editor window, click the Environment tab.
If the application you are packaging can be installed on more than one platform, you can create environment variables for each platform. Click the tab that corresponds to the platform for which you want to create environment variables.
- 2 Click Add.
- 3 In the Variable Name box, specify a name for the environment variable you want to add.

Environment variable names are not case-sensitive.

- 4 In the Value box, specify the contents of the environment variable.

For example, to append the PATH variable for the new Whammo application from Acme Corp., type:

```
%Path%;C:\Program Files\Acme\Whammo
```

- 5 Click OK.

The new variable appears in the list of environment variables.

Editing an environment variable

You can edit an environment variable that you previously created. You can change contents (value) of the variable but you cannot change its name. To change the variable name, you must delete the variable and recreate it with the new name.

► To edit an environment variable

- 1 On the Package Editor window, click the Environment tab.

If the application you are packaging can be installed on more than one platform, you can have environment variables for each platform. Click the tab that corresponds to the platform for which you want to edit environment variables.

System environment variables—Administrators can change or add environment variables that apply to the system, and thus to all users of the system.

User environment variables—The user environment variables are different for each user of a particular computer. The variables include any that are set by the user, as well as any variables defined by applications, such as the path to the location of the application files.

- 2 Select the variable you want to edit and click Edit.
- 3 In the Value box, specify your changes to the environment variable.
- 4 Click OK.

The edited variable value appears in the list of environment variables.

Removing an environment variable

You can remove any of the environment variables that appear in the Package Editor list.

► To remove an environment variable

- 1 On the Package Editor window, click the Environment tab.

If the application you are packaging can be installed on more than one platform, you can have environment variables for each platform. Click the tab that corresponds to the platform for which you want to remove environment variables.

- 2 To remove variables, perform one of the following steps:
 - Select the variable you want to remove, and click Remove.
 - To remove all variables from the list, click Remove All.
- 3 Click Close or Add/Close.

Environment variables on different Windows platforms

If the application you are packaging is intended for Windows NT, Windows 2000, or Windows XP, you can have both system and user environment variables. Click the System tab or the User tab, depending on which type of environment variable you want.

When you are installing a package to a Windows NT, 2000, or XP endpoint and the tuner on that endpoint is running as a non-NT service (just a regular are installed to the user environment variables of the Windows NT, 2000, or XP endpoint. The environment variables on the Windows 95/98 tab override any user environment variables on the Windows NT/2000 tab for a package. For example, if both tabs have a variable named `myvar` and the Windows 95/98 tab has `win95value` and the Windows NT/2000 tab has `winntvalue`, `win95value` overrides `winntvalue` when the `myvar` variable is installed on a Windows NT, 2000, or XP endpoint.

Configuring system and channel dependencies

You can use the Package Editor to configure system and channel dependencies for a channel. Dependencies are checked before downloading the channel. If any of the configured dependencies cannot be validated, the channel is not downloaded, installation of the channel fails, and, depending on how the channel is configured, the user at the endpoint is informed of the failure.

You can configure these system dependencies:

- The amount of system memory (in megabytes) required
- The type of processor required
- The video resolution required
- The connection speed required

- Whether an environment variable, which is part of System variables or User variables, is required to exist or not exist at the endpoint, and whether the environment variable is required to contain a certain value
- The minimum amount of disk space (in megabytes) required for a specific drive
- Whether a file is required to exist or not exist at the endpoint, and whether the file is required to have a certain size, date, or version
- Whether a registry key is required to exist or not exist at the endpoint, and whether the registry key is required to contain a certain value

You can also configure channel dependencies by creating a list of channel URLs. You can require that:

- Channels be present
- Channels not be present

In UNIX, only file dependencies and channel dependencies are available.

Note: You might want to use the feature for delaying file downloads with dependencies. For example, you might want to download files only after you have checked the dependency for disk space. For more information, see “Delaying the download of a channel’s files” on page 109.

► To set system and channel dependencies

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Dependencies tab.

You can use the Dependencies page to specify the following requirements for installing the channel successfully.

- system memory (in megabytes)
 - processor
 - video resolution
 - connection speed (in kilobytes per second)
- Note: The connection speed is validated against the marimba.bandwidth.max tuner property
- environment variable requirements

For more information, see “Configuring environment variables requirement” on page 134 .

-disk space requirements (For more information, see “Configuring disk space requirements” on page 132.)

-files (For more information, see “Configuring file requirements” on page 133.)

-registry entries (For more information, see “Configuring registry entry requirements” on page 136.)

-channels (For more information, see “Configuring channel requirements” on page 138.)

- 3 If you want dependencies to be checked during major updates, select the Check dependencies on major updates check box. Otherwise, dependencies are checked when installing channels only.
- 4 You can also specify if you want dependencies to be checked during minor updates, verification, or repairs. For more information, see “Configuring file requirements” on page 133, “Configuring registry entry requirements” on page 136, “Configuring channel requirements” on page 138, and “Configuring environment variables requirement” on page 134.

Configuring disk space requirements

You can use the Package Editor to configure the disk space requirements for a channel. The Disk Space Requirements dialog box lists the drives and the minimum space on each drive required by the channel.

► To configure disk space requirements

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Dependencies tab.
- 3 Click Add or Edit beside Required disk space.
- 4 To add a disk space requirement, perform the following steps:
 - a From the Drive list, select a drive.
 - b In the Minimum space box, specify the minimum amount of space required for the channel.
 - c Click Add.

The new disk space requirement appears in the list.

- 5 To edit a disk space requirement, perform the following steps:
 - a From the Disk space requirements list, select the requirement you want to edit.
 - b In the Minimum space box, edit the minimum amount of space required for the channel.
 - c Click Add.
- The edited disk space requirement appears in the list.
- 6 To remove disk space requirements, perform one of the following steps:
 - Select the disk space requirement you want to remove, and click Remove.
 - To remove all disk space requirements from the list, click Remove All.
- 7 Click Close or Add/Close.

Configuring file requirements

You can use the Package Editor to configure the file requirements for a channel. The Required Files dialog box lists:

- Files that *must* exist
- Files that must *not* exist
- Files that must be a certain size
- Files that must have a certain timestamp
- Files that must have a certain version

► To configure file requirements

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Dependencies tab.
- 3 Click Add or Edit beside Required files.
- 4 To add a file requirement, perform the following steps:
 - a In the Path box, specify the name and path of the file.
 - b Set the existence, and optionally, the size, date and version requirements for the file.

- c If you want the file requirement to be an OR dependency, select the OR check box. If you want it to be an AND dependency, leave the OR check box unchecked. For more information, see “Combining multiple requirements by using AND and OR dependencies” on page 140.

- d Click Add.

The new file requirement appears in the list.

- 5 Specify when you want this dependency to be checked (aside from the first-time installation):

- Run on Minor Update
- Run on Verify
- Run on Repair

- 6 If you want the dependency to be checked on major updates, select the Check dependencies on major updates check box on the Configuration - Dependencies page. Otherwise, dependencies are checked when installing channels only.

- 7 To edit a file requirement, perform the following steps:

- a From the Required files list, select the file you want to edit.
 - b Edit the existence, size, and date requirements for the file.
 - c Click Add.

The edited file requirement appears in the list.

- 8 To remove file requirements, perform one of the following steps:

- Select the file requirement you want to remove, and click Remove.
- To remove all file requirements from the list, click Remove All.

- 9 Click Close or Add/Close.

Configuring environment variables requirement

You can use the Package Editor to configure the environment variable requirements for a channel. The Required Environment Variable dialog lists:

- Environment variables that must exist
- Environment variables that must not exist
- Environment variables that must have a certain value

► To configure environment variable requirements

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Dependencies tab.
- 3 Click Add or Edit beside Required environment variables.
- 4 To add an environment variable requirement, perform the following steps:
 - a In the Environment Variable box, specify the name of the environment variable.
 - b Select the appropriate value from the "Is part of variables for" drop down. The specified environment variable will be searched under the selected option("System" variables or "User" variables)
 - c Set the existence, and optionally, the value requirements for the environment variable.
 - d If you want the environment variable requirement to be an OR dependency, select the OR check box. If you want it to be an AND dependency, leave the OR check box unchecked. For more information, see “Combining multiple requirements by using AND and OR dependencies” on page 140.
 - e Click Add.

The new environment variable requirement appears in the list.

- 5 Specify when you want this dependency to be checked (aside from the first time installation):
 - Run on Minor Update
 - Run on Verify
 - Run on Repair
- 6 If you want the dependency to be checked on major updates, select the Check dependencies on major updates check box on the Configuration - Dependencies page. Otherwise, dependencies are checked when installing channels only.
- 7 To edit an environment variable requirement, perform the following steps:
 - a From the Required environment variable list, select the environment variable you want to edit.
 - b Edit the existence and the value requirements for the environment variable.

- c Click Add.

The edited environment variable appears in the list.

- 8 To remove environment variable requirements, perform one of the following steps:
 - a Select the environment variable requirement you want to remove, and click Remove.
 - b To remove all environment variable requirements from the list, click Remove All.
- 9 Click Close or Add/Close.

Configuring registry entry requirements

You can use the Package Editor to configure the registry entry requirements for a channel. The Required Registry Entries dialog box lists:

- Registry keys that *must* exist
- Registry keys that must *not* exist
- Registry keys that must have a certain value (value name, value type, and value)

The following value types are supported:

- REG_SZ—A single-line string
- REG_EXPAND_SZ—A single-line string with macros
- REG_BINARY—A hexadecimal binary value
- REG_DWORD—A 32-bit integer in hexadecimal format
- REG_MULTI_SZ—A string in hexadecimal, 16-bit Unicode format

► To configure registry entry requirements

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Dependencies tab.
- 3 Click Add or Edit beside Required registry entries.
- 4 To add a registry entry requirement, perform the following steps:
 - a In the Required Key box, specify the registry key.

- b Set the existence, and optionally, the value requirements for the registry entry. You must specify the value name before you can specify the value type and value.
- c If you want the registry entry requirement to be an OR dependency, select the OR check box. If you want it to be an AND dependency, leave the OR check box unchecked. For more information, see “Combining multiple requirements by using AND and OR dependencies” on page 140.
- d Click Add.

The new registry entry requirement appears in the list.

- 5 Specify when you want this dependency to be checked (aside from the first-time installation):

- Run on Minor Update
- Run on Verify
- Run on Repair

- 6 If you want the dependency to be checked on major updates, select the Check dependencies on major updates check box on the Configuration - Dependencies page. Otherwise, dependencies are checked when installing channels only.

- 7 To edit a registry entry requirement, perform the following steps:

- a From the Required registry entry list, select the registry entry you want to edit.
- b Edit the existence and value requirements for the registry entry.
- c Click Add.

The edited registry entry appears in the list.

- 8 To remove registry entry requirements, perform one of the following steps:

- Select the registry entry requirement you want to remove, and click Remove.
- To remove all registry entry requirements from the list, click Remove All.

- 9 Click Close or Add/Close.

Configuring channel requirements

You can use the Package Editor to configure the channels that must or must not be present (subscribed or installed on the tuner) before installing a channel. The behavior is as follows:

- If you specify a requirement that a channel must be present and the required channel is not present on the tuner, the Application Installer (for more information, see “Application Installer” on page 30) attempts to subscribe or install it. If Application Installer cannot subscribe or install the required channel, the Application Installer fails the installation and displays an error through the GUI or in the log file.
- If you specify a requirement that a channel must not be present and the channel is present on the tuner, the Application Installer fails the installation and displays an error through the GUI or in the log file.

The following examples show the expected behavior for required channels:

- If Channel A requires Channel B, where Channel B is a channel created using Application Packager, the Application Installer makes sure that Channel B is installed on the tuner before installing Channel A. If Channel B is not yet installed on the tuner, the Application Installer installs it. If Channel B cannot be installed (that is, installation is aborted or fails), Channel A is not installed.
- If Channel A requires Channel C, where Channel C is not a channel created using Application Packager (for example, a Symphony Marimba Client Automation channel like the Publisher), the Application Installer makes sure that Channel C is subscribed on the tuner before installing Channel A. If Channel C is not yet subscribed on the tuner, the Application Installer subscribes the tuner to it.
- If Channel A requires that Channel D is not present, the Application Installer makes sure that Channel C is not subscribed or installed on the tuner before installing Channel A. If Channel C is subscribed or installed on the tuner, the Application Installer fails the installation of Channel A and displays an error through the GUI or in the log file.

You can also add a new parameter in the channel’s `parameters.txt` file to control installation. For more information about the `parameters.txt` file and the other contents of the package directory, see “Package directory” on page 28 and “Editing the channel parameters and properties” on page 158.

If you set `dependChannelSilent` to true, which is the default, the required channels are subscribed and installed in silent mode. If you set `dependChannelSilent` to false, the required channels are subscribed and installed using the installation options you set using the Package Editor.

Note: If a channel that is required to install another channel is not already subscribed, the tuner attempts to subscribe it.

► To configure channel requirements

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Dependencies tab.
- 3 Click Add or Edit beside Required Symphony Marimba Client Automation channels.
- 4 To add a channel requirement, perform the following steps:
 - a In the Channel URL box, specify the URL of the channel.
 - b Specify whether the channel must or must not be present.
 - c If you want the channel requirement to be an OR dependency, select the OR check box. If you want it to be an AND dependency, leave the OR check box unchecked. For more information, see “Combining multiple requirements by using AND and OR dependencies” on page 140.
 - d Click Add.The new channel requirement appears in the list.
- 5 Specify when you want this dependency to be checked (aside from the first-time installation):
 - Run on Minor Update
 - Run on Verify
 - Run on Repair
- 6 If you want the dependency to be checked on major updates, select the Check dependencies on major updates check box on the Configuration - Dependencies page. Otherwise, dependencies are checked when installing channels only.
- 7 To edit a channel requirement, perform the following steps:

- a From the channel list, select the channel requirement you want to edit.
- b Edit the URL or the requirement that the channel be present.
- c Click Add.

The edited channel requirement appears in the list.

- 8 To remove channel requirements, perform one of the following steps:
 - Select the channel requirement you want to remove, and click Remove.
 - To remove all channel requirements from the list, click Remove All.
- 9 Click Close or Add/Close.

Combining multiple requirements by using AND and OR dependencies

You can combine multiple file requirements, registry entry requirements, and channel requirements as AND or OR dependencies. For a channel with dependencies to be installed:

- All AND dependencies must be satisfied
- At least one OR dependency must be satisfied

All AND dependencies are evaluated as one expression. If all AND dependencies are satisfied, the entire expression is true. All OR dependencies are evaluated as one expression also. If at least one OR dependency is satisfied, the entire expression is true. The overall dependency is satisfied if both AND and OR expressions are true. If there are no dependencies specified for AND, the expression is treated as true. The same also applies if there are no OR dependencies specified.

For example, a channel requires four files, and you have configured two as AND file dependencies and two as OR file dependencies. If both AND file dependencies are satisfied and one OR file dependency is not, the channel is installed. If both OR dependencies are satisfied and one AND file dependency is not, the channel is *not* installed.

Customizing a channel by using scripts and classes

You can customize the behavior of your channel on the endpoint using scripts (or executables) and Java classes that are invoked at key times in your channel's lifecycle (install, uninstall, update, verify, repair, execute, or a combination of these phases). You can also use script return codes to signal error conditions. There can be any number of scripts in a channel.

Note: MSI (Windows Installer) macros cannot be used in preinstall scripts.

For information about using scripts to control system reboots, see “Using scripts to trigger reboots” on page 116.

Customizing a channel using an executable script

You can specify script files (executables and batch files) to be invoked at key times in your channel's lifecycle.

It is important that you place scripts not present on the endpoint in your channel's `signed\scripts` directory before publishing the channel. You can do this either by manually adding the scripts or by using the GUI. For example, in Windows, you might want to use a preinstall script called `foo.exe` and a postinstall script called `bar.bat`. You must place `foo.exe` and `bar.bat` in the channel's `signed\scripts` directory. However, a script already present on the endpoint could be entered either by using the full path or by using a macro, for example, `$sys.system$\myscript.exe`. Scripts already present on the endpoint do not need to be placed in the `signed\scripts` directory.

Limitations

If the pre or post script associated with a package generates output that is greater than 1024 bytes in a single instance, the output is truncated in SDM logs.

Note: In Windows, you can configure the process creation associated with script files having the extensions .exe or .bat by setting the processCreationFlags in the parameters.txt file of the channel. For more information, see the *Symphony Marimba Client Automation Reference Guide*, available on the Marimba Channel Store.

► To specify an executable script to use with your channel

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Customization tab.
- 3 Click Add.
- 4 Specify the required information for the new script:
 - a In the Script Name box, specify the name of the script. It can contain spaces.
 - b In the Type of script box, select the script type. The following types are available:
 - Executables (.exe) and batch files (.bat) in Windows
 - Shell scripts (.sh) in UNIX
 - Java in both Windows and UNIX. For more information, see “Customizing a channel by using a Java class” on page 145.
 - c For the Exe,Batch path box, perform one of the following steps:
 - Specify the path of the script. For more information, see “Specifying paths for a script” on page 144.
 - Choose Add from the drop-down list to browse the local machine and select the script you want to add to the channel. If you choose Add, the editor adds the script you selected to the proper location in the package directory (channel_directory\signed\scripts).
 - d To provide the script with any arguments, specify the arguments in the Arguments box. Use a space to separate multiple arguments.

- e To specify a working directory for the script, specify the path in the Working dir box. If the path is relative to the scripts folder, the current working directory is set to the scripts folder. If the path is absolute, the current working directory is set to the parent directory of the native executable. If the current working directory is not set manually, the default is the directory for which the tuner is currently set.
- f To specify when you want to run the script, select the Run script options and Installer phases check boxes.
- g Select the check boxes for the options you want to use:

Ignore script exit/return code—To ignore any exit or return codes.

Run this script in the user's context—(Windows only) To run the script in the user's context. This option might be useful when running the tuner as a service. By default, scripts started using the tuner inherit the tuner's context. Inheriting the tuner's context might cause problems when the tuner is running as a service because the script is not running in the logged-on user's context (and is missing the user's security and identity settings). If you select this option, the script has the rights and permissions of the logged-on user only.

To pass the currently logged-on user's environment to a script, you must set the property `userenv=true` in the `parameters.txt` of the package and the script must be set to run in the user's context.

Run this script as a detached process—To run the script separately. This option might be useful if you do not want to wait for the script to finish before continuing with the installation (or any of the other phases). For example, if your preinstall script launches a program, such as NotePad, you can select this option to allow your installation to continue even if NotePad is still running. Without this option, installation waits for the preinstall script to finish (including closing NotePad) before continuing.

Note: If you select this option, the exit or return code for the script is ignored. Application Packager sets the exit or return code to zero to indicate a successful launch of the detached process (even if the actual exit or return code from the script is not zero).

If you run a script as a detached process, you cannot see the output of that script redirected to the console window or in the channel or tuner history logs. This is true even if you set `disabledProcessRedirect=false` in the `parameters.txt` file of the package.

- h Click OK.

Specifying paths for a script

You can use any of the following methods and shortcuts when typing a path for the script file. You can:

- Specify the full path and file name. This file must exist at this location on the endpoints before you execute it.
- Specify the path relative to the directory (`channel_directory\signed\scripts`) that the packager prepares for you.

If you use this option, you must copy the script files you specify to the directory (`channel_directory\signed\scripts`) that the packager prepares for you.

- Specify any macro (for example, `$system$`) as a substitute for the path to the Windows directory on the user's system. Specify any additional path and file name information after the `$system$` variable, for example, `$system$\calc.exe`.

This option allows your channel to find the Windows system directory, even when the user has installed the operating system in a non-standard location.

Failure of post-operation scripts

Before version 4.7.2.1, post-operation scripts had no effect on whether the package operation succeeded or failed. Specifically, `postinstall`, `post-major update`, `post-minor update`, `post-verify`, `post-repair`, and `post-uninstall` scripts did not have any effect on whether the operation succeeded. To preserve this behavior in 4.7.2.1 and later, you must set the parameter `postscripts.fail` to `false` in the `parameters.txt` file of the channel.

When set to true, this parameter specifies that the failure of post-operation scripts (such as postinstall scripts) should cause the package operation to fail. The package operation fails if the post-operation script returns a non-zero value.

Using scripts to run a response file for SVr4 packages

For SVr4 packages, you probably want to install the Symphony Marimba Client Automation package into a temporary directory (like /tmp/pkgdir) and then run the response file that you saved for the package (see “UNIX SVr4 package” on page 61). The response file contains answers for each of the installation questions that need to be answered during a normal installation.

► To install the Symphony Marimba Client Automation package into a temporary directory and use a script to run a response file

- 1 To create the /tmp/pkgdir directory, in the Package Contents page of the Package Editor, right-click and choose New > Directory.
- 2 To add the package contents to that directory, right-click the directory and choose Add > File.
- 3 Create a postinstall script that installs your package using the bundled response file (see the “Customizing a channel using an executable script” on page 141). Add the script to your package.

The postinstall script should invoke this command:

```
pkgadd -d /tmp/pkgdir -r /tmp/pkgdir/response <package name>
```

Customizing a channel by using a Java class

You can specify a class file that extends your channel’s behavior by implementing the `com.marimba.intf.packager.IScript` interface. The method in the interface invokes the script. For more information, see “`IScript` interface” on page 147 and “`IScriptContext` interface” on page 148. For a simple example, see “Example: Simple Java script that implements the `IScript` interface” on page 149. If you do not use the `IScript` interface, you do not have access to internal channel information, such as macros.

Note: This feature requires Java developer assistance. For more information, see the “Writing Java Classes for Application Packager Install Scripts” section of the *Advanced Symphony Marimba Client Automation Programming Guide*, available on the Marimba Channel Store.

You can place a Java class either in the `signed\scripts` directory or in a ZIP file in the signed directory hierarchy. You specify the classpath of the Java class on the Customization page.

► To specify a java class to use with your channel

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Customization tab.
- 3 Click Add to create a new script.
- 4 Specify the required information for the new script:
 - a In the Script Name box, specify the name of the script. It can contain spaces.
 - b In the Type of script box, select Java.
 - c For the Java class file box, perform one of the following steps:
 - Specify the name of your `.class` file, for example, `MyScript.class`.
 - Choose Add from the drop-down list to browse the local machine and select the script file you want to add to the channel. If you use the Add option, the editor adds the script you selected to the proper location in the package directory (`channel_directory\signed\scripts`).
 - d In the Fully qualified name box, specify the fully qualified name of the class for a Java script. This is the name of the actual Java class (as opposed to the file name you specified in the Java class file box). For example, for the Java script shown in “Example: Simple Java script that implements the `IScript` interface” on page 149, type `MyScript`. The fully qualified name is case-sensitive.
 - e To provide the script with any arguments, specify the arguments in the Arguments box. Use a space to separate multiple arguments.

- f** To specify a working directory for the script, specify the path in the Working dir box. If the path is relative to the scripts folder, the current working directory is set to the scripts folder. If the path is absolute, the current working directory is set to the parent directory of the native executable. If the current working directory is not set manually, the default is the directory for which the tuner is currently set.
- g** Specify when you want to run the script by using the Run script options and Installer phases check boxes.
- h** Select the check boxes for the options you want to use:
 - Ignore script exit/return code**—To ignore any exit or return codes.
 - Run this script in the user's context**—(Windows only) To run the script in the user's context. This option might be useful when running the tuner as a service. By default, scripts started using the tuner inherit the tuner's context. Inheriting the tuner's context might cause problems when the tuner is running as a service because the script is not running in the logged-on user's context (and is missing the user's security and identity settings). If you select this option, the script has the rights and permissions of the logged-on user only.
 - Run this script as a detached process**—Does not apply to Java scripts.

- i** Click OK to add the script and return to the Customization page.
- 5** In the Classpath for Java scripts box, specify the classpath of the Java class (relative to the package directory). If the Java script files are in the package directory, you can type signed/scripts. The classpath can be a colon-delimited list of files and directories.

Note: Currently, you cannot specify paths that include colons (:) because they are interpreted as delimiters.

IScript interface

```
/*
 * Confidential and Proprietary Information of BMC, Inc.
 *
 * Copyright (c) 1998 BMC, Inc.
 * All rights reserved.
 *
 * @(#)IScript.java, 1.2, 08/06/1999
 */
```

```
package com.marimba.intf.packager;

/**
 * This interface should be implemented by a Java-based script that
is used in
 * an Application Packager channel.
 *
 * <P>Classification: public.
 *
 */

public interface IScript {
    /**
     * Invoke this script.
     */
    public int invoke(IScriptContext context, String[] args);
}
```

IScriptContext interface

```
/*
 * Confidential and Proprietary Information of BMC, Inc.
 *
 * Copyright (c) 1998 BMC, Inc.
 * All rights reserved.
 *
 * @(#)IScriptContext.java, 1.4, 08/03/2002
 */

package com.marimba.intf.packager;

/**
 * Context for Java-based Script. will The script can use this class
to access
 * macros and other features of the Application Packager.
 *
 * <P>Classification: public.
 *
 */

public interface IScriptContext {
    /**
     * script phase constants.
     */

    int SCRIPT_PREINST = 1;
    int SCRIPT_POSTINST = 2;
    int SCRIPT_PREUNINST = 3;
    int SCRIPT_POSTUNINST = 4;
    int SCRIPT_PREUPDATE = 5;
    int SCRIPT_POSTUPDATE = 6;
    int SCRIPT_PREVERIFY = 7;
```

```

int SCRIPT_POSTVERIFY = 8;
int SCRIPT_PREEEXEC = 9;
int SCRIPT_POSTEXEC = 10;
int SCRIPT_PREUPDATEMINOR = 11;
int SCRIPT_POSTUPDATEMINOR = 12;
int SCRIPT_PREREPAIR = 13;
int SCRIPT_POSTREPAIR = 14;

/**
 * Parse the given string for macros. The return value is the string
 * with all macros expanded
 */
String parse(String str);

/**
 * set the given macro value
 */
boolean setMacro(String name, String value);

/**
 * get the invocation phase. See SCRIPT_RUN constants above
 */
int getPhase();

/**
 * access advanced features
 * "context" return the IApplicationContext for the channel
 * other tuner features may also be accessed.
 */
Object getFeature(String name);
}

```

Example: Simple Java script that implements the IScript interface

```

import com.marimba.intf.packager.*;
import java.lang.*;
import java.util.*;

public class MyScript implements IScript {

    /**
     * Access point into the script from the IScript interface
     */
    public int invoke(IScriptContext context, String[] args) {
        System.out.println ("Welcome to my script");
        return 0;
    }

}

```

Managing major and minor updates

A channel can have two types of updates: major and minor. The type of update determines whether the contents of a channel need to be installed, changed, or removed during an update.

- **Major Update**—An update is considered *major* if the contents of the channel are changed. Content changes mean files, registries, environment variables, or text modifiers have been added, changed, or removed from the channel. For files, content changes include any changes in file attributes also, for example, attribute flags in Windows, permission attributes in UNIX, or timestamps on either platform.

The following situations are also considered major updates:

- Using the Package Editor to remove content and then later re-add the content is considered a major update. For example, if you remove File A and later add File A again to a channel, Application Packager considers that a content change.
- Any changes to the update and uninstall policies, whether global or object-level, are considered major updates.
- Using File Packager to repackage a channel is considered a content change.
- **Minor Update**—An update is considered *minor* if the contents of the channel are not changed. No channel contents have changed if there are no new, deleted, or changed files, registry values, or environment variables. The following types of changes are considered minor updates:
 - Any changes to the Application Packager itself, such as an internal update of its classes and, in Windows, internal DLL files
 - Any changes to the properties.txt and parameters.txt files
 - Any changes to the scripts or dependencies
 - Any changes to the install, verify, and repair policies, whether global or object-level

If you make any changes (except for the [major update changes](#) described previously) to a channel in the Package Editor, undo your changes, and save the time property changes. (Application Packager uses the time property to track changes for the channel.) As a result, the channel is marked for a minor update, even though you have not really changed anything.

Setting scripts and dependencies for major updates

A major update triggers the major update scripts and major update dependencies. Updates that contain both minor and major updates are considered major updates. During the updates, only major update scripts (not minor update scripts) are executed. If you upgrade a packaged application that was created using a pre-4.7.1 version of Application Packager (which did not distinguish between major and minor updates), the scripts are set to run during major updates by default. Major updates also detect if there is a need for a reboot.

You can set scripts and dependencies to be triggered during major updates in the following places:

- **Dependencies tab**—Check dependencies on major updates check box
- **New/Edit Script dialog box**—Major Update check box

For more information, see “Configuring system and channel dependencies” on page 130 and “Customizing a channel by using scripts and classes” on page 141.

Setting scripts and dependencies for minor updates

A minor update triggers the minor update scripts and minor update dependencies if they are set. Updates that contain both minor and major updates are considered major updates. During the updates, only major update scripts (not minor update scripts) are executed. Minor updates also detect if there is a need for a reboot, just like major updates. If Always reboot after successful installation is selected, the application reboots the machine on minor updates.

You can set scripts and dependencies to be triggered during minor updates in the following places:

- **Required Symphony Marimba Client Automation Channels dialog box**—Run on Minor Update check box
- **Required Registry Entries dialog box**—Run on Minor Update check box
- **Required Files dialog box**—Run on Minor Update check box
- **New/Edit Script dialog box**—Minor Update check box
- **Required Files dialog box**—Run on Minor Update check box

- **Required Environment variables dialog box**—Run on Minor Update check box

For more information, see “Configuring system and channel dependencies” on page 130 and “Customizing a channel by using scripts and classes” on page 141.

Managing services (Windows only)

In Windows, services are programs or processes that perform a specific system function, usually as background tasks, to support other programs. Typically, they do not require user interaction and do not have a user interface. Some Windows applications include and install services. If you are familiar with services and want to manage services for the applications you are packaging, you can use the Services page in the Package Editor to edit, add, and remove services.

You can capture and install services only on a system running either Windows NT, Windows 2000, or Windows XP. If a single channel is installed on different versions of Windows, services are ignored when the channel is installed on systems not running these versions of Windows.

Note: Application Packager currently does not support packaging and distributing device drivers.

Editing services

If an application you packaged includes services, you can edit the properties of those services using the Services page in the Package Editor. You can edit general properties such as the display name and startup configuration of the service, as well as installation, uninstallation, update, verify installed, verify uninstalled, and repair policies for the service.

► To edit a service

- 1 On the Package Editor window, click the Services tab.
- 2 On the Services page, select a service and click Edit to change its properties.

The NT Service Properties dialog box appears. For more information about the properties you can change, see the descriptions for the various tabs in the dialog box:

- “General properties page” on page 153
- “Security properties page” on page 154
- “Advanced properties page” on page 154
- “Install policies page” on page 155
- “Uninstall policies page” on page 156

3 Click OK.

General properties page

Use this page to change general information about the service.

- **Display name**—Specifies the name that is displayed in the Services Control Panel.
- **Service program**—Specifies the name and path of the executable file associated with the service.
- **Type**—Specifies whether the service works on hardware (Kernel driver) or on the operating system (File system driver). It can also indicate whether the service runs as its Own process or as a Shared process.
- **Start**—Specifies when the service is configured to start:
 - **Boot**—Starts when the computer is booted up. This usually applies to services for hardware that are required for the computer to function, such as disk drives.
 - **System**—Starts when the operating system starts. This usually applies to services that are critical for the operation of the operating system, such as display drivers.
 - **Automatic**—Starts when the operating system starts, like system services, but is not crucial to the operation of the operating system.
 - **Demand**—Can be started by a user or a dependent service. The service does not start automatically when the operating system starts.
 - **Disabled**—Cannot be started by a user or a dependent service.
- **Error control**—Specifies the service’s level of error control:
 - **Critical**—Does not start up the system.
 - **Severe**—Switches to the “Last Known Good” setup, or, if already using that setup, continues.

- **Normal**—Continues system startup, but displays an error message stating that the service did not start.
- **Ignore**—Does not stop the system or display an error message, but just skips the service.

Security properties page

Use this page to specify which account a service uses to log in.

- **System account**—Specifies that the service logs on using the system account, rather than a user account. Most services log into a system account.
- **Allow interaction with desktop**—Specifies whether the service provides a user interface that can be used by the logged on user when the service is started.
- **User account**—Specifies a logon user account for a service to use.
Although most services log into the system account, some services can be configured to log into specific user accounts so that the user can have access to resources such as files and directories protected by Windows.
Provide the user name and password for the logon user account.

Advanced properties page

Use this page to specify which group the service is associated with and also the groups and services on which the service depends.

- **Group name**—Specifies the group a service is associated with. The group usually determines the order in which the services start.
- **Dependencies on other groups and Dependencies on other services**—Specify the groups and services on which a particular service depends. If a service is stopped or is not running properly, other services that depend on that service can be affected. Click these icons:

Table 4-4: Icons

Icon	Description
	To add a new group dependency.
	To add a new service dependency.

Table 4-4: Icons

Icon	Description
or	To change the order of dependencies.
	To remove dependencies.

Install policies page

Use this page to set installation policies for services.

- **Inherit global policy**—Uses the installation policy set as the default for the entire channel. For more information, see “Setting default policies for channels” on page 123.

If you do not want to use the global or default policy, select Use a policy below and select one of the following policies:

- **Always install service**—Determines if the service is already installed by using native Windows APIs. If the service is not yet installed, it is installed. If the service is already installed but the attributes of the service are different, the service in the channel is installed.

The service in the channel cannot be installed on top of a previously installed service that is currently running. You can stop the service before installing the channel; otherwise, the service is installed at the next system reboot.

- **Install service if it doesn't exist**—Determines if the service is already installed. If the service is not yet installed, it checks if the service exists in the system. If a service with the same name is found, the service is skipped and not installed. Otherwise, the service is installed.

- **Install service only if it exists**—Determines if the service is already installed. If the service is not yet installed, it checks if the service exists in the system. If a service with the same name is *not* found, the service is skipped and not installed. Otherwise, the service is installed.

The service in the channel cannot be installed on top of a previously installed service that is currently running. You can stop the service before installing the channel; otherwise, the service is installed at the next system reboot.

Uninstall policies page

Use this page to set uninstallation policies for services.

- **Inherit global policy**—Uses the uninstallation policy set as the default for the entire channel. For more information, see “Setting default policies for channels” on page 123.
If you do not want to use the global or default policy, select Use a policy below and select one of the following policies:
- **Uninstall an installed service**—Determines if Application Packager was used to install the service. If not, uninstallation is not attempted. Then, it determines if the service is already uninstalled from the system by using native Windows APIs. If the service is not already uninstalled, uninstallation is performed by removing the service, or by replacing it with an older version that was overwritten during installation.

Note: The service in the channel cannot be uninstalled if it is currently running. You can stop the service before uninstalling the channel; otherwise, the service is uninstalled at the next system reboot.

- **Always uninstall service**—Currently the same as Uninstall an installed service because uninstallation is attempted only if Application Packager was used to install the service.
- **Don't uninstall service**—Service is skipped and not uninstalled during uninstallation.

Update policies page

Use this page to set update policies for services.

- **Inherit global policy**—Uses the policy set as the default for the entire channel. For more information, see “Setting default policies for channels” on page 123.

If you do not want to use the global or default policy, select Use a policy below and select one of the following policies:

- **Always update the service.**
- **Update service if it doesn't exist.**
- **Update service only if it exists.**

- Never update service.

Verify installed and Verify not installed policies pages

Use this page to set verify policies for services.

- **Inherit global policy**—Uses the policy set as the default for the entire channel. For more information, see “Setting default policies for channels” on page 123.

If you do not want to use the global or default policy, select Use a policy below and select from the following policies:

- Verify the service using the following policies.
 - Verify contents of the service.
 - Verify the existence of the service.
- Never verify service.

Repair policies pages

Use this page to set repair policies for services.

- **Inherit global policy**—Uses the policy set as the default for the entire channel. For more information, see “Setting default policies for channels” on page 123.

If you do not want to use the global or default policy, select Use a policy below and select from the following policies:

- Repair contents of the service.
- Never repair the service.

Adding services

If the application you are packaging will be installed in Windows NT, Windows 2000, or Windows XP, you can add services using the Services page in the Package Editor.

► To add a service

- 1 On the Package Editor window, click the Services tab.
- 2 On the Services page, click Add.

- 3 In the Add New NT Service dialog box, specify the information for the service you want to add.
- 4 Click OK.

The service you added now appears on the Services page.

- 5 Click the arrow buttons on the right side of the window to specify the order in which the services should be installed on the endpoint.

Services are installed in the order in which they appear in the list. By specifying the installation order of services, you can preserve any dependencies needed between services.

Removing services

You can remove one or more services from a channel using the Services page in the Package Editor.

► To remove a service

- 1 On the Package Editor window, click the Services tab.
- 2 To remove services, perform one of the following steps:
 - Select the service you want to remove, and click Remove.
 - To remove all services from the list, click Remove All.
- 3 Click Close or Add/Close.

Editing the channel parameters and properties

You can control how applications are packaged and published using the following files:

- `parameters.txt`—controls how the packaged application is installed and launched.
- `properties.txt`—contains publishing information and instructions on how to launch the channel

Some channel parameters have default values. Some are additional parameters that are not found in the `parameters.txt` file by default. Entries in the `parameters.txt` file have the form:

`<channel_parameter>=<parameter_value>`

For example:

```
silent=true
```

For more information about the channel properties and parameters, see the *Symphony Marimba Client Automation Reference Guide*.

You can edit the channel parameters for a packaged application by:

- Using Channel Copier to edit the channel parameters when you publish the channel
- Editing the files directly in the package directory. For more information about the package directory and its contents, see “Package directory” on page 28.
- Editing the files through the Application Packager graphical interface.

► To edit channel properties and parameters

- 1 From the Package Editor window, select the package and then click the channel property icon in the toolbar.
- 2 From the Channel Properties window, enter the parameter and property values.
- 3 To add additional parameters or properties that are not listed, scroll to the end of the dialog, click the last row, and then enter the property and value.
- 4 To save your changes, click Save.

Minimizing bandwidth usage

You can minimize the amount of bandwidth used when downloading and installing a packaged application by setting the `preload` and `delayfiledownload` parameters, as described in this section.

Note: You cannot use the `preload` and `nofilemap` parameters together.

► To set the `preload` and `delayfiledownload` parameters

- 1 Locate the `parameters.txt` file in the package directory. For more information, see “Package directory” on page 28.
- 2 Open the `parameters.txt` file in a text editor.

- 3 Add the following lines to the file:

```
preload=true  
delayfiledownload=true
```

- 4 Save the file.

You can now publish the packaged application for distribution.

Preload and delayfiledownload parameters

- **preload**—If a channel you are packaging uses files that might already be on users' systems, you can use the `preload` parameter to minimize the number of files that they need to download.

When you subscribe to the channel and install it, the installer checks for files that already exist on the hard drive. It collects any files on the endpoints that match files with the same paths in the channel. Then it connects to the transmitter and downloads only the files that are not already on the endpoints. If many files on the endpoint already match those that are in the channel, they do not have to be downloaded, thus reducing the bandwidth usage.

Setting this property implicitly enables the `delayfiledownload` property.

- **delayfiledownload**—This parameter delays the download of a channel's files to the endpoints until any pre- scripts have run. Usually, when subscribing an endpoint to a channel, all the files in the channel are downloaded to the endpoint before running any scripts and starting installation.

In some cases, you might want to delay the download of a channel's files to the endpoints. For example, to use a preinstall script to run some commands or check that some conditions are met before installing an application, you might want to delay the download so that time and bandwidth are not wasted if the commands fail or if the conditions are not met.

Tip: You can also set this parameter through the Configuration – Installer tab by selecting the Delay download of files until pre scripts have run check box.

If you use this parameter, the channel download occurs in two parts. The initial download includes only the installer and the manifest file (`manifest.ncp`), which contains the code and instructions required to install and update the application. After any pre- scripts have run, the second download brings over the files necessary to install or update the application.

Note: If the packaged application contains any files that are added by reference, the `delayfiledownload` parameter does not take effect during updates.

Working with packages that contain user-specific files and registry entries

If you are packaging and distributing applications that are used by multiple local users on the same machine, make sure that any current user keys, user profile files, and other user-specific files and registry entries are properly installed. This section describes the feature you can use to package and distribute applications that include user-specific file and registry entries to a machine that is used by multiple local users. As a result of having this feature, a package has a state for each local user in addition to its state for the machine. The local user state is either *installed* or *none*.

This feature is supported for packages created using the following Application Packager components:

- Custom Application Packager
- File Packager
- .NET Packager
- Packager for Shrinkwrap Windows Applications

Consider the following questions before you enable the multiple-user feature:

- Does the application include files that are common to all users as well as files that are installed for specific local users?
- Which directories contain user-specific files? Which registry keys contain user-specific values?

- Do you need to run any scripts before or after installing user-specific items? Before or after updating user-specific items?

► To enable the multiple-user feature

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Installer tab.
- 3 In the Installer Options area, select the Multi-user package check box.

When you select this check box, the following properties in the package directory are set:

In the properties.txt file—local.user.settings=true

In the parameters.txt file—userobjs=true

Note: When you create a package with the property userobjs=true , after the Subscription Service pushes the channel to the endpoint, if the user at the endpoint is not logged on, the tuner on the endpoint attempts to install the channel every three minutes.

- 4 Identify and mark user-specific directories and registry keys:
 - a Right-click a directory or registry entry, and choose Properties.
The Properties dialog box appears.
 - b Select the User specific check box.

The user-specific attribute applies to all files in a directory and all the values in a registry key.

- 5 Specify any scripts you want to run before or after the *user install* and *user update* phases for a package:

User install phase—After all the common objects (those used by all users) in a package are installed on an endpoint machine, the user install phase runs for each user who installs the package. In this phase, only the user-specific objects are checked, and, if necessary, installed or repaired.

User update phase—After all the common objects (those used by all users) in a package are updated on an endpoint machine, the user update phase runs for each user who updates the package. In this phase, only the user-specific objects are checked, and, if necessary, updated or repaired.

For more information, see “Customizing a channel by using scripts and classes” on page 141.

Using the multiple-user feature with Policy Management

To use the multiple-user feature with Policy Management, you must set the following properties for Policy Service running on the endpoints:

```
logon.notify=true  
logon.action=updatestart
```

You can set these properties by using Policy Manager. For more information, see the section about setting package properties in the *Policy Management Administrator’s Guide*, available on the Marimba Channel Store.

Setting these properties causes the tuner at the endpoint to update and start Policy Service when a user logs on. These properties make sure that the user-specific objects are installed for each user who logs on to the machine and who has been assigned the multi-user package through a policy.

Package behavior with the multiple-user feature

The following table describes the behavior for packages when the multiple-user feature is enabled and a local user is currently logged on.

Table 4-5: Behavior for packages with the multiple-user feature enabled

Command given or state assigned	Resulting state	State for the currently logged-on local user
<i>Using the tuner, channel manager, or tuner administrator</i>		
subscribe	staged	none
install	installed	installed
update	installed	installed If an update exists for the application, the new update is downloaded and installed for the currently logged-on user.
verify	installed	installed
repair	installed	installed

Table 4-5: Behavior for packages with the multiple-user feature enabled(Continued)

Command given or state assigned	Resulting state	State for the currently logged-on local user
uninstall	uninstalled	none The application is uninstalled, and user-specific files are removed for the local user.
delete	none	none The application is uninstalled, the package is deleted, and user-specific files are removed for the local user.
start	installed	State does not change for the local user.
stop	installed	State does not change for the local user.

Using Policy Manager

advertise	advertised	none
exclude	none	none
install	installed	installed
install-start	installed	installed
install-persist	installed	installed
install-start-persist	installed	installed
primary	installed	installed
stage	staged	none
uninstall	none	The application is uninstalled, the package is deleted, and user-specific files are removed for the local user.

Verifying and repairing channels

Sometimes applications and content that you package and distribute to users as channels might become unusable or not run properly. You can use the verify and repair capabilities to automatically check and fix problems, such as missing or corrupted files.

You can use the Package Editor to specify verification policies for your channels (see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121), as well as to set a schedule for automatically verifying and repairing your channels.

Table 4-6: Verifying and repairing

Action	Description
Verify	When you verify a channel, the file and registry objects in the channel are compared with the file and registry information for the channel when it was originally installed. Any object mismatches found are recorded in the log file.
Repair	When you repair a channel, the file and registry objects in the channel are first verified as described previously. After verification is complete, one of the following occurs: <ul style="list-style-type: none"> ▪ There are no object mismatches. No repairs are needed. ▪ There are object mismatches. The tuner re-installs the affected files or registry entries if they are available from the tuner storage. If not, the tuner contacts the transmitter to download the files or registry entries needed to complete the repair.

Note: If the transmitter cannot be contacted, or the channel on the transmitter has been removed or modified so that the needed objects are no longer available, the log has a record of the objects that cannot be repaired.

Verify and repair operations do not check the channel’s dependencies. Dependencies are checked when installing or downloading the channel. For more information about dependencies, see “Configuring system and channel dependencies” on page 130.

Verify and repair behavior is different for Windows Installer packages. For more information, see “Packaging a set of files into a channel” on page 222.

Verifying channels before publishing and distributing

Before publishing and distributing, you can verify the contents of a channel so that you discover problems before installing or updating the channel on endpoints. When you verify a channel, the file objects in the channel are compared with the file information for the channel when it was originally created.

This feature is supported for packages created using the following Application Packager components:

- Custom Application Packager
- File Packager
- .NET Packager
- Packager for Shrinkwrap Windows Applications
- Windows Installer Packager

► To verify channels before publishing and distributing

- 1 On the main Application Packager window, click one of the packagers.

Depending on the packager you select, a list of previously packaged channels appears.

- 2 Select a channel from the list and click Verify or Verify verbose.

Any object mismatches found are displayed in the Verification Results window. In addition, if you select Verify verbose, the Verification Results window shows the files in the packaged application and the corresponding file objects in the channel.

Scheduling verification and repair for channels

You can use the Package Editor to set the verify and repair schedule for a channel. When verify is scheduled (or a user manually verifies a channel from the tuner), verification checks that each object in the channel is installed correctly and records any verification errors in the channel's log file. If you choose to repair a channel, objects that have changed since the channel was installed are re-installed. Before repairing the channel, the tuner connects to the transmitter from which the channel gets updates.

Note: Scheduled verify and repair usually does not require any user interaction. At most, users might see a verify or repair progress bar displayed. However, if the channel's transmitter cannot be contacted, or the channel has been updated and the needed files are no longer available, the user is informed that the transmitter cannot be reached.

► To set up the verify and repair schedule for a channel

- 1 On the Package Editor window, click the Configuration tab.
- 2 On the Configuration page, click the Verify/Repair tab.
- 3 Select the Verify channel according to schedule check box to specify a schedule for verifying that a channel is installed correctly.
Select this check box to set the verify and repair schedule.
- 4 To make any necessary repairs to a channel at the same time that verification is scheduled, select the Repair also check box.
- 5 In the Days area, specify how often you want to verify the channel:

Never—The tuner never verifies the channel; you have to manually verify and repair the channel when necessary.

Daily—The channel can be verified every day, only on weekdays (Monday through Friday), or every 1 to 100 days, depending on the number you set.

Weekly—The channel gets verified every 1 to 100 weeks, depending on the number you set. During the week when verification is scheduled, verification occurs only on the days you check. If no days are checked, verification occurs on Monday.

Monthly—The channel can be verified every 1 to 100 months on the day you set. If you select day 30 or 31 and the month when verification occurs does not have that day (for example, February), verification occurs on the first day of the next month.

- 6 In the Frequency area, specify the time of the day when you want to verify the channel:

Verify at—Select to choose one time of the day when verifications occur.

Verify every—Select to specify multiple times of the day when verifications occur. You can set verifications to happen every 1-100 minutes or hours during the day, and you can limit verifications to a single range of hours during the day. For example, you can have verification occur every hour between 9 a.m. and 5 p.m., or every 30 minutes from 5 p.m. to 9 a.m.

If the scheduled verification or repair cannot take place because the package is already running, Application Packager logs that it attempted to start a package that is already running. For example, you have specified verification to occur every day between 9 to 9:10 a.m., every 30 minutes, and the verification cannot take place because the package is already running. Application Packager logs that it attempted to start a package that is already running, and it attempts the verification again between 9 to 9:10 a.m. the next day.

Manually running verify and repair for channels

Usually, you set the policy and schedule for automatically verifying and repairing your channel when you package the channel. However, there can be situations when you want to manually run verify and repair on the endpoint machine. You can manually run verify and repair from the tuner or from the command line.

► To run verify or repair from the tuner GUI

- In the tuner, right-click the channel you want to verify or repair, and choose Verify or Repair.

► To run verify or repair from the command line

- 1 Go to the directory where the tuner is installed. This is where the runchannel1 program is located. For example:

C:\Program Files\Marimba\

- 2 On the command line, type one of the following

```
runchannel <channel_URL> -verify  
runchannel <channel_URL> -repair
```

For example:

```
runchannel http://acme.com:5282/MyChannel -repair
```

Updating and repairing applications using shortcuts before startup (Windows only)

In some cases, you might want to update or repair an application before it starts. This practice makes sure that the application at the endpoint is always updated and consistent with the one that was packaged. Using the Package Editor, you can configure the Windows shortcut used to start an application, so that before the application starts, one of the following operations take place:

- The application gets updated.
- The application gets verified, and if necessary, repaired.
- The application gets both updated and repaired.

These operations are performed before the application is started, whenever the shortcut is used. You can use this feature to seamlessly update and repair an application, without requiring users at the endpoints to manually perform any operations, other than double-clicking the shortcut. This is also known as just-in-time (JIT) application deployment.

► To configure the shortcut for application updates and repairs

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 On the Package Contents page, find the shortcut you want to configure.
 You might need to expand directories or use the find feature.
- 4 Right-click the shortcut and choose Properties.
- 5 Near the bottom of the dialog box, click the Before Running drop-down list and select from the following options:

No action—Do not perform any updates or repairs before running the application associated with the shortcut. This is the default.

Update application—Update the channel before running the application associated with the shortcut.

Repair application—Verify the channel and perform any necessary repairs before running the application associated with the shortcut. For more information, see “Verifying and repairing channels” on page 165.

Update and repair application—Update the channel, verify the channel, and perform any necessary repairs before running the application associated with the shortcut.

- 6 Click OK.



Shortcuts that have been configured with automatic update and repair appear with this icon.

- 7 Click the Configuration tab.
- 8 Click the Installer tab.
- 9 Make sure that the Install updates after update is brought down check box is selected.

Symphony Teleca recommends that you select this check box, because it makes sure that a downloaded update is installed. Otherwise, if an update is downloaded but not installed, the channel is in an install-pending state and is not able to do any other operations, such as repairing the application. If the user at the endpoint double-clicks the shortcut (configured to update and repair the application) again after an update was downloaded but not installed, the channel does not perform a repair as expected.

- 10 Click Save to save the channel.
- 11 In the package directory, open the parameters.txt file using a text editor and add the property rundetach=true.

When you set this property to true, the application can be launched by the shortcut to run as a detached process. When an application is running as a detached process, you can start and stop it independently from the tuner, even if it was launched using the tuner.

Associating packages with software library items in the Definitive Software Library (DSL)

If you adopt the Symphony Marimba Client Automation solution, a route to value in BMC Business Service Management (BSM), you can store a standardized list of available packages (Software Library Items, or SLIs) in the Definitive Software Library (DSL). Use the Action Request User Tool or Action Request Mid-Tier to navigate to the DSL console.

BSM users can use the DSL to standardize application names in a central repository of all authorized versions of software. BMC calls this repository the Product Dictionary. Use the Product Dictionary (PD) to standardize the names of all developed and brought-in software packages available in your enterprise. You can then deploy, manage, and maintain license compliance knowing that all references describe the same software.

For more information, see the configuration information in the *Definitive Software Library 7.0 Administrator's Guide*, available on the Marimba Channel Store.

Anyone with a DSL User role can review existing Product Dictionary entries using the DSL console. With a DSL Administrator role, you can also add entries to the Product Dictionary. If a Product Dictionary entry exists for the software, you can add packages to the list of SLIs automatically when you publish a package in Application Packager. Follow the standard workflow described here.

► To store packages you create in as Software Library Items

- 1 If a Product Dictionary entry for your package does not yet exist, create a PD entry in the DSL.
- 2 Create the Application Package.
- 3 Add metadata information on the DSL tab to relate your package to the PD entry.

Symphony Teleca supports this feature for packages you create using the following Application Packager modules:

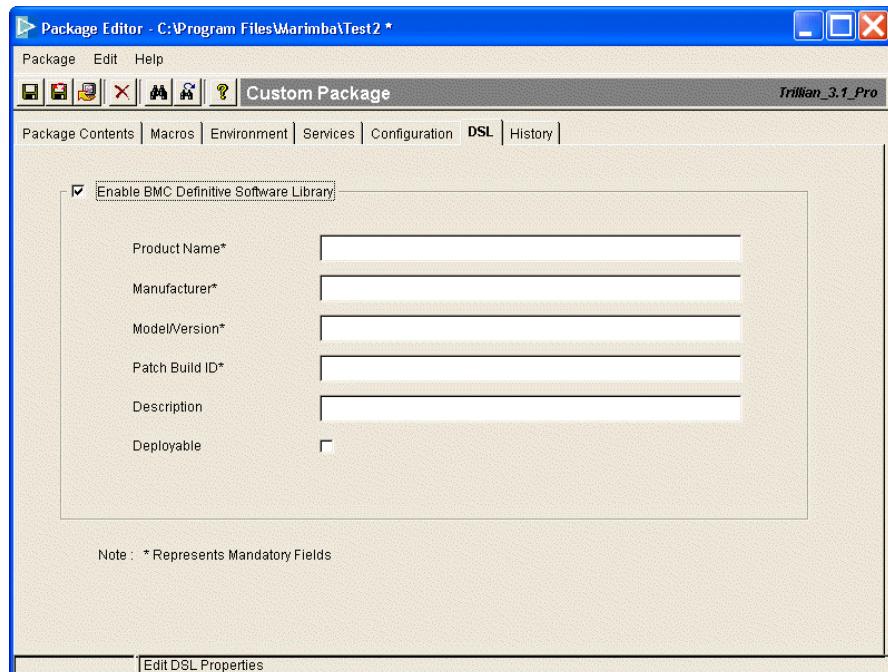
- Shrinkwrap Windows Application Packager
- Custom Application Packager
- Windows Installer Packager

-File Packager

► **To add a new Software Library Item to the DSL**

- 1 Click the DSL tab.

Figure 4-2: DSL tab



- 2 Click the Enable BMC Definitive Software Library check box.
- 3 Specify the normalized values you entered (or referenced) in the DSL in the following fields:
 - Product Name
 - Manufacturer
 - Model/Version
 - Patch Build ID

Note: If you specify a value in one of these fields, you must complete all four values. Accurate data entry is important, because the system does not validate data you specify on the DSL tab.

- 4 Optionally, for Description, specify additional information about the software.
- 5 To share this package with CCM users across your enterprise, select the Deployable check box.

Editing ASCII text files

When installing a packaged application at an endpoint, you might need to edit ASCII text files such as INI files, BAT files, and other configuration files. You can use the Package Editor to edit these files by adding text modifiers to your channel. You can use text modifiers to insert and replace text in ASCII text files. The files to be edited can be part of your channel or they can be any files on the endpoint. The files are modified at the endpoint during the installation of the channel.

You can add text modifiers when editing channels you have created using Packager for Shrinkwrap Windows Applications, Custom Application Packager, and File Packager.

You can create three types of text modifiers:

- **Insert Line**—To insert or replace a line of text at a specified line number. Select this text modifier if you know the line number where you want to insert or replace a line of text.
- **Search and Replace Line**—To find a line of text, and then insert a line of text or replace the entire line of text. Select this text modifier if you know all or part of the text in the line where you want to insert or replace text.
- **Search and Replace Text**—To search for certain text and replace it with the specified text. Select this text modifier if you know all or part of the text you want to replace.

In addition to ASCII text, you can also include system and user-defined macros in the search text. For more information, see “Using macros” on page 95.

Note: You cannot use text modifiers to edit Unicode or non-ASCII text files.

During installation time, text modifiers are installed after any files, directories, registry, and metabase entries in the channel. This is because the text modifiers might edit files that are part of the channel. However, the text modifiers are installed before any services (that are part of the channel) are started and before any postinstall scripts are ran. If you have two or more text modifiers, they are installed in the order they appear in the Package Editor.

Creating a text modifier group

To create text modifiers, you must first create a text modifier group. A text modifier group contains all the text changes you want to make to one ASCII text file. The text modifier group specifies two things:

- The name of the text modifier group.
- The path of the ASCII text file to be edited.

After creating a text modifier group, you can add to it one or more text modifiers that affect a single ASCII text file.

Note: The instructions in this section describe creating a text modifier group for a text file that is *not* part of the channel you are editing. If the text file is part of the channel you are editing, you do not need to create a text modifier group separately. When you create a text modifier, the Package Editor creates a text modifier group after asking you to provide a name. You do not need to specify the path of the text file if it is part of the channel.

► To create a text modifier group for a file *not* in the channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 On the Package Contents page, right-click My Computer and choose New > Text Modifier.
- 4 Specify the following information:

- a A name for the text modifier group (for example, application.ini Modifiers).

To make it easier to identify which file a modifier group affects, include the name of the text file you are editing in the text modifier group name.

- b The full path of the ASCII text file you want to edit, for example, C:\Program Files\MyApplication\application.ini.

Make sure you have write permission to the file (and that the file is not locked) on the endpoints. Otherwise, the text modifier cannot make changes to the file.

Also, if the file you specify is on a network drive, do not use the form \\machine_name\directory. Instead map the network drive and use the assigned drive letter (not the machine name) when you specify the file path.

- 5 Click OK.

The text modifier group you created appears in the left pane under Text Modifiers.

You can now add one more text modifier to this group.

Creating an insert line modifier

If you know the line number in the ASCII text file where you want to insert or replace a line of text, you can create an insert line modifier. You can use this type of text modifier to insert or replace a line of text at a specified line number.

► To create an insert line modifier

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 If the text file you want to edit is *not* in the channel:
 - a Create a text modifier group for that file, as described in “Creating a text modifier group” on page 174.
 - b In the left pane, right-click the text modifier group you just created and choose Add Text Modifier > Insert Line.

- c Proceed to step 5.
- 4 If the text file you want to edit *is* part of the channel:
 - a On the Package Contents page, locate the text file you want to edit.
 - b Right-click the text file and choose Add Text Modifier > Insert Line.
The New Text Modifier Group dialog box appears.
 - c Specify a name for the text modifier group (for example, application.ini Modifiers) and click OK. The Package Editor has already completed the path for the text file.
 - d Proceed to step 5.
- 5 Specify the following information:

A name for the text modifier—For example, Replace Line 4

The text to insert or replace—In the text, you can use any macros defined in the channel. When you use macros within a string, make sure you include a dollar sign (\$) preceding and following the macro name, for example, \$macro_name\$. Also, remember that macros are case-sensitive. For more information about macros, see “Using macros” on page 95.

The action to perform—You can choose from the following steps:

 - Replace at**—The entire line at the given line number is replaced with the text you specify.
 - Insert at**—The text is inserted on the line you specify. For example, if you specify line number 4, the text you insert appears in that line. The text that used to be on that line now appear on line number 5.
 - The line number in the text file**—To insert a line at the end of the file, use -1 for the line number.
- 6 Click OK.

The text modifier you added appears in the right-hand pane.

Creating a search and replace line modifier

If you know all or part of the text in the line where you want to insert or replace text, you can create a search and replace line modifier. You can use this type of text modifier to insert or replace a line of text based on the text that appears in that line. If the text appears in more than one line, the text modifier finds each occurrence and performs the action you specify.

► To create a search and replace line modifier

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 If the text file you want to edit is *not* in the channel:
 - a Create a text modifier group for that file. For more information, see “Creating a text modifier group” on page 174.
 - b In the left pane, right-click the text modifier group you just created and choose Add Text Modifier > Search and Replace Line.
 - c Proceed to step 5.
- 4 If the text file you want to edit *is* part of the channel:
 - a On the Package Contents page, locate the text file you want to edit.
 - b Right-click the text file and choose Add Text Modifier > Search and Replace Line.
 - c Specify a name for the text modifier group (for example, application.ini Modifiers) and click OK. The Package Editor has already completed the path for the text file.
 - d Proceed to step 5.
- 5 Specify the following information:

A name for the text modifier—For example, Insert machine name

The text to search for—In the text, you can use any macros defined in the channel. When you use macros within a string, make sure you include a dollar sign (\$) preceding and following the macro name, for example, \$macro_name\$. Also, remember that macros are case-sensitive. For more information about macros, see “Using macros” on page 95.

The search criteria—You can choose from the following criteria:

Exact match—The text you specify must match the entire line of text you want to find in the text file.

Starts with—The text you specify must match the beginning of the line you want to find in the text file. You must specify at least the first character of the line you want to find. For example, to find this line:

host=triad.bmc.com

search for:

host, h, or host-

Substring match—The text you specify must match part of the line you want to find in the text file. The text can appear anywhere in the line, including the beginning and end, but you must specify at least one character that appears in the line you want to find. For example, to find this line:

host=triad.bmc.com

search for:

triad, host, com, t, or -

Ends with—The text you specify must match the end of the line that appears in the text file. You must specify at least the last character of the line you want to find. For example, to replace this line:

host=triad.bmc.com

search for:

com, m, or bmc.com

Whether to consider case when searching—Uppercase and lowercase letters

The text to insert or replace with—In the text, you can use any macros defined in the channel. When you use macros within a string, make sure you include a dollar sign (\$) preceding and following the macro name, for example, \$macro_name\$. Also, remember that macros are case-sensitive. For more information about macros, see “Using macros” on page 95.

The action to perform—You can choose from the following steps:

Replace line—The entire line where the text is found is replaced with the text you specify.

Insert before line—The text you specify is inserted into the line where the search text is found. The text you were searching for is moved to the next line. For example, you are searching for host=triad.bmc.com and inserting port=5282. If the host=triad.bmc.com is found on line number 4, port=5282 is inserted into line number 4.

host=triad.bmc.com is moved to line number 5.

Insert after line—The text you specify is inserted into the line following the line where the search text is found. The text you were searching for remains on the same line, but the rest of the text in the file moves down one line. For example, you are searching for host=triad.bmc.com and inserting port=5282. If the host=triad.bmc.com is found on line number 4, port=5282 is inserted into line number 5.
host=triad.bmc.com remains on line number 4.

- 6 Click OK.

The text modifier you added appears in the right-hand pane.

Creating a search and replace text modifier

If you know all or part of the text you want to replace, and you do not want to replace an entire line, you can create a search and replace text modifier. You can use this type of text modifier to search for certain text and replace it with the specified text. If the text appears more than once in the file, the text modifier finds each occurrence and replaces it with the text you specify.

► To create a search and replace text modifier

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 If the text file you want to edit is *not* in the channel:
 - a Create a text modifier group for that file. For more information, see “Creating a text modifier group” on page 174.
 - b In the left pane, right-click the text modifier group you just created and choose Add Text Modifier > Search and Replace Text.
The New Search and Replace Text Modifier dialog box appears.
 - c Proceed to step 5.
- 4 If the text file you want to edit is part of the channel:
 - a On the Package Contents page, locate the text file you want to edit.
 - b Right-click the text file and choose Add Text Modifier > Search and Replace Text.

- c Specify a name for the text modifier group (for example, application.ini Modifiers) and click OK. The Package Editor has already completed the path for the text file.
- d Proceed to step 5.

5 Specify the following information:

A name for the text modifier—For example, Insert machine name

The text to search for—In the text, you can use any macros defined in the channel. When you use macros within a string, make sure you include a dollar sign (\$) preceding and following the macro name, for example, \$macro_name\$. Also, remember that macros are case-sensitive. For more information about macros, see “Using macros” on page 95.

The search criteria—You can choose from the following criteria:

Exact match—Must match exactly the string you want to find and replace in the text file. Also, the string must be delimited by spaces, tabs, and new lines to be found with an exact match search.

Substring match—Must match the string or part of the string you want to find and replace in the text file. It does not need to be delimited by spaces, tabs, and new lines.

Whether to consider case when searching—Upper or lowercase letters

The text to replace with—In the text, you can use any macros defined in the channel. When you use macros within a string, make sure you include a dollar sign (\$) preceding and following the macro name, for example, \$macro_name\$. Also, remember that macros are case-sensitive. For more information about macros, see “Using macros” on page 95.

6 Click OK.

The text modifier you added appears in the right-hand pane.

Setting installation and update policies for text modifiers

You can use the Package Editor to set installation and update policies for text modifiers. Installation policies determine whether to install the text modifier during channel installation, while update policies determine whether to install the text modifier during a channel update. You can set installation and update policies for text modifiers at two levels:

- You can set the installation and update policies for the text modifier group. By default, any text modifiers in the group inherit the group's installation and update policies.
- You can also override the text modifier group policies and set installation and update policies for individual text modifiers.

This section describes how you can set installation and update policies at both these levels:

- “Setting policies for text modifier groups” on page 181
- “Setting policies for text modifiers” on page 182

For a description of the policies, see Appendix B, “Policies for installation, update, uninstallation, verification, and repair”“Policies for installation, update, uninstallation, verification, and repair” on page 331website.

Setting policies for text modifier groups

The following policies are available for text modifier groups:

Table 4-7: Policies for text modifier groups

Policy type	Description
Installation policies	<ul style="list-style-type: none"> ■ Inherit policy from parent object. ■ Always install modifier group. ■ Never install modifier group.
Update policies	<ul style="list-style-type: none"> ■ Inherit policy from parent object. ■ Update if changed. ■ Always reapply this modifier group. ■ Never update modifier group.

► **To set installation and update policies for a text modifier group**

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 On the Package Contents page, locate the text modifier group for which you want to set installation and update policies.
- 4 Right-click the text modifier group and choose Properties.
- 5 Click the Install Policy tab.
- 6 Select one of the available installation policies.
- 7 Click the Update Policy tab.
- 8 Select one of the available update policies.
- 9 Click OK.

The policies you selected are applied to the text modifier group and any text modifiers that inherit its policies.

Setting policies for text modifiers

The following policies are available for text modifiers:

Table 4-8: Policies for text modifiers

Policy type	Description
Installation policies	<ul style="list-style-type: none">▀ Inherit policy from group.▀ Always install modifier.▀ Never install modifier.
Update policies	<ul style="list-style-type: none">▀ Inherit policy from group.▀ Update if changed.▀ Always reapply this modifier.▀ Never update modifier.

► **To set installation and update policies for a text modifier**

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.

- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 On the Package Contents page, locate the text modifier for which you want to set installation and update policies.

Click on a text modifier group to see the text modifiers in it.
- 4 Right-click the text modifier and choose Properties.
- 5 Click the Install Policy tab.
- 6 Select one of the available installation policies.
- 7 Click the Update Policy tab.
- 8 Select one of the available update policies.
- 9 Click OK.

The policies you selected are applied to the text modifier.

Verifying and repairing files modified by text modifiers

Verification of a file’s contents fails for a file (in a packaged application) that is modified using text modifiers. This is because the file’s contents are different from those in the channel that was originally installed. The workaround for this is to set the individual verify policy for that file to Verify the existence of the file. However, any changes done by the text modifiers cannot be verified or repaired. You can also set the verify and repair policies to Don’t verify file during package verification and Don’t repair file, so that it is not verified or repaired at all during the verify or repair phase.

For more information about verify and repair policies, see “Setting policies for installation, update, uninstallation, verification, and repair” on page 121.

Troubleshooting text modifiers

This section describes some common problems when using text modifiers:

- If the text file to be modified is not part of the channel and the file does not exist at the endpoint, the text modifier is not installed. There is no error message at the endpoint.
- Text modifiers cannot be used to edit Unicode or non-ASCII text files.

- Make sure you have write permission to the file (and that the file is not locked) on the endpoints. Otherwise, the text modifier cannot make changes to the file.
- Also, if the file you specify is on a network drive, do not use the form `\machine_name\directory`. Instead map the network drive and use the assigned drive letter (not the machine name) when you specify the path.

Recording change history for channels



If more than one person makes changes to the channels that you create in your organization, you can keep a record or history of the changes. By recording your changes, you or other users editing the channel can keep track of the settings and contents.

The Package Editor prompts you to add a history entry when you save a channel after making changes. Adding history entries is recommended best practice. You can also add, edit, and delete history entries at any time by following the steps described later in this section.

The following fields are available for each history entry:

- Timestamp
- Author
- E-mail
- Phone
- Subject
- Notes

You can sort the list of history entries on the History page by clicking on the column headings. Also, after you have added a history entry, the Package Editor remembers what you specify in the Author, E-mail, and Phone fields (until you close the Package Editor). It automatically completes those fields when you add another history entry.

Note: When you save a channel after making changes, the Package Editor prompts you to add a history entry before saving your changes. If you select the Do not show this prompt again check box, you or other users editing the channel are no longer prompted to add a history entry when saving changes. To reset this feature so that the prompt again appears every time the channel is edited and saved, edit the parameters.txt file by setting the editor.save.history.prompt.suppress property to false.

► To add a new history entry

- 1 In the Package Editor, click the History tab.
- 2 Click Add to add a new history entry to the channel.
- 3 Specify the following information:
 - Author
 - E-mail
 - Phone
 - Subject
 - Notes
- 4 Click OK.

The history entry you added appears on the History page. In addition to the information you provided, the Package Editor automatically adds a timestamp that records the date and time when the entry is added.

Also, after you have added a history entry, the Package Editor remembers what you specify in the Author, E-mail, and Phone fields during that session (until you close the Package Editor). It automatically completes those fields when you add another history entry

► To edit a history entry

- 1 In the Package Editor, click the History tab.
- 2 Select a history entry and click Edit.
- 3 Edit the information for the history entry.
- 4 Click OK.

The only information you cannot change is the timestamp.

► To remove a history entry

- 1 In the Package Editor, click the History tab.
- 2 Select a history entry and click Remove.

The history entry no longer appears on the History page.

Saving changes to channels

While editing your channel in the Package Editor, you can save your changes at any time. If you exit the editor before saving changes you have made to a channel, you are prompted to save your changes.

If you have made changes to a channel and have not yet saved them, an asterisk (*) appears after the channel name in the title bar.

► To save changes to a channel

- Choose Package > Save.

Saving channels to a network drive

The tuner on which Application Packager is running must be running as a user application and not as a service to access network shared directories.

Application Packager must know the root share directory to browse Universal Naming Convention (UNC) formatted paths or IP addresses. For example, if there is a shared directory called `shared_directory` on the machine called `machine` or with the IP address `xxx.xx.x.xxx`, then to browse the shared directory, the packager would have to type `\machine\shared_directory` or `\xxx.xx.x.xxx\shared_directory` and click Refresh to browse the network resource. Entering `\machine` or `\xxx.xx.x.xxx` by itself does not work, and you cannot browse the machine to find the shared directory.

On Windows, to save a package in a shared drive, the drive should be mapped to the machine that Application Packager is using for packaging.

Reverting to the previously saved version of a channel

While editing a channel in the Package Editor, you can revert to the last version of the channel that was saved. You might want to do this if you have made changes to a channel you do not want to save.

► To revert to the previously saved version of a channel

- Choose Package > Revert to Saved.

The Package Editor undoes any changes you have made after the last time you saved the channel.

Saving a copy of a channel in a different directory

You might want to save a copy of an existing channel to a different directory, so that you can edit it for different environments. When you save a copy of the channel, you can edit the copy without changing the original.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► To save a copy of a channel in a different directory

- 1 Choose Package > Save as.
- 2 Specify the directory where you want to save a copy of the channel and click OK.

The files in the channel are copied to the directory you specified. Any changes you make through the Package Editor now affect the channel in the new location. The path to the new location of the channel now appears at the top of the Package Editor window.

Recovering from channel corruption

When files in the tuner's workspace directory or in a channel are corrupted, a channel can recover from the corruption by getting updates from the transmitter to replace the corrupted files. Unless the version of the channel on the transmitter is also corrupted, a channel can recover from having corrupted files.

Most failures can be attributed to the following causes:

- Java out-of-memory exceptions during installation time might cause the process to be unable to successfully write out parts of the channel (including the index file, which describes the different files that make up the channel).
- Java might fail to execute a process and obtain the process ID. This failure might be caused by timing issues, system busy responses, and no execution permissions issues.
- Index and script files might be damaged on disk as a result of file corruption or removal.

The following log IDs and messages indicate that some files in the channel might have been corrupted and that the channel has attempted to recover so that the channel can run correctly.

Table 4-9: File corruption log IDs and messages

Log ID	Log message
9237	Failed to retrieve old manifest from the end point; creating a blank manifest
9238	Failed to parse new manifest from the tuner storage; check the original package's manifest.ncp for consistency
9239	Failed to parse old uninstaller on the end point; attempting to create a best effort uninstaller
9240	Best effort recovery yielded a blank uninstaller
9241	Failed to load the object state configuration file, creating blank one
9242	The adapter cannot execute because it has user objects and nobody is logged on to the system
9243	There was a problem in determining the currently logged on user; defaulting to no one as logged in
9244	Failed to create scripts directory, trying again
9245	Failed to execute native script, trying again
9246	Path to native script does not exist
9247	Failed to launch the native script in a separate process

Chapter

5 Using Windows Installer Packager

This section shows you how to use the Windows Installer Packager to package Microsoft installations (MSI) created for the Microsoft Windows Installer.

The following topics are provided:

- Overview (page 190)
- Packaging a Windows Installer or MSI application into a channel (page 191)
- Packaging an MSI database into a channel (page 192)
- Customizing Windows Installer packages (page 196)
- Editing the contents of MSI packages (page 213)
- Configuring applications to use the Windows Installer command line (page 214)
- Repackaging Windows Installer packages (page 215)
- (page 216)
- Using Windows Installer redirection (page 216)

Overview

This section shows you how to use Windows Installer Packager GUI. You can also use Windows Installer Packager from the command line. For more information, see “Windows Installer command-line options” on page 311.

Microsoft Windows Installer technologies are divided into two parts that work in combination: a client-side installer service (`msiexec.exe`) and a Windows installation database file (`.msi` file). Windows Installer uses the information contained within a database file to install the application.

- **Installer service**—Windows Installer is an operating system service that allows the operating system to manage the installation process. The `msiexec.exe` program is a component of Windows Installer. This program uses a dynamic link library, `msi.dll`, to read the database files (`.msi`), apply transforms (`.mst`), and incorporate command-line options. The installer performs all installation-related tasks: copying files onto the hard disk, making registry edits, creating shortcuts on the desktop, and displaying dialog boxes to query user installation preferences when necessary.

When Windows Installer is installed on a computer, the file association capabilities of the operating system are modified to recognize the `.msi` file type. When a file with the `.msi` extension is double-clicked, the operating system associates the `.msi` file with Windows Installer and runs the `msiexec.exe` application.

The Windows Installer service ships as part of the Microsoft Windows 2000, Windows XP, and Windows Millennium Edition (Windows ME) operating systems; it can also be installed (from a redistributable on [Microsoft’s website](#)) in Windows 95, Windows 98, and Windows NT version 4.0.

- **Installation database file**—Each database file contains a database that stores all the instructions and data required to install (and uninstall) the software across many installation scenarios. For example, a database file could contain instructions for installing an application when a prior version of the application is already installed. The database file could also contain instructions for installing the software on a computer where that application has never been present. This file has a `.msi` file name extension.

In addition to a database file (also known as an MSI database or MSI file), a Microsoft installation can also include the following other components:

- **External source files or cabinet files required by the installation**
- **Patch files**—Windows Installer uses patch files to patch local or administrative installations. A patch file is a storage file, similar to a descriptor file, and typically has an .msp file name extension. Patches can be applied to more than one application or can upgrade an application into another application or version. A patch does not include a database like a regular installation. Instead, it contains at least one database transform that adds patching information to the database of its target installation.
- **Transform files**—Windows Installer uses transform files to edit the installation database at installation time and dynamically affect the installation behavior. A transform file typically has an .mst file name extension. Transforms are applied at initial installation; they cannot be applied to an already installed application.

Packaging a Windows Installer or MSI application into a channel

You can use the Windows Installer Packager to package Microsoft installations (MSI) that were created for the Microsoft Windows Installer. Windows Installer Packager is useful if the software vendor provided it in Windows Installer format (.msi), and the application does not require frequent updates or changes. You might also want to use this packager if you do not want to violate Microsoft logo compliance.

Note: The Windows Installer Packager component of Application Packager requires Microsoft Windows Installer (`msiexec.exe`) version 1.1 and above. To find out what version you currently have, run `msiexec.exe` from the command line.

The Windows Installer service ships as part of the Microsoft Windows 2000, Windows XP, and Windows Millennium Edition (Windows ME) operating systems; it can also be installed (from a redistributable file on [Microsoft's website](#)) in Windows 95, Windows 98, and Windows NT version 4.0.

► **To package a Windows Installer or MSI application into a channel**

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Select Windows Installer Packager.
- 3 Select the MSI database file (and additional files) you want to package.
- 4 Package the files and settings into a channel.
- 5 (Optional) Use the Package Editor to review or edit the channel.
- 6 Use the channel publishing wizard to publish the channel to a transmitter.

Packaging an MSI database into a channel

The MSI database, contained in a file with the `.msi` extension, contains the core content of an installation and can also control the user interface during installation. When you set properties and options for a Windows Installer package in Application Packager, you edit values of installation options in the MSI database.

Windows Installer Packager prompts you for essential information and then creates a package directory that you publish to a transmitter. This section guides you through the packaging process.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► **To package an MSI database into a channel**

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 To start Windows Installer Packager, perform the following steps:
 - a On the Application Packager main window, click Windows Installer.
 - b Click Create.
- 3 For Database location, specify the path to the location of the MSI database file (with the `.msi` extension) you want to package.
- 4 Select the options you want to use for packaging:

Include all files found at database location—Some MSI databases require supporting files, typically stored in the same location as the .msi file. Select this check box to include these files in the download. If your MSI database contains all the files that are part of the installation, clear this check box.

Note: Be sure you clear this check box if the MSI database is in the root directory or a directory that contains files not related to the MSI database. Those files might get included in the package accidentally.

Package by reference—Select this check box to package files by reference. When you package by reference, the MSI database and supporting files are published directly to the transmitter. For more information, see “Benefits and limitations of packaging files by reference” on page 193.

- 5 In Package location, specify a path to the directory where you want the packager to store the channel.
- 6 Click Create.

Windows Installer Packager creates a package directory in the location you specified. Then it opens the Package Editor, where you specify options for the installation. For information about the Package Editor settings, see “Customizing Windows Installer packages” on page 196.

- 7 When you are ready to publish the channel that Windows Installer Packager has prepared, click Publish in the Package Editor, or choose Package > Publish. You are not required to publish the channel now. The packager has prepared the package directory; you can publish it at any time.
- 8 The publishing wizard appears and guides you through the process of publishing the channel. For more information, see “Publishing channels” on page 279.

Benefits and limitations of packaging files by reference

When you use Windows Installer Packager to package applications, you can choose to package the files by reference. When you package by reference, the MSI database file and supporting files are published directly to the transmitter during publishing.

Packaging files by reference has the following benefits and limitations:

Table 5-1: Benefits and limitations of packaging files by reference

Benefits	Limitations
<ul style="list-style-type: none">▀ File references are useful when you are packaging an application that includes large files. When you add files to a channel, the files' contents are copied to the package directory. As a result, the storage required by the package directory is at least the total size of the original files. The additional storage can be significant, especially for large files. However, if you add files by reference, you do not need the additional storage because the contents of the files are not stored in the package directory.	<ul style="list-style-type: none">▀ Channels that contain files added by reference are not portable because they point to an absolute path on the local machine. Therefore, the channel cannot be transferred to another machine without some changes to the channel.
<ul style="list-style-type: none">▀ File references are also useful when you have a channel with files that are, for the most part, static except for one or two files that are occasionally updated. When updating the channel, you do not need to replace the original files in the channel with the newer versions. If the channel includes references to the updated files in the file system, the contents of the updated files are used when you republish the channel.	<ul style="list-style-type: none">▀ If a channel contains any files that are added by reference, the <code>delayfiledownload</code> parameter does not take effect during updates. For more information, see “Delaying the download of a channel’s files” on page 109.
<ul style="list-style-type: none">▀ You can also take advantage of file references if you have multiple channels that use the exact same file. You save space because you do not need multiple copies of the file in each package directory.	<ul style="list-style-type: none">▀ File attributes, such as timestamps, might not be accurate for files that are added by reference. Because files might have changed from when they were added by reference, the file attributes might be outdated.

Packaging an MSI database with cabinet files

This section describes what to do when an MSI application uses source files compressed in internal or external cabinet (.cab) files, or the application uses external source files that have long file names. A file is considered to have a long file name if it has more than an eight-character name, a period, and a three-character extension (8.3).

This procedure is required if you want to use any of the download options for Windows Installer packages that require contacting the transmitter for additional or deleted files, such as the following options:

- Only download required files
- Download .msi file only
- Delete downloaded files

For more information about these download options, see “Setting the file download options for Windows Installer packages” on page 207.

For example, when you use the Delete downloaded files option for your Windows Installer package, any files downloaded for the installation are deleted after the installation is complete. Later, if the Windows Installer package requires a repair, the repair operation must connect to the transmitter to download any needed files because the files that were previously downloaded during the installation have been deleted.

For the repair operation to succeed, run an administrative installation of the MSI application before you create a Windows Installer package. You can use the administrative installation to extract any source files that have been included with the MSI database file. When you extract these source files and separate them from the MSI database file, you can take advantage of the Delete downloaded files option. When you use the administrative installation, you can make sure that files have short file names by setting the SHORTFILENAMES property. The repair operation looks for files by their short file names.

► To perform an administrative installation of the MSI application and set the SHORTFILENAMES property

- 1 Run the following command on the command prompt:

```
msiexec /a <MSI_application.msi> SHORTFILENAMES=1
```

- 2 After you have performed the administrative installation, you can use the Windows Installer Packager to package the MSI application. For more information, see “Packaging an MSI database into a channel” on page 192.

Packaging an MSI database with merge modules

Merge modules are precompiled bundles of components (files, registry entries, and other modules) that developers can use to add third-party features to an installation. You do not have to do anything special to package an MSI database that contains a merge module—Application Packager treats an MSI database that contains a merge module just as it would any other MSI database.

Customizing Windows Installer packages

After you have packaged an MSI database file, you can customize the package using the Package Editor.

The following sections describe the options you have for customizing Windows Installer packages:

- “Setting the installation modes for Windows Installer packages” on page 197
- “Setting the user interface level for Windows Installer packages” on page 200
- “Customizing MSI error codes and script error codes” on page 201
- “Managing patches for Windows Installer packages” on page 204
- “Managing transforms for Windows Installer packages” on page 205
- “Setting the file download options for Windows Installer packages” on page 207
- “Setting Windows Installer policies” on page 209
- “Setting Windows Installer logging” on page 212

These sections describe options that are available for Windows Installer packages only. Options in the Package Editor that are available for all packages are described in “Using the Package Editor” on page 67.

These sections assume that you have created the Windows Installer package (see “Packaging an MSI database into a channel” on page 192) and opened the Windows Installer package in the Package Editor. For information about opening Windows Installer package in the Package Editor, see “Opening a Windows Installer package in the Package Editor” on page 197.

Opening a Windows Installer package in the Package Editor

Before you can customize a Windows Installer package, open it in the Package Editor.

► To open a Windows Installer package in the Package Editor

- 1 On the main Application Packager window, click Windows Installer.

A list of the packages you have created using the Windows Installer Packager appears. If a package does not appear in this list, import that package. For more information, see “Importing channels” on page 64.

- 2 Select the package you want to edit and click Edit.
- 3 After making changes, be sure to save your package.

Setting the installation modes for Windows Installer packages

You can control the way Windows Installer installs files after the package has been downloaded by setting the Windows Installer installation mode.

► To set the Windows Installer installation mode

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197.
- 2 Click the Windows Installer – Installation tab.
- 3 Select one of the following installation modes:

Perform application installation (default installation mode)—To install the packaged application on target endpoints.

Perform administrative installation, for redistribution—To perform an administrative installation. The Microsoft Windows Installer can perform an administrative installation of an application or product to a network for use by a workgroup. An administrative installation installs a source image of the application onto the network that is similar to a source image on a CD-ROM. Users in a workgroup who have access to this administrative image can then install the product from this source. Users must first install the product from the network to run the application. Users can choose to run-from-source during installation, and the installer uses most of the product's files directly from the network.

Install packaged files “as is”—To download all patches as separate files rather than applying them to the installation image. Use this option to allow users to control which patches are applied to the application.

- 4 Depending on the installation mode you selected in step 3, provide additional information for installation:

-If you selected the default installation mode, you can select the following check boxes:

Reinstall application if it was previously installed prior to the first deployment—If you want the installer to remove a previously installed version of the application and install the new version.

Advertise application—To use the advertisement feature. For more information about advertisement, see “Advertising Windows Installer applications” on page 199.

-If you selected either the administrative installation or “as is” installation, you can specify a default location and whether you want users to be able to specify a different location:

Default location—Default location where the administrative installation or the patch files are stored. For an administrative installation, this location should be your administrative installation point (AIP), which is usually a shared directory on a network server. For patch files, this location is usually a directory on the endpoints.

User settable—To allow users to select a location other than the default. This option prompts users for a location.

- 5 For Enter command line property settings, specify any command-line property settings you want to apply to the Windows Installer when it runs.

For a list of command-line options and properties, see Microsoft Windows Installer Help or [Microsoft’s website](#).

- 6 To enter command-line MSI properties (public properties only) that will be used during the installation of the Windows package, click Set MSI Properties.

For more information on the public MSI properties, see the Microsoft web site at <http://msdn.microsoft.com/en-us/default.aspx>.

- 7 From the MSI Property List Dialog, select the MSI property value, enter a value, and click Add.
- 8 After you have added all of the properties, click Save.

Advertising Windows Installer applications

If you distribute applications that not all users might use, you might want to use the advertisement feature of Windows Installer applications. You can use advertisement to display certain available applications without actually installing the application itself. For example, you can display a shortcut on users' desktops. The Windows Installer application associated with the shortcut is installed only if a user double-clicks the shortcut. Using this feature is identical to using the `msiexec /j` command-line option for advertising a Windows Installer package. The advertisement feature requires Windows Installer (`msiexec.exe`) version 2.0 and higher on the endpoints.

The Windows Installer application is advertised for all users who log into the machine. Application Packager currently does not support user-specific advertisement. Also, if you want to use desktop shortcuts for advertising, use a tool like InstallShield's AdminStudio to add them. Otherwise, the advertised applications appear in the Add/Remove Program Control Panel only.

► To advertise a Windows Installer application

- 1 Open the Windows Installer package in the Package Editor, as described in "Opening a Windows Installer package in the Package Editor" on page 197.
- 2 Click the Windows Installer – Installation tab.
- 3 Select the Advertise application check box.
- 4 For Enter command line property settings at the bottom of the page, type `ALLUSERS=2`.
- 5 Save and publish the Windows Installer package.

When users subscribe to the package, the application is advertised but not installed.

If a package is set to advertise and on an update the Advertise application check box is cleared, the application on the endpoint is installed.

Setting the user interface level for Windows Installer packages

You can control the dialog boxes, error messages boxes, and progress dialog boxes that Windows Installer presents to users during installation by specifying the user interface (UI) level.

The Windows Installer provides Windows Installer developers the ability to author an internal user interface that has levels of UI functionality. Because the internal UI must be authored, the behavior of the UI at each level depends on the Windows Installer package.

► To set the UI level

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197).
- 2 Click the Windows Installer – UI Level tab.
- 3 Select the Set independent UI level for Windows Installer operations check box.

When you select this option, you can set a UI level for Windows Installer operations separate from the settings on the Configuration – Installer page. If you do not select this check box, the Windows Installer presents a UI level equivalent to the installer mode you selected on the Configuration – Installer page.

- 4 Select one of the following UI levels:

Default—To allow Windows Installer to select a default UI from the contents of the package.

None—To specify a completely silent installation.

Basic—To show simple progress dialog boxes and error message boxes. The basic UI does not display any authored dialog boxes.

If you select the basic UI, you have the following additional options:

Omit error handling dialogs—To suppress any modal dialog boxes and show progress dialog boxes only. If there are errors, this option prevents warning error message boxes from appearing to users. This option is useful for silent installations that show progress dialog boxes only. (Selecting this check box has the same effect as setting the channel parameter `msi.ui.basic.progressonly` to true.)

Do not display the Cancel button (MSI 2.0 and higher only)—To hide the Cancel button on progress dialog boxes. This option is available only with Windows Installer 2.0 and later.

Reduced—To suppress authored modal dialog boxes. The reduced UI still shows authored modeless dialog boxes and built-in modal error message boxes. Modal dialog boxes require user input before the installation can continue and are specified by setting the Modal Dialog Style Bit in the Attributes column of the Dialog table. A modeless dialog box does not require user input for the installation to continue.

Full—To show all authored, progress, and error dialog boxes. A full UI commonly exhibits user interface wizard behavior.

- 5 To display a dialog box that shows whether the installation was successful, select the Display success/failure dialog at end of install check box.

This option is useful if you specify a silent installation (UI level is none), but want to see a confirmation at the end of installation.

- 6 To display the dialog boxes used for source resolution even if you specify a silent installation (UI level is none), select the Force display of source resolution even if quiet (MSI 2.0 and higher only) check box.

Because Windows Installer already displays the source resolution dialog boxes for the other UI levels, this option has no effect if the UI level is not none. This option is available only with Windows Installer 2.0 and later.

Customizing MSI error codes and script error codes

MSI packages return error codes that indicate one of two conditions:

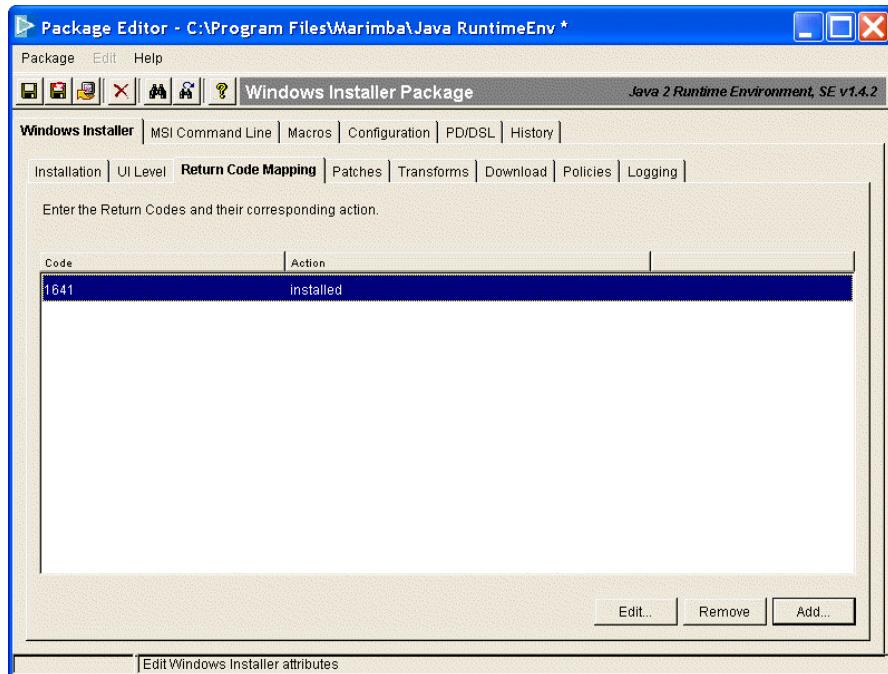
- Installed
- Failed

The same is true for pre- and postinstallation scripts. The Installed state signifies that the process ran successfully, and installed the channel.

Typically, Application Packager interprets as Failed any non-zero code from an MSI package or script. (One exception is code 3010, indicating that a repair operation requires a reboot.)

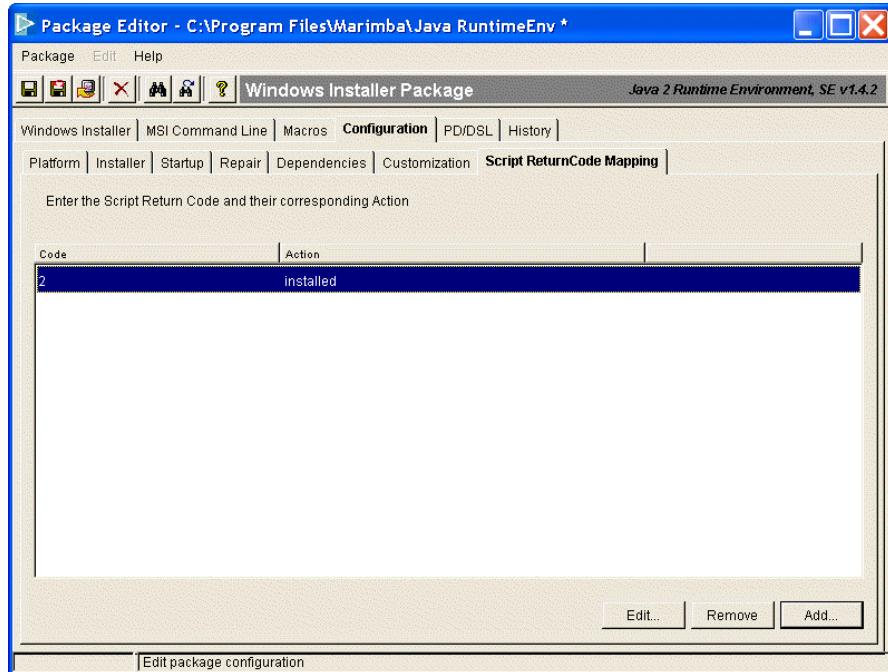
You can manually map error codes related to vendor-specific MSI installers to either the Installed or Failed state. You do this on the Return Code Mapping tab, available on the Windows Installer tab.

Figure 5-1: Return Code Mapping tab



You can use a similar tab, the Script ReturnCode Mapping tab, to map error codes related to pre- and post-installation scripts to the Installed or Failed state. For more information about installation scripts, see Customizing a channel by using scripts and classes (page 141).

Figure 5-2: Script ReturnCode Mapping tab



► To map a return code to the installed or failed state

- 1 Perform one of the following steps:
 - To map an MSI return code, click the Return Code Mapping tab under the Windows Installer tab.
 - To map a script return code, click the Script ReturnCode Mapping tab under the Configuration tab.
 - 2 Click Add.
- If you are mapping an MSI return code, the Add Return Code Entry window appears. If you are mapping a script return code, the Add Script Return Code Entry window appears.
- 3 For Enter the Code, specify the number of the code you want to map.
 - 4 For Enter the Action, select the state you want the code to signify.
 - 5 Click OK.

The custom mapping you specified appears on the list of codes and actions.

Managing patches for Windows Installer packages

A patch is a storage file, similar to a descriptor file, and typically has an .msp file name extension. Patches can be applied to more than one application or can upgrade an application into another application or version. A patch does not include a database like a regular installation. Instead, it contains at least one database transform that adds patching information to the database of its target installation.

When creating and installing patches, be aware of the order in which patches should be installed. You can use the Package Editor to change the order of patches; however, patches themselves can contain information about their order in the installation process. Complying with that order is a good practice.

► To manage patches for a Windows Installer package

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197.
- 2 Click the Windows Installer – Patches tab.
- 3 To add a patch file, perform the following steps:
 - a Click Add.
 - b Specify the following information:

Path to patch—Full patch for the patch file you want to add.

Command-line property settings—Any command-line property settings you want to apply when installing the patch.

For a list of command-line options and properties, see Microsoft Windows Installer Help or [Microsoft’s website](#).

- c Click OK.

The Windows Installer – Patches page appears, with the patch you added. Application Packager checks the validity of the patch and warns you if the patch is invalid.

- 4 To remove a patch file, select the patch file and click Remove.
- 5 To change the order in which Windows Installer should apply patch files, use the up and down arrows:

Up Arrow—To move up the selected patch in the list of patches. Patches are applied in the order in which they appear in the list, top patch first.

Down Arrow—To move down the selected patch in the list of patches.

- 6 If you want the Windows Installer to reinstall the application if the patch list shown for the Windows Installer package is different from the one found on the target system, select the Force reinstall if patch list changes check box.

If you force a reinstall, you can make sure that the target system hosts a stable version of the application. If you do not force a reinstall, only the new patches are applied.

Managing transforms for Windows Installer packages

A transform is a collection of changes applied to an installation. Transforms alter the installation database and can be used to encapsulate the various customizations of a base package required by different groups of users. For example, in organizations where the finance and engineering departments require different installations of a spreadsheet, the spreadsheet's base package can be made available to everyone at one administrative installation point, with the appropriate customizations distributed to each group of users separately, as transforms. Administrators can also apply multiple transforms, on the fly, during an installation to efficiently assign the most appropriate installation to different users.

Windows Installer transforms can be embedded in the .msi database file, or provided standalone as .mst files. You can use Application Packager to provide transforms in either form. The only difference between the two is that embedded transforms are included in the MSI database, while standalone (or external) transforms are in separate files. To apply an embedded transform, you do not need to specify its location. The advantage of an embedded transform is that you always have access to it. Because an external transform is not included in the MSI database, you must identify its location before applying it.

Unlike patches, transforms can be applied only at installation time—double-clicking a transform does not launch it.

Note: When you package an MSI database for silent installation, you can use transforms created by third-party tools to control what is installed. When you create the transforms, make sure that the tool you use creates transforms that can be applied during a silent installation.

► To manage transforms for a Windows Installer package

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197.
- 2 Click the Windows Installer – Transforms tab.
- 3 To add a standalone or external transform file, perform the following steps:
 - a Choose Add > Transform File.
 - b Select the transform file you want to add. Transform files usually have the .mst file extension.
 - c Click Open.

The Windows Installer –Transforms page appears with the transform you added.

- 4 To add an embedded transform, perform the following steps:
 - a Choose Add > Embedded Transform.
 - b Select the embedded transform you want to add.
 - c Click Open.

The Windows Installer – Transforms page appears with the transform you added. Embedded transform names are prefixed with a colon (:)

- 5 To remove a transform, select the transform and click Remove.
- 6 To change the order in which Windows Installer should apply transforms, use the up and down arrows:

Up Arrow—To move up the selected transform in the list of transforms. Transforms are applied in the order in which they appear in the list, top transform first.

Down Arrow—To move down the selected transform in the list of transforms.

- 7 For Path, specify paths for directories in which transform files are stored. Separate the path names with semicolons.

During the download, all .mst files found in these locations are added to the transform list during installation.

- 8 If you want the Windows Installer to reinstall the application if the transform list changes, select the Force reinstall if transform list changes check box.

Transforms can be applied to applications only during installation, so this option is useful if you want to remove the previously installed application and reinstall it with the new transform list.

Setting the file download options for Windows Installer packages

You can control the way Windows Installer downloads files in the package, and whether they are deleted after the installation.

► To set the file download options for a Windows Installer package

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197.
- 2 Click the Windows Installer – Download tab.
- 3 Select one of the following file download options:

Only download required files—To download only those files required for the installation. An installation might have optional features that do not need to be installed; selecting this option omits the files that implement these features. When you select this option, the Windows Installer Packager examines the MSI database file, applies transforms that are included in the package to the MSI database, determines which files pertain to optional features, and omits those files from the package. The Windows Installer does not need to go to the transmitter or a URL for sources during installation.

Download all files—To download all files, including those required for optional features. When all files are included in the download, an installation including these features can proceed without network access. This option is the most reliable because all files are downloaded to the package directory on the endpoint, and that directory is used as the source during installation. However, this option also takes up the most time and resources (such as disk space).

Download .msi file only—To download only the MSI database file and associated patches and transforms. During installation or repair, the Windows Installer gets files from a transmitter or another source location that you have specified using the Alternate source or Additional sources text boxes described in the following steps. For best results, use this option with an administrative installation point (AIP), which is usually a shared directory on a network server, from which the Windows Installer can get additional files.

Important: You can use the option Delay download until prescripts are run to download files only after running prescripts. Do not use the delay download option with staging. You specify the Stage state expressly to download files before running scripts or installing, therefore this state conflicts with the delay download option. For more information about the delay download option, see “Delaying the download of a channel’s files” on page 109.

- 4 To specify an alternate source from which the Windows Installer gets files when installing or repairing an application, specify the source location for Alternate source. You can type a URL or a network path in the Universal Naming Convention (UNC) format, for example,
\\servername\sharename\path\directory_containing_MS1_file or
\\ipaddress\sharename\path\directory_containing_MS1_file.
- 5 To specify additional sources from which the Windows Installer gets files when installing additional features or repairing an application, specify a list of source locations for Additional sources. You can specify a semicolon-delimited list of URLs, local paths, or network paths in the UNC format. You can use a channel URL if you want to use a channel on a transmitter as an additional source. Use this format: http://transmitter:5282/
applications/channelurl?segment=any/any&path=/
where the channel is followed by segment information (?segment=any/any)
and &path=/.

Note: If you specify both alternate and additional sources, the Windows Installer first tries the path specified in the Alternate source text box. Then it tries the paths specified in the Additional sources text box in the order they are listed. The Windows Installer tries both types of sources when installing additional features and repairing the application, but it tries the alternate source only when installing an application for the first time. Alternate and additional sources are most useful if you have selected the Download .msi file only option.

- 6 To remove files from the target endpoints after the installation has been completed, select the Delete downloaded files check box.

This option frees up system resources on the target system, but repair or reinstallation requires another download from a transmitter.

Setting Windows Installer policies

You can set user and machine policies that affect the installation behavior of Windows Installer. When you set these policies for a Windows Installer package, the tuner overrides the current Windows Installer policies during installation. After installation is complete, Windows Installer policies are reset to their original settings.

Note: These policies are applied only when the tuner is operating under an account with the appropriate system privileges.

► To set the Windows Installer policies for a Windows Installer package

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197).
- 2 Click the Windows Installer – Policies tab.
- 3 For Enable/disable Windows Installer, select one of the following levels:
 - Use system setting**—Uses the enabling level of the machine on which the Windows Installer runs.
 - Enable for all applications**—Enables Windows Installer for all applications. All install operations are allowed.
 - Enable for managed applications only**—Enables Windows Installer for managed applications, but disables it for non-managed applications. Non-elevated per-user installations are blocked. Per-user elevated and per-machine installs are allowed.

- 4 For Install with elevated privileges, select one of these settings to control the privilege level of the Windows Installer:

Use system setting—Uses the privilege level allowed by the machine on which the Windows Installer will run.

Use elevated privileges—The Windows Installer always installs with elevated privileges.

Don’t use elevated privileges—The Windows Installer uses elevated privileges to install managed applications and to use the current user’s privilege level for non-managed applications.

Note: As of MSI version 3.1, you can perform a per-user installation for a general Domain User with elevated privileges.

- 5 For Rollback during install, select one of these policies to control whether an installation can be rolled back (uninstalled):

Use system setting—Uses the rollback setting of the machine on which the Windows Installer will run.

Store rollback files during install—Stores rollback files, enabling installation rollback.

Don't store rollback files during install—Prevents the Windows Installer from storing rollback files during installation, disabling installation rollback. Do not select this policy unless it is absolutely essential because it might leave users with no way to recover from an unsuccessful installation.

- 6 For Browsing for alternative install sources, select one of these policies to determine whether users can browse for sources during installation:

Use system setting—Uses the browsing permissions of the machine on which the Windows Installer will run.

Allow browsing—Allows users to browse for installer sources during installation.

Disallow browsing—Prevents users from browsing to locate installer sources. In the Windows Installer, the Use feature from combo box for direct input is locked and the Browse button is disabled. If you make this selection, the following policy selection is disabled.

- 7 For Browsing by non-admin users, select one of these settings to control who can browse for installation sources during elevated installations:

Use system setting—Uses the browsing permissions of the machine on which the Windows Installer will run.

Allow browsing—Allows non-administrative users to browse for new sources while running an installation at elevated privileges. Setting this policy also enables non-administrative users to run programs during an elevated installation.

Any user can browse for new sources during a non-elevated installation. Setting this policy gives non-administrative users the additional flexibility of browsing for new sources during an elevated installation.

Disallow browsing—Allows only administrators to browse for sources during an elevated installation.

- 8 For Use of alternate media sources, select one of these options to control whether nor not users and administrators can use media sources such as a CD-ROM for installations:

Use system setting—Uses the permissions of the machine on which the Windows Installer will run.

Allow—Allows users and administrators to use media sources for installations.

Disallow—Prevents users and administrators from using media sources for installations, regardless of whether the installation is with elevated privileges.

- 9 Use the **Allow User to postpone Installation** option to decide the postpone behavior. Select this option if the user is allowed to postpone installation of the package.

You can configure the maximum postpone time based on:

- **Until a specified time**

This option allows users to postpone installation any number of times until the specified time exceeds (format: mm/dd/yyyy hh:mm a).

- **Until specified number of attempts**

This will allow users to postpone installation only upto the specified number of times.

In the Postpone dialog, you can also type a custom message which will be displayed to the user.

Note: When a package is configured to allow users to postpone installation, the postpone dialog is displayed irrespective of the package UI mode like fully interactive, semi-interactive or silent

The postpone option is applicable only during installation of the package (for packages in “install-pending” or “uninstalled” states).

Setting Windows Installer logging

You can control Windows Installer logging by specifying the location where the log file is stored as well as by selecting the type of information to include in the log file.

You must specify the path for the log file before you can select logging modes. Also, the logging options on this page are not supported on Windows machines that have the tuner running as a service, if the Windows Installer package needs impersonation.

Note: If a file already exists on the endpoint with the same name and path as the one you specified, the file is overwritten. However, if a directory already exists on the endpoint with the same name and path as the log file you specified, the log file is created with a unique name. For example, if you specify the log file path C:/msi.log and there is a directory named msi.log in the C:/ root directory, the log file is created with a unique name, such as C:/msi.log-ncp-0.

► To set Windows Installer logging

- 1 Open the Windows Installer package in the Package Editor, as described in “Opening a Windows Installer package in the Package Editor” on page 197.
- 2 Click the Windows Installer – Logging tab.
- 3 For Log file path, specify the full name and path for the log file that Windows Installer creates on the endpoint.
- 4 Select the check boxes that correspond to the logging modes you want to enable. The logging modes determine the type of information included in the log file.

Setting MSI logging options

You can use the tuner properties in the following table to specify the type of information that is logged in the MSI logs. For example, to log error messages (value = 2) and warning messages (value = 4), set the value of `msi.log.mode`= 6 (2+4).

Property	Value	Description
<code>msi.log.mode</code>	1	Logs out of memory or fatal exit information.
	2	Logs the error messages.
	4	Logs the warning messages.
	8	Logs the user requests.
	16	Logs the status messages that are not displayed.
	64	Request to determine a valid source location.
	128	Indicates insufficient disk space.
	256	Logs the start of new installation actions.
	512	Logs the data record with the installation action.
	1024	Logs the property values at termination.
<code>msi.log.attr</code>	2048	Logs the parameters for user-interface initialization.
	4096	Log verbose information.
<code>msi.log.attr</code>	2	Force the log buffer to be flushed after every 20 log lines.

Editing the contents of MSI packages

If you enable advanced editing, you can perform any tasks in Windows Installer that you can do using the Package Contents tab elsewhere in Application Packager, such as adding files from a new directory, adding registry keys, or creating user-defined macros. For more information, see Chapter 4, “Using the Package Editor”.

► To edit the contents of MSI packages

- 1 Choose Package > Advanced.
- 2 Click the Package Contents tab.

- 3 Edit the contents of your MSI package.

Configuring applications to use the Windows Installer command line

You can bypass the Windows Installer APIs and use the Windows Installer command line to install applications. This is useful when you have applications that need to be installed using a setup.exe file (instead of an MSI database file).

Note: When you use the Windows Installer command line and bypass the Windows Installer APIs, select the Download all files option on the Windows Installer – Download tab in the Package Editor. When bypassing the Windows Installer APIs, some of the customizations to the installation using the Package Editor pages under the Windows Installer tab (Installation, UI Level, Patches, Transforms, Policies, and Logging) are not applied.

► To use the Windows Installer command line

- 1 Open the Windows Installer package for editing.
- 2 In Package Editor, click the MSI Command Line tab.
- 3 For the operation that you want to use on the Windows Installer command line, perform the following steps:
 - a Select the Use command line check box.
 - b In the corresponding text box, specify the command you want to run and the arguments you want to use.
- 4 Use the macro \$msidir to see the ch.xx\data\msi directory in the tuner's workspace directory. For example, to run the setup.exe file during installation, perform the following steps:
 - a Select the Use command line check box beside Install.
 - b In the text box beside Install, type \$msidir\$\setup.exe <args> and any arguments you want to use.
- 5 Save and publish the Windows Installer package.

In the example given, when endpoints download and install the Windows Installer package, the command and arguments you specified are used.

Rewrapping Windows Installer packages

Rewrapping a Windows Installer package involves finding the difference in content between the existing package and the content of the source Windows Installer product that is located on the packaging machine. This difference is used to update the content of the existing package to reflect the content of the source Windows Installer product. Instead of simply replacing the entire contents of the existing package (which would be very time-consuming), only the content that needs to be added or removed from the existing package is processed by the Windows Installer Packager.

► To repackage a Windows Installer package

- 1 On the Application Packager window, click Windows Installer and select the Windows Installer package you want to repackage.

If the Windows Installer package does not appear in the Windows Installer Packages list, choose File > Import to import it. To import the package, determine the location of its package directory.

- 2 Make sure that the `repackage.xml` file you created in the previous procedure appears in the package directory for your Windows Installer channel.

This XML template file is used to apply the configuration to your package after repackaging. If there is no `repackage.xml` file in the package directory, the Application Packager defaults are applied to the package after repackaging.

- 3 Click Repackage and choose one of the following options:

Repackage only—If only supporting files (such as MSI patches and transforms) have changed in your Windows Installer Packager channel, and not the main MSI database file. Repackaging a Windows Installer Packager channel involves finding the difference in content between the existing package and the content of the source Windows Installer product that is located on the packaging machine. This difference is used to update the content of the existing package to reflect the content of the source Windows Installer product. This makes it easier to publish updates of the channels you create.

Reconfigure—If the MSI database file has changed in your Windows Installer Packager channel. You can change the name and path of the MSI database file to the one you want to use. Also choose this option if you want to change the configuration settings in your Windows Installer Packager channel, such as selecting or clearing the Include all files found at database location and Package by reference check boxes.

- 4 Edit, save, and publish the Windows Installer package.

Verify and repair behavior for Windows Installer packages

When you verify or repair a Windows Installer package, Application Packager invokes the Windows Installer repair functionality. The Windows Installer repair functionality reinstalls missing or corrupted files and registry entries, or if necessary reinstalls the application. However, the Windows Installer repair functionality does not support verifying an installed application only. If you verify a Windows Installer package, Application Packager invokes the same Windows Installer repair functionality as it does for a repair. The only difference is that Application Packager runs any scripts that are specified to run for a verify operation.

Using Windows Installer redirection

You can dynamically change source locations (instead of having a fixed source location) for Windows Installer applications that are distributed or managed by Symphony Marimba Client Automation. This feature allows Windows Installer applications to use the scalable transmitter infrastructure (with mirrors and repeaters) for storing source files that are required for repairs and installations of additional features.

In Application Packager, set the channel parameter `msi.redirect` to true or false to switch this feature on or off. The default is false.

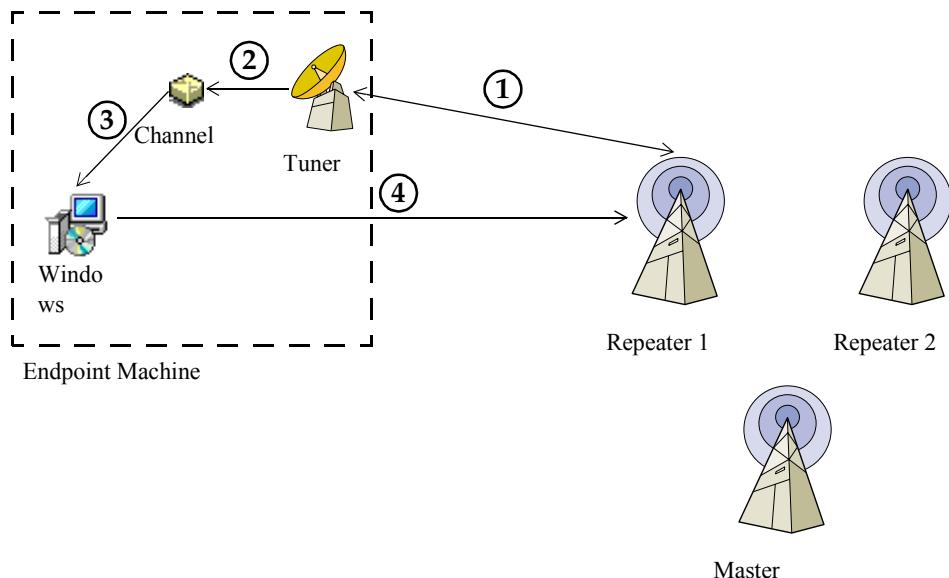
When the Windows Installer redirection feature is on, the following items occur on the endpoints:

- **Installation time**—The Windows Installer channel is aware of the repeater it is coming from and sets the Windows Installer source to use that repeater. The Windows Installer channel sets the Windows Installer source list to include the entire repeater list so that if files are not cached locally by Windows Installer, the channel connects to the repeaters instead of the master transmitter.
- **Update time**—At every major or minor update, the Windows Installer channel resets the Windows Installer source list with any changes in the repeater list.
- **Repair time**—When a repair is triggered through the Symphony Marimba Client Automation/Symphony Marimba Client Automation infrastructure, the Windows Installer channel first validates the repeater list, refreshing it.
- **New action**—Installed Windows Installer channels accept a new argument `-refreshsourcelist` (see “`-refreshsourcelist`” on page 327) on endpoints to get a new repeater list and reset the repeater being used.

Depending on whether the particular repeater (from which a channel is subscribed) is available or not, the Windows Installer might download files from a different repeater. If all of the repeaters are down, the Windows Installer channel defaults to the master transmitter as its URL source and the Windows Installer downloads the file from it during installation. The same behavior occurs during repairs and updates if the Windows Installer application is reinstalled.

“Initially setting a windows installer application’s source location” shows how the Windows Installer application’s source location is set the first time a channel is installed:

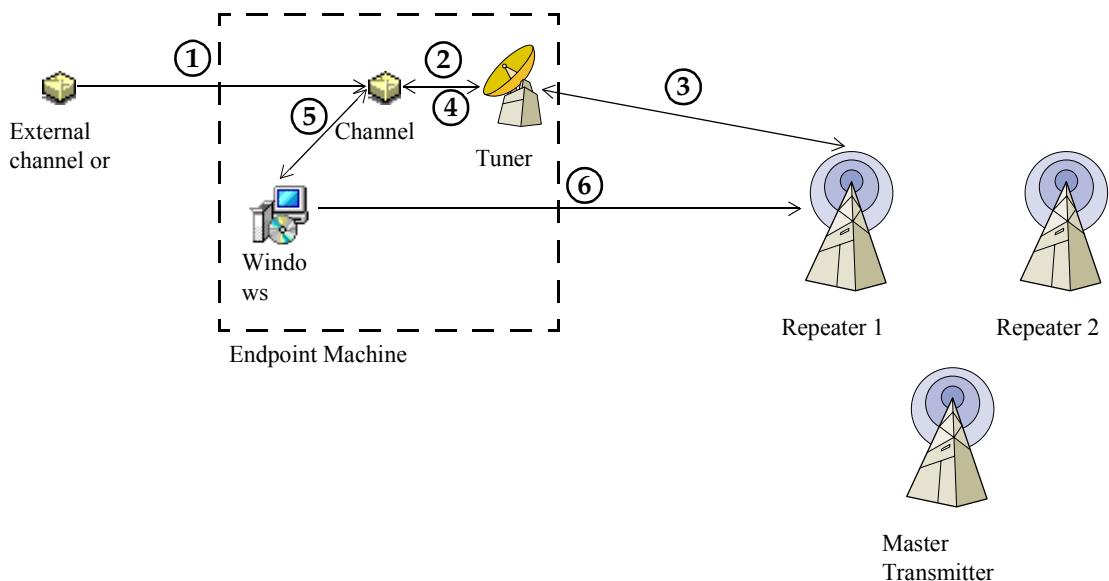
Figure 5-3: Initially setting a windows installer application's source location



- 1 The tuner requests and gets a Windows Installer channel from a repeater.
- 2 The tuner passes the repeater information (including the list of repeaters) to the channel.
- 3 The channel (specifically the adapter) passes the repeater information to Windows Installer.
- 4 Windows Installer identifies the repeaters in the list as sources. It connects to those repeaters when it needs source files for installation, update, or repair.

Figure 5-4 shows how the application source location is refreshed.

Figure 5-4: Refreshing the source location for a Windows Installer application when repeater information changes



- 1 An external channel or script calls the Application Packager API by passing the `-refreshsourcedir` argument.
- 2 The Windows Installer channel (specifically the adapter) asks the tuner to perform an update to obtain the most up-to-date repeater list.
- 3 The tuner requests and gets the latest list of repeaters.
- 4 The tuner passes the repeater information (including the list of repeaters) to the channel.
- 5 The channel (specifically the adapter) passes the repeater information to Windows Installer.
- 6 Windows Installer updates the list of the repeaters that it uses as the source location. It connects to those repeaters when it needs source files for installation, update, or repair.

Chapter

6

Using File Packager

File Packager is one of the packagers included with Application Packager. You can use the Application Packager window to start File Packager and assemble a group of files into a channel. After packaging the channel, you publish it to make it available to users.

The following topics are provided:

- Overview (page 222)
- Packaging a set of files into a channel (page 222)
- Updating a channel (page 226)

Overview

File Packager is best suited for packaging directories of files. Typically, you use File Packager with spreadsheets, HTML files, templates, and other file types that people want to *view* on their local system. You can also use File Packager to redistribute the load from a heavily accessed file server to a group of file servers that mirror the original. It helps keep files up to date on any number of endpoints.

After the files are published as a channel and installed using a tuner, they can be used normally on the endpoint. However, changes you make to the files are overwritten by the next channel update. Accordingly, File Packager is best suited to files that are to be viewed and not modified.

You can configure the channel to launch an application that can be used to view the files. Running the channel from the tuner launches the configured application.

Currently, File Packager produces channels that are platform specific. For example, if you package a channel using File Packager in Windows (95/98/NT/XP), UNIX users are not able to use the channel you publish. To support multiple platforms, create a version of the channel on each platform you want to support.

You can also use File Packager from the command line. For more information, see “File Packager command-line options” on page 313.

Packaging a set of files into a channel

File Packager prompts you for essential information and then creates a package directory that you publish to a transmitter. This section guides you through the packaging process.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

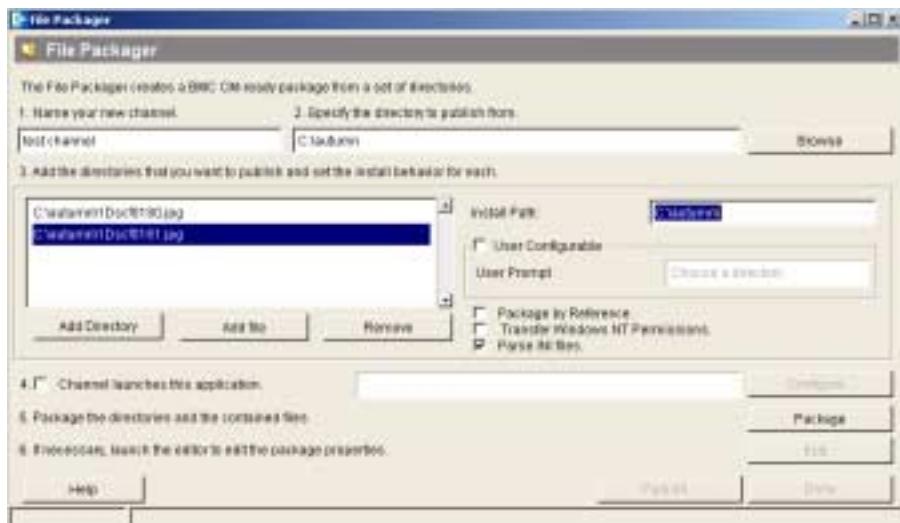
► To package a set of files into a channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.)

- 2 From the list of package types in Application Packager, select File Packager and click Create.

The File Packager dialog box is displayed as shown in Figure 6-1.

Figure 6-1: File Packager dialog box



- 3 Specify a name for the new channel.

The name you select appears in transmitter listings and in the channel listing of tuners that subscribe to the channel.

- 4 Specify a path to the directory where you want File Packager to assemble the channel for publishing.

- 5 To select a directory containing the files you want to package, click Add Directory and select the directory.

All of the directory's files and subdirectories become part of the channel you are creating.

- 6 To select individual files to package, take the following steps:

- a Click Add file.
- b Browse to the directory where the files are located.
- c Click List Files.
- d Select the files that you want to add.

You can select multiple files.

- 7 To suggest an installation directory on the endpoint that differs from the original source directory, perform the following steps:

- a Select the directory or file from the list.
- b For Install Path, specify the installation path.

To install the contents of multiple source directories to the same directory on the endpoint, specify the same installation directory for multiple source directories.

- 8 To allow users to override the default directory at installation time, perform the following steps:

- a Select the User configurable check box.
- b For User prompt, specify a message for users.

Provide a useful message to the user because it might be the only information they receive about the directory where the files are to be installed. For example, if you are creating a channel that gathers spreadsheets from one area and policy documents from another area, you might provide the following user prompt messages: “Where would you like to save the Acme financial spreadsheets?”; “Where would you like to save the Acme human resources policy documents?”

- 9 To package a directory by reference and not include its contents in the channel until publish time, select the Package by Reference check box.

When you package a directory by reference, directory contents reside on the local file system and are not added to the channel until publish time. This is useful when you are packaging an application that includes large files. When you add files to a channel, their contents are copied to the package directory. As a result, the storage required by the channel is at least the total size of the original files. The additional storage can be significant, especially for large files. However, if you add files by reference, you do not need the additional storage because the contents of the files are not stored in the package directory. Packaging by reference does not affect users. For more information about file references, see “Adding files by reference” on page 72.

Note: If you use the Package by Reference option, you cannot transfer Windows permissions.

- 10 (Windows NT, Windows 2000, and Windows XP only) If you want the files in the directory you are packaging to retain their original Windows file permissions, select the Transfer Windows NT Permissions check box.

Windows supports two kinds of file systems: FAT and NTFS. The latter offers more advanced file permission settings. To take full advantage of this option, both the person who is creating the channel and the user must be using NTFS. Otherwise, only basic Windows file permissions or attributes (read-only, archive, hidden, system) are preserved.

Note: Users installing the channel must have administrative privileges to set permissions.

- 11 (Windows only) If you want INI files to be treated as special file objects, select the Parse INI files check box. Application Packager parses the INI files and captures any changes made to the key and value pairs in them. Any changes found by Application Packager are merged with the existing file found at the endpoint. By default, this check box is selected. If you want Application Packager to process INI files as ordinary files and not merge changes, clear this check box. For more information, see “Working with INI files” on page 78.
- 12 Repeat step 5–step 10 for each directory or file that you want to include in the channel you are packaging.

To remove a directory or a file that you added, select it from the list and click Remove.

- 13 To launch an application when the channel runs, select the Channel launches this application option and specify the path of the application executable. The application you specify starts when the channel is started.

If you want more control over when the application is launched, you might want to add some properties to the channel. In the package directory, open the parameters.txt file using a text editor and add one or both of these properties:

```
run.install=true  
run.update=true
```

These properties control whether the specified application should be launched after the initial installation or after updates (both major and minor). For more information, see the *Symphony Marimba Client Automation Reference Guide*, available on the Marimba Channel Store.

- 14 To specify any arguments or a working directory for the application, click Configure and specify any arguments or the name of the working directory.
- 15 After selecting and configuring the directories and files that you want to include in the channel, click Package.

The files are assembled and the package directory is prepared for publishing.

- 16 To edit the properties and contents of the channel, click Edit to start the Package Editor.

For more information about editing the channel, see “Using the Package Editor” on page 67.

- 17 If you are ready to publish the channel that File Packager has prepared, click Publish. (You are not required to publish the channel now. File Packager has prepared the package directory; you can publish it at any time.)

The publishing wizard appears and guides you through the process of publishing the channel. For more information, see “Publishing channels” on page 279.

- 18 Return to the File Packager window and click Done.

Updating a channel

To make it easier to publish updates of the channels you create, File Packager offers the following options:

- **Reconfigure**—To add or remove directories from your File Packager channel. Also select this option if you want to reconfigure the default installation path, user prompt, and application settings in your File Packager channel.
- **Repackage only**—If only files (not directories) have changed in your File Packager channel, and you do not want to publish the channel yet. This option does not republish the channel; as its name implies, it only repackages the channel (that is, it gathers the set of required files into a package directory for publishing). Select this option if you want to publish your repackaged channel to a destination different from where you originally published it. For example, use this option if you originally published the channel to your staging transmitter and now you want to publish the repackaged channel to the production transmitter.

- **Repackage and Publish**—If only files (not directories) have changed in your File Packager channel, and you want to publish the channel to the same destination where you originally published it. This option automatically publishes the repackaged channel to the destination where you previously published it.

► To reconfigure an existing channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Click File Packager.
- 3 Select the channel you want to reconfigure from the list of file packaged channels.
- 4 Click Repackage.
- 5 From the pop-up menu that appears, choose Reconfigure.
- 6 Use File Packager to add and remove directories or files from the channel. You can also configure other settings for the channel. For more information, see “Packaging a set of files into a channel” on page 222.

When you have finished making changes to the channel, you can package and publish it.

► To repackage an existing channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Click File Packager.
- 3 Select the channel you want to repackage from the list of file packaged channels.
- 4 Click Repackage.
- 5 From the pop-up menu that appears, choose Repackage only or Repackage and Publish.

-Choose Repackage only to publish the channel to a destination other from where you previously published it.

File Packager gathers the files and copies them to the package directory using the parameters you set for the channel. Publish the channel to complete the update process.

-Choose Repackage and Publish to publish the channel to the destination where you previously published it.

File Packager gathers the files and copies them to the package directory using the parameters you set for the channel. After repackaging, File Packager automatically publishes the channel to the destination location where you previously published it.

Installing Microsoft Office 2007 using File Packager

The following procedure describes how to use File Packager to package and install Office 2007.

Before you begin

You can configure Office 2007 installation options such as, silent installation, reboot control, installation directory, etc., in the config.xml file in the <PackageID>.WW folder. <PackageID> can be Enterprise, Prop, etc. For more information on editing the config.xml file, see the following Microsoft TechNet article: <http://technet2.microsoft.com/Office/en-us/library/41f07f9b-f0d0-489d-a185-d7b96f21f5611033.mspx>

► To install Office 2007 using File Packager

- 1 Modify the config.xml file if needed.
- 2 Using Application Packager, create a file package for Office 2007.
- 3 For the file package you created in step 2, create a post-installation batch script that contains:
 - a the command to invoke setup.exe after installation has finished from File Packager (File Packager puts the file in the destination folder without invoking setup or installing any .exe files.)
 - b any reboot options required after a successful installation
- 4 Publish the package as a channel.
- 5 Install the published Office 2007 file package. After installation, it executes the post-installation batch script which invokes the setup.exe file.

► **To repair an Office 2007 installation using File Packager**

To repair Office 2007, follow the previous steps for installing using File Packager with one difference. When you invoke the `setup.exe` file in the post-installation batch script, you must specify the repair option:

`<path>\Setup.exe /repair <PackageID>`

Where `/repair` is the repair command and `<packageID>` is the ID of the package, such as Enterprise, Prop, etc.

Chapter

7

Using .NET Packager

This section shows you how to use the .NET Packager to package .NET applications.

The following topics are provided:

- Overview (page 232)
- Components of .NET applications (page 232)
- Packaging a .NET application into a channel (page 234)
- Editing a .NET Packager channel (page 236)
- The .NET Assembly properties (page 237)
- Adding virtual directories for Internet Information Services (page 244)
- Updating a channel (page 244)

Overview

.NET Packager is a component of Application Packager that you can use to package applications that have been created using Microsoft's .NET framework. The .NET framework is Microsoft's platform for building, deploying, and running Web services and applications.

You use .NET Packager to specify the files and directories that make up the .NET application. If you have used the File Packager component of Application Packager previously, using .NET Packager should be familiar. After packaging, you can use the Package Editor to set the configuration for the application. Then you can publish the packaged application or channel to a transmitter, so that it can be distributed to other machines.

Components of .NET applications

Applications that have been created using the .NET framework include the following components. You can use .NET Packager and the Package Editor to configure these components.

Applications created using the .NET framework consist of assemblies. An assembly is the primary building block of a .NET framework application. It is a collection of functionality that is built, versioned, and deployed as a single implementation unit (one or multiple files). All managed types and resources are marked either as accessible only within their implementation unit or as exported for use by code outside that unit. In the common language runtime (CLR), the assembly establishes the name scope for resolving requests and the visibility boundaries are enforced. The runtime can determine and locate the assembly for any running object because every type is loaded in the context of an assembly.

Assemblies can be static or dynamic. Static assemblies can include .NET framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on). Static assemblies are stored on disk in portable executable (PE) files. You can also use the .NET framework to create dynamic assemblies, which are run directly from memory and are not saved to disk before execution. You can save dynamic assemblies to disk after they have executed.

There are several ways to create assemblies. You can use development tools, such as Visual Studio .NET, that you have used in the past to create files with the .dll or .exe extensions. You can use tools provided in the .NET framework SDK to create assemblies with modules created in other development environments. You can also use common language runtime APIs, such as `Reflection.Emit`, to create dynamic assemblies.

An assembly file usually has the file extension `.netmodule`. It can be compiled as part of an EXE or DLL file.

■ **Assembly manifest**—The assembly manifest is an integral part of every assembly that renders the assembly self-describing. The assembly manifest contains the assembly's metadata. The manifest:

- Establishes the assembly identity.
- Specifies the files that make up the assembly implementation.
- Specifies the types and resources that make up the assembly.
- Itemizes the compile-time dependencies on other assemblies.
- Specifies the set of permissions required for the assembly to run properly.

This information is used at run time to resolve references, enforce version binding policy, and validate the integrity of loaded assemblies. The self-describing nature of assemblies also helps makes zero-impact install and XCOPY deployment feasible.

■ **Global assemblies cache**—Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache (GAC). The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer.

Share assemblies by installing them into the global assembly cache only when necessary. As a general guideline, keep assembly dependencies private and locate assemblies in the application directory unless sharing an assembly is explicitly required. In addition, you do not have to install assemblies into the global assembly cache to make them accessible to COM interop or unmanaged code.

Packaging a .NET application into a channel

.NET Packager prompts you for essential information and then creates a package directory that you publish to a transmitter. If you have previously used File Packager, the packaging process is similar. This section guides you through the packaging process.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► To package a .NET application into a channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.)
- 2 From the list of package types in Application Packager, select .NET and click Create.
- 3 Specify a name for the new channel.

The name you select appears in transmitter listings and in the channel listing of tuners that subscribe to the channel.

- 4 Specify a path to the directory where you want .NET Packager to assemble the channel for publishing.
- 5 Click Add and select a directory containing files you want to package.

All of the directory’s files and subdirectories become part of the channel you are creating.

- 6 To suggest an installation directory on the endpoint that differs from the original source directory, perform the following steps:
 - a Select the directory from the list.
 - b For Install Path, specify the installation path.

To install the contents of multiple source directories to the same directory on the endpoint, specify the same installation directory for multiple source directories.

- 7 To allow users to override the default directory at installation time, perform the following steps:

- a Select the User configurable check box.
- b For User prompt, specify a message for users.

Provide a useful message because it might be the only information they receive about the directory where the files are to be installed.

- 8 To package a directory by reference and not include its contents in the channel until publish time, select the Package by Reference check box.

-When you add files to a channel, their contents are copied to the package directory. As a result, the storage required by the channel is at least the total size of the original files. The additional storage can be significant, especially for large files.

-However, when you package a directory by reference, directory contents reside on the local file system and are not added to the channel until publish time. You do not need additional storage because the contents of the files are not stored in the package directory. This is useful when you are packaging an application that includes large files. Packaging by reference does not affect users. For more information, see “Adding files by reference” on page 72.

Note: If you use the Package by Reference option, you cannot transfer Windows permissions.

- 9 (Windows NT, Windows 2000, and Windows XP only) If you want the files in the directory you are packaging to retain their original Windows file permissions, select the Transfer Windows NT Permissions check box.

Windows supports two kinds of file systems: FAT and NTFS. The latter offers more advanced file permission settings. To take full advantage of this option, both the person who is creating the channel and the user must be using an NTFS file system. Otherwise, only basic Windows file permissions or attributes (read-only, archive, hidden, system) are preserved.

Note: Users installing the channel must have administrative privileges to set permissions.

- 10 If you want INI files to be treated as special file objects, select the Parse INI files check box.

Application Packager parses the INI files and captures any changes made to the key and value pairs in them. Any changes found by Application Packager are merged with the existing file found at the endpoint. By default, this check box is selected. If you want Application Packager to process INI files as ordinary files and not merge changes, clear this check box. For more information, see “Working with INI files” on page 78.

- 11 Repeat step 5–step 10 for each directory you want to include in the channel you are packaging.

To remove a directory you have added, select it from the list and click Remove.

- 12 If you want assembly policy configuration files to be treated as special file objects, select the Parse assembly policy configuration files check box.

- 13 After selecting and configuring the directories you want to include in the channel, click Package.

The files are assembled and the package directory is prepared for publishing.

- 14 To edit the properties and contents of the channel, click Edit to start the Package Editor.

For more information about editing the channel, see “Using the Package Editor” on page 67.

- 15 If you are ready to publish the channel that .NET Packager has prepared, click Publish. (You do not need to publish the channel now. File Packager has prepared the package directory; you can publish it at any time.)

The publishing wizard appears and guides you through the process of publishing the channel. For more information, see “Publishing channels” on page 279.

- 16 Return to the .NET Packager window and click Done.

Editing a .NET Packager channel

You can edit .NET Packager channels using the Package Editor. The appropriate time to use the editor is after you package the channel and before you publish it. You can use this editor to refine the channel by adding, removing, and renaming files and directories. You can also configure .NET assembly policy configuration properties. For more information, see “Using the Package Editor” on page 67.

Note: If you import and edit a channel that was packaged using a previous version of Application Packager, you are asked if you want to upgrade the channel. You must upgrade the channel before you can edit it with the current version of Application Packager.

► To edit the .NET Packager channel

- 1 On the Application Packager main window, click .NET Packager.
- 2 Select the channel you want to edit.
- 3 Click Edit.
- 4 Make the necessary changes using the Package Editor. For .NET applications, configure the .NET assembly policy configuration properties, which is described in step 5.

For more information about configuration and settings that apply to all packages, see “Using the Package Editor” on page 67.

- 5 To configure .NET assembly policy configuration properties, perform one of the following steps:

-Select a file with a .dll or .exe file extension, right-click the file, and choose .NET Properties.



These files usually appear with the .NET assemblies icon.

-Select a file with a .config file extension, right-click the file, and choose Properties.



These files usually appear with the XML assembly policy configuration icon.

The .NET Assembly Properties or .NET Assembly Policy Configuration Properties dialog box appears. You can use the various pages in this dialog box to configure assembly properties. For more information, see “The .NET Assembly properties” on page 237.

The .NET Assembly properties

The following sections describe the options available in the .NET Assembly Properties and .NET Assembly Policy Configuration Properties dialog box:

“Properties page” on page 238

“Application Policy Configuration page” on page 238

- “Assembly binding” on page 239
- “Publisher policy” on page 239
- “Probing” on page 239
- “Garbage collection” on page 240
- “Dependent assemblies” on page 240
- “Development mode” on page 242
- “Assembly qualifiers” on page 243

Properties page

Use this page to register an assembly into the global assembly cache (GAC) and to view properties for an assembly.

Register assembly into Global Assembly Cache (GAC) — Select this check box to register an assembly into the GAC and to make the assembly available to other applications. The GAC stores assemblies specifically designated to be shared by several applications on the computer.

This page shows the following .NET assembly properties:

- Name
- Version
- Locale
- Public key token

Application Policy Configuration page

An application policy configuration file is used to control the runtime behavior of a .NET assembly. The pages in this section configure the different policies that can be applied to a .NET assembly. The configuration file is a standard XML file that contains a set of elements (defined by the .NET framework) that implement configuration settings. To directly edit the configuration file, you should be familiar with XML and be aware that XML tags and attributes are case-sensitive. The pages in this section allow you to create or modify the configuration file without having to directly edit the XML file.

Create application policy configuration file — Select this check box to create a policy configuration file for the assembly. You must select this check box before you can set any of the options and pages in this section.

Assembly binding

This page contains information about assembly version redirection and the locations of assemblies. The text box specifies the XML namespace required for assembly binding. The default is the string `urn:schemas-microsoft-com:asm.v1`.

Publisher policy

This page specifies whether the runtime applies the publisher policy for one or more assemblies. The publisher policy determines which version of an assembly applications use. When an assembly vendor releases a new version of an assembly, the vendor can include a publisher policy so applications that use the old version now use the new version. You can specify whether to apply publisher policy in the application's configuration file for either a specific assembly or all assemblies the application uses.

Enable property — Select this check box to specify whether to apply the publisher policy. By default, this check box is selected.

Apply publish policy for all assemblies — Select this check box to specify whether to apply the publisher policy for all assemblies the application uses. By default, this check box is selected.

Probing

This page specifies application base subdirectories for the common language runtime (CLR) to search when loading assemblies.

Enable property — Select this check box to specify whether to specify subdirectories to search. By default, this check box is selected.

In the text box, specify the subdirectories of the application's base directory that might contain assemblies. Delimit each subdirectory with a semicolon.

Garbage collection

This page specifies whether the runtime runs garbage collection concurrently. If your application is single-threaded and involves heavy user interaction, leave concurrent garbage collection enabled so the application does not pause to perform garbage collection. If your application is a server application that is heavily multithreaded and is not user-interface intensive, turn off concurrent garbage collection to improve performance.

Enable property and Use concurrent garbage collection — Select these check boxes to specify whether the runtime runs garbage collection concurrently. By default, both check boxes are selected.

Dependent assemblies

This page specifies the dependent assemblies along with their binding policy and location. For each dependent assembly, this page shows the following information:

- Name
- Culture
- Public key
- Publisher policy

Add — Click to add a dependent assembly to the configuration file. A dialog box appears, where you can specify information for the assembly.

Edit — Click to edit the selected dependent assembly. A dialog box appears, where you can change the information for the assembly.

Remove — Click to remove the selected dependent assembly from the configuration file.

Add or edit .NET dependent assembly

Contains identifying information about the assembly.

Assembly name — Enter the name of the assembly (for example, myAssembly)

Specify culture — Select the check box, and, in the text box, enter a string that specifies the language and country/region of the assembly (for example, en-us for English-United States).

Specify public key — Select the check box, and, in the text box, enter a hexadecimal value that specifies the strong name of the assembly (for example, 32ab4ba45e0a69a1).

Code base

Specifies where the common language runtime can find an assembly. For the runtime to use the codebase setting in a machine configuration file or publisher policy file, the file must also redirect the assembly version.

Application configuration files can have a codebase setting without redirecting the assembly version. After determining which assembly version to use, the runtime applies the codebase setting from the file that determines the version. If no codebase is indicated, the runtime probes for the assembly in the usual way.

If the assembly has a strong name, the codebase setting can be anywhere on the local intranet or the Internet. If the assembly is a private assembly, the codebase setting must be a path relative to the application's directory.

Version/Name — Specifies the version of the assembly the codebase applies to. The format of an assembly version number is *major.minor.build.revision*. Valid values for each part of the version number are 0 to 65535 (for example, 2.0.0.0).

Href/Location — Specifies the URL where the runtime can find the specified version of the assembly (for example, <http://www.litwareinc.com/myAssembly.dll>).

Redirection

Redirects one assembly version to another. When you build a .NET framework application against a strong-named assembly, the application uses that version of the assembly at run time by default, even if a new version is available. However, you can configure the application to run against a newer version of the assembly. For details on how the runtime uses these files to determine which assembly version to use, see Microsoft's .NET documentation.

You can redirect more than one assembly version by including multiple redirection elements in a dependent assembly element.

Old version — Specifies the version of the assembly that was originally requested. The format of an assembly version number is *major.minor.build.revision*. Valid values for each part of this version number are 0 to 65535 (for example, 1.0.0.0).

You can also specify a range of versions in the following format:

n.n.n.n - n.n.n.n

New version — Specifies the version of the assembly to use instead of the originally requested version in the format: *n.n.n.n* (for example, 2.0.0.0).

Publisher policy

This page specifies whether the runtime applies the publisher policy for a particular assembly. The publisher policy determines which version of an assembly applications use. When an assembly vendor releases a new version of an assembly, the vendor can include a publisher policy so applications that use the old version now use the new version.

Enable property and Apply publisher policy for dependent assembly — Select these check boxes to specify whether to apply the publisher policy to this particular assembly. By default, these check boxes are not selected.

Development mode

This page specifies whether the runtime searches for assemblies in directories specified by the DEVPATH environment variable.

Note: Important! Use this setting only at development time. The runtime does not check the versions on strong-named assemblies found in the DEVPATH. It simply uses the first assembly it finds.

Enable property and Use development mode — Select these check boxes to search for assemblies in directories specified by the DEVPATH environment variable. By default, neither check box is selected.

Assembly qualifiers

This page specifies the full name of the assembly that should be dynamically loaded when a partial name is used. Calling the `Assembly.Load` method using partial assembly names causes the common language runtime to look for the assembly only in the application base directory. Add assembly names to this page to provide the full assembly information (name, version, public key token, and culture) and cause the common language runtime to search for the assembly in the global assembly cache. This page displays the following information:

- **Partial name**—Specifies the partial name of the assembly as it appears in the code. It must partially reference an assembly. You must specify at least the assembly's text name (the most common case), but you can also include version, public key token, or culture (or any combination of the four, but not all four). The partial name must match the name specified in your call. For example, you cannot specify `math` as the partial name and call `Assembly.Load("math, Version=3.3.3.3")` in your code.
- **Strong name**—Specifies the full name of the assembly as it appears in the global assembly cache. It must include the four fields of assembly identity: name, version, public key token, and culture.

Example

Partial name=`math`

Strong name=

`math,version=1.0.0.0,publicToken=a1690a5ea44bab32,culture=neutral`

Add—Click to add an assembly qualifier to the configuration file. A dialog box appears, where you can specify the partial name and strong name for the assembly.

Edit—Click to edit the selected assembly qualifier. A dialog box appears, where you can change the partial name or strong name for the assembly.

Remove—Click to remove the selected assembly qualifier from the configuration file.

Adding virtual directories for Internet Information Services

Virtual directories are aliases on an Internet Information Services (IIS) web server website that point to content located on a local path or network path. You can use the Package Editor to add IIS virtual directories to a package and specify the alias, path, and simple permissions (such as read, run scripts, execute, write, and browse). When the package is installed, it automatically locates IIS 5.0 (if installed) and registers the packaged application with IIS as a web service.

► To add a virtual directory for IIS

- 1 In the Package Editor, click the Package Contents tab.
- 2 Right-click Internet Information Services (IIS), (If IIS does not appear, right-click My Computer, and choose New > IIS Virtual Directory.)
- 3 Specify the following information:
 - Alias
 - Path
 - Access Permissions
- 4 Click OK.

The IIS virtual directory you added appears in the Package Contents page.

Updating a channel

To make it easier to publish updates of the channels you create, .NET Packager offers the following options:

- **Reconfigure**—To add or remove directories from your .NET Packager channel. Also select this option if you want to reconfigure the default installation path, user prompt, and application settings in your .NET Packager channel.
- **Repackage only**—If only files (not directories) have changed in your .NET Packager channel (for example, if you have added, changed, or removed some files from the directories already included in the channel). This option repackages the files and directories, but it retains any settings you previously made to the channel using Package Editor.

Before repackaging, save the previous configuration of the .NET Packager channel in an XML template file so that it can be used on a subsequent repackage. After the .NET Packager channel has been repackaged with updated content, the XML template file is applied to restore the previous configuration.

► To save the configuration for your .NET Packager channel

- 1 On the Application Packager window, click .NET Packager and select the .NET Packager channel you want to repackage.

If the .NET Packager channel does not appear in the .NET Packages list, choose File > Import to import it. To import the package, determine the location of its package directory.

- 2 Click Edit.

The package opens in the Package Editor.

- 3 Choose Package > Save as XML Template File.
- 4 From the Specify XML Template File dialog box, select the Include content settings check box.
- 5 Click Browse and locate the package directory for your .NET Packager channel.
- 6 Type `repackage.xml` for the XML template file name.
- 7 In the Specify XML Template File dialog box, click Continue.

The package configuration is saved in the `repackage.xml` file.

- 8 Exit the Package Editor.

- 9 If necessary, you can edit the `repackage.xml` file and add additional tags.

For information about the tags you can use in these XML template files, see “XML syntax” on page 369.

► To reconfigure an existing channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Click .NET Packager.
- 3 Select the channel you want to reconfigure from the list of channels.

- 4 Make sure that the `repackage.xml` file appears in the package directory for your .NET Packager channel.

This XML template file is used to apply the configuration to your package after repackaging. If there is no `repackage.xml` file in the package directory, the Application Packager defaults are applied to the package after repackaging.

- 5 Click Repackage.
- 6 From the pop-up menu that appears, choose Reconfigure.

The channel you selected opens in .NET Packager.

- 7 Use .NET Packager to add and remove directories from the channel. You can also configure other settings for the channel. For more information, see “Packaging a .NET application into a channel” on page 234.
- 8 When you have finished making changes to the channel, you can package and publish it.

► To repackage an existing channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 Click .NET Packager.
- 3 Select the channel you want to repackage from the list of file packaged channels.
- 4 Make sure that the `repackage.xml` file appears in the package directory for your .NET Packager channel.

This XML template file is used to apply the configuration to your package after repackaging. If there is no `repackage.xml` file in the package directory, the Application Packager defaults are applied to the package after repackaging.

- 5 Click Repackage.
- 6 From the pop-up menu that appears, choose Repackage only.
.NET Packager gathers the files and copies them to the package directory using the parameters you set for the channel.
- 7 When you have finished making changes to the channel, you can package and publish it.

Chapter

8

Using PDA Packager

You can use the PDA Packager to package directories of files or PDA (personal digital assistant) applications in the form of CAB files.

The following topics are provided:

- Overview (page 248)
- Packaging files and PDA applications into a channel (page 248)
- Deploying PDA packages to mobile devices (page 252)

Overview

You can use the PDA Packager to create packages for Personal Digital Assistant (PDA) applications that can be distributed directly to a device tuner on a mobile device. Directories of files might include HTML files, text files, or even .exe files. The CAB file format is a format often used for applications that run on mobile devices. The specific supported platform for CAB files is Strong Arm 1100.

This strategy alleviates the need to first download the package to a desktop PC and then synchronize the mobile device to the desktop to place the files or applications on the mobile device.

Packaging files and PDA applications into a channel

The PDA Packager is similar to the File Packager. The package directory created by the PDA Packager is somewhat different, however. For more information about the package directory for PDA packages, see the *Device Management Administrator's Guide*.

Note: When you specify the directory in which to create or save the package, make sure the directory does not already contain another package. Otherwise, the package you create might not be usable.

The following features, which are included in the packaged applications for desktop PCs, are not included in packages for mobile devices:

- Verification of applications
- Creation of shortcuts
- Checks for sufficient disk space
- Checks for dependencies
- Support for policies (for example, install and update policies)
- Support for adding files by reference

Note: Before you use the PDA Packager to package a CAB file, find out the exact name of the installed application. Therefore, first install the application on one mobile device by using ActiveSync, as you would if you did not have Symphony Marimba Client Automation products. Then use the mobile device's Remove Programs list to find the complete name of the application, and use that name in the PDA Packager. (To find the Remove Programs list, on the mobile device choose Start > Settings > System > Remove Programs.)

► To package an application into a channel

- 1 On the desktop computer you will use for creating PDA packages, make sure your tuner is running.
- 2 Double-click the Application Packager channel, which appears in the Channel Manager.
- 3 From the list of package types in Application Packager, select PDA and click Create.
- 4 Specify a name for the new channel you want to create.

The name you select appears in transmitter listings and in the channel listing of tuners that subscribe to the channel.

- 5 Specify a path to the directory where you want the PDA Packager to assemble the channel for publishing.

This directory name is also used as the URL name when the package is copied to your transmitter.

- 6 To package a directory of files, perform the following steps:
 - a Click Add dir, and browse to the directory that contains files you want to package.

All of the directory's files and subdirectories become part of the channel you are creating.

- b For Install Path, specify the path to the installation directory you want to use on the mobile device.

In most cases, use a short path. For example, if you package C:\Program Files\Microsoft ActiveSync\MyApplication, you probably want the install path to be \MyApplication.

Tip: To install the contents of multiple source directories to the same directory on the endpoint, specify the same installation directory for multiple source directories.

- c To package a directory by reference and not include its contents in the channel until publish time, select the Package by Reference check box.

When you package a directory by reference, directory contents reside on the local file system and are not added to the channel until publish time. This is useful when you are packaging an application that includes large files. When you add files to a channel, their contents are copied to the package directory. As a result, the storage required by the channel is at least the total size of the original files. The additional storage can be significant, especially for large files. However, if you add files by reference, you do not need the additional storage because the contents of the files are not stored in the package directory. Packaging by reference does not affect users.

- d Repeat step a–step c for each directory you want to include in the channel you are packaging.

7 To package a CAB file, perform the following steps:

- a Click Add CAB, and browse to the CAB file you want to package.

In some cases, you should select the CAB file that corresponds to the supported platform, which is Strong Arm 1100.

Note: The CAB file must be in the same directory as the Publish directory.

- b For App Name, specify the name of the application.

Tip: To find out what the application name should be, first install the application on one mobile device by using ActiveSync, as you would if you did not have Symphony Marimba Client Automation products. Then use the mobile device's Remove Programs list to find the complete name of the application, and use that name in the PDA Packager. (To find the Remove Programs list, on the mobile device choose Start > Settings > System > Remove Programs.)

- c To package a CAB file by reference, select the Package by Reference check box.
 - d Repeat step a-step c for each CAB file you want to include in the channel you are packaging.
- 8 After selecting and configuring the directories or CAB files (or both), click Package.
- The files are assembled and the package directory is prepared for publishing.
- 9 If you are ready to publish the channel that the PDA Packager has prepared, click Publish. (You are not required to publish the channel now. The PDA Packager has prepared the package directory; you can publish it at any time.)
- The publishing wizard appears and guides you through the process of publishing the channel.
- 10 Complete the wizard pages to publish the channel to the transmitter of your choice. Each wizard page contains a Help button that can give you information about the fields on that page.
- 11 After the channel has been successfully published and the PDA Packager window appears, you can exit the Application Packager.

Using the PDA packager to update a channel

To make it easier to publish updates of the channels you create, the PDA Packager contains the following options:

- **Reconfigure**—To add or remove directories or CAB files from your channel. You should also select this option if you want to reconfigure the default installation path or application name.
- **Repackage only**—If only files (not directories) have changed in your channel, and you do not want to publish the channel yet. This option does not republish the channel; as its name implies, it only repackages the channel (that is, it gathers the set of required files into a package directory for publishing). You should select this option if you want to publish your repackaged channel to a destination different from where you originally published it. For example, you should use this option if you originally published the channel to your staging transmitter and now you want to publish the repackaged channel to the production transmitter.

- **Repackage and Publish**—If only files (not directories) have changed in your channel, and you want to publish the channel to the same destination where you originally published it. This option automatically publishes the repackaged channel to the destination where you previously published it.

Deploying PDA packages to mobile devices

Now that you have published the application package, you can deploy the package to your mobile devices by subscribing the device tuners to the package.

► To deploy PDA packages to mobile devices

- 1 Subscribe a mobile device tuner to the package, either by manually subscribing or by using Policy Manager from the Symphony Marimba Client Automation console.
- 2 To manually subscribe, on the mobile device, use Internet Explorer to browse to your transmitter and subscribe to the channel. Perform the following steps:
 - a If necessary, choose View > Address Bar to display the address bar, so that you can specify the URL for your transmitter. Be sure to use the fully qualified name of the machine that hosts the transmitter.
 - b If necessary, scroll to find the correct channel name, and then click to subscribe to the channel. When the configuration Download dialog box appears, click Yes.

After your channel is downloaded, which can take several seconds, the MarimbaClient's Current Files & Applications page appears, and your new channel appears in the list.

Chapter

9

Using Java Packager

Java Packager is one of the packagers included with Application Packager. You can use Java Packager to package a Java application into a channel, and run a channel packaged with an alternate Java Virtual Machine (JVM).

The following topics are provided:

- Overview (page 254)
- About Java Virtual Machines (page 254)
- Packaging a Java application into a channel (page 255)
- Passing arguments to the JVM (page 258)
- Editing a Java Packager channel (page 259)
- Updating a channel (page 259)

Overview

Normally, channels run using the tuner’s Java Virtual Machine (JVM). Java Packager supports this feature too, but you can also use it to specify an alternate JVM for your Java application. However, the JVM you specify must be supported by Java Packager.

The list of currently supported JVMs appears in the drop-down list on Java Packager’s External JVM Selection page.

- The JVM must already be installed on the endpoint. This tool does not install alternate JVMs.
- The redirection of standard input to a Java application is not supported, so you cannot use Java Packager for console-only Java applications that require standard input from users.

After packaging your channel, you can publish it to a transmitter as a channel. Users can use their tuner to subscribe, start, and update your channel. When your channel is started, the JVM in which it runs should be transparent to the user.

About Java Virtual Machines

Basically, a Java Virtual Machine (JVM) loads, verifies (for validity and security), and converts platform-independent Java bytecodes (produced by a Java compiler) to platform-specific native code for execution. For more information, see the Java specification or your favorite programming reference.

You can use Java Packager to specify the JVM that is used to run a Java application when you package it into a channel. You can run your Java application channel in:

- The same JVM as the tuner
- Another instance of the tuner’s JVM

While theoretically unnecessary, this option might provide additional protection to the tuner if your application misbehaves.

- A JVM produced by another manufacturer

The JVM you specify must be supported by Java Packager and it must be preinstalled on the endpoint. (Java Packager does not install JVMs.)

Packaging a Java application into a channel

Java Packager prompts you for all essential information and then creates a package directory you can publish. The procedure in this topic guides you through all the steps in the packaging process.

Java Packager modifies the source directory you specify. However, nothing that is added by the packager should affect your application. If you prefer, you can package a *copy* of your application's source directory.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► To package a Java application into a channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.)
- 2 From the list of package types in Application Packager, select Java Packager and click Create.
- 3 From the Java Packager welcome page, click Next.
- 4 Specify the directory that contains the Java application.
- 5 Click Next.
- 6 To set the basic publish properties, perform the following steps:
 - a In the Title of the Application box, specify the name you want to appear in the tuner’s channel list area for your application.
 - b In the Main Class box, specify the name of the class that starts your application. Do *not* include the .class extension.
 - c In the Classpath box, specify the name and relative path of any zip files that are needed by the application and should be published as part of your application.

The classpath you specify is applied to the user’s environment each time your application runs. Use a slash (“/”) as the path separator (for all platforms). If you have multiple classpath entries, separate them with a colon.

For example, if your application's source directory is c:\new_app and it uses c:\new_app\extras\zips\xyz.zip, your classpath should be set to extras/zips/xyz.zip (the relative path to the zip file).

- d From the Please select the type of application list, select the application type.
- If your application was designed to run as a channel, select Symphony Marimba Client Automation Application. (An application that is designed to run as a channel must implement the Symphony Marimba Client Automation IApplication interface.)
 - If you are packaging a standard Java application, select static main() class. (A standard Java application does not use Symphony Marimba Client Automation APIs and uses the conventional static main() as its entry point.)

7 Click Next.

8 Select the run mode for your Java application channel:

Run the channel in the tuner—If you want to run your channel in the tuner using the tuner's JVM.

If you select this option, the channel uses the same version and type of JVM as the tuner. For example, if the tuner is using JRE version 1.1.8, selecting this option makes the channel use JRE version 1.1.8 as well.

Run the channel in an external JVM, but keep channel files in the tuner—If your channel is required to run in a particular version of a JVM, but you want to store the channel files in the tuner.

The required JVM must be preinstalled on the user's system. If you select this option, the channel files are stored in the tuner's workspace directory, a tuner-controlled area of the user's file system where channels are saved in a proprietary file format. This is the recommended option when running in an external JVM because it makes the most efficient use of user disk space.

Run the channel in an external JVM, and copy the files into the filesystem—If your channel is required to run in a particular version of a JVM, and you want to copy the channel files to the user's standard file system (that is, outside of the tuner's workspace directory).

The required JVM must be preinstalled on the user's system. If you select this option, you might have duplicate storage of your channel files. Your channel accesses files and classes based on the current working directory. This option also circumvents the use of a class loader, which is required by some applications.

Note: In Application Packager 4.7.2.1 and earlier releases, the working directory is currently set to the current process, not the data directory where the main class files are. Any packaged files under the data directory are not accessible by the main class. In Application Packager 4.7.2.2 and later releases, the current working directory is now set to the path: `<channel_data_directory>\channel`. If creating that directory fails or that path is not a directory, the current working directory is set to: `<channel_data_directory>`.

- 9 Click Next.
- 10 If your Java channel requires particular version of a JVM, select the external JVM for your channel:
 - To run the channel in a separate instance of the same JVM used by the tuner, select Use the JRE included with the tuner.

This option is intended for those who want to use the same JVM as the tuner while minimizing the risk that their application might crash the tuner. For example, you might be running a transmitter on the same tuner as your application and want to minimize the risk that the transmitter might become unavailable to endpoints.
 - To run your application in one of the supported external JVMs, select Use a preinstalled JVM. Then select a JVM and version from the associated drop-down lists.

If the JVM you want to use is not listed, it cannot be used with Java Packager at this time. The JVM you select must already be present on the endpoint; Java Packager does not install external JVMs. This option is available in Windows only.
- 11 Click Next.
- 12 If you are ready to publish the channel that Java Packager has prepared, click Publish. You are not required to publish the channel now. Java Packager has prepared the package directory and you can publish it at any time.

The publishing wizard appears, preloaded and displaying the properties of the channel you just assembled. You can use the channel-signing page to set the security options for your channel. You can also edit other channel properties before publishing the channel to your transmitter. For more information, see “Publishing channels” on page 279.

- 13 Return to the Java Packager window and click Done.

Passing arguments to the JVM

You can configure your channel to pass arguments to the Java Virtual Machine (JVM) at runtime. When the channel is launched from the command line, the arguments are passed by the channel. Specify the arguments you want to pass when you publish your channel. Use the Parameters page to specify the arguments you want to pass. Each JVM has its own set of arguments that it supports. For more information about the arguments that it supports, see the documentation for your JVM.

Note: If your channel is configured to use the same instance of the JVM that the tuner is using, you cannot pass any arguments.

The syntax for typing the arguments on the Parameters page is:

`vm.extraargs=<JVM_arg>`

where:

`<JVM_arg>` is the argument you want to pass. Do not type the angle brackets (`<>`) with your command.

If you have multiple arguments to pass to the JVM, you can append them to the same command, separating each argument with a space. For example,

`vm.extraargs=<JVM_arg1> <JVM_arg2> <JVM_arg3>`

Additional parameters include:

- **vm.redirection**—Set this parameter to false to disable redirection. Disabling redirection is useful in situations where the Java application hangs during execution.
- **vm.userprocess**—Set this parameter to false to run the Java application in the context of the tuner. By default, this parameter is set to true.

Editing a Java Packager channel

You can edit the properties of a Java Packager channel before you publish it.

► To edit the channel

- 1 From the Application Packager main window, click Java Packager.
- 2 Select the channel you want to edit.

Channel directories of the channels you packaged previously are linked to Application Packager. If you delete Application Packager and re-subscribe, the channels you have packaged previously do not appear in the list. However, you can import them to add them to the list of editable channels.

- 3 Click Edit.
- 4 Make the necessary changes on the packager pages.

Updating a channel

If a Java application you have packaged changes after publishing, you can publish an updated channel using the latest version of the Java application. Users receive your updated channel automatically when their tuner checks for updates (at its scheduled time) or when they manually update the channel.

If the changes are simple, such as one of the class files has been modified, you can bypass Java Packager and use Channel Copier to republish the channel directly. Make sure the new class file is in the source directory that Java Packager prepared, and then republish the channel.

To change the JVM specifications or make substantial changes to the way the Java application channel is packaged, you can edit the channel using Java Packager and then republish the channel. For more information, see “Editing a Java Packager channel” on page 259.

Chapter 10 Using Virtual Packager

- Overview (page 262)
- Requirements for packaging virtual applications (page 262)
- Packaging a virtual application into a channel (page 263)
- Publishing a virtual package (page 264)

Overview

The Virtual Packager packages ThinApp virtual applications for deployment to the endpoint.

About virtualized applications

With application virtualization, you can decouple applications from the underlying operating system, allowing more independence and isolation from the operating system.

Requirements for packaging virtual applications

If you package a SoftGrid MSI file, which contains multiple application files in a single MSI file, you must manually provide the following information for each of the bundled applications in the GUI. An equivalent command-line option is not available. For more information, see “To package a Windows Installer or MSI application into a channel” on page 192.

Table 10-1: Virtual file properties

Property	Description	Notes
Application Name	name of the virtual application.	Required field for ThinApp applications
Vendor name	name of application vendor.	
File size (in bytes)	size of the .exe file	Required field for ThinApp applications
Version	version of the .exe file	
Description	description of the virtual application	
Primary Data Container	the Primary Data Container associated with the .exe file	Required field for ThinApp applications.

Note: To distribute Softgrid applications, use the Windows Installer packager and package the application as a .msi file.

Packaging a virtual application into a channel

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► To package a virtual application into a channel

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.)
- 2 From the list of package types in Application Packager, select Virtual Package and click Create.
- 3 From the Virtual Package window, specify the following information:
Virtual exe location — the path to the ThinApp .exe file that you want to package.

Note: To package multiple .exe files in the same package, place all of the .exe files in one directory, select any one .exe file in that directory as the source path, and select the option “Include all exe files found at this location.”

Package location — the path to the directory where the package is stored.

Install location — the installation path for the package on the endpoint.

- 4 To save disk space on the packaging machine, select the option “Package by reference (saves disk space).”
- 5 Click Create.
- 6 To configure the virtual package (optional), select the Configuration => Virtual Package tab on the Package Editor and select the appropriate options:

Register for all users (HKLM), requires admin rights — registers the application for all users on the endpoint and provides a shortcut to the application. By default, the application is registered for all users.

Don't relaunch to get elevated privileges on Vista — allows a standard users to run the application on Vista. The application is not relaunched with elevated privileges.

Don't register in Add/Remove Programs — select if you don't want the application to appear in the Add/Remove Programs list

Note: By default, a shortcut to the virtual application is automatically created on the endpoint.

- 7 To save your configuration settings, from the Package menu, select Save.

Editing a Virtual Packager channel

You can edit a Virtual Packager channel before you publish it.

► To edit the channel

- 1 From the Application Packager main window, click Virtual Packager.
- 2 Select the channel you want to edit.

If you delete Application Packager and re-subscribe, the channels you have packaged previously do not appear in the list. To add them to the list of editable channels, you can import them.

- 3 Click Edit.

Publishing a virtual package

To publish the virtual package, from the Package Editor menu, select the Package => Publish and use the wizard to guide you through the publishing steps. For more information, see “Publishing channels” on page 279.

Troubleshooting virtual packages

Virtual Packager channels have the following behavior if a user manually removes the packaged virtual application using the "Add/Remove Program files" command:

- The adapter status is not changed from Installed to Uninstalled
- Application packager does not remove the installed package files
- The Application Packager Verify/Repair functionality does not re-register the virtual application or re-create the virtual application shortcuts.

Chapter 11 Using Custom Application Packager

You can use Custom Application Packager to package any application, especially ones developed by companies for internal use.

The following topics are provided:

- Overview (page 266)
- Packaging a custom application into a channel (page 266)

Overview

You can use Custom Application Packager to package any application, especially ones developed by companies for internal use.

To use this packager, you need access to the application executable file, as well as knowledge about and access to all files and settings used by the application. This packager is useful when you know exactly which files, registry keys, or services you want to distribute to endpoints. It is also useful when you want to deliver some configuration data, such as a few registry keys, or a text modifier to search and replace some text in a certain file that exists on endpoints.

You can also use Custom Application Packager from the command line. For more information, see “Custom Application Packager command-line options” on page 324.

Packaging a custom application into a channel

You can use Custom Application Packager to package any application into a channel. To use this packager, you need knowledge about and access to all shared files used by the application.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

► **To package a custom application into a channel**

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.)
- 2 From the list of package types in Application Packager, select Custom and click Create.
- 3 In the Package Name box, specify a name for the custom application you are packaging.
- 4 In the Package Location box, specify the path to the directory where you want the packager to assemble the files for publishing.
- 5 Click Edit to begin assembling the files in your channel.

The Package Editor appears.

- 6 In the Package Contents page of the editor, add the files and registry entries you want to include in the channel.

For more information about adding files and registry entries, see “Editing the contents of a channel” on page 70.

- 7 If the application requires any shared files (for example, in Windows, DLL files, ActiveX controls, and so on) to exist on the user’s system, use the Package Editor to specify the required files.

For information about specifying required files, see “Configuring system and channel dependencies” on page 130 and “Configuring file requirements” on page 133.

Any files you specify should already be present on the endpoint. The packager does not install these files for you.

Note: If you make the existence of these files a dependency, installation of the packaged application fails if these files are not present.

- 8 To select the platforms you want this application to support, perform the following steps:

a Click the Configuration tab.

b Click the Platform tab.

c Select the platforms you want this application to support.

- 9 Use the Package Editor to configure other settings for the channel. For more information, see the following sections:

“Setting installation modes” on page 106

“Setting policies for installation, update, uninstallation, verification, and repair” on page 121

“Setting channel startup options” on page 104

“Configuring file requirements” on page 133

“Configuring system and channel dependencies” on page 130

“Using environment variables” on page 127

“Scheduling verification and repair for channels” on page 167

“Using macros” on page 95

“Customizing a channel by using scripts and classes” on page 141

10 To save the channel, choose Package > Save.

11 To publish the channel, choose Package > Publish.

This starts the publishing wizard, which guides you through the process of publishing your channel. For more information, see “Publishing channels” on page 279.

Chapter 12 Using XML template files

You can use XML template files to save, edit, and apply default settings and configurations for Application Packager.

The following topics are provided:

- Overview (page 270)
- Saving Package Editor settings to an XML template file (page 270)
- Configuring Application Packager to use an XML template file for default settings (page 271)
- Using XML template files with Packager for Shrinkwrap Windows Applications (page 272)
- Applying an XML template file to existing channels (page 276)

For information about the tags you can use in XML template files, see “XML syntax” on page 369.

Overview

You can use XML template files to save, edit, and apply default settings and configurations for Application Packager.

These settings include installation policies, macros, scripts, installation modes, and other settings that can apply to multiple applications.

Saving Package Editor settings to an XML template file

After you have packaged an application and configured it using the Package Editor, you might want to save your settings so that you can apply them to other packaged applications. You can do this by saving the settings to an XML template file.

After you have saved your settings to an XML template file, you can use that file to apply the settings to an existing channel. You can also apply those settings to future channels that you create by setting the XML template file as the default template for Application Packager.

The following items are not saved to the XML template files:

- text modifier groups and text modifiers
- MSI patches
- MSI transforms

► To save your Package Editor settings to an XML template file

- 1 If you do not already have Application Packager running, start Application Packager.
- 2 Open a channel for editing, as described in “Opening a channel for editing” on page 69.
- 3 In the Package Editor, choose Package > Save as XML Template File.
- 4 Specify where you want to save the XML template file and the name you want to give it.

Configuring Application Packager to use an XML template file for default settings

When you use Application Packager to package an application, the settings specified in the default XML template file are used. This file is located in Application Packager's channel directory in the tuner's workspace directory. (In Windows, the default path to the file is

c:\winnt\.marimba\Marimba\ch.*X*\data\defaults.xml, where *X* is the appropriate channel number. In UNIX, the default is \$MARIMBAROOT/.marimba/Marimba/ch.*X*/data/defaults.xml. If \$MARIMBAROOT is not defined, it defaults to the user's login directory.)

You might want to create your own XML template file and use it as the default for Application Packager. If there are particular settings (for example, installation and uninstallation policies) you want to have for every application you package, you can use an XML template file that has those settings as the default.

For an example of using an XML template file to apply settings to all new channels, see “Example: Configuring the default settings for newly packaged channels” on page 275.

► To configure Application Packager to use an XML template file as the default

- 1 If Application Packager is not already running, start Application Packager.
- 2 On the main Application Packager window, choose File > Set XML Template File.
- 3 Specify the XML template file you want to use as the default.

The file you specified is now used every time you package an application.

Later, if you decide to use again the default XML template file that originally shipped with Application Packager, you can do so by following these instructions:

► To configure Application Packager to use the default XML template file that originally shipped with it

- 1 If Application Packager is not already running, start Application Packager.
- 2 On the main Application Packager window, choose File > Reset XML Template File.

Using XML template files with Packager for Shrinkwrap Windows Applications

You can use XML template files to customize Packager for Shrinkwrap Windows Applications with the configuration and filters that you want to use. The XML template files can contain the following information:

- The directories you want to include in the snapshot
- The registry keys you want to include in the snapshot
- The metabase keys you want to include in the snapshot
- Whether you want to capture the removal of files, directories, registry entries, and metabase entries when taking the snapshot
- The filters for ignoring files, directories, registry entries, and metabase entries when taking the snapshot

Loading XML template files for configuration and filters

When you use Packager for Shrinkwrap Windows Applications to create channels, you can exclude certain Windows objects, such as directories, files, registry keys, and metabase keys, by using filters. You can use filters to ignore these objects when taking snapshots to create a channel.

There are two types of filters:

- **Application Packager default filters**—These filters are provided by Symphony Marimba Client Automation as the default.
- **User-defined filters**—You can create these filters and configure Application Packager to use them when you create a channel.

Both types of filters can be enabled or disabled when you take snapshots.

► To load an XML template file that contains filters

- 1 Start Application Packager, as described in “Starting Application Packager” on page 62.
- 2 From the list of package types in Application Packager, select Shrinkwrap Windows Applications and click Create.
- 3 To proceed to the Preinstall Snapshot page, click Next twice.
- 4 Select Generate a new preinstall snapshot now.

- 5 Click Next.
- 6 On the Generate Preinstall Snapshot page, click Load User Configuration and User-Defined Filters.
- 7 In the dialog box that appears, select the XML template file you want to use.

The file contains the configuration and user-defined filters you want Packager for Shrinkwrap Windows Applications to use when taking snapshots.

- 8 On the Generate Preinstall Snapshot page, click Load Default Filters to load the default filters from an XML template file.
- 9 In the dialog box that appears, select the XML template file you want to use. The file contains the default filters you want Packager for Shrinkwrap Windows Applications to use when taking snapshots.

For more information about the tags to use in the XML template files, see “Snapshot tags” on page 428website.

Saving and setting configuration and filters using XML template files

On the Snapshot Customization Options page, you have the following XML template file options:

- **Save User Configuration and User-Defined Filters**—To save the current configuration and user-defined filters used by Packager for Shrinkwrap Windows Applications to an XML template file you can reuse. This includes any configurations you have set on the File System Selector page, the Registry Selector page, and the Metabase Selector page. Later, you can reuse those configurations and filters when packaging other applications.
- **Save Default Filters**—To save the current default filters used by Packager for Shrinkwrap Windows Applications to an XML template file you can reuse. This includes any filters on the File System Selector (also Registry and Metabase) - Default Ignore page. Later, you can set Packager for Shrinkwrap Windows Applications to use this XML template file as the source for default filters.

- **Set Default Filters Template**—To select an XML template file and configure Packager for Shrinkwrap Windows Applications to use the default filters in that XML template file when taking snapshots. The filters specified in the XML template file appear in the File System Selector (also Registry and Metabase) - Default Ignore page. The XML template file you select is used by the packager everytime you take snapshots until you unset the XML template file.
- **Unset Default Filters Template**—To unset the XML template file you (or another user) previously configured Packager for Shrinkwrap Windows Applications to use as the source for the default filters. When you click this button, you can go back and use the default filters that originally shipped with Application Packager.

► Example: XML template files to use with snapshots

The following XML template file contains examples of snapshot configuration, such as:

- The directories you want to include in the snapshot
- The registry keys you want to include in the snapshot
- The metabase keys you want to include in the snapshot
- whether you want to capture the removal of files, directories, registry entries, and metabase entries when taking the snapshot

```
<SNAPSHOT ini="true">
  <FILESYSTEM includedeletes="false" include="true">
    <DIRECTORY path="C:\"/>
    <DIRECTORY path="D:\\"/>
  </FILESYSTEM>
  <REGISTRY includedeletes="false" include="true">
    <REGKEY key="\HKEY_CURRENT_CONFIG"/>
    <REGKEY key="\HKEY_CURRENT_USER"/>
    <REGKEY key="\HKEY_LOCAL_MACHINE"/>
    <REGKEY key="\HKEY_USERS"/>
  </REGISTRY>
  <METABASE includedeletes="false" include="false">
    <METKEY key="\LM"/>
    <METKEY key="\SCHEMA"/>
  </METABASE>
</SNAPSHOT>
```

The following XML template file contains examples of filters you can use to ignore directories, files, registry entries, and metabase entries:

```
<SNAPSHOT>
  <FILESYSTEM>
```

```

<FILTER enabled="true" verb="is" adverb="$SYS.WINDOWS"
noun="parent" description="Windows NT backup directories"
type="directory">
    <FILTER enabled="true" verb="begins_with" adverb="$Nt"
noun="name" description="" type="directory"/>
</FILTER>
    <FILTER enabled="true" verb="is" adverb="temp" noun="name"
description="TEMP directory" type="directory"/>
        <FILTER enabled="true" verb="is" adverb="tmp" noun="name"
description="TMP directory" type="directory"/>
</FILESYSTEM>
<REGISTRY>
    <FILTER enabled="true" verb="is"
adverb="\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
\Policies\System" noun="path" description="System Boot&logon
Message" type="key"/>
        <FILTER enabled="true" verb="is"
adverb="\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\
Explorer\SessionInfo" noun="path" description="Explorer's
Session Information" type="key"/>
    <FILTER enabled="true" verb="is" adverb="\HKEY_LOCAL_MACHINE\SYSTEM"
noun="parent" description="\HKLM\System except for CurrentControlSet"
type="key">
        <FILTER enabled="true" verb="isn't"
adverb="CurrentControlSet" noun="name" description=""
type="directory"/>
        </FILTER>
        <FILTER enabled="true" verb="is"
adverb="\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet" noun="parent"
description="\HKLM\System\CurrentControlSet except for Services"
type="key">
        <FILTER enabled="true" verb="isn't" adverb="Services"
noun="name" description="" type="directory"/>
        </FILTER>
    </REGISTRY>
    <METABASE>
    </METABASE>
</SNAPSHOT>

```

► **Example: Configuring the default settings for newly packaged channels**

If you create many channels for your enterprise and you want them to be consistent, there might be settings you want to apply to all the channels you create. This section describes how you can use an XML template file to apply settings to all newly packaged channels. First, you save the settings to an XML template file. Then you configure Application Packager to use that XML template file as the default.

The following example shows how you can configure Application Packager so that all newly packaged channels have a global install policy of “Always install object” and have the parameter `delayfiledownload` set to true.

► To configure the default settings for new channels

- 1 Using Custom Application Packager, create a channel that does not contain any objects.
- 2 In the Package Editor, go to the Configuration – Policies – Install tab.
- 3 Select the Always install object option.
- 4 Click the Configuration – Installer tab.
- 5 Make sure that the Delay download of files until pre scripts have run check box is selected. (It is selected by default.)
- 6 Choose Package Editor > Package > Save as XML Template File.
- 7 In the dialog box that appears, specify the name and path for the XML template file.
- 8 Go to the main Application Packager window and choose File > Set XML Template File.
- 9 In the dialog box that appears, specify the name and path where you saved your XML template file in step 6.

The XML template file you specified is applied to any subsequent channels you create using Application Packager. More specifically, the “Always install object” policy and the `delayfiledownload` parameter are set for all newly created channels after you have set the default XML template file.

Applying an XML template file to existing channels

If you have existing channels that were packaged using Application Packager, you might want to apply certain settings to them without having to edit each one manually. You can do this by saving those settings in an XML template file, and then applying that file to the existing channels. For example, if you have a set of macros you want to add to many channels, you can add those macros to an XML template file and then apply that file to your channels.

Before you can apply an XML template file to a channel, you need the following information:

- The full path of the package directory.

- The full paths of the XML template files.

► To apply an XML template file to a channel

Use the `runchannel` program and the `-applytemplate` command-line option as follows:

```
runchannel <App_Packager_URL> -applytemplate <channel_directory>
<paths_of_XML_template_files>
```

where:

`<channel_directory>`

specifies the full path of the directory where you want to place the contents of the new channel you are creating.

`<paths_of_XML_template_files>`

specifies the paths of the XML template files you want to apply to the channel. To use more than one XML template file, separate the paths with semicolons (;).

For example:

```
runchannel http://acme.com:80/Marimba/ApplicationPackager
-applytemplate "C:\Channels\MyCustomChannel"
"C:\Channels\Templates\shrinkwrap.xml;
C:\Channels\Templates\policies.xml;
C:\Channels\Templates\macros.xml"
```


Chapter 13 Publishing channels

This section describes how to publish channels from Application Packager using the publishing wizard.

The following topics are provided:

- Overview (page 280)
- Publishing channels to a transmitter (page 280)
- Creating a backup copy of a published channel (page 283)

Overview

After packaging a channel using Application Packager, publish the channel to make it available to users. When you click the Publish button from any of the packagers in Application Packager, you start the channel publishing wizard. The channel publishing wizard guides you through the process of publishing a channel.

Publishing channels to a transmitter

To publish a channel, Application Packager interacts with Channel Copier, which appears to you as the channel publishing wizard. Thus, any channel publishing operations you perform using the channel publishing wizard appear in the main window of Channel Copier. You can easily republish a channel by starting Channel Copier, selecting a publish operation, and clicking Copy.

► To publish a channel

- 1 After creating the channel using Application Packager, click Publish. This starts the channel publishing wizard, which guides you through the process of publishing a channel.
- 2 Select a signing option for the channel you are publishing:
 - Sign using this certificate**—To sign a channel using a selected certificate. You can use the drop-down list to select a certificate.
 - Do not sign this channel**—If you do not want to sign the channel.
- 3 Click Next.
- 4 If needed, specify values for the basic properties of the channel:
 - Title**—Name of the channel displayed in the tuner. It can be an ASCII string, but you should be brief—the tuner might truncate the title if it is too long.
 - Description**—Any text description you want to include. This can include contact or support information.

Category—Name of the tuner category where you want the channel to appear. If the category does not exist in the user's tuner, the tuner creates the category when the user subscribes to the channel. If you have a category name you want shared by multiple channels, make sure you type the name exactly. Categories are case-sensitive.

Author—Usually the name of the person or group who developed the channel.

Copyright—Any copyright string you want to include, but it should be brief (fewer than 50 characters).

- 5 To set additional properties of the channel, select the Set up Advanced properties check box.
- 6 Click Next.
- 7 If you selected the Set up Advanced properties check box, use this page to edit the values of channel properties.

Property column—Displays the names of the channel properties.

Value column—Displays the values of the channel properties. You can edit and type property values in the Value column. To save your changes, be sure to press ENTER after editing each property value.

For more information about the individual channel properties, see Channel Copier Help (click Help from within Channel Copier).

- 8 Click Next.

- 9 Set the schedule for updating the channel by selecting the appropriate day and frequency:

Table 13-1: Days of schedule

Days	Description
Never	The tuner never updates the channel; you must manually update the channel when necessary.
Daily	The channel can be updated every day, only on weekdays (Monday through Friday), or every 1 to 100 days, depending on the number you set.
Weekly	The channel gets updated every 1 to 100 weeks, depending on the number you set. During the week of the update, the update occurs only on the days you check. If no days are checked, the update occurs on Monday.
Monthly	The channel can be updated every 1 to 100 months on the day you set. If you select day 30 or 31 and the month when the update occurs does not have that day (for example, February), the update occurs on the first day of the next month.

Table 13-2: Frequency of schedule

Frequency	Description
Update at	Select to choose one time of the day when updates occur.
Update every	Select to specify multiple times of the day when updates occur. You can set updates to happen every 1 to 100 minutes or hours during the day, and you can limit the updates to a single range of hours during the day. For example, you can have the update occur every hour between 9 a.m. and 5 p.m., or every 30 minutes from 5 p.m. to 9 a.m.

- 10 Click Next.

- 11 To select the destination location for the channel, perform the following steps:



- a Click the Transmitter icon to specify a transmitter as the destination type.
- b In the Select Destination drop-down list, specify the destination transmitter.

The Destination box contains the destination URL you selected.

- 12 Click Publish to publish the channel to the destination location you selected.
The wizard shows you the progress of publishing the channel and informs you when the channel is successfully published.
- 13 Click Close.

Creating a backup copy of a published channel

When you publish a channel update to the transmitter, the update overwrites the previous version of the channel on the transmitter. Before you package and publish a new version of the channel to the same channel URL, it is recommended that you create a backup copy of the previous version of a channel. You can use Channel Copier to create a backup copy of the channel on the transmitter.

► To create a backup copy of the previous version of a channel

- 1 On the packaging machine, double-click to start Channel Copier from the tuner.

The Copy window appears.

- 2 To create a new copy operation, click New.
- 3 For the source, perform the following steps:
 - a Make sure that Transmitter is selected.
 - b For Select source, type the URL of the channel you previously published, and press ENTER. For example:

`http://trans.acme.com:5282/MyApplication`

Note: If you used a different port number than 5282 for the transmitter, use that port number here.

Tip: If you do not know the exact URL of the channel, you can type the transmitter URL and press ENTER. When the channels on the transmitter appear, click the channel for which you want to create a backup copy.

- 4 For the destination, perform the following steps:
 - a Make sure that Transmitter is selected.



- b For Select destination, type the same URL you typed for the source.
 - c Change the URL of the channel so that you know it is a backup copy, and press ENTER. For example, if the URL of the channel is originally `http://trans.acme.com:5282/MyApplication`, you can change it to `http://trans.acme.com:5282/MyApplicationBackup`.

5 Click Add/Close.

The Copy window appears, and the copy operation you created is selected.

6 Click Copy.

Channel Copier creates a backup copy of the channel on your transmitter.

7 Choose File > Quit.

You now have a backup copy of the channel. When you publish an update of the channel to the transmitter, the update uses the original channel name, for example, `MyApplication`. When you deploy the update, the original channel name is preserved and byte-level differencing can be used when downloading the channel to the target servers.

To go back and deploy the previous version of the channel to your target servers, you can use the backup copy you created.

Chapter 14 Agentless Deployment

This section describes how to deploy channels from Application Packager using the Agentless Deployment.

The following topics are provided:

- Overview (page 286)
- Limitations (page 287)
- Creating a generic package in Application Packager (page 287)
- Application Packager Command-Line Interface (CLI) Commands for creating a generic package for agentless deployment (page 289)
- Using Content Replicator for creating a package for agentless deployment (page 290)
- Command for repackaging (page 289)
- Logging (page 291)
- Agentless Discovery (page 292)
- Enabling Agentless Discovery in generic package using Application Packager (page 292)
- Enabling Agentless Discovery in generic package using Application Packager CLI (page 292)
- Enabling Agentless Discovery in generic package using Content Replicator CLI (page 293)

Overview

You can also deploy packages and run inventory scan on target machines that do not have the BBCA Tuner installed. The capabilities of the existing packaging tools now include support for the creation of a new type of generic package that can be installed on any agentless machine. You can use this type of generic package to install the contents and run inventory scan on the target machines.

The advantage of this feature is that you need not install and maintain the tuner on the target machine, but you can distribute the contents to these targets and perform inventory scan. This feature can be used for the following purposes:

- Control and centralize the management of the existing content distribution system

In some organizations common installers may be stored in a file share location, however this location can be remote for all the computers in an organization to connect and get the required installers or files. In this scenario, these installers can be packaged and published on the server instead of having them on the file share. From the server, the package deployment can be automated through a GPO script that will take care of deploying the package on target machine at specified schedule.

- Self service portal

Employees of the organization can choose to self -install the generic packages on their computers.

The Application Packager, Transmitter, Channel Copier and Content Replicator support agentless deployment of generic packages.

The workflow for performing an agentless deployment of a generic package is as follows:

- 1 Use either the Application Packager or Content Replicator to create the generic package.
- 2 Publish the package on your Transmitter.

Note: If packages are created through Application Packager, ensure that the Channel Copier is updated to 8.3.00 version and is available at the same Transmitter location as Application Packager.

- 3 From the target computer which does not have the BBCA Tuner installed, open a browser and browse through the Transmitter to locate the required package, and install the package.

Limitations

- The agentless deployment feature is not supported on non-Windows platform.
- UI mode configurations like silent, semi-interactive, and fully interactive are not supported for generic installer package installation or uninstallation.
- Configuration of pre or post-install scripts in Application Packager is not supported for generic installer packages.
- Agentless Packaging is supported only for self-installing rep package types. It is not applicable for rep data packages.
- In the Package Editor Window of Application Packager, only Platform and Installer settings under Configuration tab are applicable for Generic Installer packages.
- Agentless Discovery supports scanning of limited parameters, which include
 - Host name
 - Scan Time
 - Installed Memory
 - Machine Domain
 - Package Name
 - Package install location

Creating a generic package in Application Packager

To create a package for agentless deployment, you must create a generic package. You can use the Agentless Installer tab of Application Packager to create a generic package.

Note: You cannot use any other package type other than Generic Installer package for agentless deployment.

► **To create a generic package:**

- 1 Start Application Packager.
- 2 Click Agentless Installer tab.
- 3 The Agentless Installer tab displays.
- 4 Click Create.
- 5 In the Package Name text box, type the name of the package.
- 6 In the Package Location text box, type the URL of the location where the files are located. You can also click on the Browse button to locate and select the package.
- 7 If you want to locate the package by reference, then select the Package by reference check box. If you select this option, it saves disk space.
- 8 In the Source Path text box, type the location of the source directory that needs to be packaged. It can contain executables, binaries and dependency files needed to run the executable.

You can also click on the Browse button to locate and select the source folder.

- 9 If you want to launch the executable after deployment, then select the Launch executable check box.
- 10 To specify the path of the executable, in the Executable Path text box, type the path.

You can also click on the Browse button to locate and select the folder where the executable is located.

- 11 In the Install Path text box, type the location of the folder which will be created during package installation on the target machine.
- 12 Click Create.

Agentless Installer creates the generic package and opens the generic package in the Package Editor for further configurations. In Application Packager, all the package configuration options on the Edit Configuration page are not applicable to generic packages. After the package is created, you can use Channel Copier to publish the package.

Application Packager Command-Line Interface (CLI)

Commands for creating a generic package for agentless deployment

You can also use the following Application packager CLI commands to create a generic package for agentless deployment:

```
runchannel <application packager channel url> -agentlesspackage -  
pkgName <package name> -pkgDir <package directory> -source <source  
content location> -target <install path> [-byreference <true|false>]  
[-launchExec <executable location>] [-defaults <path to XML template  
file>]
```

where:

-agentlesspackage

Specifies the type of package to be created for agentless deployment.

-pkgName

Specifies the name of the package.

-pkgDir

Specifies the directory in which the package has to be created.

-source

Specifies the location of the content which has to be packaged.

-target

Specifies the location where the package is installed.

You can also specify the following optional parameters:

[-byreference <true|false>]

Specifies whether you want to use referencing to locate the content for packaging.

[-launchExec <executable location>]

Specifies the location of the executable for launching.

[-defaults <path to XML template file>]

Specifies the XML template file you want to use when creating the new channel.

Command for repackaging

```
-agentlesspackage -repackage -pkgDir <package directory>
```

Where:

-repackage

Specifies that the package has to be repackaged

-pkgDir

Specifies the location of the package which has to be repackaged.

You can use this command for repackaging.

Note: If you want to use Application Packager UI to edit or publish a generic package which you have created using Application Packager CLI, then you must use the Import option of Application Packager to import the generic package and then edit or publish it.

Note: To download the update of a generic package, select MSI wrapper for this package from Add or Remove Programs and click on **Repair**. This will reinstall the package.

Using Content Replicator for creating a package for agentless deployment

The Content Replicator CLI now supports a new command line parameter to create generic package for agentless deployment.

You can use the following CLI parameter:

■ -agentless

For example:

```
runchannel <rep_URL> publish -package -agentless -dir <source directory> -installDir <target installation directory> -url <publish URL>
```

Note: You can use Content Replicator only to create self-subscribing packages for agentless deployment.

Agentless Deployment

The generic package created through any of the methods mentioned above can be deployed on the target machine using a browser. The Steps mentioned below can be followed to trigger the deployment

- 1 Open any browser.
- 2 In the address bar of the browser, type the URL of the Transmitter where the generic package is published.
- 3 Click the package which you want to deploy.

The Transmitter sends the MSI file which should be downloaded on the target machine to trigger the package installation. You can either choose to run this MSI file or save it to a local directory on the computer to install it later.

Logging

Generic Package Deployment generates a log file at the following location:

"%ALLUSERSPROFILE%\MarimbaGenericInstaller.txt

The log file contains information related to package deployment and scan progress, and any problems associated during deployment.

By default, the log file contains limited logging information. To get additional log messages, set the following properties:

- With Application packager, set the **debug=true** property in the parameters.txt file of packages before publishing the package to transmitter.
- With Content Replicator, specify the **-debug** argument during package creation.

Agentless Discovery

Agentless Discovery is a feature where the inventory scan is performed after the package is installed at the target machine that does not have the Tuner installed. After the scan is completed, the target machine sends the scan report to the inventory plugin (the plugin URL is provided while creating the generic package) and the plugin inserts the scan report in the database. In Report Center, you can fetch the list of these target machines where agentless deployment is performed. Report center categorizes these target machines as Agentless. The administrator can use this feature to track the deployment of agentless packages.

Enabling Agentless Discovery in generic package using Application Packager

While creating a generic package through Agentless Installer package option, select the **Scan Machine after successful deployment** check box to enable scan of the target machines after package installation. Ensure to specify the plugin URL as specified in Step 10 of “Creating a generic package in Application Packager”.

Enabling Agentless Discovery in generic package using Application Packager CLI

You can use the `-scan` CLI parameter to enable Agentless Discovery while creating a generic package for agentless deployment.

For example:

```
runchannel <application packager channel url> -agentlesspackage -  
pkgName <package name> -pkgDir <package directory> -source <source  
content location> -target <install path> [-byreference <true|false>]  
[-launchExec <executable location>] [-defaults <path to XML template  
file>] [-scan <inventory plugin URL>]
```

where:

```
-scan <inventory plugin URL>
```

Specifies that scanning has to be enabled for agentless deployment. Specify the URL of inventory plugin that should be used by the target machine to send the scan report.

Enabling Agentless Discovery in generic package using Content Replicator CLI

A new parameter has been added in Content Replicator to support discovering agentless machines.

```
runchannel <rep_URL> publish -package -agentless -dir <source directory> -installDir <target installation directory> -url <publish URL> -scan <inventory plugin URL>
```

where:

-agentless

Specifies the type of package to be created for agentless deployment.

-package

specifies that the type of package is self installing package.

-dir

Specifies the location of the content which has to be packaged.

-installDir

Specifies the location where the package has to be installed.

-url

Location on Transmitter where the package needs to be published

-scan

Specifies that scanning has to be enabled for agentless deployment. Specify the URL of inventory plugin that should be used by the target machine to send the scan report.

Chapter 15 Troubleshooting

This section describes some issues you might encounter when using Application Packager.

The following topics are provided:

- Overview (page 296)
- Issues with packages created using Application Packager (page 296)
- Looking at the log files (page 301)
- Turning on the debugging feature (page 302)

Overview

This section describes some issues you might encounter when using Application Packager. Most issues are specific to the application you have packaged, but this section discusses some common issues. In most cases, it is useful to start by looking at the history logs for the tuner and the package at the endpoint where the package is being installed.

Issues with packages created using Application Packager

Problem

I cannot find the log files. Where are they?

Possible solution

Log files for the packages are stored in the tuner's workspace directory. The tuner's workspace directory usually has the following paths:

- In Windows, `C:\Program Files\Marimba\Tuner\.marimba\<keyword>`
For the console server (the tuner you install during the initial setup and deployment), the default installation directory in Windows is `C:\Program Files\Marimba\Tuner\.marimba\Marimba`.
- In UNIX, `/opt/Marimba/Tuner/.marimba/<keyword>`.
For the console server (the tuner you install during the initial setup and deployment), the default installation directory in UNIX is `/usr/local/Marimba/Tuner/.marimba/ws3`.

You or the person who installed the tuner might have changed the location of this directory when you were creating the installer during setup and deployment.

Problem

A package is stuck in the install-pending state.

Possible solutions

Investigate the following possible causes:

- **The transmitter hosting the channel is not trusted**—Verify that the transmitter host name, subnet, or IP address that are listed for the following properties are correct:

`marimba.security.trusted.transmitters`

or

`marimba.security.identity.transmitters`

or both.

- The channel parameter `userobjs` is set to true, and the user is not logged on to the machine—If the `userobjs` parameter is set in the package directory or in the `channel.txt` file of the package in the tuner's workspace, verify that a user is logged on to the machine to install the package.

Note: When you create a package with the property `userobjs=true`, after the Subscription Service pushes the channel to the endpoint, if the user at the endpoint is not logged on, the tuner on the endpoint attempts to install the channel every three minutes.

Problem

A package is in the failed state.

Possible solutions

Investigate the following possible causes:

- A script included in the package failed to install.
- One or more files and registry entries failed to install.
- A dependency specified for the package is not met by the endpoint.
- The user at the endpoint does not have sufficient permissions to install the package.
- Installation of the package was interrupted by the user manually or by another process on the endpoint.

To troubleshoot, perform the following steps:

- Look at the package's history logs to determine the possible cause of failure.
- Try installing the application without packaging it using Application Packager, and confirm that it installs successfully.

Problem

Preinstall and postinstall scripts fail to run on the endpoint.

Possible solutions

Investigate the following possible causes:

- **The script did not get included with the package**—Verify that the script is in the `signed/scripts` directory in the package directory before publishing the package to the transmitter.
- **The command specified in the script does not match the absolute path on the endpoint**—Run the script on the endpoint to confirm that the script command is valid. You might also want to select the Run this script in the user's context option before publishing and re-installing the package.
- **The script returns a return code other than 0**—Check the history logs for the package on the endpoint for the return code.
- **A dependency specified for the package is not met by the endpoint**—Find out which dependencies are required for the package, and confirm that the endpoint meets those dependencies.

Problem

The package does not update on the endpoint.

Possible solutions

Investigate the following possible causes:

- **The tuner or package's schedule is set to never update**—Check the tuner property `marimba.schedule.filter` on the endpoint.
- **The transmitter where the package is published is offline**—Try to browse the transmitter to make sure it is available.
- **The tuner's workspace directory is corrupted**—Look at the tuner's history logs to find out if there were any tuner-related errors. You might also want to check the tuner's proxy settings.

Problem

Files included in the package are not installed or removed as needed on the endpoint.

Solution

Investigate the following possible causes:

- **Files have the incorrect install, update, or uninstall policy**—In Package Editor, check that the install, update, and uninstall policies are set correctly.

- The user at the endpoint does not have sufficient permissions to install or uninstall files—Verify that the user has sufficient permissions for the files. Also, check the history logs for the package, and search for the words “ignored” and “failed.” You might also want to find out if there are any other applications that might share or have a lock on the files.
- The package itself might be corrupted—Check the tuner’s history logs to see if there are any tuner storage errors.

Problem (Microsoft Windows Installer or MSI only)

A package created using Windows Installer Packager fails to install.

Possible solutions

When troubleshooting Microsoft Windows Installer or MSI packages, verify that the application installs successfully without being packaged using Application Packager. You can do this by running the installer using the `msiexec` program from the command line. For example:

```
C:\msiexec /i C:\ExampleApplication.msi
```

For more information about the `msiexec` program, see the following page on Microsoft’s website: <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winxppro/proddocs/msiexec.asp>

You can also investigate the following possible causes:

- The logged-in user does not have permissions to install objects in the user’s profile or the HKEY_CURRENT_USER section of the registry—Open the package in Package Editor, and go to the Windows Installer – Installation tab. In the text box below “Enter command line property settings”, type `ALLUSERS=2`. Then save, republish, and reinstall the package.

For more information about the MSI properties, see Microsoft Windows Installer Help or Microsoft’s website (at the time of this writing, the URL for the property reference is http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/property_reference.asp).

- The required version of Microsoft Windows Installer (`msiexec.exe`) is not available on the endpoint—To find out what version of the Windows Installer you currently have, run `msiexec.exe` from the command line at the endpoint. (It is usually located under the Windows system directory.) To upgrade, the Windows Installer is available as a redistributable file (from Microsoft’s website) for various Windows platforms.
- The package might contain corrupt .msi, .msp, or .mst files—Try the following solutions:
 - Try running the .msi, .msp, or .mst files on a machine similar to the target endpoint using the `msiexec` program from the command line. For example:

```
msiexec /I C:\package.msi <options>
```

For more information about the MSI command-line options, see Microsoft Windows Installer Help or [Microsoft’s website](#) (at the time of this writing, the URL for the reference is http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/command_line_options.asp).

- Open the package in Package Editor and configure it for full GUI installation (see “Setting installation modes” on page 106) and verbose logging (see “Setting Windows Installer logging” on page 212).

Logging codes for Application Packager are available in the logging codes chapter in the *Symphony Marimba Client Automation Reference Guide*, available on the Marimba Channel Store.

Error messages for the Windows Installer are described on [Microsoft’s website](#) (at the time of this writing, the URL for the reference is http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/windows_installer_error_messages.asp). Error codes for the Windows Installer are also available (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/error_codes.asp).

Problem

A Java package fails to install.

Solution

Investigate the following possible causes:

- **Java classes did not get compiled correctly**—Make sure the Java classes are compiled correctly and include all the required libraries (JAR, ZIP, TAR, and other files).
- **The Java application has permission issues**—For the external virtual machine (VM), make sure that the user at the endpoint has access and permission to launch and execute `java.exe`. Set capabilities to “all” for the Java package’s `properties.txt` file before publishing
- **The PATH or CLASSPATH settings are not valid**—Confirm that the required CLASSPATH is set.

Looking at the log files

These types of logs are available with Application Packager to aid in your troubleshooting efforts:

- **Application Packager logs**—When you are using the graphical user interface or the command-line interface, and an error occurs, the error is recorded in Application Packager’s channel history log.
- **Endpoint channels’ history logs**—Every Symphony Marimba Client Automation channel has a history log that records the events and errors that occur when using the channel. These events include starting, stopping, and updating the channel, as well as installing and repairing. For information about where these files are stored and how to read them, see “Endpoint logs” on page 302.

Application Packager’s history logs

- **What information is recorded in the history logs**—Like all other Symphony Marimba Client Automation channels, the Application Packager channel has a history log that records the events and errors that occur when you use the application. In the case of Application Packager, these events include starting, stopping, and updating the channel, as well as installing and repairing.
- **Location of the history logs**—Application Packager history logs are written to the history log in the Application Packager channel directory:

`<tuner_workspace_dir>\Ch.X\history-1.log`

where:

<tuner_workspace_dir> is the tuner's workspace directory, and *Ch.X* specifies the channel number for the Application Packager channel. This log is rolled according to size, when it reaches 32 kilobytes, and one version besides the current version is retained. (In Windows, the default location of the tuner's workspace is *c:\Program Files\Marimba\Tuner\marimba\Marimba*. In UNIX, the default is */usr/local/Marimba/Tuner/.marimba/ws3/*.) To determine the channel number for Application Packager, open the *map.txt* file in the workspace. For information about the other types of logs you can use, see “Looking at the log files” on page 301.

Endpoint logs

For the packaged channels or packages on each endpoint, logs are created that record actions that take place on the endpoint. Each package's history log is kept in the following location:

<tuner_workspace_dir>\Ch.X\history-1.log

where:

<tuner_workspace_dir> is the tuner's workspace directory, and *Ch.X* specifies the channel number for the package. Depending on the channel, a data directory might be included in this *Ch.X* channel directory.

(In Windows, the default location of the tuner's workspace is *c:\Program Files\Marimba\Tuner\marimba\<keyword>*. In UNIX, the default is */opt/Marimba/Tuner/.marimba/ws3/*. In these paths, *<keyword>* is the endpoint's keyword, which was assigned during installation. The default keyword is the profile name.) To determine the channel number for the service channel, open the *map.txt* file in the workspace.

Turning on the debugging feature

When to use debugging

You might want to turn on debugging to find out what caused one of the following situations:

- A package fails to install or update.
- You suspect that some items included in a package, such as files, directories, or registry entries, failed to install or update.

- Installation scripts that are included with a package fail to run.

- You are having problems related to repairing packages.

Before you turn on debugging, contact Customer Support for guidance.

Turning on debugging

To use the debugging feature on an endpoint, add a “debug flag.” You can set this for an individual package or for all the packages you install on a tuner.

Note: The debugging level you set for all the packages on the tuner overrides the one you set for an individual package.

- In Windows and UNIX, you can set the following property in the parameters.txt file in the package directory:

`debug.level=<integer from 0 to 10>`

For example:

`debug.level=2`

Setting the level higher increases the detail of debugging information that appears in the log files.

- In Windows and UNIX, you can add this flag by starting the tuner at the command line with the -java option. This is an example of the syntax for UNIX:

`./tuner -java -DDEBUGFLAGS=<flag>=<level>,<flag>=<level> [<other_tuner_args>]`

Note: If you start the tuner with other tuner arguments in addition to -java -DDEBUGFLAGS, always place the -java -DDEBUGFLAGS argument first, before the other arguments.

Using this -java option at the command line is the preferred way to turn on debugging because debugging is automatically turned off the next time you start the tuner without the debug flag. That is, you do not need to remember to turn it off.

- In Windows and UNIX, you can add this flag by adding a tuner property using Tuner Administrator. You can also add the tuner property directly to the prefs.txt file in the tuner's workspace. In Windows, for example, the default location of the tuner's workspace is

c:\Program Files\Marimba\Tuner\.marimba\<keyword>\,

where:

<keyword> is the endpoint's keyword, which was assigned during installation. (The default keyword is the profile name.) That is, turn on debugging by adding the following line to prefs.txt and then restarting the tuner:

```
marimba.launch.javaArgs=-DDEBUGFLAGS=<flag>=<level>,<flag>=<level>
```

Specifying flags

For <flag> and <level>, use the following values:

For <flag> use SOFTDIST. You set this debugging flag on the endpoint machine where you are installing or updating the package. The debugging information is written to both the tuner's history logs and the package's history logs.

Tip: You can also view the log messages by starting a Console Window channel. In this case, the messages are written to both the Console Window and the logs. In UNIX, the messages are also written to the command-line window following the tuner command.

For <level> use 2. The levels actually range from 1 to 10. To enable debugging, set this number to be greater than 1. Symphony Teleca recommends that you use 2.

Example

This is an example of the tuner property to set to turn on debugging for an endpoint:

```
marimba.launch.javaArgs=-DDEBUGFLAGS=SOFTDIST=2
```

Important: The debugging feature can generate a large volume of log messages, which can cause the history logs to roll sooner than expected. If logs roll too frequently and older logs get deleted, you could lose important information.

Therefore, as soon as you finish debugging, be sure to turn debugging off. If you edited the prefs.txt file to turn debugging on, turn the feature off by deleting the -DDEBUGFLAGS line from the prefs.txt file and then restarting the tuner. If you turned debugging on by starting the tuner at the command line and using the -java -DDEBUGFLAGS option, turn the feature off by restarting the tuner without the -java -DDEBUGFLAGS option.

A Command-line reference

This section describes the command-line options for using Application Packager to package applications, to upgrade channels packaged with previous versions of Application Packager, and to start channels packaged using Application Packager. You can use the command-line options as an alternative to using the graphical user interface (GUI).

The following topics are provided:

- Overview (page 308)
- Packager for Shrinkwrap Windows Applications command-line options (page 309)
- Windows Installer Packager command-line options (page 311)
- File Packager command-line options (page 313)
- .NET Packager .NET Packager command-line options (page 315)
- PDA Packager command-line options (page 319)
- Custom Application Packager command-line options (page 324)
- Applying an XML template file to channels (page 325)
- Upgrading channels (page 325)
- Checking the version of Application Packager and channels (page 326)
- Starting packaged applications from the command line (page 327)
- Staging MSI applications on endpoints (page 329)

Overview

Each tuner comes with a program named `runchannel`, which you can run to start any channel on that tuner from the command line. For more information about using the `runchannel` program with other channels, see the command-line chapter in the *Symphony Marimba Client Automation Reference Guide* on the Marimba Channel Store.

The `runchannel` program

You can use the `runchannel` program to start a channel from the command line. The `runchannel` program is located in the tuner's installation directory (for example, in Windows, the default installation directory is `C:\Program Files\Marimba\Tuner`). As always when you are issuing commands from the command line, switch to the directory in which the executable file is located, or set that directory in the path your system is configured to use for finding commands.

Use the following syntax for the `runchannel` program:

```
runchannel <channel_URL> [options]
```

where:

`<channel_URL>` is the URL of the Application Packager channel that is subscribed to by the local tuner. It has the form:

```
http://<transmitter_name>[:<port>]/<channel_name>
```

That is, it must be the full URL for the channel. Even though the channel URL specifies the original location of that channel on the transmitter, `runchannel` still starts the channel from the tuner to which it was downloaded.

For example, to view the list of command-line options for the Application Packager channel that originate from the transmitter named `trans.acme.com`, type:

```
runchannel http://trans.acme.com:5282/ApplicationPackager -help
```

Note: Using `Tuner -start` (instead of `runchannel`) is not recommended because the console is not displayed. You are not able to see any output from Application Packager (for example, the command-line options are not displayed when you use the `-help` option).

Even when you do not specify any file in the <FILE> and <SCRIPT> tags, the package is created and no error occurs while the command line package is scripted. Thus, although the operation is not successful, no error message appears. Hence, to avoid this scenario, you can use the -strict command line. If you have provided the -strict command line and the runchannel program fails, because you have not specified any file for the <FILE> and <SCRIPT> tags, an error message appears. Using the -strict command line ensures that the user is informed of the operation failure.

Syntax

Here are some important notes regarding syntax:

- For easy reference, lists of options are ordered alphabetically.
- Regardless of the order in which command-line options are shown in the syntax, you can specify them in any order (with a few exceptions, as noted).
- Some options that take multiple arguments require the arguments to be enclosed in quotation marks, as noted in the descriptions of those options. Arguments that contain spaces must be enclosed in quotation marks.
- Command syntax and examples are sometimes shown on several lines, but each command must of course be typed from the command line without any break until the end of the command.

Note: When you specify the directory in which to create or save the channel, make sure the directory does not already contain another channel. Otherwise, the channel you create might not be usable.

Packager for Shrinkwrap Windows Applications command-line options

You can start Application Packager with the runchannel program and specify Packager for Shrinkwrap Windows Applications options as follows:

```
runchannel <App_Packager_URL> -shrinkwrappackage <options>
```

where:

the options can be any of those described in this section.

Creating preinstall and postinstall snapshots

First, you take two snapshots of the system: one before and one after installing the application you want to package. Later, Packager for Shrinkwrap Windows Applications compares the two snapshots to determine the files, registry entries, and other items that make up the packaged application or channel.

► To create preinstall and postinstall snapshots

- 1 Create the preinstall snapshot.
- 2 Install the application.
- 3 Create the postinstall snapshot.

You can create a snapshot as follows:

```
runchannel <App_Packager_URL> -snapshot <snapshot_file_path>  
<XML_template_file_path>
```

creates a snapshot of the system using the filters in the specified XML file and saves the snapshot to the specified path.

<file_path> specifies the full path and name for the file where you want to save the snapshot.

<XML_template_file_path> specifies the XML template file that contains the filters you want to use for the snapshot. For more information, see “Using XML template files with Packager for Shrinkwrap Windows Applications” on page 272 and “Snapshot tags” on page 428 in “XML syntax”.

Example

```
runchannel http://trans.acme.com:5282/ApplicationPackager -snapshot  
C:\SystemSnapshots\preinstall.msp  
C:\SystemSnapshots\Templates\snapshot.xml
```

Comparing the snapshots and creating a channel

After taking preinstall and postinstall snapshots, you compare the snapshots and create the channel using the following command-line options:

```
runchannel <App_Packager_URL> -shrinkwrappackage <channel_name>  
<presnapshot_file_path> <postsnapshot_file_path> <channel_directory>  
[-includedeletes <args>] [-defaults <XML_template_file_path>]
```

creates a new channel with the specified name in the specified directory.

<channel_name> specifies the name you want to give the channel.

<presnapshot_file_path> specifies the full path of the pre-snapshot file you previously created.

<postsnapshot_file_path> specifies the full path of the post-snapshot file you previously created.

<channel_directory> specifies the full path of the directory where you want to place the contents of the new channel you are creating.

[*-includedeletes <args>*] specifies whether to capture the removal of items and to include them in the channel. *<args>* specifies one or more of the following keywords:

```
filesystem
registry
metabase
```

If you omit this option, the packager does not capture the removal of items.

[*-defaults <XML_template_file_path>*] specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

Example

```
runchannel http://trans.acme.com:5282/ApplicationPackager
-shrinkwrappackage MyShrinkwrapChannel
C:\SystemSnapshots\preinstall.msp C:\SystemSnapshots\postinstall.msp
C:\Channels\MyShrinkwrapChannel -includedeletes filesystem registry
-defaults C:\Channels\Templates\shrinkwrap.xml
```

Windows Installer Packager command-line options

You can start Application Packager with the `runchannel` program and specify Windows Installer Packager options as follows:

```
runchannel <App_Packager_URL> -wipackage <MSI_database_path>
<channel_directory> [-include <true|false>] [-byreference
<true|false>] [-defaults <XML_template_file_path>]
```

creates a new channel with the specified name in the specified directory.

<MSI_database_path> specifies the full path of the MSI database file (with the `.msi` extension) you want to package.

<channel_directory> specifies the full path of the directory where you want to place the contents of the new channel you are creating.

[**-include**] specifies whether you want to include in the channel all the files that are in the same directory as the MSI file. If you omit this option, the other files in that directory are not included in the channel.

[**-byreference**] specifies whether you want to package files by reference.

[**-defaults <XML_template_file_path>**] specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

Example

```
runchannel http://trans.acme.com:5282/ApplicationPackager -wipackage  
C:\MSI\W2Kapp\application.msi C:\Channels\MyWindowsInstallerChannel  
-defaults C:\Channels\Templates\msi.xml
```

Repackaging the contents of an existing Windows Installer Packager channel

```
runchannel <App_Packager_URL> -wipackage <channel_directory>  
-repackage [-include <true|false>] [-byreference <true|false>]  
[-defaults <XML_template_file_path>]
```

repackages the contents of an existing Windows Installer Packager channel in the specified directory.

[**-include**] specifies whether you want to include in the channel all the files that are in the same directory as the MSI file. If you omit this option, the other files in that directory are not included in the channel.

[**-byreference**] specifies whether you want to package files by reference.

[**-defaults <XML_template_file_path>**] specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

Note: You can now retain the configuration settings and attributes on the endpoints during installation, uninstallation, and after updates by adding the `savemanifest` property in the `properties.txt` file before publishing.

File Packager command-line options

You can start Application Packager with the runchannel program and specify File Packager options as follows:

runchannel <App_Packager_URL> -filepackage <options>

where:

the options can be any of those described in this section.

Packaging a set of directories and files

runchannel <App_Packager_URL> -filepackage <title> <channel_directory> <directory_list> [-defaults <XML_template_file_path>] [-props <property_list>] [-publish <tx location>] [-txauth <username>:<password>]

creates a channel in the specified directory from the source directories listed in *<directory_list>*.

<title> specifies the title of the package

<channel_directory> specifies the full path of the directory where you want to place the contents of the new channel you are creating.

Each directory in *<directory_list>* has the form

*<directory> [as <install_directory>]
[prompt "<text>"]
[ref]
[ntperm]*

Either specify an install directory or prompt the user for a directory where the files should be installed.

<directory> is the full path of the directory containing content to package.

as *<install_directory>* specifies the directory on the user's system where the packaged files are installed when the channel is run.

prompt "*<text>*" queries the user for the directory in which to install the packaged files when the channel is run, displaying the specified text as a prompt for the user's input. The text should be a message asking the user to specify the target directory on the user's system.

ref packages by reference the specified directory.

Default value: false

ntperm specifies whether to capture Windows file permissions when packaging the specified directory. This option is available only in Windows NT, Windows 2000, and Windows XP. Windows supports two kinds of file systems: FAT and NTFS. The latter offers more advanced file permission settings. To take full advantage of this option, both the person who is creating the file package and the user must be using an NTFS file system. Otherwise, only basic Windows file permissions and attributes (read-only, archive, hidden, system) are preserved.

Default value: *false*

Note: Users installing the channel must have administrative privileges to set permissions.

`[-defaults <XML_template_file_path>]` specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used.

`[-props <property_list>]` specifies the list of properties to be set for the package. `<property_list>` specifies the list of properties to be set and it takes the format `<property1>=<value1>:<property2>=<value2>:<property3>=<value3>`

`[-publish <tx_location>]` specifies whether you want to publish the packaged channel. If your transmitter is configured with publish credentials, use `-txAuth` option to specify the publish credentials

Example

```
runchannel http://trans.acme.com:5282/ApplicationPackager -filepackage  
MyFilePackage C:\Channels\MyFilePackage C:\ContentFiles\src1 as C:\Install\src1  
prompt "Where to install src1?" ref ntperm C:\ContentFiles\src2 as C:\Install\src2  
prompt "Where to install src2?" ref ntperm -  
props=testprop1=testvalue1:testprop2=testvalue2 -publish http://myserver:5282/  
filepackage1 -txAuth user1:password
```

Repackaging the contents of an existing File Packager channel

```
runchannel <App_Packager_URL> -filepackage -repackage [-publish]  
<channel_directory> [-defaults <XML_template_file_path>]
```

repackages the contents of an existing File Packager channel in the specified directory. Optionally, you can immediately publish the repackaged channel by using the `-publish` option.

[**-defaults <XML_template_file_path>**] specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

Example (Windows)

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-filepackage -repackage C:\Channels\MyFilePackage
```

Example (UNIX)

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-filepackage -repackage /home/user1/MyFilePackage
```

Note: You can now use the `filepackager.savemanifest=true` property in the `properties.txt` file before publishing to retain the configuration settings during installation, unistallation, and after updates.

.NET Packager .NET Packager command-line options

You can start Application Packager with the `runchannel` program and specify .NET Packager options as follows:

```
runchannel <App_Packager_URL> -dotnetpackage  
-packagedir <channel_directory>  
[-channelname <channel_name>]  
[-ini_file_objects <true|false>]  
[-assembly_policy_config_objects <true|false>]  
[-sourcedir <source_directory_path>  
[as <target_directory_path>]  
[prompt <install_directory_prompt_text>]  
[filebyref <true|false>]  
[ntfileperm <true|false>]  
]  
[-defaults <XML_template_file_path>]
```

creates a new channel with the specified name in the specified directory.

<channel_directory> specifies the full path of the directory where you want to place the contents of the new channel you are creating.

<channel_name> specifies a name for the channel.

`-ini_file_objects` specifies whether INI files should be treated as special file objects. If this option is set to `true`, Application Packager parses the INI files and captures any changes made to the key and value pairs in them. Any changes found by Application Packager are merged with the existing file found at the endpoint. If you want Application Packager to process INI files as ordinary files and not merge changes, you should set this option to `false`. For more information, see “Working with INI files” on page 78.

Default value: `true`

`-assembly_policy_config_objects` specifies whether you want assembly policy configuration files to be treated as special file objects. Application Packager parses the configuration files and capture any changes made to them. Any changes found by Application Packager are merged with the existing file found at the endpoint. If you want Application Packager to process configuration files as ordinary files and not merge changes, you should set this option to `false`.

Default value: `true`

`[-defaults <XML_template_file_path>]` specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

For each source directory you want to include in the channel, you specify the following information:

`<source_directory_path>` is the full path of the directory containing content to package.

`as <target_directory_path>` specifies the directory on the endpoint where the packaged files are installed when the channel is run.

`prompt "<install_directory_prompt_text>"` queries the endpoint user for the directory in which to install the packaged files when the channel is run, displaying the specified text as a prompt for the user’s input. The text should be a message asking the user to specify the target directory on the user’s system.

`filebyref` specifies whether to package files (in the specified directory) by reference. For more information about packaging files by reference, see “Adding files by reference” on page 72.

Default value: `false`

`ntfileperm` specifies whether to capture Windows file permissions when packaging the specified directory. This option is available only in Windows NT, Windows 2000, and Windows XP. Windows supports two kinds of file systems: FAT and NTFS. The latter offers more advanced file permission settings. To take full advantage of this option, both the person who is creating the file package and the user must be using an NTFS file system. Otherwise, only basic Windows file permissions and attributes (read-only, archive, hidden, system) are preserved.

Default value: `false`

Note: Users installing the channel must have administrative privileges to set permissions.

Example

```
runchannel http://trans.acme.com:5282/ApplicationPackager -dotnetpackage -packagedir C:\Channels\MyDotNetPackage -channelname My_Dot_Net_Pkg -sourcedir C:\AssemblyFiles\src1 as C:\Install\src1 prompt "Where to install src1?" filebyref true ntfileperm true -sourcedir C:\AssemblyFiles\src2 as C:\Install\src2 prompt "Where to install src2?" filebyref true ntfileperm true
```

Repackaging the contents of an existing .NET Packager channel

```
runchannel <App_Packager_URL> -dotnetpackage -packagedir <channel_directory>
[-sourcedir <source_directory_path>
[as <target_directory_path>]
[prompt <install_directory_prompt_text>]
[filebyref <true|false>]
[ntfileperm <true|false>]
]
-repackage
[-defaults <XML_template_file_path>]
```

repackages the contents of an existing .NET Packager channel in the specified directory.

For each source directory you want to include in the channel, you specify the following information:

`<source_directory_path>` is the full path of the directory containing content to package.

`as <target_directory_path>` specifies the directory on the endpoint where the packaged files are installed when the channel is run.

`prompt "<install_directory_prompt_text>"` queries the endpoint user for the directory in which to install the packaged files when the channel is run, displaying the specified text as a prompt for the user's input. The text should be a message asking the user to specify the target directory on the user's system.

`filebyref` specifies whether to package files (in the specified directory) by reference. For more information about packaging files by reference, see "Adding files by reference" on page 72.

Default value: `false`

`ntfileperm` specifies whether to capture Windows file permissions when packaging the specified directory. This option is available only in Windows NT, Windows 2000, and Windows XP. Windows supports two kinds of file systems: FAT and NTFS. The latter offers more advanced file permission settings. To take full advantage of this option, both the person who is creating the file package and the user must be using an NTFS file system. Otherwise, only basic Windows file permissions and attributes (read-only, archive, hidden, system) are preserved.

Default value: `false`

Note: Users installing the channel must have administrative privileges to set permissions.

`[-defaults <XML_template_file_path>]` specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see "Using XML template files" on page 269 and "XML syntax" on page 369.

Before repackaging, save the previous configuration of the Windows Installer package in an XML template file so that it can be used on a subsequent repackage. You can also add the `netpackager.savemanifest=true` property in the `properties.txt` file to save the configuration. After the Windows Installer package has been repackaged with updated content, the XML template file is applied to restore the previous configuration. For more information, see "To save the configuration for your .NET Packager channel" on page 245.

Note: You can now use the `netpackager.savemanifest=true` property in the `properties.txt` file before publishing to retain the configuration settings during installation, uninstallation, and after updates.

PDA Packager command-line options

This section describes how to use the command line to create packages for your mobile devices. You can use the `-pdapackage` command-line option as an alternative to using the PDA Packager graphical user interface.

Syntax for runchannel—Use the following syntax for the `runchannel` program:

```
runchannel <App_Packager_URL> -pdapackage <arg1> <arg2>
```

where:

`<App_Packager_URL>` is the URL of the Application Packager channel that is subscribed to by the local tuner. It has the form:

```
http://<transmitter_name>[:<port>]/ApplicationPackager
```

That is, it must be the full URL for the channel. Even though the channel URL specifies the original location of that channel on the transmitter, `runchannel` still starts the channel from the tuner to which it was downloaded.

Syntax for the -pdapackage option—The arguments for the `-pdapackage` option are described in the rest of this section. The following options and arguments apply to both packaging content and packaging applications:

```
-pdapackage -packagedir <channel_directory>
[-channelname <name_of_channel>] [-repackage]
```

specifies that you want to create a package for a mobile device. The `-packagedir` option creates a channel in the directory specified by `<channel_directory>`. This directory name is also used as the URL name when the package is copied to your transmitter (unless you use Channel Copier to change the URL name). Regardless of whether you want to package an application (CAB file) or content (files in a directory), specify both the `-pdapackage` and `-packagedir` options.

If you are packaging the channel for the first time, be sure to use the option `-channelname` to specify the name you want to give the channel.

If you are *repackaging* the channel, rather than packaging the channel for the first time, you can omit the `-channelname` option. In this case, be sure to use the `-repackage` option. For more information about repackaging, see “Using the PDA packager to update a channel” on page 251.

Then, if you are creating a package for the first time, depending on the type of package you want to create, use one of the following groups of command-line options (if you are repackaging, use the following options only if you want to add directories or CAB files or change another setting):

- **Content files**—For packaging content (one or more directories of files), use the following options:

`-sourcedir <source_directory>` specifies the full path to the directory that contains the content to be packaged.

`as <target_directory>` specifies the full path to the directory on the user’s system where the packaged files are installed when the channel is run. In most cases, you can use a short path. For example, if you package `C:\Program Files\Microsoft ActiveSync\MyApplication`, you probably want the install path to be `\MyApplication`.

`[filebyref {true|false}]` means, if set to true, that the source directory is packaged by reference in the specified directory. Use this argument to save disk space on the packaging machine.

Default value: `false`

Note: You can use this group of options numerous times if you have more than one directory you want to package.

Example of packaging a directory:

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-pdapackage -packagedir C:\Channels\PriceList  
-channelname PriceList -sourcedir D:\DeviceFiles\PriceListSource  
as \PriceList
```

Note: Command syntax and examples are sometimes shown on several lines, but you must, of course, type each command from the command line without any break until the end of the command.

- **CAB files**—For packaging CAB files (applications), use the following options:

-cabfile <source_CAB_file> specifies the full path to the CAB file you want to package.

unloadname <application_name> specifies the name to use to uninstall the application from the mobile device. The application name you use *must be* the same name that would appear in the Remove Programs list on your device if you installed the application without using Symphony Marimba Client Automation products. Therefore, before you create your package, install the application on one mobile device without using Symphony Marimba Client Automation products, and find out the name used in the Remove Programs list.

[filebyref {true|false}] means, if set to true, that the source CAB file is packaged by reference. Use this argument to save disk space on the packaging machine.

Default value: false

Note: You can use this group of options numerous times if you have more than one CAB file you want to package.

Example of packaging a CAB file:

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-pdapackage -packagedir C:\Channels\MyApplication  
-channelname MyApplication  
-cabfile "D:\Device Applications\MyApp.cab"  
unloadname MyApplication
```

Tip: If you want to use the command-line interface to create a package, and then later decide to use the PDA Packager graphical user interface (GUI) to reconfigure the package, choose Application Packager's File > Import command in the GUI, and then browse to the channel directory. When you import the channel directory in this manner, the channel is listed in the PDA Packager.

Publishing the packages— When you successfully package the directory or CAB files by using the appropriate command-line options, use Channel Copier to actually publish the packaged application channels to your transmitter. In the Application Packager GUI, you click the Publish button to accomplish this task. (The Publishing wizard then appears and guides you through the publishing process.) If you are using the Application Packager’s command-line interface, however, start Channel Copier and manually copy the channels.

► **To copy a package to the transmitter**

- 1 On your packaging machine, make sure your tuner is running.
- 2 Double-click the Channel Copier channel, which appears in the Channel Manager.

The Channel Copier window appears.

- 3 To create a new copy operation, click New.
- 4 In the Select Source area, click the folder icon and browse to the channel directory (that is, the directory that contains the contents of your new channel).

This is the directory you specified by using the `-packagedir` option when creating the PDA package.

- 5 In the Select Destination area, select the transmitter to which you want to publish the channel.
- 6 Click Add/Close.

The main Channel Copier window appears, and your new copy operation is selected.

- 7 Click Copy to publish your new channel.
- 8 When the copy operation has been completed, you can exit Channel Copier.

Virtual Packager command-line options

This section describes command-line options for the Virtual Packager.

Packaging a set of directories and files

The following information describes how to use the virtual packager command-line options to package a set of directories and files:

```
runchannel <App_Packager_URL> -virtualpackage
<channel_directory><directory_list> [-defaults
<XML_template_file_path>]
```

- [-defaults <XML_template_file_path>]—specifies the XML template file to use when creating the new channel. If you omit this option, the Application Packager default XML template file is used.
- <channel_directory>—the full path of the directory to place the contents of the new channel that you are creating
- <directory_list>—the source directories

Each directory in <directory_list> has the following form:

```
<directory> [as <install_directory>]
[prompt "<text>"]
[-byreference <true|false>]
[-include <true|false>]
```

<directory>—the full path of the directory containing content to package

- as <install_directory>—the directory on the user system where the packaged files are installed when the channel is run.
- prompt "<text>"—queries the user for the directory in which to install the packaged files when the channel is run, displaying the specified text as a prompt for input. The text should be a message asking the user to specify the target directory.
- [-include]—specifies whether to include in the channel all the files that are in the same directory as the MSI file. If you omit this option, other files in that directory are not included in the channel.
- [-byreference]—specifies whether to package files by reference.

Example (Windows):

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-virtualpackage C:\Channels\MyVirtualPackage C:\ContentFiles\src1  
as C:\Install\src1 prompt "Where to install src1?" -byreference  
true -include false
```

Repackaging the contents of an existing Virtual Packager channel

Use the following command to repackage or publish an existing Virtual Packager channel in the specified directory:

```
runchannel <App_Packager_URL> -virtualpackage -repackage  
[-publish] <channel_directory> [-defaults  
<XML_template_file_path>]
```

[**-defaults <XML_template_file_path>**]—specifies the XML template file to use when creating the new channel. If you omit this option, the Application Packager default XML template file is used.

Example (Windows):

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-virtualpackage -repackage C:\Channels\MyVirtualPackage
```

Custom Application Packager command-line options

You can start Application Packager with the `runchannel` program and specify Custom Application Packager options as follows:

```
runchannel <App_Packager_URL> -custompackage <channel_name>  
<channel_directory> [-defaults <XML_template_file_path>]
```

creates a new channel with the specified name in the specified directory.

<channel_name> specifies the name you want to give the channel.

<channel_directory> specifies the full path of the directory where you want to place the contents of the new channel you are creating.

[**-defaults <XML_template_file_path>**] specifies the XML template file you want to use when creating the new channel. If you omit this option, the XML template file set as the default for Application Packager is used. For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

Example (Windows)

```
runchannel http://trans.acme.com:5282/ApplicationPackager  
-custompackage MyCustomChannel C:\Channels\MyCustomChannel -defaults  
C:\Channels\Templates\custom.xml
```

Example (UNIX)

```
runchannel http://trans.acme.com:5282/ApplicationPackager
-custompackage MyCustomChannel /opt/source/pkgs/MyCustomChannel
-defaults /opt/source/pkgs/templates/custom.xml
```

Applying an XML template file to channels

This command-line option applies an XML template file to an existing channel that was packaged using Application Packager. You can start Application Packager with the `runchannel` program and specify the package directory and XML template files as follows:

```
runchannel <App_Packager_URL> -applytemplate <channel_directory>
<paths_of_XML_template_files>
```

where:

`<channel_directory>` specifies the full path of the directory where you want to place the contents of the new channel you are creating.

`<paths_of_XML_template_files>` specifies the paths of the XML template files you want to apply to the channel. To use more than one XML template file, separate the paths with semicolons (;). For more information, see “Using XML template files” on page 269 and “XML syntax” on page 369.

Example (Windows)

```
runchannel http://trans.acme.com:5282/Marimba/ApplicationPackager
-applytemplate "C:\Channels\MyCustomChannel"
"C:\Channels\Templates\shrinkwrap.xml";
C:\Channels\Templates\policies.xml;
C:\Channels\Templates\macros.xml"
```

Example (UNIX)

```
runchannel http://trans.acme.com:5282/Marimba/ApplicationPackager
-applytemplate /home/user1/MyCustomChannel /home/user1/Channels/
Templates/policies.xml;
/home/user1/Channels/Templates/macros.xml"
```

Upgrading channels

This command-line option upgrades channels that were packaged using a previous version of Application Packager to the current version. You can upgrade channels that were packaged with Application Packager 4.0 and later.

You can start Application Packager with the `runchannel` program and specify the channels to upgrade as follows:

```
runchannel <App_Packager_URL> -upgrade <path_of_text_file>
```

where :

<path_of_text_file> is the full path of the text file containing the directory paths of the channels to upgrade, one entry per line.

Example (Windows)

```
runchannel http://trans.acme.com:5282/Marimba/ApplicationPackager  
-upgrade C:\packagedchannels\upgrade.txt
```

where the file upgrade.txt contains entries like these:

```
C:\packagedchannels\app1  
C:\packagedchannels\app2  
C:\packagedchannels\app3  
C:\packagedchannels\app4  
C:\packagedchannels\app5
```

Example (UNIX)

```
runchannel http://trans.acme.com:5282/Marimba/ApplicationPackager  
-upgrade /home/user1/packagedchannels/upgrade.txt
```

where the file upgrade.txt contains entries like these:

```
/home/user1/packagedchannels/app1  
/home/user1/packagedchannels/app2  
/home/user1/packagedchannels/app3  
/home/user1/packagedchannels/app4  
/home/user1/packagedchannels/app5
```

Checking the version of Application Packager and channels

You can use these command-line options to check the version number for Application Packager and channels you created using Application Packager. These command-line options work only for the following versions:

- Application Packager version 4.7.2 and higher
- Channels packaged using Application Packager version 4.7.2 and higher

► To check the version number of Application Packager

- Use the runchannel program and specify the URL for Application Packager as follows:

```
runchannel <App_Packager_URL> -version
```

For example:

```
runchannel http://trans.acme.com:5282/Marimba/ApplicationPackager  
-version
```

The version number, such as 4.7.2.0, appears.

► **To check the version number of Application Packager that was used to package a channel**

- Use the runchannel program and specify the URL for the channel as follows:

```
runchannel <channel_URL> -version
```

Example

```
runchannel http://trans.acme.com:5282/Applications/CustomApplication  
-version
```

The version number, such as 4.7.2.0, appears.

Starting packaged applications from the command line

You can start an application packaged using Application Packager from the command line as follows:

```
runchannel <channel_URL> <options>
```

where:

the options can be any of those described in this section.

```
-exec [<arguments>]
```

starts the application, if any, that is configured to start from the channel, optionally passing arguments that the application has been configured to accept. (Using `-exec` without any arguments has the same effect as using `runchannel` without any options.) If the channel has not been installed, this option launches the installer first.

The `-exec` option should appear last among any options specified because everything that follows `-exec` is considered an argument to be passed to a started instance of the application.

```
-install
```

installs on the tuner the files the channel has been configured to install.

```
-refreshsourcelist
```

refreshes the source list for an MSI channel. This option is valid only for channels created using the Windows Installer Packager.

```
-remove
```

uninstalls the files the specified channel installed on the tuner.

-repair

checks the channel integrity and fixes any errors in the channel.

-rundir <directory>

specifies a new working directory for the channel.

-runexe <application_name>

starts the specified application, usually one installed by the channel.

application_name must include the full path of the application executable or batch file.

Note: You can configure the process creation associated with this command-line option by setting the `processCreationFlags` in the `parameters.txt` file of the package. For more information, see the *Symphony Marimba Client Automation Reference Guide*, available on the Marimba Channel Store.

-setmacro <macro_name>=<macro_definition>

defines a macro for customizing channels. For example, on a Windows system it could be the following macro:

“-setmacro INSTALLDIR=c:\program files\app”

If the macro or macro definition includes spaces, you must enclose the entire command line within quotation marks (“”).

Note: The `-setmacro` command-line option by itself cannot cause a major update to occur when a packaged application is being updated. For example, using `-setmacro` to set change the `$dir1` macro’s value in a file-packaged channel to another value does not cause a major update to occur. For more information about major updates, see “Managing major and minor updates” on page 150.

-silent

installs the channel in silent mode.

-stagems

stages an MSI channel. This option is valid only for channels created using the Windows Installer Packager. For more information, see “Staging MSI applications on endpoints” on page 329.

-verify

checks the channel integrity.

-version

displays the version of Application Packager that was used to create this channel. For more information, see “Checking the version of Application Packager and channels” on page 326.

The form with no options:

Tuner -start <channel_URL>

can be used to start the application, if any, that was previously installed by the specified channel (and configured to start from that channel).

Staging MSI applications on endpoints

This command-line option is useful when used with the stage state available when using Policy Manager. You use this option as follows:

```
runchannel <channel_URL> -stagems1
```

The MSI file and any support files are staged in an `msi` directory in the channel directory (for example,

`C:\<Tuner_workspace_directory>\ch.X\data\msi`, where `X` is a number representing the channel).

B Policies for installation, update, uninstallation, verification, and repair

This section lists the policies that are available for channels created using Application Packager. It includes the policies that apply to the entire channel, as well as policies that apply to specific items in a channel. It also includes a brief description of the actions that take place for each policy.

The following topics are provided:

- Overview (page 332)
- Policies for the entire channel (page 332)
- Policies for directories and files (page 342)
- Policies for INI files (page 352)
- Policies for registry and metabase keys and values (page 356)
- Policies for Windows NT services (page 361)
- Policies for text modifier groups and text modifiers (page 366)

Overview

This section lists the policies you can apply to the entire channel. For more information about setting these policies, see “Setting default policies for channels” on page 123. By default, items in the channel inherit the policies you set for the entire channel. However, you can override those default policies by setting policies for individual items in the channel. The available policies for individual items are listed in the other sections of this section. For more information about setting these policies, see “Setting policies for individual items in a channel” on page 124.

Policies for the entire channel

The following tables describe the actions taken for each policy:

- Installation policies: Table B-1 on page 332
- Update policies: Table B-2 on page 334
- Uninstallation policies: Table B-3
- Verify policies: Table B-4 on page 339
- Repair policies: Table B-5 on page 341

Installation policies

Table B-1: Installation policies—General

Policy	Actions taken for this policy
Install if object is newer:	
n Prompt user for action if object is not newer.	<ol style="list-style-type: none">1 Determine which object is newer by comparing the versions or, if versions do not exist for both objects, by comparing the timestamps.2 If the object that is already installed is newer than the one to be installed, ask the user whether to replace the object.
n Compare objects based on timestamps.	The objects’ timestamps determine which object is newer. If the object to be installed is newer (if it has a later date), it replaces the one already installed on the endpoint.

Table B-1: Installation policies—General(Continued)

Policy	Actions taken for this policy
n Compare objects based on versions. If version doesn't exist, compare timestamps.	<p>The objects' version numbers determine which object is newer.</p> <p>At installation time:</p> <ul style="list-style-type: none"> n If the object to be installed is newer (if it has the greater version number), it replaces the one already installed on the endpoint. n If a version number exists for the object to be installed, but not for the object on the endpoint, the endpoint object is replaced. If no version number exists for the object to be installed, but one exists for the object on the endpoint, the endpoint object is not replaced. n If neither object has a version number, timestamps are used. If the object to be installed is newer (if it has a later date), it replaces the one already installed on the endpoint.
Install only if object doesn't exist.	<ol style="list-style-type: none"> 1 Check if an object is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both objects have versions, timestamps are ignored and versions are compared.) 2 If the object is already installed, skip the object and do not install. 3 If the object is not installed, check if an object with the same name already exists on the endpoint. 4 If the object exists, skip the object and do not install. Otherwise, install the object.
Install only if object already exists.	<ol style="list-style-type: none"> 1 Check if an object is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both objects have versions, timestamps are ignored and versions are compared.) 2 If the object is not installed, check if an object with the same name already exists on the endpoint. 3 If an object with the same name is found on the endpoint, replace it by installing the object in the channel. Otherwise, skip the object and do not install.
Always install object.	Always install the object, and overwrite the object if it is already installed on the endpoint.
Don't install object.	Never install the object.

Update policies

Table B-2: Update policies—General

Policy	Actions taken for this policy
Update if object is newer:	
n Prompt user for action if object is not newer.	<ol style="list-style-type: none"> 1 Determine which object is newer by comparing the versions or, if versions do not exist for both objects, by comparing timestamps. 2 If the object that is already installed is newer than the one in the update, ask the user whether to replace the object.
n Compare objects based on timestamps.	The objects' timestamps determine which object is newer. If the object in the channel update is newer (if it has a later date), it replaces the one already installed on the endpoint.
n Compare objects based on versions. If version doesn't exist, compare timestamps.	<p>The objects' version numbers determine which object is newer.</p> <p>At update time:</p> <ul style="list-style-type: none"> n If the object in the update is newer (if it has the greater version number), it replaces the one already installed on the endpoint. n If a version number exists for the object in the channel update, but not for the object on the endpoint, the endpoint object is replaced. If no version number exists for the object in the channel update, but one exists for the object on the endpoint, the endpoint object is not replaced. n If neither object has a version number, timestamps are used. If the object in the channel update is newer (if it has a later date), it replaces the one already installed on the endpoint.
Update only if object doesn't exist.	<ol style="list-style-type: none"> 1 Check if an object is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both objects have versions, timestamps are ignored and versions are compared.) 2 If the object is already installed, skip the object and do not update. 3 If the object is not installed, check if an object with the same name already exists on the endpoint. 4 If the object exists, skip the object and do not update. Otherwise, update the object.

Table B-2: Update policies—General

Policy	Actions taken for this policy
Update only if object already exists.	<p>1 Check if an object is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both objects have versions, timestamps are ignored and versions are compared.)</p> <p>2 If the object is not installed, check if an object with the same name already exists on the endpoint.</p> <p>3 If an object with the same name is found on the endpoint, replace it by installing the object in the channel update. Otherwise, skip the object and do not update.</p>
Always update object.	Always update the object, and overwrite the object if it is already installed on the endpoint.
Don't update object.	Never update the object.

Uninstallation policies

Table B-3: Uninstallation policies—General

Policy	Actions taken for this policy
Uninstall object if it was installed, prompt user before removing shared objects.	<p>1 Check if the object was ever installed using Application Packager. If not, do not attempt to uninstall the object.</p> <p>2 Check if the object was already uninstalled from the endpoint by determining if an object with that name and path exists.</p> <p>3 If the object is not already uninstalled, check if the object is shared.</p> <p>4 If it is shared, ask the user if the user wants to uninstall the object. Otherwise, uninstall the object by removing it or replacing it with an older version that Application Packager overwrote during installation or an update.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs). This policy is available in Windows only.</p>
Uninstall object if it was installed.	<p>1 Check if the object was ever installed using Application Packager. If not, do not attempt to uninstall the object.</p> <p>2 Check if the object was already uninstalled from the endpoint by determining whether an object with that name and path exists.</p> <p>3 If the object is not already uninstalled, uninstall the object by removing it or replacing it with an older version that Application Packager overwrote during installation.</p> <p>For example, an object is part of a channel, but it was not installed at installation time because there was a newer object with the same name already installed on the endpoint. If this uninstallation policy is selected, that object is not uninstalled because it was never actually installed using Application Packager.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>

Table B-3: Uninstallation policies—General(Continued)

Policy	Actions taken for this policy
Always uninstall existing object.	<p>Restore the object to its previous state (the state it was in before Application Packager was used to install the object):</p> <ul style="list-style-type: none"> <li data-bbox="534 295 1249 381">n If Application Packager overwrote an older version of the object with a newer version, this uninstallation policy causes the older version to be restored. <li data-bbox="534 393 1249 480">n If Application Packager installed the object because it did not previously exist, this uninstallation policy causes the object to be uninstalled and removed. <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Always remove existing object.	<p>Whether or not Application Packager was used to install the object, uninstall and remove the object if it exists on the endpoint and is part of the channel.</p> <p>For example, an object is part of a channel, but it was not installed at installation time because there was a newer object with the same name already installed on the endpoint. If this uninstallation policy is selected, that object on the endpoint is uninstalled and removed even though it was never actually installed using Application Packager.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not removed unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Don't uninstall object.	Never attempt to uninstall the object.

Note: For additional examples for each of the uninstallation policies, see “Determining whether files are restored or removed during uninstallation” on page 126.

Verification policies

The policies described in Table B-4 on page 339 apply to these two types of verification policies:

- **Verify Installed**—Takes effect during the verify and repair phases for objects that were installed by a channel (created using Application Packager) at one point. An object can be installed during installation, update, or repair of a channel.
- **Verify Not Installed**—Takes effect during the verify and repair phases for objects that were never installed by a channel (created using Application Packager). For example, if a newer version of the file is already installed on the endpoint and there is an older version in the channel, the newer version is not overwritten by the older one if the installation policy Install if object is newer—Compare objects based on versions is selected.



As a best practice for the Verify not installed policy, set Verify the existence of the object. Setting Verify contents of the object is not recommended.

The policies are checked in the following order, and the first policy that fails prevents the other policies from being checked:

- 1 If Don't verify object during package verification is selected, the object is not checked.
- 2 Verify the existence of the object.
- 3 Verify contents of the object.
- 4 If Verify contents of the object fails, If the contents differ, allow Windows versions to increase is evaluated.

5 Verify the attributes of the object.

Table B-4: Verification policies—General

Policy	Actions taken for this policy
Verify object using the following policies	<p>n Verify contents of the object</p> <p>Check whether the object on the endpoint matches the one described in the manifest. The manifest contains information about the objects included in the packaged application.</p> <ul style="list-style-type: none"> n If the object on the endpoint does not match the attributes specified in the manifest, verification fails. n If the object does not exist on the endpoint, no check is performed. <p>For files, check whether the contents of the file on the endpoint matches the one described in the manifest by comparing MD5 checksums.</p> <ul style="list-style-type: none"> n If the MD5 checksums do not match, verification fails. n If the file does not exist on the endpoint, no check is performed.
n If the contents differ, allow Windows versions to increase.	<p>Using this policy prevents updated and new versions of objects, such as DLL files (in Windows), from being overwritten during a repair operation, while still allowing you to repair objects with other types of changes. This policy does not apply to all objects; it applies to files only.</p> <p>Verify an object as described in “Verify contents of the object”.</p> <p>Note: This policy is valid only if “Verify contents of the object” is selected and fails.</p> <p>If the MD5 checksums of the file on the endpoint and the one in the manifest do not match, and the Windows version has not changed or has decreased, verification fails.</p>

Table B-4: Verification policies—General(Continued)

Policy	Actions taken for this policy
n Verify the attributes of the object.	<p>Using this policy allows you to check the attributes of a file. This policy does not apply to all objects; it applies to files only.</p> <p>For files, check whether the attributes of the file on the endpoint matches the attributes described in the manifest. The attributes checked include the last modified time, creation time, and the attribute flags.</p> <ul style="list-style-type: none"> n If the attributes do not match, verification fails. n If the file does not exist on the endpoint, no check is performed. <p>Note: If this policy is set, verify fails if the file's contents have been modified at the endpoint. The last modified timestamp is considered part of the attributes, so if the file's contents have been modified at the endpoint, verify fails because editing the file's contents has changed the timestamp.</p>
n Verify the existence of the object.	Check if the object exists on the endpoint. If it does not, verification fails.
Don't verify object during package verification.	Never attempt to verify the object.

Repair policies

The repair policies are checked in the following order, and the first policy that fails prevents the other policies from being executed:

- 1 If Do not repair object is selected, the object is not repaired.
- 2 Repair the contents of the object and Do not repair contents if versions increased.

3 Repair the attributes of the object.

Table B-5: Repair policies—General

Policy	Actions taken for this policy
Repair the object using the following policies	<p>n Repair the contents of the object Repair the object on the endpoint so that it matches the one described in the manifest. The manifest contains information about the objects included in the packaged application.</p> <p>For files, the object on the endpoint has the same MD5 checksum as the one in the manifest after being repaired, unless “Do not repair contents if versions increased” is selected.</p> <p>If the MD5 checksums are already the same, no repair is done.</p>
n Do not repair contents if versions increased	<p>Using this policy prevents updated and new versions of objects, such as DLL files (in Windows), from being overwritten during a repair operation, while still allowing you to repair objects with other types of changes. This policy does not apply to all objects; it applies to files only.</p> <p>If the MD5 checksum of the file on the endpoint does not match the one in the manifest, and the version of the file has increased when compared to the version in the manifest, the contents of the file are not repaired.</p>
n Repair the attributes of the object.	<p>Using this policy allows you to repair the attributes of a file. This policy does not apply to all objects; it applies to files only.</p> <p>For files, repair the file on the endpoint so that its attributes match the attributes described in the manifest. The attributes checked include the last modified time, creation time, and the attribute flags.</p> <p>If the attributes are already the same, no repair is done.</p>
Do not repair the object.	<p>Never attempt to repair the object.</p> <p>Note: Setting this policy might result in having empty directories and registry keys after a repair operation.</p>

Policies for directories and files

This section lists the policies you can apply to the directories and files in a channel. By default, directories and files in the channel inherit the policies you set for the entire channel or from the policies of a parent directory. However, you can override those default policies by setting policies for individual directories and files in the channel. For more information about setting these policies, see “Setting policies for individual items in a channel” on page 124.

- 1 Some policies apply only to particular types of files:
 - For files that have .exe, .ocx, or .dll file extensions (in Windows) and other files that include version information, policies can use that version information to determine whether to install, update, uninstall, or verify those files.
 - For INI files, different policies apply. “Policies for INI files” on page 352 lists the policies for installing, updating, uninstalling, and verifying INI files.
 - For shortcuts and link files, the following policies do not apply:
 - Install if file is newer.
 - Update if file is newer.
 - Verify file, but allow Windows versions to increase.
 - 2 When you set policies for container objects (such as directories, registry keys, and metabase keys), you are actually setting the policies for the child objects in them (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs). For example, when you set the installation policies for a directory, the name of the Install Policy tab is “Install policy for this directory’s contents.” Container objects get these policies only:
 - Always install
 - Always update
 - Uninstall if possible (Container objects are not uninstalled unless they are empty of all child objects.)
 - Always verify
- The following tables describe the actions taken for each policy:
- Installation policies: Table B-6 on page 344

- Update policies: Table B-7 on page 346
- Uninstallation policies: Table B-8 on page 348
- Verify policies: Table B-9 on page 350
- Repair policies: Table B-10 on page 352

How update policies work when removing shortcuts

This section describes how the update policies work when removing shortcuts in various scenarios.

Scenario set up—A channel is created with the default global installation policy and the global update policy “Update if object is newer.” Create a shortcut in the channel, and then publish and install the updated channel. The shortcut is installed on the desktop endpoint.

► Scenario 1

- 1 Remove the shortcut from the desktop endpoint.
- 2 Do not edit the shortcut object in the channel, but just change its update policy to “Always update.”
- 3 Publish and install the update. The shortcut is placed back on the desktop endpoint.

► Scenario 2

- 1 Remove the shortcut from the desktop endpoint.
- 2 Edit the shortcut object in the channel, but do not change any policies.
- 3 Publish and install the update. The shortcut is placed back on the desktop endpoint.

► Scenario 3

- 1 Remove the shortcut from the desktop endpoint.
- 2 Do not edit the shortcut object in the channel, but just change its update policy to “Always update.”
- 3 Publish and install the update. The shortcut is placed back on the desktop endpoint.
- 4 Remove the shortcut from the desktop endpoint again.

- 5 Update again without changing anything in the channel. The shortcut is not placed back on the desktop endpoint because there was no update to the shortcut object itself.

► Scenario 4

- 1 Remove the shortcut from the desktop endpoint.
- 2 Do not edit the shortcut object in the channel, but just change its update policy to “Always update.”
- 3 Publish and install the update. The shortcut is placed back on the desktop endpoint.
- 4 Remove the shortcut from the desktop endpoint again.
- 5 Edit the channel by adding other files, but do not edit or update the shortcut object itself.
- 6 When you update the channel, the shortcut is not placed back on the desktop endpoint, even though there was an update to the channel, because there was no update to the shortcut object itself.

Installation policies

Table B-6: Installation policies—Directories and files

Policy	Actions taken for this policy
Install if file is newer:	
n Prompt user for action if file is not newer.	<ol style="list-style-type: none">1 Determine which file is newer by comparing the versions or, if versions do not exist for both files, by comparing the timestamps.2 If the file that is already installed is newer than the one to be installed, ask the user if the user wants to replace the file.
n Compare files based on timestamps.	The files’ timestamps determine which file is newer. If the file to be installed is newer (if it has a later date), it replaces the one already installed on the endpoint.

Table B-6: Installation policies—Directories and files(Continued)

Policy	Actions taken for this policy
<ul style="list-style-type: none"> n Compare files based on versions. If version doesn't exist, compare timestamps. 	<p>The files' version numbers determine which file is newer.</p> <p>At installation time:</p> <ul style="list-style-type: none"> n If the file to be installed is newer (if it has the greater version number), it replaces the one already installed on the endpoint. n If a version number exists for the file to be installed, but not for the file on the endpoint, the endpoint file is replaced. If no version number exists for the file to be installed, but one exists for the file on the endpoint, the endpoint file is not replaced. n If neither file has a version number, timestamps are used. If the file to be installed is newer (if it has a later date), it replaces the one already installed on the endpoint.
<p>Install if file doesn't exist.</p>	<ol style="list-style-type: none"> 1 Check if a file is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both files have versions, timestamps are ignored and versions are compared.) 2 If the file is already installed, skip the file and do not install. 3 If the file is not installed, check if a file with the same name already exists on the endpoint. 4 If the file exists, skip the file and do not install. Otherwise, install the file.
<p>Install only if file exists.</p>	<ol style="list-style-type: none"> 1 Check if a file is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both files have versions, timestamps are ignored and versions are compared.) 2 If the file is not installed, check if a file with the same name already exists on the endpoint. 3 If a file with the same name is found on the endpoint, replace it by installing the file in the channel. Otherwise, skip the file and do not install.
<p>Always install the file.</p>	<p>Always install the file, and overwrite the file if it is already installed on the endpoint.</p>
<p>Don't install the file.</p>	<p>Never install the file.</p>

Update policies

Table B-7: Update policies—Directories and files

Policy	Actions taken for this policy
Update if file is newer:	
n Prompt user for action if file is not newer.	<p>1 Determine which file is newer by comparing the versions or, if versions do not exist for both files, by comparing the timestamps.</p> <p>2 If the object that is already installed is newer than the one in the channel update, ask the user if the user wants to replace the object.</p>
n Compare files based on timestamps.	The files' timestamps determine which file is newer. If the file in the channel update is newer (if it has a later date), it replaces the one already installed on the endpoint.
n Compare files based on versions. If version doesn't exist, compare timestamps.	<p>The files' version numbers determine which file is newer.</p> <p>At update time:</p> <ul style="list-style-type: none"> n If the file in the channel update is newer (if it has the greater version number), it replaces the one already installed on the endpoint. n If a version number exists for the file in the channel update, but not for the file on the endpoint, the endpoint file is replaced. If no version number exists for the file in the channel update, but one exists for the file on the endpoint, the endpoint file is not replaced. n If neither file has a version number, timestamps are used. If the file in the channel update is newer (if it has a later date), it replaces the one already installed on the endpoint.
Update if file doesn't exist.	<p>1 Check if a file is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both files have versions, timestamps are ignored and versions are compared.)</p> <p>2 If the file is already installed, skip the file and do not update.</p> <p>3 If the file is not installed, check if a file with the same name already exists on the endpoint.</p> <p>4 If the file exists, skip the file and do not update. Otherwise, update the file.</p>

Table B-7: Update policies—Directories and files(Continued)

Policy	Actions taken for this policy
Update only if file exists.	<p>1 Check if a file is already installed on the endpoint by comparing its contents and attributes. The contents are compared by determining and then comparing the MD5 checksums. These attributes are compared: creation time, last modified time, and permission attributes. (If both files have versions, timestamps are ignored and versions are compared.)</p> <p>2 If the file is not installed, check if a file with the same name already exists on the endpoint.</p> <p>3 If a file with the same name is found on the endpoint, replace it by installing the file in the channel update. Otherwise, skip the file and do not update.</p>
Always update the file.	Always update the file, and overwrite the file if it is already installed on the endpoint.
Don't update the file.	Never update the file.

Uninstallation policies

Table B-8: Uninstallation policies—Directories and files

Policy	Actions taken for this policy
Uninstall if installed, prompt for shares.	<p>1 Check if the file was ever installed using Application Packager. If not, do not attempt to uninstall the file.</p> <p>2 Check if the file was already uninstalled from the endpoint by determining whether a file with that name and path exists.</p> <p>3 If the file is not already uninstalled, check if the file is shared.</p> <p>4 If it is shared, ask the user if the user wants to uninstall the file. Otherwise, uninstall the file by removing it or replacing it with an older version that Application Packager overwrote during installation or an update.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Uninstall if installed, no prompt.	<p>1 Check if the file was ever installed using Application Packager. If not, do not attempt to uninstall the file.</p> <p>2 Check if the file was already uninstalled from the endpoint by determining whether a file with that name and path exists.</p> <p>3 If the file is not already uninstalled, uninstall the file by removing it or replacing it with an older version that Application Packager overwrote during installation.</p> <p>For example, a file is part of a channel, but it was not installed at installation time because there was a newer file with the same name already installed on the endpoint. If this uninstallation policy is selected, that file is not uninstalled because it was never actually installed using Application Packager.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>

Table B-8: Uninstallation policies—Directories and files(Continued)

Policy	Actions taken for this policy
Always uninstall the file.	<p>Restore the file to its previous state (the state the file was in before Application Packager was used to install it):</p> <ul style="list-style-type: none"> <li data-bbox="534 290 1268 376">n If Application Packager overwrote an older version of the file with a newer version, this uninstallation policy causes the older version to be restored. <li data-bbox="534 387 1221 473">n If Application Packager installed the file because it did not previously exist, this uninstallation policy causes the file to be uninstalled and removed. <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Always remove the file.	<p>Whether or not Application Packager was used to install the file, uninstall and remove the file if it exists on the endpoint and is part of the channel.</p> <p>For example, a file is part of a channel, but it was not installed at installation time because there was a newer file with the same name already installed on the endpoint. If this uninstallation policy is selected, that file on the endpoint is uninstalled and removed even though it was never actually installed using Application Packager.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not removed unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Don't uninstall the file.	Never attempt to uninstall the file.

Verification policies

The policies described in Table B-9 on page 350 apply to these two types of verification policies:

- **Verify Installed**—Takes effect during the verify and repair phases for objects that were installed by a channel (created using Application Packager) at one point. An object can be installed during installation, update, or repair of a channel.

- **Verify Not Installed**—Takes effect during the verify and repair phases for objects that were never installed by a channel (created using Application Packager). For example, if a newer version of the file is already installed on the endpoint and there is an older version in the channel, the newer version is not overwritten by the older one if the installation policy Install if object is newer—Compare objects based on versions is selected.

The policies are checked in the following order, and the first policy that fails prevents the other policies from being checked:

- 1 If Inherit policy... is selected, the inherited policy takes precedence.
- 2 If Don't verify object during package verification is selected, the object is not checked.
- 3 Verify the existence of the object.
- 4 Verify contents of the object.
- 5 If Verify contents of the object fails, If the contents differ, allow Windows versions to increase is evaluated.
- 6 Verify the attributes of the object.

Table B-9: Verification policies—Directories and files

Policy	Actions taken for this policy
Verify the file using the following policies	
▪ Verify contents of the file	For files, check whether the contents of the file on the endpoint matches the one described in the manifest by comparing MD5 checksums. <ul style="list-style-type: none">▪ If the MD5 checksums do not match, verification fails.▪ If the file does not exist on the endpoint, no check is performed.

Table B-9: Verification policies—Directories and files(Continued)

Policy	Actions taken for this policy
n If the contents differ, allow Windows versions to increase.	<p>Using this policy prevents updated and new versions of files, such as DLL files (in Windows), from being overwritten during a repair operation, while still allowing you to repair files with other types of changes. This policy does not apply to all objects; it applies to files only.</p> <p>Verify a file as described in “Verify contents of the object”.</p> <p>Note: This policy is valid only if “Verify contents of the object” is selected and fails.</p> <p>If the MD5 checksums of the file on the endpoint and the one in the manifest do not match, and the Windows version has not changed or has decreased, verification fails.</p>
n Verify the attributes of the file.	<p>Using this policy allows you to check the attributes of a file. This policy does not apply to all objects; it applies to files only.</p> <p>For files, check whether the attributes of the file on the endpoint matches the attributes described in the manifest. The attributes checked include the last modified time, creation time, and the attribute flags.</p> <ul style="list-style-type: none"> n If the attributes do not match, verification fails. n If the file does not exist on the endpoint, no check is performed.
n Verify the existence of the file.	Check if the file exists on the endpoint. If it does not, verification fails.
Don't verify the file during package verification.	Never attempt to verify the file.

Note: You can now preserve the directory permissions even after uninstallation when you change the attributes manually. When you add the property "directories.preserveattributes=true" in the package directory, install the package, modify the installation directory, and uninstall the package, the Installation Directory Permissions are retained.

Repair policies

The repair policies are checked in the following order, and the first policy that fails prevents the other policies from being executed:

- 1 If Inherit policy... is selected, the inherited policy takes precedence.

- 2 If Do not repair object is selected, the object is not repaired.
- 3 Repair the contents of the object and Do not repair contents if versions increased.
- 4 Repair the attributes of the object.

Table B-10: Repair policies—Directories and files

Policy	Actions taken for this policy
Repair the file using the following policies	
n Repair the contents of the file	For files, the object on the endpoint has the same MD5 checksum as the one in the manifest after being repaired, unless “Do not repair contents if versions increased” is selected. If the MD5 checksums are already the same, no repair is done.
n Do not repair contents if versions increased	Using this policy prevents updated and new versions of files, such as DLL files (in Windows), from being overwritten during a repair operation, while still allowing you to repair objects with other types of changes. This policy does not apply to all objects; it applies to files only. If the MD5 checksum of the file on the endpoint does not match the one in the manifest, and the version of the file has increased when compared to the version in the manifest, the contents of the file are not repaired.
n Repair the attributes of the file.	Using this policy allows you to repair the attributes of a file. This policy does not apply to all objects; it applies to files only. For files, repair the file on the endpoint so that its attributes match the attributes described in the manifest. The attributes checked include the last modified time, creation time, and the attribute flags. If the attributes are already the same, no repair is done.
Do not repair the file.	Never attempt to repair the file.

Policies for INI files

This section lists the policies you can apply to the INI files in a channel. By default, INI files in the channel inherit the policies you set for the entire channel or from the policies of a parent directory. However, you can override those default policies by setting policies for individual INI files. For more information about setting these policies, see “Setting policies for individual items in a channel” on page 124.

The following tables describe the actions taken for each policy:

- Installation policies: Table B-11 on page 353
- Update policies: Table B-12 on page 353
- Uninstallation policies: Table B-13 on page 354
- Verify policies: Table B-14 on page 355
- Repair policies: Table B-15 on page 356

Installation policies

Table B-11: Installation policies—INI files

Policy	Actions taken for this policy
Always install sections and values.	Always install the sections and values in this file, and overwrite any existing sections and values if they already exist on the endpoint.
Allow duplicate values.	Allows the INI file to contain duplicate key-value pairs as described in “INI files that contain duplicate sections” on page 79.
Never install the sections and values.	Never install the sections and values in this file.

Update policies

Table B-12: Update policies—INI files

Policy	Actions taken for this policy
Always update sections and values.	Always update the sections and values in this file, and overwrite any existing sections and values if they already exist on the endpoint.
Allow duplicate values.	Allows the INI file to contain duplicate key-value pairs as described in “INI files that contain duplicate sections” on page 79.
Never update the sections and values.	Never update the sections and values in this file.

Uninstallation policies

Table B-13: Uninstallation policies—INI files

Policy	Actions taken for this policy
Uninstall if sections and values were installed.	<ol style="list-style-type: none"> 1 Check if the file was ever installed using Application Packager. If not, do not attempt to uninstall the file. 2 Check if the section to be uninstalled already exists in the file installed on the endpoint. 3 If the section exists in the file installed on the endpoint, uninstall the section and check if it overwrote any previous value that was there before: <ul style="list-style-type: none"> ▪ If yes, replace the current value with the previous value. ▪ If no, remove both the section and its corresponding value.
Always uninstall sections and values.	<p>Restore the file to its previous state (the state it was in before Application Packager was used to install any sections or values):</p> <ol style="list-style-type: none"> 1 Check if the section to be uninstalled already exists in the file installed on the endpoint. 2 If the section exists in the file installed on the endpoint, always uninstall the section and check if Application Packager overwrote any previous value that was there before: <ul style="list-style-type: none"> ▪ If yes, replace the current value with the previous value. ▪ If no, remove both the section and its corresponding value.
Always remove sections and values.	<p>Whether or not Application Packager was used to install any sections or values, uninstall and remove sections and values if they exist on the endpoint and are part of the channel.</p> <p>For example, INI file sections or values are part of a channel, but they were not installed at installation time because they already existed on the endpoint. If this uninstallation policy is selected, those sections and values on the endpoint are uninstalled and removed even though they were never actually installed using Application Packager.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not removed unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Don't uninstall sections and values.	Never uninstall any sections and values in this file.

Verification policies

The policies described in Table B-14 on page 355 apply to these two types of verification policies:

- **Verify Installed**—Takes effect during the verify and repair phases for objects that were installed by a channel (created using Application Packager) at one point. An object can be installed during installation, update, or repair of a channel.
- **Verify Not Installed**—Takes effect during the verify and repair phases for objects that were never installed by a channel (created using Application Packager). For example, if a newer version of the file is already installed on the endpoint and there is an older version in the channel, the newer version is not overwritten by the older one if the installation policy Install if object is newer—Compare objects based on versions is selected.

Table B-14: Verification policies—INI files

Policy	Actions taken for this policy
Verify sections and values using the following policies:	
Verify contents of the sections and values.	<ol style="list-style-type: none"> 1 Check if the file was ever installed using Application Packager. If not, do not attempt to verify the file. 2 Check if the section to be verified already exists in the file installed on the endpoint. If the section does not exist in the file installed on the endpoint, warn the user that the file has changed and is not verified. 3 If the section exists in the file installed on the endpoint, check if its value is the same: <ul style="list-style-type: none"> ▪ If the value is the same, consider the file verified. ▪ If the value is not the same, warn the user that the file has changed and is not verified.
Verify the existence of the INI file.	Verifies that the INI file exists on the endpoint.
Don't verify sections and values during package verification.	Never attempt to verify the file and the sections in it.

Repair policies

Table B-15: Repair policies—INI files

Policy	Actions taken for this policy
Repair sections and values using the following policies:	
Repair contents of the sections and values.	<ol style="list-style-type: none">1 Check if the file was ever installed using Application Packager. If not, do not attempt to repair the file.2 Check if the section to be repaired already exists in the file installed on the endpoint. If the section does not exist in the file installed on the endpoint, repair the file on the endpoint so that it matches the one in the packaged application.3 If the section exists in the file installed on the endpoint, check if its value is the same:<ul style="list-style-type: none">▀ If the value is the same, consider the file verified.▀ If the value is not the same, repair the file on the endpoint so that it matches the one in the packaged application.
Don't repair sections and values during package verification.	Never attempt to repair the file and the sections in it.

Policies for registry and metabase keys and values

This section lists the policies you can apply to the registry and IIS metabase keys and values in a channel. By default, registry and IIS metabase keys and values in the channel inherit the policies you set for the entire channel or from the policies of a parent key. However, you can override those default policies by setting policies for individual keys and values in the channel. For more information about setting these policies, see “Setting policies for individual items in a channel” on page 124.

The following tables describe the actions taken for each policy:

- Installation policies: Table B-16 on page 357
- Update policies: Table B-17 on page 358
- Uninstallation policies: Table B-18 on page 358
- Verify policies: Table B-19 on page 360
- Repair policies: Table B-20 on page 360

Note: When you set policies for container objects (such as directories, registry keys, and metabase keys) you are actually setting the policies for the child objects in them (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs). For example, when you set the installation policies for a directory, the name of the Install Policy tab is “Install policy for this directory’s contents.” Container objects get these policies only:

- Always install
- Always update
- Uninstall if possible (Container objects are not uninstalled unless they are empty of all child objects.)
- Always verify

Installation policies

Table B-16: Installation policies—Registry and metabase keys

Policy	Actions taken for this policy
Always install value.	Always install this key or value, and overwrite the existing value if the specified key is already on the endpoint.
Install if value doesn’t exist.	Check if the name of the key or value already exists under the specified key on the endpoint: <ul style="list-style-type: none"> ▀ If it exists, do not install the key or value. ▀ If it does not exist, install the key or value.
Install only if value exists.	Check if the name of the key or value already exists under the specified key on the endpoint: <ul style="list-style-type: none"> ▀ If it does not exist, do not install the key or value. ▀ If it exists, install the key or value.
Never install value.	Never install this key or value.

Update policies

Table B-17: Update policies—Registry and metabase keys

Policy	Actions taken for this policy
Always update value.	Always update this key or value, and overwrite the existing key or value if the specified key is already on the endpoint.
Update if value doesn't exist.	<p>Check if the name of the key or value already exists under the specified key on the endpoint:</p> <ul style="list-style-type: none"> ▀ If it exists, do not update the key or value. ▀ If it does not exist, install the key or value.
Update only if value exists.	<p>Check if the name of the key or value already exists under the specified key on the endpoint:</p> <ul style="list-style-type: none"> ▀ If it does not exist, do not install the key or value. ▀ If it exists, update the key or value.
Never update value.	Never update this key or value.

Uninstallation policies

Table B-18: Uninstallation policies—Registry and metabase keys

Policy	Actions taken for this policy
Uninstall if value was installed.	<p>Check if the key or value was installed using Application Packager:</p> <ul style="list-style-type: none"> ▀ If yes, uninstall the key or value either by removing it or by replacing it with an older one that Application Packager overwrote during a previous installation. ▀ If no, do not attempt to uninstall the key or value. <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>
Always uninstall value.	<p>Whether or not Application Packager was used to install the key or value, uninstall it if it exists on the endpoint.</p> <p>Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).</p>

Table B-18: Uninstallation policies—Registry and metabase keys(Continued)

Policy	Actions taken for this policy
Always remove the value.	Whether or not Application Packager was used to install the key or value, uninstall it if it exists on the endpoint. Note: Container objects (such as directories, registry keys, and metabase keys) are not uninstalled unless they are empty of all child objects (subdirectories/files, registry subkeys/value pairs, and metabase subkeys/value pairs).
Don't uninstall value.	Never attempt to uninstall this key or value.

Verification policies

The policies described in Table B-19 on page 360 apply to these two types of verification policies:

- **Verify Installed**—Takes effect during the verify and repair phases for objects that were installed by a channel (created using Application Packager) at one point. An object can be installed during installation, update, or repair of a channel.
- **Verify Not Installed**—Takes effect during the verify and repair phases for objects that were never installed by a channel (created using Application Packager). For example, if a newer version of the file is already installed on the endpoint and there is an older version in the channel, the newer version is not overwritten by the older one if the installation policy Install if object is newer—Compare objects based on versions is selected.

The policies are checked in the following order, and the first policy that fails prevents the other policies from being checked:

- 1 If Inherit policy... is selected, the inherited policy takes precedence.
- 2 If Don't verify object during package verification is selected, the object is not checked.
- 3 Verify the existence of the object.
- 4 Verify contents of the object.
- 5 If Verify contents of the object fails, If the contents differ, allow Windows versions to increase is evaluated.

6 Verify the attributes of the object.

Table B-19: Verification policies—Registry and metabase keys

Policy	Actions taken for this policy
Verify the value using the following policies	
n Verify the value	<ol style="list-style-type: none"> 1 Check if the key or value was ever installed using Application Packager. If not, do not attempt to verify the key or value. 2 Check if the key or value is already installed on the endpoint by determining if the key name and value are the same as what Application Packager previously installed. 3 If the key or value is already installed, consider it verified. Otherwise, warn the user that the key or value has changed and is not verified.
n Verify the existence of the value.	Check if the value exists on the endpoint. If it does not, verification fails.
Don't verify the value during package verification.	Never attempt to verify this key or value.

Repair policies

The repair policies are checked in the following order, and the first policy that fails prevents the other policies from being executed:

- 1 If Inherit policy... is selected, the inherited policy takes precedence.
- 2 If Do not repair object is selected, the object is not repaired.
- 3 Repair the contents of the object and Do not repair contents if versions increased.
- 4 Repair the attributes of the object.

Table B-20: Repair policies—Registry and metabase keys

Policy	Actions taken for this policy
Repair the value using the following policies	
n Repair the value	Repair the value so that it matches what Application Packager previously installed.
Do not repair the value.	Never attempt to repair the key or value.

Policies for Windows NT services

This section lists the installation and uninstallation policies for Windows NT services. In Windows NT and Windows 2000, NT services are programs or processes that perform a specific system function, usually as background tasks, to support other programs. Typically, they do not require user interaction and do not have a user interface. Some Windows applications include and install services. If you are familiar with services and want to manage services for the applications you are packaging, you can use the Services page in the Package Editor to edit, add, and remove services.

By default, NT services inherit the policies that are set for the entire channel. For more information, see “Setting default policies for channels” on page 123.

Using the Package Editor, you can override those default policies and set installation and uninstallation policies for the individual NT services. For more information about setting installation and uninstallation policies for NT services, see “Editing services” on page 152.

The following tables describe the actions taken for each policy:

- Installation policies: Table B-21 on page 362
- Uninstallation policies: Table B-22 on page 363
- Verify policies: Table B-24 on page 365
- Repair policies: Table B-25 on page 366

Installation policies

Table B-21: Installation policies—Windows NT services

Policy	Actions taken for this policy
Always install service	<p>Determine if the service is already installed by using native Windows APIs:</p> <ul style="list-style-type: none"> ▀ If the service is not yet installed, install it. ▀ If the service is already installed on the endpoint but the attributes of the service are different, install the service in the channel. <p>Note: The service in the channel cannot be installed on top of a previously installed service that is currently running on the endpoint. You can stop the service before installing the channel; otherwise, the service is installed at the next system reboot.</p>
Install service if it doesn't exist	<ol style="list-style-type: none"> 1 Determine if the service is already installed. 2 If the service is not yet installed, check if the service exists on the endpoint. 3 If a service with the same name is found, skip the service and do not install. Otherwise, install the service.
Install service only if it exists	<ol style="list-style-type: none"> 1 Determine if the service is already installed. 2 If the service is not yet installed, check if the service exists on the endpoint. If a service with the same name is <i>not</i> found, skip the service and do not install. Otherwise, install the service. <p>Note: The service in the channel cannot be installed on top of a previously installed service that is currently running on the endpoint. You can stop the service before installing the channel; otherwise, the service is installed at the next system reboot.</p>

Uninstallation policies

Table B-22: Uninstallation policies—Windows NT services

Policy	Actions taken for this policy
Uninstall an installed service	<p>1 Determine if Application Packager was used to install the service. If not, do not attempt to uninstall the service.</p> <p>2 Determine if the service is already uninstalled from the endpoint by using native Windows APIs. If the service is not already uninstalled, uninstall it by removing the service, or by replacing it with an older version that was overwritten during installation.</p> <p>Note: The service on the endpoint cannot be uninstalled if it is currently running. You can stop the service before uninstalling the channel; otherwise, the service is uninstalled at the next system reboot.</p>
Always uninstall service	This is currently the same as “Uninstall an installed service” because uninstallation is attempted only if Application Packager was used to install the service.
Always remove the service	<p>Whether or not Application Packager was used to install the service, uninstall and remove the service if it exists on the endpoint and is part of the channel.</p> <p>For example, a service is part of a channel, but it was not installed at installation time because there was a newer service with the same name already installed on the endpoint. If this uninstallation policy is selected, that service on the endpoint is uninstalled and removed even though it was never actually installed using Application Packager.</p>
Don't uninstall service	Skip the service and do not uninstall during uninstallation.

Update policies

Table B-23: Update policies—Windows NT services

Policy	Actions taken for this policy
Always update the service	<p>Determine if the service is already updated by using native Windows APIs:</p> <ul style="list-style-type: none"> ▀ If the service is not yet installed, install it. ▀ If the service is already installed on the endpoint but the attributes of the service are different, update the service in the channel. <p>Note: The service in the channel cannot be updated on top of a previously installed service that is currently running on the endpoint. You can stop the service before updating the channel; otherwise, the service is updated at the next system reboot.</p>
Update service if it doesn't exist	<ol style="list-style-type: none"> 1 Determine if the service is already installed. 2 If the service is not yet installed, check if the service exists on the endpoint. 3 If a service with the same name is found, skip the service and do not update. Otherwise, install the updated service.
Update service only if it exists	<ol style="list-style-type: none"> 1 Determine if the service is already installed. 2 If the service is not yet installed, check if the service exists on the endpoint. If a service with the same name is <i>not</i> found, skip the service and do not update. Otherwise, update the service. <p>Note: The service in the channel cannot be updated on top of a previously installed service that is currently running on the endpoint. You can stop the service before updating the channel; otherwise, the service is updated at the next system reboot.</p>
Never update the service	Never attempt to update the service.

Verification policies

The policies described in Table B-9 on page 350 apply to these two types of verification policies:

- ▀ **Verify Installed**—Takes effect during the verify and repair phases for objects that were installed by a channel (created using Application Packager) at one point. An object can be installed during installation, update, or repair of a channel.

- **Verify Not Installed**—Takes effect during the verify and repair phases for objects that were never installed by a channel (created using Application Packager). For example, if a newer version of the file is already installed on the endpoint and there is an older version in the channel, the newer version is not overwritten by the older one if the installation policy Install if object is newer—Compare objects based on versions is selected.

The policies are checked in the following order, and the first policy that fails prevents the other policies from being checked:

- 1 If Inherit policy... is selected, the inherited policy takes precedence.
- 2 If Don't verify object during package verification is selected, the object is not checked.
- 3 Verify the existence of the object.
- 4 Verify contents of the object.
- 5 If Verify contents of the object fails, If the contents differ, allow Windows versions to increase is evaluated.
- 6 Verify the attributes of the object.

Table B-24: Verification policies—Windows NT services

Policy	Actions taken for this policy
Verify the service using the following policies	
n Verify contents of the service	Check whether the contents of the service on the endpoint match those described in the manifest by comparing MD5 checksums. <ul style="list-style-type: none"> n If the MD5 checksums do not match, verification fails. n If the service does not exist on the endpoint, no check is performed.
n Verify the existence of the service	Check if the service exists on the endpoint. If it does not, verification fails.
Never verify service	Never attempt to verify the service.

Repair policies

The repair policies are checked in the following order, and the first policy that fails prevents the other policies from being executed:

- 1 If Inherit policy... is selected, the inherited policy takes precedence.
- 2 If Do not repair object is selected, the object is not repaired.

- 3 Repair the contents of the object and Do not repair contents if versions increased.
- 4 Repair the attributes of the object.

Table B-25: Repair policies—Windows NT services

Policy	Actions taken for this policy
Repair the service using the following policies	
Repair the contents of the service	Repair the service so that it matches what Application Packager previously installed.
Never repair the service.	Never attempt to repair the service.

Policies for text modifier groups and text modifiers

This section lists the installation and update policies for text modifier groups and text modifiers. You can use text modifiers to insert and replace text in ASCII text files that are part of your packaged application or that are already installed on the endpoint. A text modifier group contains all the text modifiers that affect one ASCII text file.

Using the Package Editor, you can set installation and update policies for text modifiers at two levels:

- You can set the installation and update policies for the text modifier group. By default, any text modifiers in the group inherit the group's installation and update policies.
- You can also override the text modifier group policies and set installation and update policies for individual text modifiers.

For more information about setting installation and update policies at both these levels, see “Setting installation and update policies for text modifiers” on page 181.

The following tables describe the actions taken for each policy:

- Installation policies: Table B-26 on page 367
- Update policies: Table B-27 on page 367

Installation policies

Table B-26: Installation policies—Text modifiers

Policy	Actions taken for this policy
Always install modifier group.	Always install the specified text modifier or all the text modifiers in the specified group.
Always install modifier.	
Never install modifier group.	Never install the specified text modifier or all the text modifiers in the specified group.
Never install modifier.	

Update policies

Table B-27: Update policies—Text modifiers

Policy	Actions taken for this policy
Update if changed.	<ol style="list-style-type: none"> 1 Check if the text modifier or the text modifier group in the channel update is the same as what was previously installed. If it is the same, the update is skipped. 2 If they are not the same, install the changed text modifier or text modifier group.
Always reapply this modifier group.	Always install the specified text modifier or the text modifiers in the specified group, even if they have not changed from the previous installation.
Always reapply this modifier.	
Never update modifier group.	Never update the specified text modifier or all the text modifiers in the specified group.
Never update modifier.	

Appendix C XML syntax

This section describes the syntax for Application Packager's XML template files.

The following topics are provided:

- Overview (page 370)
- Generic tags (page 370)
- Non-MSI tags (page 400)
- MSI tags (page 414)
- Search and modify tags (page 418)
- Snapshot tags (page 428)

Overview

This section describes the syntax for Application Packager’s XML template files.

The beginning of each section shows the hierarchy of XML tags.

Generic tags

Generic tags apply to all channels created using Application Packager.

This section describes the following generic tags:

“<PACKAGE>”
“<CUSTOMMACRO>”
“<MACRO>”
“<ENVIRONMENT>”
“<ENV_PROPERTY>”
“<SERVICES>”
“<SERVICE>”
“<SERV_GENERAL>”
“<SERV_SECURITY>”
“<SERV_ADVANCED>”
“<SERV_DEPEND>”
“<SERV_POLICIES>”
“<CONFIGURATION>”
“<PLATFORM>”
“<INSTALLER>”
“<REBOOT>”
“<POLICIES>”
“<STARTUP>”
“<VERIFYREPAIR>”
“<DEPENDENCIES>”

```
“<DEPEND_MEMORY>”
“<DEPEND_PROCESSOR>”
“<DEPEND_RESOLUTION>”
“<DEPEND_CONNECTION_SPEED>”
“<DEPEND_DISKSPACE>”
    “<DEPEND_DRIVE>”
“<DEPEND_FILES>”
    “<DEPEND_FILE>”
“<DEPEND_REGISTRY>”
    “<DEPEND_REGKEY>”
“<DEPEND_CHANNELS>”
    “<DEPEND_CHANNEL>”
“<DEPEND_ENVIRONMENT_VARS>”
    “<DEPEND_ENVIRONMENT_VAR>”
“<CUSTOMIZATION>”
    “<SCRIPTS>”
        “<SCRIPT>”
“<PROPERTY>”
“<PARAMETER>”
```

<DEPEND_CONNECTION_SPEED>

Specifies the required connection speed for the channel.

Syntax

```
<DEPEND_CONNECTION_SPEED
op="|=|>|<|=|>|=|
speed="required connection speed"
/>
```

Attributes

- op—An operator (=, <>, <, <=, >, or >=).
- speed—A float or integer specifying the required connection speed(in kbps).

Example

```
<DEPEND_CONNECTION_SPEED speed="500" op=">=" />
```

<PACKAGE>

Defines a channel created using Application Packager. The “<PACKAGE>” tag encloses all the other tags for the channel.

Syntax

```
<PACKAGE  
name="name for the channel"  
>  
    <!--child tags -->  
</PACKAGE>
```

Attributes

- name—A string representing the name of the channel.

<CUSTOMMACRO>

Contains the user-defined macros in a channel.

Syntax

```
<CUSTOMMACRO>  
    <!-- child macro tags -->  
</CUSTOMMACRO>
```

Attributes

None.

Example

```
<CUSTOMMACRO>  
    <MACRO definition="C:\MyPrograms" query="Install the program in  
which directory?" name="$InstallDir" type="user"/>  
    <MACRO definition="machine1" query="Computer name?"  
name="$ComputerName" type="user"/>  
</CUSTOMMACRO>
```

<MACRO>

Defines a macro in a channel.

Syntax

```
<MACRO
    type = "user|registry|environment|package|tuner|msi"
    name = "the name of the macro"
    definition = "the default macro definition"
    registrykey = "registry key name - valid only if type is registry"
    registrykeyvalue = "registry key value name - valid only if type
is registry"
    query = "optional query user for definition prompt description"
/>
```

Attributes

- **type**—A string that indicates the type of macro. The following types are available:
 - **user**—A user-defined macro with the default user namespace `$name`. Macros in this namespace cannot automatically resolve themselves. They either have default values, or they must be resolved by the user at runtime.
 - **registry**—A registry value macro with the namespace `$REG.name`. It expects a second macro called `$name` (in the default namespace) to exist. This macro should contain a complete path to a registry value.
 - **environment**—An environment variable macro with the namespace `$ENV.name`. It resolves to the value of the environment variable `name`.
 - **package**—A packaged application property macro with the namespace `$APP.name`. It resolves to the value of the application property `name`.
 - **tuner**—A tuner property macro with the namespace `$TUNER.name`. It resolves to the value of the tuner property `name`.
 - **msi**—A Windows Installer property macro with the namespace `$MSI.name`. It resolves to the value of the Windows Installer property `name`.
 - **name**—A string representing the name of the macro.

Note: Macros are case-sensitive.

- **definition**—A string representing the default definition for the macro. The default definition is used if the macro is not resolved during installation at the endpoint.

- **registrykey**—Registry key name for a registry value macro. This attribute is valid only if the macro type is registry.
- **registrykeyvalue**—Registry key value name for a registry value macro. This attribute is valid only if the macro type is registry.
- **query**—Optional. A string representing the question you want to ask endpoint users when prompting them for the macro definition.

Example

```
<MACRO definition="C:\MyPrograms" query="Install the program in which
directory?" name="$InstallDir" type="user"/>
```

<ENVIRONMENT>

Contains the environment variables in a channel.

Syntax

```
<ENVIRONMENT>
  <!--child variable tags -->
</ENVIRONMENT>
```

Attributes

None.

Example

```
<ENVIRONMENT>
  <ENV_PROPERTY value="var" segment="win9x" name="TEST ENV VAR"/>
</ENVIRONMENT>
```

<ENV_PROPERTY>

Defines an environment variable in a channel.

Syntax

```
<ENV_PROPERTY
  segment="win9x|winnt"
  type="system|user"
  name="name of environment variable"
  value="value of environment variable"
/>
```

Attributes

- **segment**—A string representing the platforms for which the environment variable is valid. The following platforms are available:
 - **win9x**—Windows 95 and Windows 98

- **winnt**—Windows NT and Windows 2000
- **type**—A string representing the type of environment variable. This attribute is valid only if the attribute segment is **winnt**. The following types are available:
 - **system**—A system environment variable that applies to the system, and thus to all users of the system.
 - **user**—A user environment variable that is different for each user of a particular computer. The variables include any that are set by the user, as well as any variables defined by applications, such as the path to the location of the application files.
 - **name**—A string representing the name of the environment variable.
 - **value**—A string representing the value of the environment variable.

Example

```
<ENV_PROPERTY value="var" segment="win9x" name="TEST ENV VAR"/>
```

<SERVICES>

Contains the Windows NT or Windows 2000 services in a channel.

Syntax

```
<SERVICES>
  <!--child service tags -->
</SERVICES>
```

Attributes

None.

Example

```
<SERVICES>
  <SERVICE/>
</SERVICES>
```

<SERVICE>

Defines a Windows NT or Windows 2000 service in a channel.

Syntax

```
<SERVICE>
  <!--child service tags -->
</SERVICE>
```

Attributes

None.

Example

```
<SERVICE>
    <!--child service tags -->
</SERVICE>
```

<SERV_GENERAL>

Defines the general properties of a Windows NT or Windows 2000 service in a channel. This contains the same information that appears in the “General properties page” tab of the Package Editor’s NT Service dialog box.

Syntax

```
<SERV_GENERAL
    servicename="the name of the service"
    displayname="the displayed name of the service"
    servicepath="path of the program to run"
    type="kernel_driver|filesystem_driver|own_process|share_process"
    start="boot|system|automatic|demand|disabled"
    errorcontrol="ignore|normal|severe|critical"
/>
```

Attributes

- **servicename**—A string representing the name of the Windows NT or Windows 2000 service.
- **displayname**—A string specifying the name for the service that is displayed in the Services Control Panel.
- **servicepath**—Name and path of the executable associated with the service.
- **type**—A string specifying whether the service works on hardware (Kernel driver) or on the operating system (File system driver). It can also indicate whether the service runs as its Own process or as a Shared process. The following types are available:
`kernel_driver|filesystem_driver|own_process|share_process`
- **start**—A string specifying when the service is configured to start. The following start options are available:
 - **boot**—Starts when the computer is booted up. This usually applies to services for pieces of hardware that are required for the computer to function, such as disk drives.

- system—Starts when the operating system starts. This usually applies to services that are critical for the operation of the operating system, such as display drivers.
- automatic—Starts when the operating system starts, like system services, but is not crucial to the operation of the operating system.
- demand—Can be started by a user or a dependent service. The service does not start automatically when the operating system starts.
- disabled—Cannot be started by a user or a dependent service.
- errorcontrol—A string specifying the service’s level of error control. The following levels of control are available:
 - critical—Does not start up the system.
 - severe—Switches to the Last Known Good setup, or, if already using that setup, continues.
 - normal—Continues system startup, but displays an error message stating that the service did not start.
 - ignore—Does not stop the system or display an error message, but just skips the service.

Example

```
<SERV_GENERAL errorcontrol="normal" displayname="Test Service Display Name" type="own_process" servicepath="TestServiceProgram" start="demand" servicename="Test Service"/>
```

[**<SERV_SECURITY>**](#)

Defines the security properties of a Windows NT or Windows 2000 service in a channel. This contains the same information that appears in the “Security properties page” tab of the Package Editor’s NT Service dialog box.

Syntax

```
<SERV_SECURITY
  account="system|user"
  desktop="true|false"
  accountname="the name of the user account"
  password="the password associated with accountname"
/>
```

Attributes

- account—A string specifying whether the service logs in using the system account or the user account.

- desktop—A boolean value (`true` or `false`) specifying whether the service provides a user interface that can be used by the logged on user when the service is started.
- accountname—A string specifying the name of a user account for the service to use. Although most services log into the system account, some services can be configured to log into specific user accounts so that the user can have access to resources such as files and directories protected by Windows. This attribute is valid only if the `account` attribute is `user`.
- password—A string specifying the corresponding password for the user account specified in the `accountname` attribute.

Example

```
<SERV_SECURITY desktop="false" account="system"/>
```

<SERV_ADVANCED>

Defines the advanced properties of a Windows NT or Windows 2000 service in a channel. This contains the same information that appears in the “Advanced properties page” tab of the Package Editor’s NT Service dialog box.

Syntax

```
<SERV_ADVANCED  
    groupname="name of the group this service belongs to"  
/>
```

Attributes

- groupname—A string specifying the group a service is associated with. The group usually determines the order in which the services start.

Example

```
<SERV_ADVANCED group="name of the group this service belongs to"/>
```

<SERV_DEPEND>

Specifies the groups and services on which a particular service depends. If a service is stopped or is not running properly, other services that depend on that service can be affected.

Syntax

```
<SERV_DEPEND
    type="GROUP|SERVICE"
    name="the name of the group or service this service depends on"
/>
```

Attributes

- type—A string specifying whether the service depends on a group (group) or another service (service).
- name—A string specifying the name of the group or service on which this service depends.

Example

```
<SERV_DEPEND type="group" name="the name of the group or service this
service depends on"/>
```

<SERV_POLICIES>

Defines the policies for a Windows NT or Windows 2000 service in a channel. This contains the same information that appears in the “Install policies page” and “Uninstall policies page” tabs of the Package Editor’s NT Service dialog box.

Syntax

```
<SERV_POLICIES
    installpol="inherit|always|exists|notexist|never"
    uninstallpol="remove|inherit|always|installed|never"
/>
```

Attributes

- installpol—A string specifying the installation policies for a service. The available policies are: inherit, always, exists, notexist, and never. For more information about each policy, see “Policies for Windows NT services” on page 361.
- uninstallpol—A string specifying the uninstallation policies for a service. The available policies are: remove, inherit, always, installed, and never. For more information about each policy, see “Policies for Windows NT services” on page 361.

Example

```
<SERV_POLICIES installpol="inherit|always|exists|notexist|never"
uninstallpol="inherit|always|installed|never"/>
```

<CONFIGURATION>

Contains the following tags:

- “<PLATFORM>”
- “<INSTALLER>”
- “<POLICIES>”
- “<STARTUP>”
- “<VERIFYREPAIR>”
- “<DEPENDENCIES>”
- “<CUSTOMIZATION>”

Syntax

```
<CONFIGURATION>
    <!-- child tags -->
</CONFIGURATION>
```

Attributes

None.

<PLATFORM>

Specifies the platforms for which the channel is valid.

Syntax

```
<PLATFORM
    segment.win="WIN9X|WINNT|WIN2K|WINXP|WINVISTA|WINANY"
    segment.unix="LINUX|SCO|OSF1"
/>
```

Attributes

- arch—A string representing the architecture for the UNIX platform.
- segment.win—A string representing the Windows platforms for which the channel is valid. You can specify the following platforms: WIN9X, WINNT, WIN2K, WINXP, WINVISTA, and WINANY.
- segment.unix—A string representing the UNIX platforms for which the channel is valid. You can specify the following platforms: LINUX, SCO, and OSF1.
- segmentation.hierarchy – A string which represents the segmentation hierarchy for the package. The valid values are default and flat.

Example

```
<PLATFORM segment.win="WINNT"/>
```

<INSTALLER>

Specifies the installation options for the channel.

Syntax

```
<INSTALLER
    guimode="full|semisilent|silent"
    preview="true|false"
    rollback="true|false"
    nobackup="true|false"
    delayfiledownload="true|false"
    updateinstall="true|false"
    loggingrollover="hourly|daily|weekly|monthly|never|an integer
representing the size of the log file in kilobytes"
    multiuser="true|false"
/>
```

Attributes

- **guimode**—A string representing the graphical user interface (GUI) mode in which the channel runs when installing the packaged application. You can specify the following modes:
 - **full**—Select this mode if you want an interactive installation that prompts the user during the process. This mode displays any dialog boxes or progress bars to the user during installation.
 - **semisilent**—Select this mode if you do not want to display any dialog boxes to the user during installation. This mode displays installation progress bars for the user.
 - **silent**—Select this mode if you do not want to display any dialog boxes or progress bars to the user during installation. This mode uses all of the default channel settings during installation.
- **preview**—A boolean value (true or false) specifying if you want to create a channel that simulates the installation process without installing any files. This option can be used for testing purposes.

- **rollback**—A boolean value (`true` or `false`) specifying if you want the channel to automatically roll back to the previous state if installation fails. If installation fails when a channel is being installed for the first time, it automatically rolls back the channel to the uninstalled state. Files and other items in the channel that were installed are uninstalled, while items that were removed or modified during installation are restored. If an update to a channel fails, it automatically rolls back the channel to its state before the update.
- **nobackup**—A boolean value (`true` or `false`) specifying if you do not want to store any directories, files, registry entries, and metabase entries for backup during an installation. Usually, when a channel is installed, existing directories, files, registry entries, and metabase entries that are deleted or overwritten are stored in a backup directory (the `unfiles` directory of the channel in the tuner's workspace directory). These are used to restore the system to its previous state when the channel is uninstalled. If you specify `true`, those directories, files, registry entries, and metabase entries are not restored if the channel is uninstalled.

This option might be useful if you want to make sure that an application does not reside on endpoints after a channel is uninstalled. It also allows you to save some disk space.

WARNING: Selecting this option might cause the removal of directories, files, registry entries, and metabase entries that are critical to other applications.

- **delayfiledownload**—A boolean value (`true` or `false`) specifying if you want to delay the download of a channel's files to the endpoints until any pre- scripts have run. If you specify `true`, the channel download occurs in two parts. The initial download includes only the installer and the manifest file (`manifest.ncp`), which contains the code and instructions required to install and update the application. After any prescripts have run, the second download brings over the files necessary to install or update the application.

This option might be useful if you use a preinstall script to run some commands or check that some conditions are met before installing an application, you might want to delay the download so that time and bandwidth are not wasted if the commands fail or if the conditions are not met. You might also want to use this option with the `preload` parameter (see “Minimizing bandwidth usage” on page 159). Using these together allows you to prevent files that are already on the endpoint from being unnecessarily downloaded again.

- `updateinstall`—A boolean value (`true` or `false`) specifying if you want to install the channel automatically after it is updated. If you specify `true`, the channel automatically installs after it is updated. If you specify `false`, the channel is in the install-pending state after it is updated.
- `loggingrollover`—A string value or integer specifying how often the log file for the channel should roll over. You can specify this based on time and date (hourly, daily, weekly, monthly, or never), or based on the size in kilobytes of the log file by specifying an integer.
- `multiuser`—A boolean value (`true` or `false`) specifying if you want to enable the feature that allows you to package and distribute applications that include user-specific file and registry entries to a machine that is used by multiple local users.

Example

```
<INSTALLER updateinstall="true" rollback="false" loggingrollover="32"
delayfiledownload="false" preview="false" guimode="full"
nobackup="false" multiuser="false"/>
```

<REBOOT>

Specifies the reboot options for installing the packaged application in the channel. This option applies only to tuners prior to version 8.0. Tuner of version 8.0 and higher use the Common Reboot Service which can be configured on the Profile tab of Setup and Deployment.

Syntax

```
<REBOOT
    type="always|allow|disallow"
    prompt="true|false"
    allowCancel="true|false"
    force="true|false"
/>
```

Attributes

- **type**—A string value (`always`, `allow`, or `disallow`) specifying whether you want to allow the channel to determine if a system reboot is required.
- **prompt**—A boolean value (`true` or `false`) specifying if you want to prompt users if a system reboot is required. This attribute is valid only if `type` is `allow`.
- **allowCancel**—A boolean value (`true` or `false`) specifying if you want to inform users of the required system reboot without actually rebooting the machine for them. Users see a dialog box that indicates a reboot is required, but the reboot does not take place automatically; it waits for users to click OK. (The dialog box does not contain a Cancel button.)
- **force**—A boolean value (`true` or `false`) specifying if you want to force the system to automatically reboot after installation. This attribute is valid only if `type` is `allow`.

Example

```
<REBOOT force="false" prompt="true" type="allow"/>
```

<POLICIES>

Specifies the default installation, update, uninstallation, and verification policies for the objects in the channel.

Syntax

```
<POLICIES  
installpol="newer_version_prompt|newer_version_noprompt|  
notexist|exists|always|newer_timestamp_version_prompt|  
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|  
newer_timestamp_noprompt"  
updatepol="newer_version_prompt|newer_version_noprompt|  
notexist|exists|always|newer_timestamp_version_prompt|  
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|  
newer_timestamp_noprompt"  
uninstallpol="remove|installed_prompt|always|installed|never"  
verifypol="content|attributes|version_increase|exists|never"  
verifynipol="content|attributes|version_increase|exists|never"  
repairpol="content|attributes|version_increase|never"  
postpone.allow="true|false"  
postpone.type="date|attempts"  
postpone.maxDate="date in mm/dd/yyyy hh:mm a format"  
postpone.maxAttempts="valid integer specifying maximum number of  
attempts to allow postpone"  
postpone.msg="custom message to be displayed to user in postpone  
dialog"  
/>
```

Attributes

For more information about each policy, see “Policies for the entire channel” on page 332.

- `installpol`—A string specifying the default installation policies for the channel.
- `updatepol`—A string specifying the default update policies for the channel.
- `uninstallpol`—A string specifying the default uninstallation policies for the channel.
- `verifypol`—A string specifying the default verification policies for the channel.
- `verifynipol`—A string specifying the default verification policies for objects in the channel that were not installed using Application Packager.
- `repairpol`—A string specifying the default repair policies for the channel.
- `postpone.allow` – specifies whether the user is allowed to postpone installation of the package.
- `postpone.type` – A string specifying whether the user should be allowed to postpone installation until a maximum time limit or until maximum number of attempts is reached. Possible values: date, attempts
- `postpone.maxDate` – A date string specifying the maximum time limit until which the user is allowed to postpone the installation. Format: mm/dd/yyyy hh:mm a
- `postpone.maxAttempts` – A integer specifying the maximum number of times user is allowed to postpone installation
- `postpone.msg` – A string specifying any custom message to be displayed in the postpone dialog to user.

Example

```
<POLICIES updatepol="always" installpol="newer_version_prompt"
uninstallpol="installed_prompt" verifypol="verify"
postpone.allow="true" postpone.type="date" postpone.maxdate="11/10/
2014 09:44 AM" postpone.msg="This software can be installed at your
convenient time" />
```

<STARTUP>

Specifies the options for starting the packaged application.

Syntax

```
<STARTUP
    runexe="true|false"
    runuser="true|false"
    launch="true|false"
    launchpath="path of executable to run from the tuner"
    launchargs="command-line arguments to pass to the executable"
    workingdir="working directory of executable"
    execmode="detach|tuner"
    exectime="seconds before forced termination"
/>
```

Attributes

- **runexe**—A boolean value (true or false) specifying if you want to allow users to execute *any* application using the command-line option -runexe *<application_name>*

where:

<application_name> includes the full path of the application executable or batch file. If you specify false, you can use the command-line option -runexe without any arguments to start only the application with the executable file specified in the attribute launchpath.

- **runuser**—A boolean value (true or false) specifying if you want the application (or scripts) in this channel to run in the user's context instead of the tuner's context.

This option might be useful when running the tuner as a service. By default, applications started using the tuner inherit the tuner's context. This might cause problems when the tuner is running as a service because the application is not running in the logged-on user's context (and is missing the user's security and identity settings).

- **launch**—A boolean value (true or false) specifying if you want to start or launch the application from the tuner. If the attribute launch is set to true, you can use the following attributes:

- **launchpath**—A string representing the name and path of the executable file that starts the application. You can include a macro name in the path if necessary, for example, \$macro_name\$.
- **launchargs**—A string representing any command-line arguments you want the application to use when you start it, for example, starting in a minimized state.
- **workingdir**—A string representing the name and path of a working directory for the application, if necessary.

- `execmode`—A string indicating whether you want the application to run independently from the tuner after it is launched. If you specify `detach`, the application can continue to run even if the tuner is no longer running. If you specify `tuner`, the tuner manages the application, including exiting the application.
- `execetime`—An integer specifying the number of seconds you want the tuner to wait before forcing the application to exit. This attribute is valid only if `launch` is `true` and `execmode` is `tuner`.

Example

```
<STARTUP runuser="false" runexe="false"/>
```

<VERIFYREPAIR>

Specifies the options for automatically verifying and repairing the channel.

Syntax

```
<VERIFYREPAIR
    repair="true|false"
    schedule="schedule string"
/>
```

Attributes

- `repair`—A boolean value (`true` or `false`) specifying if you want to make any necessary repairs to a channel at the same time that verification is scheduled.
- `schedule`—A string specifying the schedule for verifying and repairing the channel. For details and examples, see “Schedule string” on page 387.

Schedule string

A schedule string entered from the command line can have any of the following forms:

```
every <number> days update <when>
every <number> weeks on <day_of_week> update <when>
weekdays update <when>
never
```

where:

<number>

is an integer. Even if the number is 1, the word following it in the syntax must remain plural, for example, 1 days, not 1 day.

<day_of_week>

is mon, tue, wed, thu, fri, sat, or sun (lowercase and with no punctuation). To specify more than one day of the week, use a plus sign, for example, mon+tue+fri.

<when>

specifies the exact time of day or time interval at which to perform an update.

It can be either of the following strings:

at <time>

every <number> minutes | hours [between <time> and <time>]

where:

<time> is <hour>:<minute>{am|pm} (with no space before am or pm). Precede single-digit hours with a zero (for example, 01:30pm). For midnight, use 12:00am, and for noon, use 12:00pm. If you do not specify a between interval, all day (between 12:00am and 11:59pm) is assumed.

These are examples of schedule strings:

```
every 2 days update at 12:00am  
every 3 weeks on mon+sat update every 4 hours between 06:00am and  
10:00pm  
every 2 weeks on mon update at 04:00am
```

Example

```
<VERIFYREPAIR repair="true" schedule="every 2 weeks on mon update at  
04:00AM"/>
```

<DEPENDENCIES>

Contains the dependencies or requirements for the channel.

Syntax

```
<DEPENDENCIES  
  runonupdate="true|false"  
  <!-- child dependency tags -->  
</DEPENDENCIES>
```

Attributes

- runonupdate—A boolean value (true or false) specifying if you want dependencies to be checked when updating channels

Example

```
<DEPENDENCIES runonupdate="true">  
  <DEPEND_MEMORY value="32" op="&gt;=/">  
  <DEPEND_PROCESSOR op="&gt;=" type="p"/>  
  <DEPEND_RESOLUTION value="800" op="&gt;=/">
```

```

<DEPEND_DISKSPACE>
    <DEPEND_DRIVE mb="32" drive="C"/>
</DEPEND_DISKSPACE>
<DEPEND_FILES>
    <DEPEND_FILE sizeop="&gt;=" op="exists" dateop="&gt;=" path="G:\My Folder\test.txt" isOR="false" sizevalue="100" date="09-21-2001"/>
    <DEPEND_FILE op="notexist" isOR="true" path="G:\My Folder\training.txt"/>
</DEPEND_FILES>
<DEPEND_REGISTRY>
    <DEPEND_REGKEY op="notexist" key="HKEY_CURRENT_CONFIG\Software\FONTs" isOR="false"/>
</DEPEND_REGISTRY>
<DEPEND_CHANNELS>
    <DEPEND_CHANNEL op="exists" url="http://products/Software_Distribution/462/ApplicationPackager" isOR="false"/>
</DEPEND_CHANNELS>
</DEPENDENCIES>

```

<DEPEND_MEMORY>

Specifies the required amount of system memory (in megabytes) for the channel.

Syntax

```

<DEPEND_MEMORY
    op="="|<>|<|=|>|="
    value="required amount of memory in megabytes"
/>

```

Attributes

- **op**—An operator (=, <>, <, <=, >, or >=).
- **value**—An integer specifying the required amount of system memory (in megabytes).

Example

```
<DEPEND_MEMORY value="32" op=">=" />
```

<DEPEND_PROCESSOR>

Specifies the required type of processor for the channel.

Syntax

```

<DEPEND_PROCESSOR
    op="="|<>|<|=|>|="
    type="386|486|P|PP|P2|P3|P4"
/>

```

Attributes

- op—An operator (=, <>, <, <=, >, or >=).
- value—A string specifying the required type of processor.

Example

```
<DEPEND_PROCESSOR op=">=" type="P" />
```

<DEPEND_RESOLUTION>

Specifies the required screen resolution for the channel.

Syntax

```
<DEPEND_RESOLUTION  
    op="="|<>|<|=|>|=|>=  
    value="640|800|1024|1280|1600"  
/>
```

Attributes

- op—An operator (=, <>, <, <=, >, or >=).
- value—A number (640, 800, 1024, 1280, or 1600) specifying the required screen resolution.

Example

```
<DEPEND_RESOLUTION value="800" op=">=" />
```

<DEPEND_DISKSPACE>

Contains the disk space requirements for the channel.

Syntax

```
<DEPEND_DISKSPACE>  
    <!-- DEPEND_DRIVE child tags -->  
</DEPEND_DISKSPACE>
```

Attributes

None.

Example

```
<DEPEND_DISKSPACE>  
    <DEPEND_DRIVE mb="32" drive="C" />  
</DEPEND_DISKSPACE>
```

<DEPEND_DRIVE>

Specifies the disk space requirement for one disk drive.

Syntax

```
<DEPEND_DRIVE
    drive="a drive letter"
    mb="an integer representing the required disk space in megabytes"
/>>
```

Attributes

- **drive**—A letter representing the drive for which you want to specify the disk space requirement.
- **mb**—An integer specifying the required minimum amount of disk space (in megabytes) for the drive.

Example

```
<DEPEND_DRIVE mb="32" drive="C"/>
```

<DEPEND_FILES>

Contains the file requirements for the channel.

Syntax

```
<DEPEND_FILES>
    <!-- DEPEND_FILE child tags -->
</DEPEND_FILES>
```

Attributes

None.

Example

```
<DEPEND_FILES>
    <DEPEND_FILE sizeop=">=" op="exists" dateop=">="
        versionop=">=" path="G:\MyFolder\test.txt" isOR="false"
        sizevalue="100" versionvalue="1.0.0.0" date="09-21-2001"/>
    <DEPEND_FILE op="notexist" isOR="true" path="G:\My
    Folder\training.txt"/>
</DEPEND_FILES>
```

<DEPEND_FILE>

Specifies the information for one required file.

Syntax

```
<DEPEND_FILE
```

```
    path="path of the file"
    isOR="true|false"
    op="exists|notexist"
    sizeop="=|<>|<|=|=|>|>="
    sizevalue="an integer representing the size of the file in bytes"
    dateop="=|<>|<|=|=|>|>="
    date="the required date for the file in the format MM-DD-YYYY"
    versionop="=|<>|<|=|=|>|>="
    versionvalue="the required version for the file"
/>>
```

Attributes

- **path**—A string representing the name and path of the required file.
- **isOR**—A boolean value (`true` or `false`) specifying if you want to make this an OR dependency.
- **op**—A string (`exists` or `notexist`) specifying whether you want to require the file to exist or not exist at the endpoint.
- **sizeop**—An operator (`=`, `<>`, `<`, `<=`, `>`, or `>=`) for the size of the file. This attribute is valid only if **op** exists.
- **sizevalue**—An integer specifying the required size (in bytes) of the file. This attribute is valid only if **op** and **sizeop** exist.
- **dateop**—An operator (`=`, `<>`, `<`, `<=`, `>`, or `>=`) for the date of the file. This attribute is valid only if **op** exists.
- **date**—A date (in the `MM-DD-YYYY` format) specifying the required date of the file. This attribute is valid only if **op** and **dateop** exist.
- **versionop**—An operator (`=`, `<>`, `<`, `<=`, `>`, or `>=`) for the version of the file. This attribute is valid only if **op** exists.
- **versionvalue**—A version specifying the required version of the file. This attribute is valid only if **op** and **versionop** exist.

Example

```
<DEPEND_FILE sizeop="&gt;=|" op="exists" dateop="&gt;=|" versionop="&gt;=|" path="G:\My Folder\test.txt" isOR="false" sizevalue="100" date="09-21-2001" versionvalue="1.0.0.0" />
    <DEPEND_FILE op="notexist" isOR="true" path="G:\My Folder\training.txt"/>
```

<DEPEND_REGISTRY>

Contains the registry requirements for the channel.

Syntax

```
<DEPEND_REGISTRY>
    <!-- DEPEND_REGKEY child tags -->
</DEPEND_REGISTRY>
```

Attributes

None.

Example

```
<DEPEND_REGISTRY>
    <DEPEND_REGKEY op="notexist"
key="\HKEY_CURRENT_CONFIG\Software\Fonts" isOR="false"/>
</DEPEND_REGISTRY>
```

<DEPEND_REGKEY>

Specifies the information for one required registry key.

Syntax

```
<depend_regkey
    key="the name of the registry key"
    valuename="the name of the value"
    valuetype="reg_sz|reg_expand_sz|reg_binary|reg_dword|reg_multi_sz"
    valuevalue="the value of the value"
    op="exists|notexist"
    isOR="true|false"
/>
```

Attributes

- key—A string representing the name of the required registry key.
- valuename—A string representing the value name in the registry key.
- valuetype—A string (reg_sz, reg_expand_sz, reg_binary, reg_dword, or reg_multi_sz) representing the type of value in the registry key. This attribute is valid only if valuename exists.
- valuevalue—A string representing the registry key value. This attribute is valid only if valuename exists.
- op—A string (exists or notexist) specifying whether you want to require the registry key value to exist or not exist at the endpoint.
- isOR—A boolean value (true or false) specifying if you want to make this an OR dependency.

Example

```
<DEPEND_REGKEY op="notexist"
key="\HKEY_CURRENT_CONFIG\Software\Fonts" isOR="false"/>
```

<DEPEND_CHANNELS>

Contains the channel requirements (the other channels required) for the channel.

Syntax

```
<DEPEND_CHANNELS>
    <!-- child DEPEND_CHANNEL tags -->
</DEPEND_CHANNELS>
```

Attributes

None.

Example

```
<DEPEND_CHANNELS>
    <DEPEND_CHANNEL op="exists" url="http://products/
Software_Distribution/462/ApplicationPackager" isOR="false"/>
</DEPEND_CHANNELS>
```

<DEPEND_CHANNEL>

Specifies the information for one required channel.

Syntax

```
<DEPEND_CHANNEL
    url="URL of the channel"
    op="exists|notexist"
    isOR="true|false"
    available_control="wait|no_wait|stop"
/>
```

Attributes

- **url**—The URL of the Symphony Marimba Client Automation channel you want to require.
- **op**—A string (exists or notexist) specifying whether you want to require the channel to exist or not exist at the endpoint.
- **isOR**—A boolean value (true or false) specifying if you want to make this an OR dependency.
- **available_control**—A string (wait, no_wait, or stop) specifying whether you want the package to wait for, not wait for, or stop the dependency channel running at the endpoint.

Example

```
<DEPEND_CHANNEL op="exists" url="http://products/
Software_Distribution/462/ApplicationPackager" isOR="false"/>
```

<DEPEND_ENVIRONMENT_VARS>

Contains the environment variables requirements for the channel.

Syntax

```
<DEPEND_ENVIRONMENT_VARS>
<!-- DEPEND_ENVIRONMENT_VAR child tags -->
</DEPEND_ENVIRONMENT_VARS>
```

Attributes

None.

Example

```
<DEPEND_ENVIRONMENT_VARS>
<DEPEND_ENVIRONMENT_VAR valueop="=" op="exists" path="test"
partof="partofsystem" isOR="false" value="testvalue" />
</DEPEND_ENVIRONMENT_VARS>
```

<DEPEND_ENVIRONMENT_VAR>

Specifies the information for one required environment variable.

Syntax

```
<DEPEND_ENVIRONMENT_VAR
path="environment variable name"
isOR="true|false"
op="exists|notexist"
partof="the required environment variable should be part of the
variables defined for System or User variables"
valueop="=|<>|contains|does not contain"
value="the required value for the environment variable"
/>
```

Attributes

- path—A string representing the name of the environment variable.
- isOR—A boolean value (true or false) specifying if you want to make this an OR dependency.
- op—A string (exists or notexist) specifying whether you want to require the environment variable to exist or not exist at the endpoint.
- partof - Specifies whether the environment variable need to be validated against the variables defined for "System" or "User"

- **valueop**—An operator (=, <>, contains or does not contain) for the value of the environment variable. This attribute is valid only if op exists.
- **value**—A string value specifying the required value of the environment variable. This attribute is valid only if op and valueop exist.

Example

```
<DEPEND_ENVIRONMENT_VAR valueop="=" op="exists" path="test"
partof="partofsystem" isOR="false" value="testvalue" />
```

<CUSTOMIZATION>

Contains the classes, scripts, or executables used to customize the behavior of your channel on the endpoint.

Syntax

```
<CUSTOMIZATION>
    <!-- child customization tags -->
</CUSTOMIZATION>
```

Attributes

None.

<SCRIPTS>

Specifies the information for the scripts used to customize the behavior of your channel.

Syntax

```
<SCRIPTS
    classpath="an additional classpath for Java IScripts">
        <!-- child script tags -->
<SCRIPTS/>
```

Attributes

- **classpath**—The fully qualified name of the class for a Java script. This attribute is valid only if the script type is java.

Example

```
<CUSTOMIZATION>
    <SCRIPTS>
        <SCRIPT path="test.bat" name="test" type="exe_bat_sh"
args="argument1" flags="pix"/>
        <SCRIPT path="C:\scripts\test2.bat" name="test2"
type="exe_bat_sh" args="argument1" flags="qrcs" withchannel="true"/>
    </SCRIPTS>
```

```
</CUSTOMIZATION>
```

<SCRIPT>

Specifies the information for one class, script, or executable used to customize the behavior of your channel.

Syntax

```
<SCRIPT  
    name="the name of the script"  
    type="exe_bat_sh|java"  
    path="the path of the script file"  
    javapath="the fully qualified name of the class for a Java script"  
    args="arguments to pass to the script"  
    flags="pqiumrvfxcsdab"  
    withchannel="true|false"  
/>
```

Attributes

- name—The name of the script.
- type—A string specifying the type of script. Use `exe_bat_sh` for .exe and .bat files in Windows, and shell scripts in UNIX; use `java` for Java files.
- path—The path of the script file at the endpoint, or the full path of the script file on the packaging machine if it is to be included as part of the channel. (For more information, see “Specifying paths for a script” on page 144.) You can specify if you want to include the script file with the channel using the `withchannel` attribute.
- javapath—The fully qualified name of the class for a Java script. This attribute is valid only if the script type is `java`.
- args—The arguments to pass to the script when it runs.
- flags—A character that specifies when the script runs and whether to ignore error codes.
 - p—Before the specified phase or phases
 - q—After the specified phase or phases
 - i—Install
 - u—Major update
 - m—Minor update
 - r—Uninstall (removal)

- v—Verify
- f—Repair
- x—Execute
- c—Ignore the script's exit or return code
- s—Run the script in the user's context
- d—Run the script as a detached process
- a—User install
- b—User update
- withchannel—A boolean value (true or false) specifying whether the script specified is to be copied from the path specified in the path attribute to the channel's scripts directory.

Note: You have to make the withchannel=true script available while packaging and the withchannel=false script while installation to execute these scripts during installation.

Note: When a package is repackaged using an XML template through the CLI, the script files are not removed from the package directory even though they are removed from the XML template.

Example

```
<CUSTOMIZATION>
  <SCRIPTS>
    <SCRIPT path="test.bat" name="test" type="exe_bat_sh"
args="argument1" flags="pix"/>
    <SCRIPT path="C:\scripts\test2.bat" name="test2"
type="exe_bat_sh" args="argument1" flags="qrcs" withchannel="true"/>
  </SCRIPTS>
</CUSTOMIZATION>
```

<PROPERTY>

Specifies the name and value of one property you want to add to the properties.txt file of the channel.

Syntax

```
<PROPERTY
    name="the name of the property to add to the channel's
    properties.txt file"
    value="the value of the property"
/>
```

Attributes

- **name**—The name of the property to add to the channel’s properties.txt file.
- **value**—A string specifying value you want to set for the property.

For a list of properties you can use in a channel’s properties.txt file, see the *Symphony Marimba Client Automation Reference Guide* on the Marimba Channel Store.

Example

```
<PROPERTY name="author" value="My Company Inc."/>
```

<PARAMETER>

Specifies the name and value of one property you want to add to the parameters.txt file of the channel.

Syntax

```
<PARAMETER
    name="the name of the property to add to the channel's
    parameters.txt file"
    value="the value of the property"
/>
```

Attributes

- **name**—The name of the property to add to the channel’s parameters.txt file.
- **value**—A string specifying value you want to set for the property.

For a list of properties you can use in a channel’s parameters.txt file, see the *Symphony Marimba Client Automation Reference Guide* on the Marimba Channel Store website.

Example

```
<PARAMETER name="rollback" value="true"/>
```

Non-MSI tags

Non-MSI tags all belong inside the “<PACKAGE>” tag and apply only to non-MSI channels.

This section describes the following non-MSI tags:

“<DIRECTORY>”
“<FILE>”
“<SHORTCUT>”
“<SYMLINK>”
“<REGKEY>”
“<REGVALUE>”
“<METABASEKEY>”
“<METABASEVALUE>”
“<MODIFIERGROUP>”
“<MODIFIER>”
“<INI>”

<DIRECTORY>

Specifies the information for one directory to be created on the endpoint for the channel.

Syntax

```
<DIRECTORY  
    path="directory to create on the endpoint"  
    attributes="rshac"  
    delete="true|false"  
    uid="in UNIX, the user's user ID in integer form or  
(if uidname is true) in string form"  
    uidname="true|false"  
    gid="in UNIX, the user's group ID in integer form or  
(if uidname is true) in string form"  
    gidname="true|false"  
    user_specific="true|false"  
/>
```

Attributes

- **path**—The path of the directory to create on the endpoint.

- attributes—One or more letters specifying the attributes for the directory:
 - r—Read-only
 - s—System
 - h—Hidden
 - a—Archive
 - c—Compressed
 - delete—A boolean value (true or false) specifying whether the directory specified is to be deleted from the endpoint (instead of being created).
- Windows
- user_specific—A boolean value (true or false) specifying whether the object is specific to each local user. For more information, see “Working with packages that contain user-specific files and registry entries” on page 161.
- UNIX
- uid—An integer or a string (if uidname is true) representing the user’s user ID.
 - uidname—A boolean value (true or false) specifying whether the user ID should be treated as an integer or as a string (the login name).
 - gid—An integer or a string (if gidname is true) representing the user’s group ID.
 - gidname—A boolean value (true or false) specifying whether the group ID should be treated as an integer or as a string (the login name).

<FILE>

Specifies the information for one file to be created on the endpoint for the channel.

Syntax

```
<FILE  
    filename="name of the file to create on the endpoint"  
    contentpath="absolute path to the parent file"  
    attributes="rshac"  
    shared="true|false"  
    delete="true|false"  
    selfregister="true|false"  
    reference="true|false"  
    uid="in UNIX, the user's user ID in integer form or  
(if uidname is true) in string form"  
    uidname="true|false"  
    gid="in UNIX, the user's group ID in integer form or  
(if uidname is true) in string form"  
    gidname="true|false"  
/>
```

Attributes

- **filename**—The name of the file to create on the endpoint.
- **contentpath**—The path to the parent file. This attribute is required and it must be the absolute path to the file. For example:
`c:\source\question.txt`.
- **attributes**—One or more letters specifying the attributes for the file:
 - **r**—Read-only
 - **s**—System
 - **h**—Hidden
 - **a**—Archive
 - **c**—Compressed
- **shared**—A boolean value (`true` or `false`) specifying whether the file specified is to be shared.
- **delete**—A boolean value (`true` or `false`) specifying whether the file specified is to be deleted from the endpoint (instead of being created).
- **selfregister**—A boolean value (`true` or `false`) specifying whether the file (with the `.ocx` or `.dll` file extensions in Windows) should self-register.

- reference—A boolean value (true or false) specifying whether the file should be packaged by reference. For more information, see “Adding files by reference” on page 72.
- UNIX
- uid—An integer or a string (if uidname is true) representing the user’s user ID.
 - uidname—A boolean value (true or false) specifying whether the user ID should be treated as an integer or as a string (the login name).
 - gid—An integer or a string (if gidname is true) representing the user’s group ID.
 - gidname—A boolean value (true or false) specifying whether the group ID should be treated as an integer or as a string (the login name).

<SHORTCUT>

Specifies the information for one shortcut to be created on the endpoint for the channel.

Syntax

```
<SHORTCUT
    shortcutname="name of the shortcut"
    path="path that the shortcut points to"
    arguments="any arguments to pass to the shortcut"
    startlocation="the working directory"
    description="description for the shortcut"
    run=normal|minimized|maximized
    iconpath="full path of the icon image"
    iconpos="0 to 100"
    jitupdate="true|false"
    jitrepair="true|false"
/>
```

Attributes

- shortcutname—The name of the shortcut to create on the endpoint.
- path—The name and path of the file to which the shortcut points.
- arguments—The arguments to pass to the shortcut.
- startlocation—The path to the directory where the original item is or a directory that contains some files necessary for a program (that the shortcut points to) to run.
- description—A string describing the shortcut.

- `run`—A string specifying how you want the window to display this item when the shortcut is opened:
 - `normal`—Standard window
 - `minimized`—A button on the task bar
 - `maximized`—Full screen
 - `iconpath`—A path to the icon for the shortcut.
- `iconpos`—An integer from 0 to 100 specifying the position of the icon specified in the `iconpath` attribute.
- `jitupdate`—A boolean value (`true` or `false`) specifying whether the application pointed to by the shortcut should get updated before running when the shortcut is opened. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.
- `jitrepair`—A boolean value (`true` or `false`) specifying whether the application pointed to by the shortcut should get repaired before running when the shortcut is opened. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.

<SYMLINK>

Specifies the information for one symbolic link to be created on the endpoint for the channel (in UNIX only).

Syntax

```
<SYMLINK
    name="name of the file"
    link="a symbolic link to the file or directory"
    attributes="file permission mode, an octal value (000-777)"
    delete="true|false"
    installpol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    updatepol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    uninstallpol="remove|installed_prompt|always|installed|
never|inherit"
    verifypol="inherit|content|exists|never"
    verifynipol="inherit|content|exists|never"
    repairpol="inherit|content|never"
/>
```

Attributes

- **shortcutname**—The name of the shortcut to create on the endpoint.
- **path**—The name and path of the file to which the shortcut points.
- **arguments**—The arguments to pass to the shortcut.
- **startlocation**—The path to the directory where the original item is or a directory that contains some files necessary for a program (that the shortcut points to) to run.
- **description**—A string describing the shortcut.
- **run**—A string specifying how you want the window to display this item when the shortcut is opened:
 - **normal**—Standard window
 - **minimized**—A button on the task bar
 - **maximized**—Full screen
- **iconpath**—A path to the icon for the shortcut.
- **iconpos**—An integer from 0 to 100 specifying the position of the icon specified in the **iconpath** attribute.

- jitupdate—A boolean value (true or false) specifying whether the application pointed to by the shortcut should get updated before running when the shortcut is opened. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.
- jitrepair—A boolean value (true or false) specifying whether the application pointed to by the shortcut should get repaired before running when the shortcut is opened. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.
- installpol—A string specifying the installation policy for this object.
- updatepol—A string specifying the update policy for this object.
- uninstallpol—A string specifying the uninstallation policy for this object.
- verifypol—A string specifying the verification policy for this object.
- verifynipol—A string specifying the verification not installed policy for this object.
- repairpol—A string specifying the repair policy for this object.

<REGKEY>

Specifies the information for one registry key to be created on the endpoint for the channel.

Syntax

```
<REGKEY
    key="the registry key"
    installpol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    updatepol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    uninstallpol="installed_prompt|always|installed|
never|inherit"
    verifypol="verify|ignore|verify_flexible|inherit"
>
    user_specific="true|false"
    <!-- child registry tags -->
</REGKEY>
```

Attributes

- key—The registry key to create on the endpoint.
- installpol—A string specifying the installation policy for this object.
- updatepol—A string specifying the update policy for this object.
- uninstallpol—A string specifying the uninstallation policy for this object.
- verifypol—A string specifying the verification policy for this object.
- user_specific—A boolean value (true or false) specifying whether the object is specific to each local user. For more information, see “Working with packages that contain user-specific files and registry entries” on page 161.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

<REGVALUE>

Specifies the information for one registry value to be created on the endpoint for the channel.

Syntax

```
<REGVALUE
    name="the name of the key-value pair"
    value="the value of the key-value pair"
    type="reg_sz|reg_dword|reg_binary"
    installpol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|
newer_timestamp_prompt|newer_timestamp_noprompt|inherit"
    updatepol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    uninstallpol="installed_prompt|always|installed|
never|inherit"
    verifypol="verify|ignore|verify_flexible|inherit"
/>
```

Attributes

- name—A string specifying the name of the registry key-value pair to create on the endpoint.

- **value**—A string specifying the value of the registry key-value pair to create on the endpoint.
- **type**—A string specifying the type of registry key-value pair to create on the endpoint:
 - **reg_sz**—A string
 - **reg_dword**—A 32-bit integer in hexadecimal format.
 - **reg_binary**—A hexadecimal binary value.
 - **installpol**—A string specifying the installation policy for this object.
 - **updatepol**—A string specifying the update policy for this object.
 - **uninstallpol**—A string specifying the uninstallation policy for this object.
 - **verifypol**—A string specifying the verification policy for this object.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

<METABASEKEY>

Specifies the information for one metabase key to be created on the endpoint for the channel.

Syntax

```
<METABASEKEY
    key="the metabase key"
    installpol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    updatepol="newer_version_prompt|newer_version_noprompt|
notexist|exists|always|newer_timestamp_version_prompt|
newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
newer_timestamp_noprompt|inherit"
    uninstallpol="installed_prompt|always|installed|
never|inherit"
    verifypol="verify|ignore|verify_flexible|inherit"
>
    <!-- child metabase tags -->
</METABASEKEY>
```

Attributes

- **key**—The metabase key to create on the endpoint.
- **installpol**—A string specifying the installation policy for this object.

- `updatepol`—A string specifying the update policy for this object.
- `uninstallpol`—A string specifying the uninstallation policy for this object.
- `verifypol`—A string specifying the verification policy for this object.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

<METABASEVALUE>

Specifies the information for one metabase value to be created on the endpoint for the channel.

Syntax

```
<METABASEVALUE
    id="integer ID"
    usertype="integer user type"
    attribute="inherit|insert_path|reference|
    secure|volatile"
    value="the metabase value"
    type="met_sz|met_dword|met_binary|
    met_expand_sz|met_multi_sz"
    installpol="newer_version_prompt|newer_version_noprompt|
    notexist|exists|always|newer_timestamp_version_prompt|
    newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
    newer_timestamp_noprompt|inherit"
    updatepol="newer_version_prompt|newer_version_noprompt|
    notexist|exists|always|newer_timestamp_version_prompt|
    newer_timestamp_version_noprompt|never|newer_timestamp_prompt|
    newer_timestamp_noprompt|inherit"
    uninstallpol="installed_prompt|always|installed|
    never|inherit"
    verifypol="verify|ignore|verify_flexible|inherit"
    />
```

Attributes

- `id`—An integer specifying the ID of the metabase value to create on the endpoint.
- `usertype`—An integer specifying metabase value’s user type. (For descriptions of the available user types, see “Editing metabase value data” on page 89.)
- `attribute`—A string specifying the attributes for the metabase value. (For descriptions of the available attributes, see “Editing metabase value data” on page 89.)
- `inherit`

- insert_path
- reference
- secure
- volatile
- value—The data for the metabase value to create on the endpoint.
- type—A string specifying the type of the metabase value: (For descriptions of the available types, see “Adding metabase entries to a channel” on page 87.)
 - met_sz
 - met_dword
 - met_binary
 - met_expand_sz
 - met_multi_sz
- installpol—A string specifying the installation policy for this object.
- updatepol—A string specifying the update policy for this object.
- uninstallpol—A string specifying the uninstallation policy for this object.
- verifypol—A string specifying the verification policy for this object.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

<MODIFIERGROUP>

Specifies the information for one text modifier group in the channel. It contains the text modifiers that affect a single file.

Syntax

```
<MODIFIERGROUP
    name="name"
    filepath="the filepath that this modifier group is associated
    with"
    installpol="always|never|inherit"
    updatepol="always|never|reinstall|inherit"
    >
    <!--child text modifier tags -->
</MODIFIERGROUP>
```

Attributes

- name—A name for the text modifier group.

- **filepath**—The path of the ASCII text file to be modified.
- **installpol**—A string specifying the installation policy for this object.
- **updatepol**—A string specifying the update policy for this object.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

<MODIFIER>

Specifies the information for one text modifier in the channel.

Syntax

```
<MODIFIER
    name="a name for the text modifier"
    type="insertline|searchline|searchreplace"
    installpol="always|never|inherit"
    updatepol="always|never|reinstall|inherit"
    <!-- type-specific tags -->
/>
```

where the type-specific tags depend on the type of text modifier

■ For Insert Line Text Modifiers

```
inserttext="the text to insert"
insertaction="replace|insert"
linenumber="an integer line number to insert/replace text at"
```

■ For Search and Replace Line Text Modifiers

```
searchtext="the text to search for on a line"
inserttext="the text to insert or replace with"
match="exact|startswith|substring|endswith"
insertaction="replace|before|after"
matchcase="true|false"
```

■ For Search and Replace Text Modifiers

```
searchtext="the text to search for"
replacetext="the text to replace the search text with"
match="exact|substring"
matchcase="true|false"
```

Attributes

- **name**—A name for the text modifier.
- **type**—A string (`insertline`, `searchline`, or `searchreplace`) specifying the type of text modifier. For more information about the types of text modifiers and the options available for each one, see “Editing ASCII text files” on page 173.

- `installpol`—A string specifying the installation policy for this object.
- `updatepol`—A string specifying the update policy for this object.

For a description of the policies available, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

<INI>

Specifies the information for one ini file to be created on the endpoint for the channel.

Syntax

```
<INI filename="name of the ini file to create on the endpoint"
      contentpath="absolute path to the parent ini file"
      delete="true|false" />
```

Attributes

`filename`—The name of the ini file to create on the endpoint.

`contentpath`—The path to the parent ini file. This attribute is required and it must be the absolute path to the ini file. For example:

`C:\source\settings.ini`.

`delete`—A boolean value (true or false) specifying whether the file specified is to be deleted from the endpoint (instead of being created).

► Example: XML template file that contains non-MSI tags

This XML template file uses non-MSI tags to perform the following actions:

- Adds a directory called `c:\1` to the package.
- Adds a file called `file1.txt` with content sourced from `c:\filecontent.txt` under the directory `c:\1`.
- Adds a shortcut called `my shortcut name.lnk` under the directory `c:\1`.
- Adds a registry key called `HKEY_LOCAL_MACHINE\Software` to the package.
- Adds a registry key value pair `name=value` under the `HKEY_LOCAL_MACHINE\Software` registry key.
- Adds a metabase key called `LM` to the package.
- Adds a metabase key value pair `0=value` under the `LM` metabase key.
- Adds an NT Service with the binary located at `c:\myservice.exe` to the package.

```

<PACKAGE>
  <DIRECTORY path="c:\1" attributes="a">
    <FILE filename="file1.txt" contentpath="c:\filecontent.txt"
      attributes="a" />
    <SHORTCUT shortcutname="my shortcut name.lnk" path="c:\"
      arguments="arg1 arg2" startlocation="c:\" description="my
      description" run="maximized" installpol="inherit" />
  </DIRECTORY>
  <REGKEY key="\HKEY_LOCAL_MACHINE\Software" >
    <REGVALUE name="value name" value="value value" type="reg_sz"
      installpol="inherit" updatepol="inherit" uninstallpol="inherit"
      verifypol="inherit,content,exists,never"
      verifynipol="inherit,content,exists,never"
      repairpol="inherit,content,never"/>
  </REGKEY>
  <METABASEKEY key="\LM" installpol="inherit" updatepol="inherit"
    uninstallpol="inherit" verifypol="inherit,content,exists,never"
    verifynipol="inherit,content,exists,never"
    repairpol="inherit,content,never">
    <METABASEVALUE id="0" usertype="0" attribute="secure"
      value="value" type="met_sz" installpol="inherit"
      updatepol="inherit" uninstallpol="inherit"
      verifypol="inherit,content,exists,never"
      verifynipol="inherit,content,exists,never"
      repairpol="inherit,content,never"/>
  </METABASEKEY>
  <SERVICES>
    <SERVICE>
      <SERV_GENERAL errorcontrol="normal" displayname="my service"
        type="own_process" servicepath="c:\myservice.exe"
        start="demand" servicename="service name"/>
      <SERV_SECURITY desktop="false" account="system"/>
      <SERV_POLICIES installpol="inherit"
        verifypol="inherit,content,exists,never"
        verifynipol="inherit,content,exists,never"
        updatepol="inherit"
        repairpol="inherit,content,attributes,never"
        uninstallpol="inherit"/>
    </SERVICE>
  </SERVICES>
</PACKAGE>

```

MSI tags

MSI tags all belong inside the “<PACKAGE>” tag and apply to MSI channels only.

This section describes the following MSI tags:

- “<MSI_INSTALLATION>”
- “<MSI_UILEVEL>”
- “<MSI_PATCHES>”
- “<MSI_PATCH>”
- “<MSI_TRANSFORMS>”
- “<MSI_TRANSFORM>”
- “<MSI_DOWNLOAD>”
- “<MSI_POLICIES>”
- “<MSI_LOGGING>”

<MSI_INSTALLATION>

Specifies how Windows Installer installs files after the package has been downloaded.

Syntax

```
<MSI_INSTALLATION
    installmode="normal|admin|asis"
    reinstall="true|false"
    usersettable="true|false"
    msidir="the default directory to hold the .msi, patch, and
    transform files prior to install"
    properties="installation properties in name=value form, each
    separated by at least one space"
/>
```

Example

```
installmode="NORMAL|ADMIN|ASIS"
    reinstall="TRUE|FALSE"
    usersettable="TRUE|FALSE"
    msidir="the default directory to hold the .msi, patch, and
    transform files prior to install"
    properties="installation properties in name=value form, each
    separated by at least one space"
```

<MSI_UILEVEL>

Specifies the level of user interface (UI) that Windows Installer presents to the user when it installs the packaged application.

Syntax

```
<MSI_UILEVEL
    type="default|none|basic|reduced|full"
    omiterrordialogs="true|false"
/>
```

Example

```
type="DEFAULT|NONE|BASIC|REDUCED|FULL"
    omiterrordialogs="TRUE|FALSE"
```

<MSI_PATCHES>

Contains the patches for a Windows Installer application.

Syntax

```
<MSI_PATCHES
    reinstall="true|false"
>
    <!-- child MSI patch tags -->
</MSI_PATCHES>
```

Example

```
reinstall="TRUE|FALSE"
```

<MSI_PATCH>

Specifies the information for one patch file.

Syntax

```
<MSI_PATCH
    path="full path to the patch file"
    properties="patch properties in name=value form, each separated by
at least one space"
/>
```

Example

```
path="the path to the patch file"
    properties="patch properties in name=value form, each
separated by at least one space"
```

<MSI_TRANSFORMS>

Contains the transforms for a Windows Installer application. It also specifies how transforms are packaged and applied to an installation.

Syntax

```
<MSI_TRANSFORMS
    reinstall="true|false"
    globalpath="a semicolon-separated list of paths that contain
transforms to apply"
>
    <!-- child MSI transform tags -->
</MSI_TRANSFORMS>
```

Example

```
reinstall="TRUE|FALSE"
    globalpath="a semi-colon separated list of paths that contain
transforms to apply"
```

<MSI_TRANSFORM>

Specifies the information for one transform file.

Syntax

```
<MSI_TRANSFORM
    path="full path to the transform file"
/>
```

Example

```
path="the path to the transform file"
```

<MSI_DOWNLOAD>

Specifies which files are downloaded in the package, and if they are removed after the installation.

Syntax

```
<MSI_DOWNLOAD
    option="required|all|msi"
    deleteafterdownload="true|false"
    srclist="the URL, local, or network path to search for files"
/>
```

Example

```
option="REQUIRED|ALL|MSI"
    deleteafterdownload="TRUE|FALSE"
    srclist="the url, local, or network path to search for files"
```

<MSI_POLICIES>

Specifies the user and machine policies that affect the installation behavior of Windows Installer.

Syntax

```
<MSI_POLICIES
    installer="system|all|managed"
    elevated="system|true|false"
    storerollback="system|true|false"
    alternativebrowsing="system|true|false"
    nonadminbrowsing="system|true|false"
    alternatemedia="system|true|false"
    logging="system|true|false"
    loggingargs="the command line arguments for logging"
/>
```

Example

```
installer="SYSTEM|ALL|MANAGED"
elevated="SYSTEM|TRUE|FALSE"
storerollback="SYSTEM|TRUE|FALSE"
alternativebrowsing="SYSTEM|TRUE|FALSE"
nonadminbrowsing="SYSTEM|TRUE|FALSE"
alternatemedia="SYSTEM|TRUE|FALSE"
logging="SYSTEM|TRUE|FALSE"
loggingargs="the command line args for logging"
```

<MSI_LOGGING>

Specifies Windows Installer logging options.

Syntax

```
<MSI_LOGGING
    logfilepath="the path of the log file to be generated"
    outofmemory_fatalexit="true|false"
    errormessages="true|false"
    warningmessages="true|false"
    userrequests="true|false"
    statusmessages="true|false"
    validsourcelocation="true|false"
    diskspace="true|false"
    installactions="true|false"
    datarecord="true|false"
    userinterface="true|false"
    propertyvalues="true|false"
    verbose="true|false"
    flushevery20="true|false"
    flushline="true|false"
/>
```

Example

```
logfilepath="the path of the log file to be generated"
    outofmemory_fatalexit="TRUE|FALSE"
    errormessages="TRUE|FALSE"
    warningmessages="TRUE|FALSE"
    userrequests="TRUE|FALSE"
    statusmessages="TRUE|FALSE"
    validsourcelocation="TRUE|FALSE"
    diskspace="TRUE|FALSE"
    installactions="TRUE|FALSE"
    datarecord="TRUE|FALSE"
    userinterface="TRUE|FALSE"
    propertyvalues="TRUE|FALSE"
    verbose="TRUE|FALSE"
    flushevery20="TRUE|FALSE"
    flushline="TRUE|FALSE"
```

Search and modify tags

Search and modify tags allow you to search for specific objects (based on a pattern-matching syntax) and to edit the objects' attributes. These tags all belong inside the “<PACKAGE>” tag.

These tags are useful for editing many objects in a package without having to manually open the package with the Package Editor GUI and edit each object that they want to change. They can be used with the command line.

This section describes the following search and modify tags:

“<MODIFY_OBJECT>”
“<MODIFY_OBJECT_ATTRIBUTES>”

<MODIFY_OBJECT>

Specifies the objects in the package you want to search for and edit. It is a container tag.

Syntax

```
<MODIFY_OBJECT
    types="directory,file,dll,shortcut,registry_key,registry_value,any"
    canonical_path="path of the objects to search for"
    canonical_path_casesensitive="true|false"
    >
    <!-- child modify object tags -->
</MODIFY_OBJECT>
```

Attributes

- **types**—The type of objects to search for in the package. You can specify a comma-delimited list.

Description	
Required	yes
Default	none
Possible values	directory file dll shortcut registry_key registry_value any

- **canonical_path**—The full path to the objects to search for in the package.

Description	
Required	no
Default	none
Possible values	A string pattern representing a path. You can substitute wildcard characters for parts of the path. You can use an asterisk (*) as a substitute for zero or more characters, and you can use a question mark (?) as a substitute for any single character. For example, searching for c:*.ex? locates c:\\notepad.exe, but not c:\\notepad.exe.

- **canonical_path_casesensitive**—A boolean value (true or false) specifying whether the canonical_path attribute should be case-sensitive. If you do not specify a value for this attribute, the case is ignored when searching for objects with full paths that match the given pattern.

Description	
Required	no

Description	
Default	false
Possible values	true or false

<MODIFY_OBJECT_ATTRIBUTES>

Specifies the attributes of the objects in the package you want to edit. It is not a container tag.

Syntax

```
<MODIFY_OBJECT_ATTRIBUTES
    installpol="NEWER_VERSION_PROMPT|NEWER_VERSION_NOPROMPT|
NOTEXIST|EXISTS|ALWAYS|NEWER_TIMESTAMP_VERSION_PROMPT|
NEWER_TIMESTAMP_VERSION_NOPROMPT|NEVER|NEWER_TIMESTAMP_PROMPT|
NEWER_TIMESTAMP_NOPROMPT|INHERIT"
    updatepol="NEWER_VERSION_PROMPT|NEWER_VERSION_NOPROMPT|
NOTEXIST|EXISTS|ALWAYS|NEWER_TIMESTAMP_VERSION_PROMPT|
NEWER_TIMESTAMP_VERSION_NOPROMPT|NEVER|NEWER_TIMESTAMP_PROMPT|
NEWER_TIMESTAMP_NOPROMPT|INHERIT"
    verifypol="INHERIT,CONTENT,ATTRIBUTES,
VERSION_INCREASE,EXISTS,NEVER"
    verifynipol="INHERIT,CONTENT,ATTRIBUTES,
VERSION_INCREASE,EXISTS,NEVER"
    repairpol="INHERIT,CONTENT,NEVER"
    uninstallpol="REMOVE|INSTALLED_PROMPT|
ALWAYS|INSTALLED|NEVER|INHERIT"
    delete="true|false"
    attributes="Windows:(r|s|h|a|c)*;UNIX:(0-7)(0-7)(0-7)(0-7)"
    shared="true|false"
    register="true|false"
    path="c:\winnt\system3\notepad.exe"
    arguments="argument1 argument2 argument3"
    startlocation="c:\winnt"
    description="description_string"
    run="NORMAL|MINIMIZED|MAXIMIZED"
    iconpath="c:\winnt\icon.ico"
    iconpos="integer from 1 to 100"
    jitupdate="true|false"
    jitrepair="true|false"
    value="text or hexadecimal string"
    type="REG_SZ|REG_DWORD|REG_BINARY"
    uid="in UNIX, the user's user ID in integer form"
    gid="in UNIX, the user's group ID in integer form"
    user_specific="true|false"
```

Attributes

Note: For a description of the different policies, see “Policies for installation, update, uninstallation, verification, and repair” on page 331.

- `installpol`—Install policy for the object.

Description	
Required	no
Default	none
Possible values	NEWER_VERSION_PROMPT NEWER_VERSION_NOPROMPT NOTEXIST EXISTS ALWAYS NEWER_TIMESTAMP_VERSION_PROMPT NEWER_TIMESTAMP_VERSION_NOPROMPT NEVER NEWER_TIMESTAMP_PROMPT NEWER_TIMESTAMP_NOPROMPT INHERIT
Valid object types	any

- `updatepol`—Update policy for the object.

Description	
Required	no
Default	none
Possible values	NEWER_VERSION_PROMPT NEWER_VERSION_NOPROMPT NOTEXIST EXISTS ALWAYS NEWER_TIMESTAMP_VERSION_PROMPT NEWER_TIMESTAMP_VERSION_NOPROMPT NEVER NEWER_TIMESTAMP_PROMPT NEWER_TIMESTAMP_NOPROMPT INHERIT
Valid object types	any

- **verifypol**—Verify-installed policy for the object.

Description	
Required	no
Default	none
Possible values	A comma-separated list of the following policies: INHERIT CONTENT ATTRIBUTES VERSION_INCREASE EXISTS NEVER
Valid object types	any

- **verifynipol**—Verify-not-installed policy for the object.

Description	
Required	no
Default	none
Possible values	A comma-separated list of the following policies: INHERIT CONTENT ATTRIBUTES VERSION_INCREASE EXISTS NEVER
Valid object types	any

- **repairpol**—Repair policy for the object.

Description	
Required	no
Default	none
Possible values	A comma-separated list of the following policies: INHERIT CONTENT NEVER
Valid object types	any

- `uninstallpol`—Uninstall policy for the object.

Description	
Required	no
Default	none
Possible values	REMOVE INSTALLED_PROMPT ALWAYS INSTALLED NEVER INHERIT
Valid object types	any

- `delete`—A boolean value (`true` or `false`) specifying whether the object should be deleted (rather than added) when the package is installed or updated. If this attribute is set to true, the object are marked for a delete operation.

Description	
Required	no
Default	none
Possible values	true or false
Valid object types	any

- `attributes`—In Windows, the attribute flags for the object. In UNIX, the permission flags for the object.

Description	
Required	no
Default	none
Possible values	Windows: $(r s h a c)^*$ UNIX: $(0-7)(0-7)(0-7)(0-7)$
Valid object types	file dll directory

- **shared**—A boolean value (true or false) specifying whether the object should be a shared file. If this attribute is set to true, the object is shared.

Description	
Required	no
Default	none
Possible values	true or false
Valid object types	file dll

- **register**—A boolean value (true or false) specifying whether the DLL file should be registered. If this attribute is set to true, the DLL file is registered (in Windows).

Description	
Required	no
Default	none
Possible values	true or false
Valid object types	dll

- **path**—The path to which the shortcut is pointing.

Description	
Required	no
Default	none
Possible values	A file system path. For example c:\winnt\system32\notepad.exe
Valid object types	shortcut

- **arguments**—The arguments that the shortcut should pass to the path value (specified in the path attribute) before execution.

Description	
Required	no
Default	none

Description	
Possible values	A space-separated list of arguments.
Valid object types	shortcut

- `startlocation`—The path to the working directory where the original item is or a working directory that contains some files necessary for a program (that the shortcut points to, as specified in the `path` attribute) to run.

Description	
Required	no
Default	none
Possible values	A file system path. For example: <code>c:\winnt\</code>
Valid object types	shortcut

- `description`—A string describing the shortcut.

Description	
Required	no
Default	none
Possible values	A string.
Valid object types	shortcut

- `run`—A string specifying how you want the window to display this item when the shortcut is opened (window mode).

Description	
Required	no
Default	none
Possible values	NORMAL—Standard window MINIMIZED—A button on the task bar MAXIMIZED—Full screen
Valid object types	shortcut

- `iconpath`—A path to the icon for the shortcut.

Description	
Required	no
Default	none
Possible values	A file system path to the icon. For example: <code>c:\winnt\icon.ico</code>
Valid object types	shortcut

- `iconpos`—An integer from 0 to 100 specifying the position of the icon specified in the `iconpath` attribute.

Description	
Required	no
Default	none
Possible values	An integer from 1 to 100.
Valid object types	shortcut

- `jitupdate`—A boolean value (`true` or `false`) specifying whether the application pointed to by the shortcut should get updated before running when the shortcut is opened. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.

Description	
Required	no
Default	none
Possible values	<code>true</code> or <code>false</code>
Valid object types	shortcut

- **jitrepair**—A boolean value (true or false) specifying whether the application pointed to by the shortcut should get repaired before running when the shortcut is opened. For more information, see “Updating and repairing applications using shortcuts before startup (Windows only)” on page 169.

Description	
Required	no
Default	none
Possible values	true or false
Valid object types	shortcut

- **value**—A text or hexadecimal string (depending on the type of registry value) representing the value name in the registry key.

Description	
Required	no
Default	none
Possible values	A text or hexadecimal string (depending on the type of registry value).
Valid object types	registry_value

- **type**—A string representing the type of value in the registry key.

Description	
Required	no
Default	none
Possible values	REG_SZ REG_DWORD REG_BINARY
Valid object types	registry_value

- **uid**—In UNIX, an integer representing the user’s user ID.
- **gid**—In UNIX, an integer representing the user’s group ID.

- `user_specific`—In Windows, a boolean value (`true` or `false`) specifying whether the object is specific to each local user. For more information, see “Working with packages that contain user-specific files and registry entries” on page 161.

Description	
Required	<code>no</code>
Default	<code>false</code>
Possible values	<code>true</code> or <code>false</code>
Valid object types	directories and registry keys

Snapshot tags

Snapshot tags all belong inside the “<PACKAGE>” tag and apply when taking snapshots from the command line only (see “Creating preinstall and postinstall snapshots” on page 310).

Note: You can use these tags only when taking snapshots from the command line. You cannot apply them to a packaged application.

This section describes the following snapshot tags:

- “<SNAPSHOT>”
- “<FILESYSTEM>”
- “<DIRECTORY>”
- “<FILTER>”
- “<REGISTRY>”
- “<REGKEY>”
- “<FILTER>” (same as directory)
- “<METABASE>”
- “<METKEY>”
- “<FILTER>” (same as directory)

<SNAPSHOT>

Defines a snapshot created using Packager for Shrinkwrap Windows Applications. The “<SNAPSHOT>” tag encloses all the other tags for a snapshot.

Syntax

```
<SNAPSHOT
    ini="true|false"
>
    <!-- child snapshot tags -->
</SNAPSHOT>
```

Attributes

- ini—A boolean value (true or false) specifying whether you want INI files to be treated as special file objects. If this attribute is set to true, Application Packager parses the INI files and capture any changes made to the key and value pairs in them. Any changes found by Application Packager are merged with the existing file found at the endpoint. By default, this check box is selected. If you want Application Packager to process INI files as ordinary files and not merge changes, you can set this attribute to false.

<FILESYSTEM>

Contains all the directories to include and exclude in the snapshot.

Syntax

```
<FILESYSTEM
    include="true|false"
    includedeletes="true|false"
>
    <!-- child filesystem tags -->
</FILESYSTEM>
```

Attributes

- include—A boolean value (true or false) specifying whether you want to include the specified directories in the snapshot. Set the attribute to true to include and false to exclude.
- includedeletes—A boolean value (true or false) specifying whether you want to capture the removal of the specified directories in the snapshot. Set the attribute to true to include and false to exclude.

<DIRECTORY>

Defines a directory to include in the snapshot.

Syntax

```
<DIRECTORY  
    path="a directory path"  
/>
```

Attributes

- **path**—The path of a directory you want to include in the snapshot.

<FILTER>

Defines a filter that specifies a directory, file, or key to ignore when taking the snapshot.

Syntax

```
<FILTER  
    type="directory|file|key"  
    noun="name|parent|path"  
    verb="is|isn't|contains|not_contains|begins_with|ends_with"  
    adverb="the value of the filter"  
    enabled="true|false"  
    description="description of the filter"  
/>
```

Attributes

- **type**—A string representing the type of item (directory, file, and key) for which you are creating a filter.
- **noun**—A string representing the type of information (name, parent, and path) you are using to specify a filter.
- **verb**—A string representing the comparison (is, isn't, contains, not_contains, begins_with, and ends_with) you want to make when using the filter.
- **adverb**—A string specifying the information you are using to specify a filter, for example, a directory or file name. You can also use macros; for more information, see “Macros you can use in filters” on page 44.
- **enabled**—A boolean value (true or false) specifying whether you want to enable the filter when taking the snapshot.
- **description**—A string specifying a description or label for the filter.

<REGISTRY>

Contains the registry entries to include and exclude in the snapshot.

Syntax

```
<REGISTRY
    include="true|false"
    includedeletes="true|false"
>
    <!-- child registry tags -->
</REGISTRY>
```

Attributes

- **include**—A boolean value (true or false) specifying whether you want to include the specified registry entries in the snapshot. Set the attribute to true to include and false to exclude.
- **includedeletes**—A boolean value (true or false) specifying whether you want to capture the removal of the specified registry entries in the snapshot. Set the attribute to true to include and false to exclude.

<REGKEY>

Defines a registry key to include in the snapshot.

Syntax

```
<REGKEY
    key="a registry key to include in the snapshot"
/>
```

Attributes

- **key**—The registry key you want to include in the snapshot.

<FILTER>

See the description for “<FILTER>” on page 430.

<METABASE>

Contains the metabase entries to include and exclude in the snapshot.

Syntax

```
<METABASE
    include="true|false"
    includedeletes="true|false"
>
```

```
<!-- child metabase tags -->
</METABASE>
```

Attributes

- **include**—A boolean value (true or false) specifying whether you want include the specified metabase entries in the snapshot. Set the attribute to true to include and false to exclude.
- **includedeletes**—A boolean value (true or false) specifying whether you want capture the removal of the specified metabase entries in the snapshot. Set the attribute to true to include and false to exclude.

<METKEY>

Defines a metabase key to include in the snapshot.

Syntax

```
<METKEY
  key="a metabase key to include in the snapshot"
/>
```

Attributes

- **key**—The metabase key you want to include in the snapshot.

<FILTER>

See the description for “<FILTER>” on page 430.

D Windows system macros

This section lists the system macros available for Windows machines. It also gives a brief description and example for each macro.

The following topics are provided:

- Overview (page 434)
- System macros (page 434)

Overview

In the Package Editor, these system macros are listed in the Available Macros section of the Macro tab.

System macros

The following table lists the macros you can use in packaged applications you install in Windows.

Table D-1: Windows system macros

Macro name	Description	Platforms available
SYS.BOOT	<p>The drive where Microsoft Windows is installed. Example: If Windows is installed in C:\Windows, SYS.BOOT is C:\.</p>	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.COMMON DESKTOP	<p>For Windows NT, Windows 2000, and Windows XP, it is the directory where the desktop for ALLUSERS application files goes. Example: An application with a global desktop shortcut in Windows 2000 puts it under C:\Documents and Settings\All Users\Desktop. For Windows NT, it is something similar to C:\WINNT\Profiles\All Users\Desktop. For Windows Vista/2008: C:\Users\Public\Desktop</p>	Windows NT Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008 only
SYS.COMMON PROFILE	<p>For Windows NT, Windows 2000, and Windows XP, this maps to the location of the ALLUSERS root directory. Example: This is typically the parent path to SYS.COMMONDESKTOP. For Windows 2000, this means C:\Documents and Settings\All Users. For Windows NT, it is C:\WINNT\Profiles\All Users. For Windows Vista/2008: c:\ProgramData\Microsoft\Windows</p>	Windows NT Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008 only

Table D-1: Windows system macros(Continued)

Macro name	Description	Platforms available
SYS.COMMONPROGRAMS	<p>maps to the location of the ALLUSERS Start menu's Program directory.</p> <p>Example: An application with a global desktop shortcut under the Start menu's Program files inserts a shortcut or directory under this path. Typically, it is</p> <p>C:\Documents and Settings\All Users\Start Menu\Programs for Windows 2000 and C:\WINNT\Profiles\All Users\Start Menu\Programs for Windows NT.</p> <p>For Windows Vista/2008: c:\ProgramData\Microsoft\Windows\Start Menu\Programs</p>	Windows NT Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008 only
SYS.COMMONSTARTMENU	<p>maps to the location of the ALLUSERS Start menu.</p> <p>Example: An application with a global desktop shortcut under the Start menu inserts a shortcut or directory under this path. Typically, it is</p> <p>C:\Documents and Settings\All Users\Start Menu for Windows 2000 and C:\WINNT\Profiles\All Users\Start Menu for Windows NT.</p> <p>For Windows Vista/2008: c:\ProgramData\Microsoft\Windows\Start Menu</p>	Windows NT Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008 only
SYS.COMMONSTARTUP	<p>maps to the location of the ALLUSERS Startup directory.</p> <p>Example: An application with a global desktop shortcut under the Start menu inserts a shortcut or directory under this path. Typically, it is</p> <p>C:\Documents and Settings\All Users\Start Menu\Programs\Startup for Windows 2000 and C:\WINNT\Profiles\All Users\Start Menu\Programs\Startup for Windows NT.</p> <p>For Windows Vista/2008: ---</p>	Windows NT Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008 only
SYS.COMPUTER	<p>The computer's name.</p> <p>Example: The macro returns the name for this computer if applicable, such as NTMACHINE1.</p>	Windows 95/98, Windows NT, Windows 2000, Windows 2003, and Windows XP

Table D-1: Windows system macros(Continued)

Macro name	Description	Platforms available
SYS.DATE	The system date. Example: The macro returns the current date in SHORT format according to Java's DateFormat. This represents December 13, 1952 as 12.13.1952.	Windows 95/98, Windows NT, Windows 2000, Windows 2003, and Windows XP
SYS.DESKTOP	The current user's desktop path. Example: For Windows 98, it is C:\WIN98\Desktop. But for Windows 2000, it is something similar to C:\Documents and Settings\currentusername\Desktop. For Windows Vista/2008: c:\Users\<user name>\Desktop	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.IP	The current machine's IP address, if one is assigned. Example: The macro maps to something similar to 172.1.1.102.	Windows 95/98, Windows NT, Windows 2000, Windows 2003, and Windows XP
SYS.PROFILE	The current user's profile directory. Example: The macro maps to the current user's profile. For Windows Vista/2008: c:\Users\<user name>	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.PROGRAMS	The current user's program file location. Example: For Windows 98, it is something similar to C:\WIN98\Start Menu\Programs. But for Windows 2000, it is something similar to C:\Documents and Settings\currentusername\Start Menu\Programs. For Windows Vista/2008: c:\Users\<user name>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.ROAMINGPROFILE	The current user's installation directory: Install directory for Windows OS prior to Windows Vista/2008: C:\Documents and Settings\<user name> Install directory for Windows OS for Windows Vista/2008 : C:\Users\<user name>\AppData\Roaming\Microsoft\Windows	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, Windows Vista/2008

Table D-1: Windows system macros(Continued)

Macro name	Description	Platforms available
SYS.STARTMENU	The current user's Start menu location. Example: For Windows 98, it is something similar to C:\WIN98\Start Menu. But for Windows 2000, it is something similar to C:\Documents and Settings\currentusername\Start Menu. For Windows Vista/2008: c:\Users\<user name>\AppData\Roaming\Microsoft\Windows\Start Menu	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.STARTUP	The current user's Startup directory. Example: For Windows 98, it is something similar to C:\WIN98\Start Menu\Programs\Startup. But for Windows 2000, it is something similar to C:\Documents and Settings\currentusername\Start Meny\Programs\Startup. For Windows Vista/2008: c:\Users\<user name>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.SYSTEM	The path to the SYSTEM directory. Example: For Windows 98, it is something similar to C:\WIN98\SYSTEM. But for Windows 2000, it is something similar to C:\WINNT\SYSTEM. For Windows Vista/2008: C:\Windows\System32	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008
SYS.TIME	The current system time in Java's SHORT DateFormat. Example: The current system time of 3:30:32 p.m. is displayed as 3:30 p.m.	Windows 95/98, Windows NT, Windows 2000, Windows 2003, and Windows XP
SYS.USER	The name of the current user logged in. Example: This macro resolves to the login name of the user. If a channel was installed when user A logs in, this macro resolves to "A."	Windows 95/98, Windows NT, Windows 2000, Windows 2003, and Windows XP
SYS.WINDOWS	The location where Windows is installed. Example: For Windows 98, it might be C:\WIN98. For Windows NT, it might be C:\WINNT. For Windows XP/Vista/2008, it might be C:\Windows	Windows 95/98, Windows NT, Windows 2000, Windows 2003, Windows XP, and Windows Vista/2008

E Choosing between Content Replicator and

File Packager

This section describes the differences between Content Replicator and Application Packager, specifically File Packager. For more information about Content Replicator, see the *Server Management Administrator's Guide* on the Marimba Channel Store.

Deciding which packager to use

For Server Management, you can use either of two Marimba applications (modules) to replicate content and applications to your endpoints. This section provides guidelines to help you choose which one to use in your particular circumstances.

Application Packager (Software Distribution module)

Application Packager is best suited for packaging applications (not just files). It provides an assortment of packagers, including the Packager for Shrinkwrap Windows Applications, which you can use to replicate Windows applications, such as Microsoft Office, to your endpoints. Other packagers can be used on both Windows and UNIX, such as the File Packager. If you want to replicate a small number of files, such as configuration files for applications, to your endpoints, you might choose to use the File Packager because it allows you to make file-level configuration changes. For example, File Packager's text modifier feature would enable you to easily edit the contents of a .config or .properties file.

A key benefit to using the File Packager is being able to modify the specific configuration for an application installation. After you have packaged the files that make up an application, you can use the Application Packager's Package Editor component to change file attributes or file permissions, as well as install- and update-time policies. You can also use scripts and text file modifiers to change files at install time. (For definitions of some of these terms, see the table at the end of this section.)

Content Replicator (Content Distribution module)

Whereas Application Packager is meant to be used for applications and for a small number of files, Content Replicator was designed to replicate large numbers of files and directories, and also very large files. To run pre-install or post-install scripts, you can use Deployment Manager to create remote system commands for the scripts. Content Replicator also enables you to package content (for example, HTML files) on one platform and then replicate them to various UNIX and Windows platforms.

Note: You cannot subscribe the tuner directly to a data channel created by Content Replicator. Use a Content Replicator command to install the files from the data channel. In contrast, for channels created with Application Packager, you do subscribe the tuner to the channel.

Comparison

The following table provides more detail about the differences between Content Replicator and File Packager.

Table E-1: Comparing Content Replicator and File Packager

Feature	Content Replicator	File Packager
File size	Up to 20 GB.	Up to 2GB for a single file.
Channel size	Up to 100 GB.	10 GB.
Max number of files	Up to 2,000,000 files.	65,000
Max number of files and subdirectories in one directory	Up to 100,000.	Limit has not been tested.
Directory depth	Up to 100 levels.	Limit has not been tested.
Length of file names	Windows: 246; UNIX: 255 characters The “file name length” means the number of characters in the full path to the file, including the drive letter on Windows platforms (example: c:\documents\files.txt is 22 characters).	Limit has not been tested.
Packaging steps	Packaging files and publishing files to the transmitter are done in a single step.	Packaging and publishing are discreet steps. This means that a “package” is first created, which makes a copy of all the files before publishing them. If your original source directory contains 1 GB of files, you might need another 1 GB of disk space to accommodate the package (copy).
Installation steps	If the tuner and the target installation directory are on the same drive, files are installed directly into the file system (unless you specifically “stage” them first).	Installs files first into the tuner’s storage space. This means that a copy of the files is created and remains until they are garbage-collected.

Table E-1: Comparing Content Replicator and File Packager

Feature	Content Replicator	File Packager
Scripting	Pre- and post-install. (Other actions, such as update, verify, repair, and uninstall are done by other means than scripting.)	Pre- and post-install, update, verify, repair, and uninstall. You can use Java Script to interface with the application installer and the tuner. On Windows, you can also switch contexts, meaning that you can specify whether to run scripts as a user or as a system.
Dependencies	Disk space. Other types of dependencies can be accommodated by using Deployment Manager to create remote system commands to run scripts.	Disk space. Enables you also to specify dependencies for the existence of files and other Marimba channels on both UNIX and Windows. In addition, you can specify dependencies for the CPU, registry entries, memory, and video resolution on Windows only.
File attributes	Replicates attributes from source files.	Enables you to edit the attributes of each file, if necessary, before publishing.
File permissions	On UNIX, replicates permissions for user ID and group ID. On Windows, replicates FAT file system permissions (such as read-only, hidden, and archive).	Replicates all the permissions that Content Replicator does, and in addition, on Windows, replicates NT file system permissions.
Portability	Package on one platform, and deploy to many platforms.	Package for each platform.
Policies	None.	Enables you to specify policies such as, "Install if the file is newer than the file currently on the endpoint." You can apply policies at the package level as well as the file level. You can specify policies for installation, update, uninstallation, verify, and repair.
Locked files (applies to Windows only)	For locally locked files, unlocks DLLs and executables, but cannot unlock some other types of files, such as Word or PowerPoint files. For network locks, unlocks all files. No reboot is needed.	Treats all locks the same by creating a temp file in a temporary location and signaling that a reboot is needed. Once the reboot occurs, the temp file is copied over the original file.

Table E-1: Comparing Content Replicator and File Packager

Feature	Content Replicator	File Packager
Reboot (applies only to Windows tuner versions prior to 8.0)	Cannot trigger a reboot. Reboots generally apply to application installations.	Enables you to specify whether to reboot and what reboot options, if any, to display on the endpoints. (Note: You can also specify reboots if you use the Terminal Services Adapter module.)
Links	On UNIX, replicates symbolic links; allows you to specify whether to “follow the link” and include the file that is referred to. On Windows, treats shell links (shortcuts) like binary files.	On UNIX, replicates symbolic links and hard links. On Windows, you can edit shell links (shortcuts) to specify different values.
INI files (applies to Windows only)	INI files are treated like binary files.	INI files are treated like metadata for ease of updates and merges.
Scheduling	Use Deployment Manager to set schedules.	Enables you to set the update and verify schedule in the package, so that scheduling is done outside of Deployment Manager (not recommended in server environments).
User interface for packaging	Command-line and GUI available through Deployment Manager.	GUI and command-line File Packager and Package Editor user interfaces available on the source machine.
User interface displayed on the endpoint at installation time	Command-line log messages from the endpoint are returned to Deployment Manager, to centrally monitor progress.	Enables you to specify whether you want to display a progress bar, dialog box, or nothing on the endpoint.
Macros	Supports many types of macros.	Supports many types of macros.
Target install directories	Installs all directories under one root directory.	Can install directories under multiple root directories.

Table E-1: Comparing Content Replicator and File Packager

Feature	Content Replicator	File Packager
Preload / Clean	Yes. You can use the -clean option to cause Content Replicator to index the files and directories already present in the installation directory. You can use this option to prevent Content Replicator from downloading duplicate files if the endpoint's installation directory already contains some of the same files that you are intending to install by using Content Replicator.	Yes. If the package you are replicating uses files that are already be on endpoints, you can use the preload property to minimize the number of files that need to downloaded and installed.
Staging	Yes	Yes
Copying to UNC paths/network drives (applies to Windows only)	If the tuner is running as an NT service, cannot write to mapped network drives.	If the tuner is running as an NT service, can write to mapped network drives—if you specify that it run as “system.”
Repair	Repairs both content and attributes.	Enables you to specify whether you want to repair only content, only attributes, or both.
Transmitter-less repair	No	Yes. If the nofilemap property is set to true, can repair by using files in the tuner storage.
Backup of modified or deleted files	No	Yes
Rollback	Yes, with built-in version control.	No

Glossary

Application Installer

A component that appears when a user subscribes to a channel created using Application Packager. It installs the application or content files comprising the channel.

Application Packager

A Symphony Marimba Client Automation application used to package software application (and subsequent updates) into a format that you can then distribute to target endpoints, such as desktops or servers. It provides an interface to a family of packaging components, including Packager for Shrinkwrap Windows Applications, Windows Installer Packager, Custom Application Packager, File Packager, and Java Packager.

assembly

The primary building block of a .NET framework application. It is a collection of functionality built, versioned, and deployed as a single implementation unit (one or multiple files). Assemblies can be static or dynamic:

■ Static assemblies can include .NET framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on). Static assemblies are stored on disk in portable executable (PE) files.

■ Dynamic assemblies run directly from memory and are not saved to disk before execution. You can save dynamic assemblies to disk after they have executed.

An assembly file usually has the file extension .netmodule. It can be compiled as part of an EXE or DLL file.

byte-level differencing

A Symphony Marimba Client Automation feature that allows updates to be performed easily, efficiently, and even automatically. During updates, byte-level differencing means that you do not need to transfer entire files. Only the new or changed bytes are downloaded during updates.

certificate

See “security certificate” on page 452.

certificate authority

An agency from which security certificates can be obtained, either directly or through Certificate Manager.

Certificate Manager

A Symphony Marimba Client Automation application that can be used to obtain security certificates from a certificate authority and to install and manage those certificates.

channel

A format that is used to publish an application or content to a transmitter so that it can be downloaded by a tuner to a desktop or server endpoint. It contains all the information necessary to install the application on endpoints. A channel can be:

- An application of any type (Windows, Java, and so on)
- One or more content files, containing HTML or any data
- A combination of the two previous options

channel category

Any of a number of named sets into which channels are organized in a tuner's channel list. For each category, the channel list shows a colored bar containing the category title.

Channel Copier

A Symphony Marimba Client Automation application that can be used to copy channels (and information about them, such as channel properties) from a transmitter, CAR file, or package directory to a transmitter or CAR file. Using Channel Copier to copy a channel to a transmitter constitutes publishing the channel.

channel directory

See “package directory” on page 450.

channel index

A representation of a channel's file structure, including each file's contents along with other information, such as the file's checksum. The transmitter caches channel indexes for all channels published to it, to optimize its handling of requests for channel updates.

Channel Manager

The channel that provides the user interface to the standard tuner; the tuner's default primary channel.

channel parameter

Information for installing and launching a packaged application. The information in this file is used by the tuner when you run a channel to install and launch the packaged application. See also “parameters.txt” on page 450.

channel property

One of many characteristics of a channel that provide information about the channel, such as who created or published it, what its URL is, when it was last updated, and when the next update is scheduled. Through the tuner, a user can view channel properties and set some of them; additional channel properties can be set by an administrator or a channel publisher. See also “channel parameter” on page 446 and “tuner property” on page 454.

channel segment

See “segment (of a channel)” on page 452.

channel-signing certificate

A security certificate, indicating who published a particular channel, that's installed in the tuner from which the channel is being published and is used to digitally sign the channel when it's published.

channel update

The synchronization of a tuner's subscribed channel with new channel data from the transmitter, made by the tuner either automatically according to an update schedule or specifically at the user's request.

channel URL

A channel's uniform resource locator, which is assigned when the channel is published and provides a unique identifier for that channel on the transmitter. By specifying the channel URL, a user can subscribe to the channel.

checkpoint restart

A Symphony Marimba Client Automation feature that enables the downloading of a channel (or channel update) to continue seamlessly when the tuner restarts after an interruption, as might be caused by a power outage or the loss of a network connection.

command line

An interface provided by most products that enables commands to be entered, either at a command prompt or in a script or batch file; useful for automating the entry of complex commands.

Content Replicator

Content Replicator is a Symphony Marimba Client Automation Server Management component that does the work of taking data files from a source system, sending the content to a data server (that is, a transmitter), and then previewing, staging, installing, or rolling back the content on destination machines. Content Replicator has the following capabilities, among others:

- Takes directories from a source system on one platform and publishes them in a format that allows for deployment on Windows NT, Windows 2000, and Linux operating systems
- Downloads and activates content and applications on one or more target servers
- Can install applications and data files regardless of whether some files are locked on the target server; can also unlock shared network resources
- Allows you to control the rate at which content is deployed, to accommodate bandwidth connections that vary, from high-speed backbone access to fractional T1 speeds

Custom Application Packager

An Application Packager component that allows you to package any application, especially ones developed by companies for internal use.

dependency

A requirement that you want to check at the endpoint before downloading and installing a channel. For example, you might want to require that a certain file or a certain channel exists at the endpoint. If dependencies cannot be validated, the channel is not downloaded, installation of the channel fails, and, depending on how the channel is configured, the user at the endpoint is informed of the failure.

File Packager

An Application Packager component that allows you to package a collection of files (for example, spreadsheets, HTML files, or templates).

file reference

An Application Packager feature that allows you to publish the contents of a file directly to the transmitter instead of adding it to the package directory first. Using file references allows you to save disk storage space and to update files without having to replace them manually.

global assembly cache (GAC)

For .NET framework applications, a machine-wide code cache located on each computer where the common language runtime is installed. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer. See also “assembly” on page 445.

history entry

An annotation in the package that allows you to keep a record or history of the changes to a channel. A history entry might be especially useful if multiple users are editing a package. The following fields are available for each history entry:

- Timestamp
- Author
- E-mail
- Phone
- Subject
- Notes

Java Packager

An Application Packager component that allows you to package a Java application and set options, such as using a different Java virtual machine (JVM).

just-in-time (JIT) application deployment

An operation for ensuring that an application is updated or repaired before it is used. Using the Package Editor, you can configure the Windows shortcut used to start an application, so that before the application starts, one of the following operations takes place:

- The application gets updated.
- The application gets verified, and if necessary, repaired.
- The application gets both updated and repaired.

These operations are performed before the application is started, whenever the shortcut is used. Using this feature allows you to seamlessly update and repair an application, without requiring users at the endpoints to manually perform any operations, other than double-clicking the shortcut.

macro

An Application Packager feature that allows you to customize and replace items in a channel, such as a file name or directory path. There are some predefined macros included with Application Packager, and you can also create user-defined macros.

major update

An update where the contents of the channel are changed. A major update means files, registries, environment variables, or text modifiers have been added, changed, or removed from the channel. For files, content changes would include any changes in file attributes also.

manifest file

A file that contains the code and instructions required to install and update an application packaged as a channel.

master transmitter

The transmitter that has channels which a repeater or mirror transmitter replicates.

merge modules

Pre-compiled bundles of components (files, registry entries, and other modules) that enable developers to easily add third-party features to a Microsoft installation (MSI). See also “[MSI package](#)” on page 449.

metabase

A structure similar to the registry that Microsoft’s Internet Information Server (IIS) uses to store information. Although it resembles the registry, the metabase is more sophisticated and has optimized information retrieval capabilities. Like the registry, the IIS metabase contains key nodes and value nodes.

Microsoft installation (MSI) package

This consists of at least one file with the .msi extension, which contains an installation database, a summary information stream, and data streams for various parts of the installation. An MSI package can also contain one or more transforms, internal source files, and external source files or cabinet files required by the installation. See also “[Windows Installer](#)” on page 454.

minor update

An update where the contents of the channel are not changed. A minor update means that the channel changed, but there are no new, deleted, or changed files, registry values, or environment variables.

mirror

To replicate channels from a master transmitter to another transmitter for the purpose of balancing the tuner request load; see “[mirror transmitter](#)” on page 449.

mirror transmitter

A transmitter that replicates channels from a master transmitter for the purpose of balancing the tuner request load, but that (unlike a repeater) does not have tuner requests redirected to it from the master transmitter. Users’ tuners subscribe to channels directly on the mirror transmitter.

MSI package

See “[Microsoft installation \(MSI\) package](#)” on page 449.

.NET Packager

A component of Application Packager that allows you to package applications that have been created using Microsoft’s .NET framework. The .NET framework is Microsoft’s platform for building, deploying, and running Web services and applications.

package (a channel)

To prepare some kinds of applications and content for publishing as a channel, as required by the transmitter. Java application or applet channels and HTML content do not require packaging before they can be published. See also “Application Packager” on page 445 and “channel” on page 446.

package directory

The directory in which Application Packager stores the files for a packaged application. A package directory is created when you use any of the packager components of Application Packager to package a software application. It is also sometimes referred to as the channel directory or publish directory.

Channel directory is also sometimes used to see the subdirectory of the tuner’s workspace directory in which the files for a particular channel are stored.

Package Editor

A component of Application Packager that allows you to add, edit, and remove files, directories, and registry entries in packaged applications. You can also use the Package Editor to set macros, customize environment settings, add installation scripts, and so on.

Packager for Shrinkwrap Windows Applications

An Application Packager component that allows you to package any commercial 32-bit Windows software application.

parameters.txt

A property file that contains channel parameters and is located in the package directory for that channel. It stores information for installing and launching a packaged application. The information in this file is used by the tuner when you run a channel to install and launch the packaged application.

patch files

Storage files for MSI packages that contain at least one database transform that adds patching information to the database of its target installation. They typically have the .msp file extension. Patches can be applied to more than one application or can upgrade an application into another application or version.

See also “Microsoft installation (MSI) package” on page 449.

PDA Packager

An Application Packager component that allows you to package either directories of files or PDA (personal digital assistant) applications in the form of CAB files.

policy (for installation, update, uninstallation, verify, and repair)

An Application Packager feature that determines whether files and the other contents of a channel are installed, updated, uninstalled, verified, or repaired. You can set policies at two levels:

- You can set policies as the default for all the contents of a channel.
- You can also override the default policies and set policies for individual items in a channel.

prefs.txt

A property file in the tuner's workspace directory in which tuner properties can be set when the tuner is installed, or in some cases after installation.

properties.txt

A property file in any of several locations applying to channels, tuners, or transmitters (for example, in a package directory, the tuner's workspace directory, or the transmitter's workspace directory).

property

See "channel property" on page 446 or "tuner property" on page 454.

property file

A file containing key-value pairs that define characteristics of a channel, tuner, or transmitter, depending on the name of the file and where it's stored. Channel properties and parameters, tuner properties, and transmitter properties are stored in property files.

publish (a channel)

To copy a channel (or channel update) to a transmitter in a way that enables it to be downloaded by a tuner. This operation (which includes copying information about the channel, such as channel properties) can be performed with Channel Copier or, as an alternative that users of earlier versions of Symphony Marimba Client Automation products might favor, with Publisher.

publish directory

See "package directory" on page 450.

Publisher

A Symphony Marimba Client Automation application that can be used to publish channels to a transmitter, as an alternative to Channel Copier; sometimes favored by those who have experience with earlier versions of Symphony Marimba Client Automation products.

repair

An action performed on a channel to check and fix problems. When you repair a channel, the files and other objects in the channel are first verified. After verification is complete, one of the following occurs:

- There are no object mismatches. No repairs are needed.
- There are object mismatches. The tuner re-installs the affected files or registry entries if they are available from the tuner storage. If not, the tuner contacts the transmitter to download the files or registry entries needed to complete the repair.

See also "verify" on page 454.

repeater

A transmitter on which channels from a master transmitter have been replicated, and to which requests made by tuners can be redirected from the master transmitter.

replication

The transfer of channels from a master transmitter to a repeater or mirror transmitter. This is just a duplication of the channels and not the same operation performed by Channel Copier, which publishes channels when it copies them to a transmitter.

roll (a log file)

To clear a log file before filling it with new data.

rollback install

An action triggered when an installation fails, causing the channel to be in a failed state. As a result, the channel ends up in an uninstalled state. All items are removed or restored to their previous states regardless of uninstallation policies and properties (such as nobackup). However, no scripts are executed and no dependencies are checked.

rollback update

An action triggered when a “major update” fails causing the channel to be in failed state. As a result, the channel ends up in a rollback state. This state indicates that the channel update has failed and has been brought back to its previous state. However, this state cannot successfully verify, repair, or install. It can only be “updated” with the correct version of the channel to bring it back into an “installed” state. During the rollback of an update, all files overwritten are temporarily backed up so they can be restored. Like rollback install, no scripts are executed and no dependencies are checked during this phase.

script

A file in the form of batch files, executables, or Java classes that you can use to customize the behavior of your channel on the endpoint. Scripts are invoked at key times in your channel’s lifecycle (install, uninstall, update, verify, repair, execute, or a combination of these phases). There can be any number of scripts in a channel.

security certificate

A mechanism for ensuring that a channel being subscribed to comes from the proper source (“channel-signing certificate” on page 447).

segment (of a channel)

A platform-specific or localized version of a channel. Every channel is made up of one or more segments; the transmitter determines which one to send based on feedback it received from the tuner.

segment ID

A string that identifies the specific platform and locale for which a channel segment is intended—for example, Windows NT,x86/en_US.

semisilent installation

A mode of installation (of a packaged channel) that provides user feedback in the form of progress bars but does not offer any installation options.

services

Programs or processes that perform a specific system function, usually as background tasks, to support other programs on Windows NT, Windows 2000, and Windows XP. Typically, they do not require user interaction and do not have a user interface. Some Windows applications include and install services.

signed channel

A channel (or channel update) that has been digitally signed by means of a channel-signing certificate, guaranteeing subscribers that it comes from a reliable source and is not an unauthorized, possibly corrupted, copy. When the user subscribes to a signed channel, a dialog box appears that requests permission to download the channel.

silent installation

A mode of installation (of a packaged channel) that provides no user feedback and offers no installation options. It displays no dialog boxes or progress bars to the user. The installer uses all of the default channel settings during installation.

subscribe (to a channel)

To request a channel from a transmitter via a tuner, causing the channel's files (and subsequent channel updates) to be downloaded to the workstation on which the tuner is located.

Policy Management (previously called Subscription)

A set of components that enables channels to be assigned to users or machines through a centrally located policy. See “Policy Manager (previously called Subscription Policy Manager)” on page 453.

Policy Manager (previously called Subscription Policy Manager)

An application found in the Symphony Marimba Client Automation console that an administrator runs to assign channels to users, machines, or groups of users or machines as part of a policy.

template file

See “XML template file” on page 455.

text modifier group

Contains all the text changes that you want to make to one ASCII text file. The text modifier group specifies two things:

- The name of the text modifier group
- The path of the ASCII text file to be modified

See also “text modifier” on page 453.

text modifier

An Application Packager feature that allows you to insert and replace text in ASCII text files. The files to be modified can be part of your channel or they can be any files on the endpoint.

transform files

Files for MSI packages that the Windows Installer uses to modify the installation database at installation time and dynamically affect the installation behavior. Transform files typically have a .mst file name extension. Transform files are applied at initial installation; they cannot be applied to an already installed application.

See also “Microsoft installation (MSI) package” on page 449.

transmitter

The Symphony Marimba Client Automation server component, it is essentially a server that delivers applications and content in the form of channels. It is similar to a Web server, but instead of serving Web pages, it serves channels. Channels (and channel updates) are published to a transmitter and downloaded from there to its clients (either tuners or other transmitters running as repeaters or mirror transmitters). The transmitter is itself a channel that runs on the tuner or some other tuner under the control of an administrator.

Transmitter Administrator

A channel that an administrator uses to configure one or more transmitters.

trusted channel

A channel that has been given permission to read or write data anywhere on the user's system and to call any code, including operating system code. Channels must be signed to be designated as trusted.

tuner

The Symphony Marimba Client Automation client component, it is the interface through which Symphony Marimba Client Automation components interact. The tuner is the application through which users subscribe to channels that have been published on a transmitter. The tuner downloads the channel files (or updates to them) from the transmitter to the user's workstation.

Tuner Administrator

A Symphony Marimba Client Automation application that allows you manage tuners remotely, controlling which channels appear on tuners or changing their configuration in other ways.

tuner property

One of many characteristics of a tuner, including the URL of its primary channel, the URL for tuner updates, and the update schedule for its channels, as well as proxy details. Tuner properties can be set when the tuner is configured with Tuner Packager, and some properties can also be set during or after tuner installation.

update schedule

The periodic timing with which updates are automatically delivered to channels that have been subscribed to and downloaded. This schedule is set up either when the channel is published or later at the user's request.

verify

An action performed on a channel to check if there are any problems. When you verify a channel, the files and other objects in the channel are compared with the information for the channel when it was originally installed. Any object mismatches found are recorded in the log file.

WFP

See "Windows File Protection (WFP)" on page 454.

Windows File Protection (WFP)

A feature in the Windows 2000 product family for protecting system files to make sure that only the owners or vendors of those files can modify them.

Windows Installer

An installation and configuration service that ships as part of the Microsoft Windows 2000 operating system; it can also be installed on Windows 95/98 and Windows NT 4.0. See also "Microsoft installation (MSI) package" on page 449.

Windows Installer Packager

An Application Packager component that enables you package Microsoft installations (MSI) that were created for the Microsoft Windows Installer.

Windows Terminal Services (WTS)

Services that provide clients remote access to applications that run on a server. Applications are installed on the server only, and clients access the applications through WTS client software. WTS transmits only the user interface of the application to the client. The client then returns keyboard and mouse clicks to be processed by the server.

workspace directory

The directory on the user's workstation that is, by default, where the tuner stores channel files (in called channel directories) and that is normally the only place where files accessed by channels can be located. For a transmitter, this term refers to the directory where the transmitter stores data and channels; by default it's a subdirectory of the transmitter's channel directory, but it's typically configured to be a different directory. Similarly, a proxy has a workspace directory where it stores data, including its channel cache.

WTS

See “Windows Terminal Services (WTS)” on page 454.

XML template file

A file that allows you to save, modify, and apply default settings and configurations for Application Packager and channels created using Application Packager. These settings include installation policies, macros, scripts, installation modes, and other settings that can apply to multiple applications.

Index

A

ActiveX controls 267
Administrative Installation Point. *See AIP*
advertising Windows Installer applications 198, 199
AIP 207
 performing for Windows Installer packages 198
AND dependencies 140
Application Installer 30
Application Packager
 See also names of individual components
 about 24
 benchmarks 31
 command line 307–329
 components of 24–26
 graphical user interface (GUI) 27
 importing packaged applications 64
 preparing to use 60
 starting 62
 Windows Installer applications, about 192
Application Policy Configuration page 238
applications
 custom, packaging 266
 editing with Java Packager 259
 editing with Package Editor 67
 importing into Application Packager 64
 managing with BMC Configuration Management 54
 policy configuration file 238

 starting from command line 327
-applytemplate command-line option 325
archives in UNIX, cpio 61
arguments, passing to JVM 258
ASCII files. *See* text files
assemblies
 about 232
 application policy configuration file 238
 binding 239
 codebase setting 241
 creating 233
 dependent 240
 development mode 242
 dynamic 232
 garbage collection 240
 locations 239
 manifest 233
 .netmodule file extension 233, 445
 partial names 243
 policy configuration options 237
 probing 239
 properties 237
 publisher policy for 239, 242
 qualifiers 243
 redirection for different versions 241
 registering into GAC 238
 specifying subdirectories for CLR to search 239
 static 232
 strong names 243
 version redirection 239

- XML namespace for assembly binding 239
- B**
- backups
- channels, creating 283
 - Content Replicator versus File Packager 444
 - preventing items from being stored during installation 109
- bandwidth, minimizing use of 55, 159
- benchmarks for Application Packager 31
- Best Practice icon 19
- best practices
- matching repair policies and verification policies 124
 - Verify not installed policy 338
- BMC Configuration Management
- managing applications with 54
- byte-level differencing 27
- C**
- CAB files
- MSI databases with 195
 - packaging 250
 - PDA Packager and 26, 248
- cabfile command-line option 321
- cabinet files
- packaging 320
- cabinet files. *See CAB files*
- certificate for channel signing 63
- change history
- See also* history entries
 - information available 184
 - recording for channels 184
- Channel Copier 280, 322
- channelname command-line option 319
- channels
- about 26
 - adding
 - directories to 82
 - files to 71
 - files to, by reference 72
 - metabase entries to 87
 - registry entries to 84
 - shortcuts to 77
 - applying XML template files 276
- backing up 283
- benefits 27
- command line 310
- configuring
- dependencies for 130
 - requirements for 138
- creating 51
- customizing
- with executable scripts 141
 - with Java classes 145
 - with scripts and Java classes 141
 - with Windows Installer Packager 196
- dependencies
- AND 140
 - configuring 130
 - OR 140
- editing 62, 70
- directory properties 82
 - symbolic links in 78
 - with .NET Packager 236
 - with Java Packager 259
 - with Package Editor 67
- environment variables and 127
- history in log files 112
- importing 64
- installation policies for 121
- managing updates to applications 52
- naming 51
- opening for editing 69
- packaging
- custom applications into 266
 - files into 222
 - Java applications into 255, 263
 - MSI applications into 191
 - .NET applications into 234
 - PDA applications into 248
 - shrink-wrapped Windows applications into 38
- parameters 139
- editing 158
 - parameters.txt file 29
- policies for individual items in 124
- policy defaults for 123
- properties.txt file 29
- publishing 63, 279–284

- recovering from corrupted files 187
- removing
 - files from endpoints during uninstallation of 127
 - from list 64
 - items from 92
 - items from endpoints in 93
- removing files during uninstallation of 126
- renaming items in 92
- repackaging from command line 312
- repair policies for 121
- repairing 165
 - by schedule 167
 - manually 168
- requirements, configuring 138
- restoring files during uninstallation of 126
- reverting to previously saved version 187
- running manual verification and repair of 168
- saving
 - changes to 186
 - copy in different directory 187
 - to network drives 186
 - XML template files 270
- scheduling verification and repair 167
- signing certificates for 60, 63
- starting from command line 327
- startup options for 104
- uninstallation policies for 121
- update policies for 121
- updating 53, 150, 325
 - with Java Packager 259
- verification policies for 121
- verifying 165
 - before publishing and distributing 166
 - by schedule 167
 - manually 168
- classes. *See Java classes*
- classpath for Java applications 255
- CLR
 - about 232
 - specifying subdirectories to search 239
- codebase setting, for assemblies 241
- command line
 - about 308
 - Application Packager 307–329
- channels 310
- Custom Application Packager 324
- File Packager 313
- .NET Packager 315
- Packager for Shrinkwrap Windows Applications 309
- PDA Packager 319
- runchannel program 308
- starting packages 327
- Windows Installer Packager 198, 214, 311
- XML template files 325
- common language runtime. *See CLR*
- Configuration Management, managing applications with 54
- configuration text files, editing 173
- CONFIGURATION XML tag 380
- container objects
 - marking for removal 94
 - policies for 122, 124, 342
- Content Replicator, compared with File Packager 439–444
- corrupted files in channel, recovering from 187
- cpio archive in UNIX 61
- Custom Application Packager 265–268
 - about 26, 266
 - command line 324
 - packaging custom applications into channels 266
 - selecting platforms 267
- custom applications, packaging 266
- CUSTOMIZATION XML tag 396
- customizing
 - channels
 - with executable scripts 141
 - with Java classes 145
 - with scripts and Java classes 141
 - error codes 201
 - script error codes 201
 - snapshots 46
 - Windows Installer packages 196
- CUSTOMMACRO XML tag 372
- custompackage command-line option 324

D

- database file, Windows Installer 190

-DDEBUGFLAGS debugging option 302
debugging 302
Definitive Software Library. *See* **DSL**
delayfiledownload channel parameter 160
delaying downloads during installation 109
deleting. *See* removing
DEPEND_CHANNEL XML tag 394
DEPEND_CHANNELS XML tag 394
DEPEND_DISKSPACE XML tag 390
DEPEND_DRIVE XML tag 391
DEPEND_FILE XML tag 391
DEPEND_FILES XML tag 391
DEPEND_MEMORY XML tag 389
DEPEND_PROCESSOR XML tag 389
DEPEND_REGISTRY XML tag 392
DEPEND_REGKEY XML tag 393
DEPEND_RESOLUTION XML tag 390
dependChannelSilent channel parameter 139
dependencies
 AND 140
 configuring for channels 130
 configuring for system 130
 major updates and 151
 minor updates and 151
 OR 140
 types 130
DEPENDENCIES XML tag 388
dependent assemblies 240
deploying PDA packages to mobile devices 252
Deployment Manager 27
development mode, for assemblies 242
DEVPATH environment variable, for
 assemblies 242
directories
 adding
 for snapshots 41
 to channels 82
 to file packages 223
 to .NET packages 234
 to PDA packages 249
 allowing users to select for installation 102
 Content Replicator versus File Packager 443
 editing properties of 82
 ignoring for snapshots 42
 marking for removal 94
 policies for 122, 342
 removing from snapshots 42
 virtual, adding for IIS 244
DIRECTORY XML tag 400, 430
disk space requirements, configuring 132
distributing channels, verifying before 166
DLL files
 adding to Custom Packager channels 267
 Content Replicator versus File Packager 442
 locked 114
 marking for registration 74, 76
 Windows File Protection and 57
documentation
 organization of 18
-dotnetpackage command-line option 315
downloading files
 byte-level differencing 27
 delaying during installation 109, 160
 long file names 195
 MSIDOWNLOAD XML tag 416
 options for 207
drives
 adding for snapshots 41
 ignoring for snapshots 42
 removing from snapshots 42
DSL, adding packages to 171
duplicate entries, in INI files 80
duplicate sections, in INI files 79
dynamic assemblies 232

E

editing
 ASCII text files 173
 channel parameters 158
 channels 62, 70
 with .NET Packager 236
 with Java Packager 259
 configuration files 173
 contents of MSI packages 213
 directory properties 82
 environment variables 129
 filter settings 45
 filters 44
 history entries 185
 macros 103

- metabase entry properties 88
 - registry 46
 - registry value data 85
 - symbolic links 78
 - Windows services 152
 - endpoints
 - removing shortcuts from 343
 - staging MSI applications 329
 - ENV_PROPERTY XML tag 374
 - environment variables
 - adding to channels 128
 - channels and 127
 - creating macros for 96
 - editing 129
 - macros and 100
 - removing from channels 129
 - Windows platforms 130
 - ENVIRONMENT XML tag 374
 - environment, preparing for packaging 39
 - error messages
 - customizing 201
 - suppressing during installation 201
 - exec command-line option 327
 - executable scripts, customizing channels with 141
- F**
- File Packager 221–229
 - about 25, 222
 - adding multiple files 224
 - command line 313
 - compared with Content Replicator 439–444
 - installation directory 224
 - packaging
 - files into channels 222
 - platform issues 222
 - updating channels 226
 - file systems
 - adding for snapshots 41
 - finding, with Package Editor 94
 - ignoring for snapshots 42
 - removing from snapshots 42
 - FILE XML tag 402
 - filebyref command-line argument 320
 - filepackage command-line option 313
 - files
 - ASCII. *See* text files
 - CAB. *See* CAB files
 - DLL. *See* DLL files
 - HTML. *See* HTML files
 - INI. *See* INI files
 - text. *See* text files
 - adding to channels 71
 - adding to channels, by reference 72
 - benefits of adding by reference 72
 - editing properties of 73
 - limitations of adding by reference 72
 - packaging from command line 313
 - repackaging from command line 314
 - requirements for channels 133
 - response, for SVr4 packages 145
 - troubleshooting 298
 - user-specific 161
 - viewing contents of INI and ASCII text files 76
 - files directory 29
 - FILESYSTEM XML tag 429
 - FILTER XML tag 430
 - filters for snapshots
 - adding file system items 42
 - editing 44, 46
 - editing default settings for 45
 - filtering metabase keys from 48
 - macros 44
 - removing 44
 - saving with XML template files 273
 - XML template files for 272
 - finding items in Package Editor 94
 - full installation mode 106
- G**
- GAC
 - about 233
 - registering assemblies into 238
 - garbage collection, for .NET applications 240
 - generic XML tags 370–399
 - CONFIGURATION 380
 - CUSTOMIZATION 396
 - CUSTOMMACRO 372
 - DEPEND_CHANNEL 394
 - DEPEND_CHANNELS 394
 - DEPEND_DISKSPACE 390

- DEPEND_DRIVE 391
DEPEND_FILE 391
DEPEND_FILES 391
DEPEND_MEMORY 389
DEPEND_PROCESSOR 389
DEPEND_REGISTRY 392
DEPEND_REGKEY 393
DEPEND_RESOLUTION 390
DEPENDENCIES 388
ENV_PROPERTY 374
ENVIRONMENT 374
INSTALLER 381
MACRO 373
PACKAGE 372
PARAMETER 399
PLATFORM 380
POLICIES 384
PROPERTY 398
REBOOT 383
SCRIPT 397
SCRIPTS 396
SERV_ADVANCED 378
SERV_DEPEND 378
SERV_GENERAL 376
SERV_POLICIES 379
SERV_SECURITY 377
SERVICE 375
SERVICES 375
STARTUP 385
VERIFYREPAIR 387
- Global Assembly Cache. *See* GAC
- H**
- help, online 34
history
 adding new entries 185
 editing entries 185
 information available 184
 log files 301
 removing entries 186
HTML files
 File Packager and 222
 PDA Packager and 248
- I**
- ignore list. *See* filters for snapshots
IIS metabase
 about 47
 adding entries to channels 87
 adding metabase keys to snapshots 48
 adding virtual directories to 244
 editing metabase entry properties 88
 editing metabase value data 89
 enabling capturing changes for snapshots 48
 enabling Microsoft IIS Metabase support 46
 filtering metabase keys from snapshots 48
 macros for 91
 metabase properties for 90
 removing metabase keys from snapshots 48
 selecting metabase entries for snapshots 47
importing channels into Application Packager 64
- INI files
 about 78
 applying macros 97
 Content Replicator versus File Packager 443
 duplicate entries in 80
 duplicate sections in 79
 enabling parsing 47
 handling with Application Packager 78
 policies 80, 352–356
 repair operation for duplicate entries 81
 uninstallation for duplicate entries 81
 viewing contents of 76
- insert line modifiers
 about 173
 creating 175
- install command-line option 327
- installation
 See also installation policies
 See also uninstallation
 about 30
 as is 198
 Content Replicator versus File Packager 443
 creating macros that allow users to select
 directory for 102
 delaying downloads during 109
 directory
 for File Packager channels 224
 for .NET Packager channels 234

- for PDA Packager channels 249
 - for Packager for Shrinkwrap Windows Applications 50
 - for Windows Installer packages 197
 - full mode 106
 - log files, enabling 111
 - minimizing use of bandwidth during 159
 - modes for applications 106
 - modifiers 174
 - performing administrative installation for Windows Installer packages 198
 - platforms for channels 117
 - preventing items from being stored for backup during 109
 - preview mode 107
 - rollback for failed installations 108
 - semi-silent mode 31, 107
 - silent mode 31, 107, 139, 200, 205, 328
 - troubleshooting 296
 - installation policies 155
 - See also* installation
 - channels 121
 - directories and files 344–345
 - duplicate entries in INI files 80
 - general 332–333
 - INI files 353
 - registry and metabase keys 357
 - selecting for channels 121
 - text modifier groups 181
 - text modifiers 181, 182, 367
 - Windows NT services 362
 - INSTALLER XML tag 381
 - Internet Information Services (IIS). *See* IIS metabase
 - inverting items in Package Editor 93
 - IScript interface for customizing channels 145
- J**
- Java
 - applications, packaging 253
 - classes, customizing channels with 141, 145
 - Java Packager 253–259
 - about 26, 254, 262
 - channel signing certificate 258
 - editing channels 259
- packaging an application 255, 263
 - passing arguments to JVM 258
 - troubleshooting 300
 - updating channels 259
 - Java Virtual Machine. *See* JVM
 - just-in-time application deployment 169
 - JVM
 - about 254, 262
 - Java Packager support for 254, 262
 - JVM selection 257
 - passing arguments to 258
 - run mode 256
- L**
- links, symbolic, editing in channels 78
 - log files
 - Application Packager 301
 - channel history 112
 - endpoints 301, 302
 - Package Editor 111
 - rollover options for 111
 - troubleshooting 296
 - Windows Installer 212
- M**
- MACRO XML tag 373
 - macros
 - about 95
 - allowing users to select installation directory in 102
 - applying in INI files 97
 - case-sensitivity 96
 - Content Replicator versus File Packager 443
 - creating MSI (Microsoft installation)
 - macros 97
 - editing 103
 - environment variable 96
 - environment variables 100
 - examples of 98
 - filters and 44
 - for Windows 433–437
 - in scripts 97, 141
 - in text modifiers 173, 176
 - metabase keys and values 91
 - package property 97

- registry keys in 98
- registry value 96
- removing 103
- tuner properties in 101
- tuner property 97
- types 96
- user-defined, about 96
- Windows Installer property 97
- manifest file 30, 110, 161, 382
- merge modules, MSI databases with 196
- metabase
 - about 47, 86
 - adding
 - entries to channels 87
 - metabase keys to snapshots 48
 - editing
 - metabase entry properties 88
 - metabase value data 89
 - enabling capturing changes for snapshots 48
 - filtering metabase keys from snapshots 48
 - macros and 91
 - properties for 90
 - removing metabase keys from snapshots 48
 - selecting metabase entries for snapshots 47
- METABASE XML tag 431
- METABASEKEY XML tag 408
- METABASEVALUE XML tag 409
- METKEY XML tag 432
- Microsoft installation (MSI). *See* MSI
- minimizing use of bandwidth 159
- mobile devices. *See* PDAs
- MODIFIER XML tag 411
- MODIFIERGROUP XML tag 410
- modifiers. *See* text file modifiers
- MODIFY_OBJECT XML tag 418
- MODIFY_OBJECT_ATTRIBUTES XML tag 420
- MSI
 - about 191
 - creating macros 97
 - database 191
 - packaging 192
 - packaging by reference 193
 - packaging with CAB files 195
 - packaging with merge modules 196
 - editing contents of packages 213
- error codes 201
- file 191
- macros in scripts 97, 141
- packaging
 - about 25
 - with patch files 204
- staging applications on endpoints 329
- troubleshooting 299
- Windows Installer property macros 97
- XML tags 414–418
 - MSI_DOWNLOAD 416
 - MSI_INSTALLATION 414
 - MSI_LOGGING 417
 - MSI_PATCH 415
 - MSI_PATCHES 415
 - MSI_POLICIES 417
 - MSI_TRANSFORM 416
 - MSI_TRANSFORMS 416
 - MSI_UILEVEL 415
- .msi file name extension 190
- msiexec.exe program 190
- .msp files
 - file name extension 191
 - for snapshots 40
- .mst file name extension 191, 453
- multiple-user feature 163

N

- naming channels 51
- .NET applications
 - assemblies. *See* assemblies
 - channels, editing 236
 - enabling processing of features of 47
 - framework 25, 232, 449
- .NET Packager
 - adding virtual directories for IIS 244
 - updating channels 244
 - command line 315
 - installation directory 234
 - about 25, 232
 - assembly properties 237
 - components of applications 232
 - editing channels 236
 - creating new channels 234
- .netmodule file extension 233, 445

network drives

- Content Replicator versus File Packager 444
- saving channels to 186
- non-MSI XML tags 400–413
 - DIRECTORY 400
 - FILE 402
 - METABASEKEY 408
 - METABASEVALUE 409
 - MODIFIER 411
 - MODIFIERGROUP 410
 - REGKEY 406
 - REGVALUE 407
 - SHORTCUT 403
 - SYMLINK 405

O

operating system

- registry implications 46
- selecting 117

OR dependencies

P

Package Editor

- about 25, 69
- bandwidth use and 159
- change history for channels and 184
- channel parameters and 158
- channel startup options 104
- DSL and 171
- editing
 - channels 62, 70, 236
- enabling log files for installation 111
- environment variables in 127
- installation modes 106
- installation platforms, specifying 117
- macros in 95
- major and minor updates in 150
- opening channels for editing 69
- opening Windows Installer packages in 197
- policies in 121
- publishing channels 63
- recovering from channel corruption 187
- saving changes to channels 186
- scripts and classes in 141
- shortcuts and 169

system and channel dependencies

- in 130
- system reboot options in 114
- text files and 173
- updating and repairing applications 169
- user-specific files and 161
- user-specific registry entries and 161
- verifying and repairing channels 165
- Windows services and 152

package property macros

- PACKAGE XML tag 372
- packagedir command-line option 319

Packager for Shrinkwrap Windows

- Applications 37–58
 - about 24, 38
 - command line 309
 - creating snapshots for 39
 - installing application between snapshots 50
 - managing applications with BMC Configuration Management 54
 - managing updates to applications 52
- packaging
 - applications into channels 51
 - the removal of applications 55

preparing

- environment for 39
- to install 50

- restrictions with Windows File Protection 57
- XML template files and 272

packagers

See also individual components

- components of 24–26
- Custom Application Packager 26, 265–268
- File Packager 25, 221–229
- Java Packager 26, 253–259
- .NET Packager 25, 231–246
- Package Editor 25, 67–188
- Packager for Shrinkwrap Windows
 - Applications 24, 37–58
- PDA Packager 26, 247–252, 261–??
- Windows Installer Packager 25, 189–219

packages

- adding to DSL 171
- multiple-user feature and 163
- package directory 28
- package directory in PDA Packager 30

troubleshooting 297

packaging

- applications into channels 51
- as part of software distribution process 26
- Content Replicator versus File Packager 443
- files 222
- Java applications 253
- MSI databases 192
 - by reference 193
 - with CAB files 195
 - with merge modules 196
- .NET applications 234
- sets of files 222
- shrinkwrapped Windows applications 38
- Windows Installer applications 192

PARAMETER XML tag 399

parameters, channel

- delayfiledownload 160
- dependChannelSilent parameter 139
- editing 158
- parameters.txt file 29
- preload 160

parameters.txt file 29

parsing INI files, enabling 47

partial names, for assemblies 243

patch files

- about 191
- default location 198
- packaging with MSI 204

paths for scripts 144

PDA Packager 247–252, 261–???

- about 26, 248
- command line 319
- deploying to mobile devices 252
- installation directory 249
- packaging files and applications 248
- updating channels 251

-pdapackage command-line option 319

PDAs

- deploying PDA packages to 252

PE files 232

personal digital assistants. *See* PDAs

PLATFORM XML tag 380

platforms

- about 32

Custom Application Packager channel 267

selecting 117

UNIX 33

Windows 32

plugin directory 29

policies 331–367

See also individual types of policies

Content Replicator versus File Packager 442

defaults 123

directories and files 342–356

general 332–341

registry and metabase 356–360

setting for individual items in 124

text modifiers 366–367

Windows NT services 361–366

POLICIES XML tag 384

Policy Manager 27

- multiple-user feature and 163

portable executable (PE) files 232

postinstall snapshots. *See* snapshots, postinstall

post-operation scripts 144

preinstall snapshots. *See* snapshots, preinstall

preload channel parameter 160

preloading, Content Replicator versus File Packager 444

preview installation mode 107

previous version of channels, reverting to 187

properties

- advanced, for Windows services 154
- general, for Windows services 153
- properties.txt file for channels 29
- security, for Windows services 154

PROPERTY XML tag 398

publish directory. *See* package directory

publisher policy, for assemblies 242

publishing

- channels 279–284
 - scheduling 282
 - to transmitters 280

File Packager channels 227

publishing channels

- about 63
- assemblies 239
- File Packager 226
- Java Packager 257

- .NET Packager channels 236
 - PDA Packager channels 251
 - software distribution process and 26
 - verifying before 166
 - Windows Installer Packager channels 193
- Q**
- qualifiers, for assemblies 243
- R**
- REBOOT XML tag 383
 - reboots
 - allowing cancellation of 115
 - Content Replicator versus File Packager 443
 - detecting when necessary 114, 115
 - forcing 114
 - options 114
 - preventing 115
 - showing dialog boxes 114
 - silent installation mode and 115
 - triggering with scripts 116
 - reconfiguring
 - File Packager channels 226
 - redirection
 - assembly versions 241
 - in Windows Installer 216
 - refreshsourcelist command-line option 327
 - registering DLL files 74
 - about 76
 - registry
 - about 83
 - adding entries to channels 84
 - configuring requirements 136
 - editing 46
 - entry properties 85
 - value data 85
 - files supported for importing 84
 - finding entries with Package Editor 94
 - importing entries from registry file 84
 - keys
 - in macros 98
 - marking for removal 94
 - policies 122, 342
 - selecting what to include and exclude in snapshots 45
 - user-specific entries 161
 - value macros 96
 - .reg files
 - supported for importing 84
 - REGISTRY XML tag 431
 - REGKEY XML tag 406, 431
 - REGVALUE XML tag 407
 - reinstallation 198
 - remove command-line option 327
 - removing
 - applications from endpoints 55
 - channels from list 64
 - environment variables 129
 - files
 - associated with an application 55
 - during uninstallation 126
 - from endpoints during uninstallation 127
 - filters from filter list 44
 - history entries 186
 - items
 - from channels 92
 - from endpoints in channels 93
 - registry keys associated with an application 55
 - shortcuts from endpoints 343
 - transforms from Windows Installer packages 205
 - Windows services from channels 158
 - renaming items in channels 92
 - repackage command-line option 312, 314, 317, 319
 - repackaging
 - channels
 - from command line 312
 - .NET Packager 317
 - File Packager channels 226
 - .NET Packager channels 244
 - PDA Packager channels 251
 - Windows Installer packages 215
 - repair command-line option 328
 - repair policies 157
 - See also* repairing
 - about 124
 - best practice 124
 - channels 121

- Content Replicator versus File Packager 444
 - directories and files 351–352
 - general 340–341
 - INI files 81, 356
 - registry and metabase keys 360
 - selecting for channels 121
 - Windows Installer 216
 - repairing
 - See also* repair policies
 - applications before startup 169
 - channels 165
 - files modified by text modifiers 183
 - requirements
 - channel, configuring 138
 - disk space 132
 - files 133
 - registry entries 136
 - response files for SVr4 packages 145
 - restarts. *See* reboots
 - restoring files during uninstallation 126
 - Return Code Mapping tab 202
 - rollbacks
 - Content Replicator versus File Packager 444
 - failed installations and 108
 - specifying in XML 382
 - storing files during installation 210
 - when installation fails 108
 - when major updates fail 108
 - runchannel program 308
 - rundir command-line option 328
 - runexe command-line option 328
 - runtime scripts 141
- S**
- save as command when editing channels 187
 - scalability 31
 - schedules
 - Content Replicator versus File Packager 443
 - publishing channels 282
 - repairing channels 167
 - verification of channels 167
 - Script ReturnCode Mapping tab 203
 - SCRIPT XML tag 397
 - scripts
 - customizing channels with 141
 - error codes 201
 - executable, customizing channels with 141
 - major updates and 151
 - minor updates and 151
 - MSI macros and 97, 141
 - paths for 144
 - postinstall 141
 - post-operation 144
 - preinstall 141
 - response files for SVr4 packages 145
 - runtime 141
 - triggering reboots with 116
 - troubleshooting 297
 - SCRIPTS XML tag 396
 - search and modify XML tags 418–428
 - MODIFY_OBJECT 418
 - MODIFY_OBJECT_ATTRIBUTES 420
 - search and replace
 - line modifiers
 - about 173
 - creating 176
 - text modifiers
 - about 173
 - creating 179
 - searching for items in Package Editor 94
 - security certificate 60, 258
 - security properties 154
 - Select Package page 28
 - self-healing channels 165
 - semi-silent installation mode 31, 107
 - specifying in XML 381
 - SERV_ADVANCED XML tag 378
 - SERV_DEPEND XML tag 378
 - SERV_GENERAL XML tag 376
 - SERV_POLICIES XML tag 379
 - SERV_SECURITY XML tag 377
 - SERVICE XML tag 375
 - SERVICES XML tag 375
 - services, Windows. *See* Windows services
 - setmacro command-line option 328
 - SHORTCUT XML tag 403
 - shortcuts
 - adding to channels 77
 - configuring
 - to repair applications before startup 169

- to update applications before startup 169
- removing from endpoints 343
- update policies and 343
- SHORTFILENAMES** property 195
- Shrinkwrap Windows Applications.** *See Packager*
 - for Shrinkwrap Windows Applications
- shrinkwrappackage** command-line option 310
- signed directory 30
- silent** command-line option 328
- silent installation mode 31, 107, 139, 200, 205, 328
 - reboots and 115
 - specifying in XML 381
- snapshot** command-line option 310
- SNAPSHOT** XML tag 429
- snapshots
 - adding
 - file systems for 41
 - metabase keys to snapshots 48
 - command line 310
 - creating preinstall 39
 - customizing 46
 - editing
 - default filter settings for 45
 - filters used for 44, 46
 - postinstall snapshot settings 51
 - enabling capturing IIS metabase changes for metabase 48
 - excluding file system items from 41
 - files with .msp extension 40
 - filters 44
 - metabase keys from snapshots 48
 - XML template files and 272
 - ignoring file system items for 42
 - installing application to package 50
 - loading files into Application Packager 40
 - postinstall
 - editing settings 51
 - taking 50
 - preinstall
 - adding file systems 41
 - adding metabase keys to 48
 - creating 39
 - enabling capturing IIS metabase changes for 48
 - file system selection 41
- filtering metabase keys from 48
- including and excluding registry entries 45
- loading snapshot files 40
- removing metabase keys from 48
- saving to file 49
- selecting metabase entries for 47
- selecting source 40
- preparing to install application 50
- removing
 - file systems from 42
 - filters used for 44
 - metabase keys from snapshots 48
- saving to file 49
- selecting file systems for 41
- selecting for metabase 47
- selecting source 40
- XML tags** 428–432
 - DIRECTORY 430
 - FILESYSTEM 429
 - FILTER 430
 - METABASE 431
 - METKEY 432
 - REGISTRY 431
 - REGKEY 431
 - SNAPSHOT 429
- XML template files and 272
- SOFTDIST** debugging flag 304
- software distribution
 - about 26
- software library. *See DSL*
- sourcedir** command-line option 320
- spreadsheets, packaging with File Packager 222
- stagems** command-line option 328
- staging, Content Replicator versus File Packager 444
- starting
 - Application Packager 62
 - applications, updating and repairing before 169
 - channels from tuner 104
 - packaged applications in XML 385
 - packages from command line 327
 - reboots with scripts 116
- STARTUP** XML tag 385

- static assemblies 232
 - strong names, for assemblies 243
 - subscribing tuners, as part of software distribution process 26
 - Subscription Policy Manager. *See* Policy Manager
 - Subscription *See* Policy Management
 - supported platforms. *See* platforms
 - SVr4 packages
 - in UNIX 61
 - running response files with scripts 145
 - symbolic links
 - Content Replicator versus File Packager 443
 - editing in channels 78
 - SYMLINK XML tag 405
 - system dependencies
 - configuring 130
 - types 130
 - system reboots. *See* reboots
- T**
- tar files 61
 - template files
 - about 270
 - applying to channels 276
 - command line 325
 - configuring Application Packager to use 271
 - File Packager and 222
 - filters and 272
 - saving Package Editor settings to 270
 - Terminal Services for Windows 33
 - text files
 - editing 173
 - viewing contents of 76
 - text modifiers
 - about 173
 - groups 174
 - insert line 175
 - installation of 174
 - installation policies for 181
 - macros in 176
 - order of installation 174
 - policies for 182, 366–367
 - policies for groups 181
 - search and replace 176
 - search and replace text modifiers 179
- troubleshooting 183
 - types of 173
 - update policies for 181
 - verifying and repairing files 183
 - ThinApp 262
 - ThinApp applications 262
 - transform files
 - about 191, 205
 - adding and removing from packages 205
 - types 205
 - transmitters 27
 - troubleshooting 295–305
 - debugging 302
 - descriptions of log files for 301
 - files 298
 - finding log files 296
 - installation 296
 - Java packages 300
 - MSI packages 299
 - packages 297
 - scripts 297
 - text modifiers 183
 - updates 298
 - tuner properties, macros and 97, 101
- U**
- Unicode files 174
 - uninstall scripts 141
 - uninstallation
 - See also* installation
 - See also* uninstallation policies
 - applications from endpoints 55
 - INI files and 81
 - uninstallation policies 121, 156
 - See also* uninstallation
 - directories and files 348–349
 - files 126
 - general 336–337
 - INI files 354
 - registry and metabase keys 358–359
 - registry keys for 125
 - removing files from endpoints 127
 - restoring files 126
 - selecting for channels 121
 - Windows NT services 363

UNIX

cpio archive 61
 editing symbolic links in channels 78
 preparing to package applications 61
 support in Application Packager 33
 SVr4 package 61
 tar file 61
 unloadname command-line argument 321
 update policies
See also updates
 channels 121
 directories and files 346–347
 duplicate entries in INI files 80
 general 334–335
 INI files 353
 registry and metabase keys 358
 selecting for channels 121
 shortcuts and 343
 text modifier groups 181
 text modifiers 181, 182, 367
 Windows NT services 364
 updates
See also update policies
 before startup 169
 channels 53
 File Packager channels 226
 Java Packager channels 259
 major 150
 managing for applications 52
 minor 150
 .NET Packager channels 244
 PDA Packager channels 251
 troubleshooting 298
 types of 150
 -upgrade command-line option 326
 upgrading channels 325
 user interface for Windows Installer packages 200
 user-defined macros. *See also* macros
 user-specific files 161
 user-specific registry entries 161

V

variables, environment. *See* environment variables
 verification policies 157
See also verifying channels

about 124
 best practice 124
 channels 121
 directories and files 349–351
 general 338–340
 INI files 355
 registry and metabase keys 359–360
 selecting for channels 121
 Windows Installer 216
 Windows NT services 364–365
 -verify command-line option 329
 Verify not installed policy
 best practice 338
 verifying
See also verification policies
 channels
 about 165
 before publishing and distributing 166
 manually 168
 schedules for 167
 files modified by text modifiers 183
 VERIFYREPAIR XML tag 387
 VeriSign certificates 60, 63
 -version command-line option 326, 327, 329
 version numbers
 changing 53
 checking from command line 326
 virtual directories, adding to Internet Information Services (IIS) 244
 Virtual Machine. *See* JVM
 virtual packager 263

W

Windows
 editing registry 46
 environment variables 130
 macros for 433–434
 NT services, policies for 361–366
 support in Application Packager 32
 Windows File Protection (WFP) 57
 Windows Installer Packager 189–219
 about 25, 190, 191
 advertising applications 199
 command line 198, 311
 configuring applications to use command

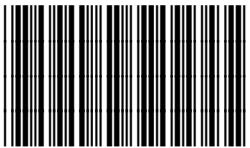
- line 214
 - customizing packages 196
 - database file 190
 - editing contents of MSI packages 213
 - log files 212
 - packaging MSI files 192
 - performing an administrative installation 198
 - policies for 209
 - property macros 97
 - redirection 216
 - repackaging 215
 - troubleshooting 299
 - verifying and repairing 216
- Windows services
- about 152
 - adding to channels 157
 - Advanced Properties page 154
 - editing 152
 - General Properties page 153
 - Install Policies page 155
 - Installer (MSI) 190
 - removing from channels 158
 - Repair Policies page 157
 - Security Properties page 154
 - Terminal Services 33
 - Uninstall Policies page 156
 - Update Policies page 156
 - Verify Installed Policies page 157
 - Verify Not Installed Policies page 157
- Windows XP 60
- wipackage command-line option 311

X

XML

- generic tags 370–399
- MSI tags 414–418
- namespace, for assembly binding 239
- non-MSI tags 400–413
- search and modify tags 418–428
- snapshot tags 428–432
- syntax 369–432
- template files
 - about 270
 - applying to channels 276
 - command line 325

- configuring Application Packager to use 271
 - filters 272
 - Packager for Shrinkwrap Windows Applications and 272
 - saving configuration and filters 273
 - saving Package Editor settings to 270
- XP (Windows) 60



439154