





Note

Before using this information and the product it supports, read the information in “Notices” on page 123.

First Edition (June 2006)

This edition applies to Autonomic Deployment Engine, Version 1.3, and to all subsequent releases and modifications until otherwise indicated.

© Copyright International Business Machines Corporation 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	v
Who should read this book	v
Related documents	v
Conventions used in this book	vi

Part 1. Introduction 1

Chapter 1. Overview of Deployment Engine 3

What is Deployment Engine?	3
What can I do with Deployment Engine?	3
What the Deployment Engine technologies do	5
More key terms used by Deployment Engine	5
The components of a Deployment Engine environment	6
How do I implement Deployment Engine?	10
Software deployment models	10
Wrappered software deployment	10
Fully enabled software deployment	10
Software deployment methods	11
Software package types	12
Application packages	13
The contents of a software package	14
Payload files	15
Deployment descriptors	15
Installable unit deployment descriptors	15
Configuration unit deployment descriptors	16
Installable units	17
Configuration units	27
Unit hierarchies	31
Optional content	34
Managed resources	38
Backing resources	38
Topologies	38
Requisites	40
Which units are most appropriate for my IU deployment descriptor?	42
Action descriptors	45
Media descriptor	49
Software life cycle	51
Life cycle states	52
Change requests	52
Change management operations	54
Dependencies	56
Checks	57
Types of dependencies and their corresponding checks	57
Relationships	61
Relationship types	61
Management of relationships	63
Relationships and integrity checking	64
Variables	64
Variable types	64
Internal variables	66

Chapter 2. The Deployment Engine run-time environment 69

User mode selection for the run-time environment	69
Deployment Engine user modes	70
Multiuser mode	70
Single-user mode	71
Database access	71
Access restrictions in multiuser mode	71
Access restrictions in single-user mode	72
Installed directories	72
Directories for users of Deployment Engine in multiuser mode	72
Directories for users of Deployment Engine in single-user mode	72
Environment variables for the installed directories	73
Removing Deployment Engine	74

Part 2. Commands 77

Chapter 3. Command summary 79

Chapter 4. Working with commands . . . 81

Command authorization	81
Locating and running the commands	82
Command syntax conventions	82
Case sensitivity in commands	82
Specifying a software instance uniquely	83
Retrieving return codes	84
Representing strings that include spaces	84

Chapter 5. Developer commands . . . 85

manageIU	86
validateIUID	101

Part 3. Problem determination . . . 103

Chapter 6. Locating the Deployment Engine log files 105

Finding the ACULogger.properties file	105
Finding the logs for components	105

Chapter 7. Message logging 107

Message identifier	107
Message text	108
Message help	108
Message log format	108

Chapter 8. Messages issued by components 111

Common messages	112
Change manager messages	113

Dependency checker messages.	114
Operating system touchpoint messages.	115
Chapter 9. Trace logging	117
Chapter 10. Troubleshooting	119
<hr/> Part 4. Appendixes	<hr/> 121

Notices	123
Open source license notices.	124
Apache Software License, Version 1.1	125
Apache Software License, Version 2.0	126
W3C Software Notice and License	128
Trademarks	129
 Index	 131

Preface

This book describes commands that you can use for software package testing, explains related concepts, and provides problem determination information—including messages—for software package developers who use the IBM® Autonomic Deployment Engine technology.

Who should read this book

This book is for developers of Deployment Engine software packages; that is, software packages that use Deployment Engine to install them. The software packages typically comprise a Deployment Engine-enabled application or some follow-on maintenance for that application. A *Deployment Engine-enabled application* is a software application whose installation and maintenance depends on the presence of a Deployment Engine run-time environment on the application's computer. Such software applications are prepackaged for installation and maintenance using Deployment Engine technology and packaging conventions. The software packages for the Deployment Engine-enabled application can be prepared manually or with the help of tooling that may include this book as part of its documentation.

Software package developers include *developers of base software packages*,—which make up the original, or base, application—and *developers of maintenance software packages*, which modify the base application with either full updates, incremental updates, or fixes. This book provides some supplemental information for these developers, including Deployment Engine commands, related concepts, and problem determination information, to help facilitate their software package testing.

This book addresses the following tasks, which a software package developer can perform on the computer where the software package is to be deployed and tested:

- Validate a deployment, action, or media descriptor in an application's software package
- Test the software package with its associated life cycle operations to determine if the application deploys as intended
- Remove the Deployment Engine run-time environment
- Access message and trace log files for troubleshooting purposes

Related documents

In addition to *Autonomic Deployment Engine for Software Package Developers*, Deployment Engine provides the following related documents:

- *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*

Provides commands for administering applications that were deployed with the Autonomic Deployment Engine technology. This book also includes some additional commands for administering Deployment Engine itself. And it provides concepts related to the administration commands, as well as problem determination information, including messages.

- *IBM Autonomic Deployment Engine: Autonomic Deployment Engine Glossary*
Defines the terminology used in Deployment Engine documentation.

- Touchpoint documentation
Deployment Engine provides the following pre-release 1.3 documents for its supported touchpoints:
 - *Solution Install for Autonomic Computing: Operating System Touchpoint Guide and Reference*
Describes the operating system touchpoint, its available actions, and the schema used to create these actions.
 - *Solution Install for Autonomic Computing: WebSphere Touchpoint Guide and Reference*
Describes the WebSphere® touchpoint, its available actions, and the schema used to create these actions.
- Autonomic Computing Architecture Board (ACAB) documentation
Deployment Engine provides the following schema documents, which were prepared by the Autonomic Computing Architecture Board:
 - *IBM Autonomic Computing: Installable Unit Deployment Descriptor Specification (ACAB.BO0402)*
Defines the schema for XML documents (IU deployment descriptors and CU deployment descriptors) that describe the characteristics of installable units and configuration units of software. These characteristics are relevant to the identity, deployment, configuration, and maintenance of the software.
 - *IBM Autonomic Computing: Installable Unit Package Format Specification (ACAB.BO0404)*
Defines the schema of an XML document (a media descriptor) that describes the binding information (physical locations) of the files in the software package. This document also describes the package format and related design, including the package structure, physical packages, security model, the relationship with existing install technologies and package standards, and tooling.

Conventions used in this book

This book uses the following typeface conventions:

Bold

- Commands names that are difficult to distinguish from surrounding text
- Keywords and parameters in text
- Interface controls

Italic

- Words and phrases that are emphasized
- New terms
- Variables and values that you must provide

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text, including user prompts
- User input
- Values for arguments or command options

This book uses UNIX[®] conventions for specifying environment variables, directory names, or directory paths:

- When using the Microsoft[®] Windows[®] command line, replace *\$variable* with *%variable%* for environment variables or directory names, and replace each forward slash (/) with a backslash (\) in any Windows directory paths that include forward slashes.
- When using the bash shell on Windows operating systems, use the UNIX conventions.

Part 1. Introduction

Chapter 1. Overview of Deployment Engine

This book is for anyone developing software that includes IBM *Autonomic Deployment Engine* (hereafter called *Deployment Engine*). This chapter introduces the Deployment Engine approach to software packaging.

A note about the Deployment Engine name: The names and content of documentation from earlier releases, as well as some specifications, troubleshooting information, output, files, and the like, refer to Deployment Engine by its previous name, Solution Install for Autonomic Computing, or Solution Install. Occasionally, such references may appear in this book. Also, commands, files, APIs, or other Deployment Engine–related entities might use the characters "si" in their names for the same reason.

This chapter describes key Deployment Engine components and related resources, and explains their relationships. Concepts pertinent to developing software packages that use Deployment Engine to install them are also discussed.

A *software package* is a collection of files that includes the application files (referred to as *payload files*) to be deployed plus all the instructions and data that Deployment Engine requires to deploy the application files. Deployment Engine and an installation program—referred to herein as a *software deployment program* because it may not only install but also configure, update, or remove software—both use this software package to deploy all or part of an application in its target environments. An application can be made up of one or more software packages. The software package and its files must have a specific package format and the software package must use designated file-naming conventions.

What is Deployment Engine?

Deployment Engine is a collection of common software packaging, deployment, and configuration technologies from IBM Tivoli®. Tivoli makes these technologies available to IBM and its business partners, whose customers in turn may use it to deploy their applications.

The common technologies provide a *standard way* for application developers to package their software for deployment to a number of popular operating system and application environments. In the customer environment, Deployment Engine works with the application's software deployment program to orchestrate the software deployment. In addition, Deployment Engine can manage subsequent functionality upgrades, fixes, maintenance, and other change management operations on the deployed software.

When developers implement Deployment Engine in their software, people who use that software can consistently and comprehensively plan for and deploy that software in less time, and use fewer resources to deploy and manage the changes to that software across complex information technology (IT) environments.

What can I do with Deployment Engine?

Deployment Engine helps developers and packagers of applications, and the people who use and maintain those applications, achieve the following results:

- Install, configure, upgrade, maintain, and remove software, either interactively, silently, or both
- Keep track of installed software
- Manage their prerequisite and corequisite software dependencies
- Manage their environmental dependencies
- Reuse or share the installed software
- Remove applied software upgrades, maintenance, or service packs
- Ensure the tolerance of all dependent software before any shared software is changed or removed
- Create, deploy, and maintain a solution or suite of applications or components
- Handle unsuccessful deployment attempts

To achieve these results, software package developers organize their code into logical installable entities, called *installable units*. Installable units are XML representations of the installable parts of an application, whose physical equivalent is the application's payload files. The developers then package the installable units, along with dependency checking and other Deployment Engine infrastructure, to form the final application they deliver to their customers.

By designing their software to be installed using the Deployment Engine infrastructure, developers can ensure a more consistent deployment experience for the people who install and maintain their software. Deployment Engine-based deployments require less human interaction and less time to perform. Additionally, the Deployment Engine infrastructure helps reduce the cost and complexity of building, deploying, and maintaining software, solutions, and suites.

Deployment Engine provides some standardized packaging mechanisms and common componentry for software package developers to use. Among them:

- *Deployment descriptors*. Deployment descriptors are XML documents that define the deployment capabilities and dependencies for a given software package. Deployment descriptors include some XML elements that describe the installable units to be deployed, as well as the units that configure them.
- *Action descriptors*. Action descriptors are XML documents that define the actions to be performed during software deployment, configuration, upgrading, maintenance, and removal. Actions include operations like creating or removing directories, installing or removing application files, and updating registries, configuration properties, environment variables, paths, and so on.
- *Media descriptors*. Media descriptors are optional XML documents that define the location of the deployable software files in specific situations, such as when the files are to be installed from multiple CDs or from archive files.
- *Deployment Engine schema*. The Deployment Engine schema is a set of XSD documents that define the structure, content, and semantics for building the deployment, action, and media descriptors.
- *A dependency checker, change manager, and installation database*. These are key components of the Deployment Engine run-time environment that application developers package with their software. The components are installed in the customer environment, where they assist with deploying software and managing software changes.

What the Deployment Engine technologies do

Deployment Engine by itself does not actually install software. Rather, Deployment Engine works *in concert with* the application's own software deployment program. To this end, Deployment Engine does the following things:

- Defines standardized XML documents that describe installable units of software and their associated requirements, dependencies, and relationships.
- Provides a run-time environment that can consume the standardized XML documents and enable the software deployment.
- Protects the target environment (the environment intended to host the new software) by checking how a new software deployment will affect the software already there. This includes validating the target environment and deployment plan before making any changes to the target environment.
- Uses a standard Web service interface to communicate with target environments and carry out change management operations.
- Provides a database and the appropriate interfaces for logging and tracking deployed software and retaining interdependency and relationship information.

Thus Deployment Engine supplies *the enabling technologies*—the *engine*—for software deployment. These enabling technologies include the standards, facilities, and tools that enable users or administrators to perform rapid and safe software deployments.

Deployment Engine itself uses open standards like XML, Java™, and Web services so that application developers and administrators can create and deploy installable units which can be managed as elements of their own software application.

More key terms used by Deployment Engine

Before reading the Deployment Engine concepts presented in this chapter, you should understand how Deployment Engine uses the following key terms. The terms as defined here are tailored to Deployment Engine. Some definitions for these terms are further developed later in this book. A complete glossary is provided in a related document, *IBM Autonomic Deployment Engine: Autonomic Deployment Engine Glossary*.

- **Software.** *Software* is any computer programming that provides instructions to the computer hardware to tell it what to do.
- **Application.** *Application* is the term Deployment Engine uses to refer to any software script, component, feature, product, application, application suite, solution, maintenance, or other software deliverable that can be (or already is) deployed in an operating environment.
- **Instance.** A particular occurrence or example of something, such as an *instance* of a deployed application or a feature *instance* in the Deployment Engine installation database.
- **Application package.** Different from a *software package*, as described on page 3, an *application package* is all the software that comprises an application, including any additional application software needed prior to installation or to accomplish installation. It includes all the software packages for the application. It can include Deployment Engine itself, if Deployment Engine is not already part of the operating environment. It can also include other things, such as a Java runtime environment or a software deployment program that an application user or administrator runs in order to deploy the software packages.

- **Feature.** A *feature* is a set of installable or configuration units that represents some specific functionality of a larger application, and whose deployment is optional. Samples, language packs, or even applications in a suite are considered features.
- **Maintenance.** *Maintenance* is the general term for any separately deployable software package that represents one or more application updates or migration actions. Fix, incremental update, and full update software packages are specific forms of maintenance.
- **Change request.** A *change request* is an object that is passed by the software deployment program to Deployment Engine to request some kind of software change. The change request includes the deployment or configuration operation to be performed as well as the location of the software package. The change request supplies values for variables defined in the software package (such as the value for the installation location). A change request can be associated either with software deployment or with follow-on configuration of the deployed software.

As previously mentioned, Deployment Engine provides an infrastructure for packaging, deploying, and configuring software. Deployment Engine defines these tasks as follows:

- **Packaging.** *Packaging* is the process of reorganizing your application files into a Deployment Engine-compatible software package. This task is usually performed by a *software package developer*.
- **Deploying.** *Deploying* is the process of placing files or installing software into an operating environment and making that software available for use. As used in this book, *deploying* software with Deployment Engine specifically refers to all of the following things, including software removal:
 - Installing or removing a software package (where *installing* also involves any initial configuration of the software)
 - Applying or removing a feature
 - Applying or removing maintenance

These tasks are usually performed by an *administrator*, a general user, a product administrator, or a system administrator.

- **Configuring.** *Configuring* is the process of setting up or customizing software for a particular use or environment. Unless otherwise stated, *configuring* with Deployment Engine refers to the one-time initial setup of an application or to the subsequent reconfiguration or setup of that application, which can be repeated. Configuration can be designed or automated by the developer who creates the software package, and by the developer who writes the software deployment program. Any real-time configuring is usually performed by the *administrator* who is deploying the application.

The components of a Deployment Engine environment

Figure 1 on page 7 shows the key Deployment Engine components (middle of figure) that application developers provide with their Deployment Engine-enabled software. These components are installed on the target computer together with the Deployment Engine-enabled application when that application is installed for the first time.

Also shown are some resources that are associated with Deployment Engine (on the right). One or more of these resources are also located on the target computer where Deployment Engine deploys software. Resources in the target environment

that are related to Deployment Engine include managed resources (see page 38) and touchpoints. Application components (on the left)—components that are related to but not provided by Deployment Engine—include the software deployment program, descriptors, and payload files.

The arrows in the figure show how the components developed for the application itself (on the left) and the resources associated with Deployment Engine (on the right) communicate with the actual Deployment Engine components (in the middle). All of the components and resources shown in Figure 1 reside on the same computer. They are each described in the text that follows the figure.

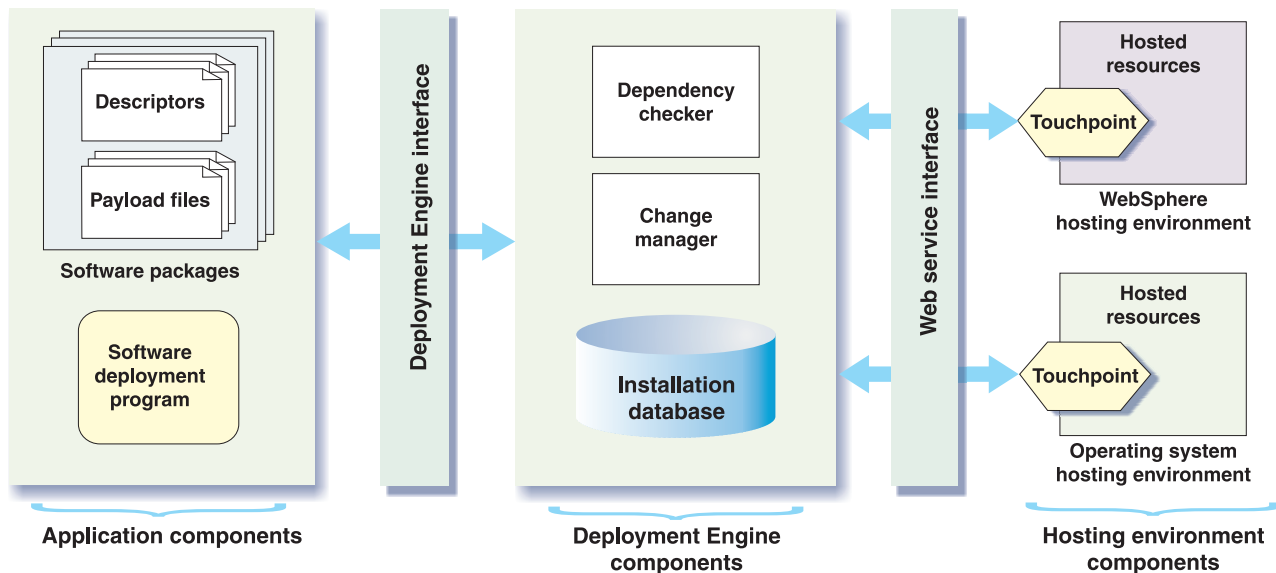


Figure 1. Deployment Engine operating environment on a single computer

A brief description of the key components found in a Deployment Engine operating environment follows. Some components are supplied by Deployment Engine. Other components, noted parenthetically as “application-defined,” are defined or supplied by the application that includes Deployment Engine. Associated resources that are found in the target environments, known as *hosting environments*, are also included in the following list. Each associated resource is noted parenthetically as a “hosting-environment resource.”

Software package (application-defined)

A collection of files that includes the payload files to be deployed plus all the descriptors that Deployment Engine requires to deploy the payload files. Deployment Engine and a software deployment program use this software package to install and configure an application in target environments. An application can be made up of one or more software packages.

Descriptors (application-defined)

XML files that contain the instructions and data that Deployment Engine requires to deploy and configure an application. In Figure 1, “Descriptor” is a general term that represents any of the following XML descriptor files:

- *Installable unit deployment descriptor (IU deployment descriptor)*. An XML descriptor file that defines the content of a software package. The file

content includes logical installable software entities, called *installable units* (or *IUs*). A software package contains exactly one IU deployment descriptor.

- *Configuration unit deployment descriptor (CU deployment descriptor)*. An XML descriptor file that defines a single configuration task, though the task might include multiple steps. The file content includes the basic entities of configuration, called *configuration units*, or *CUs*. There can be more than one CU deployment descriptor in a software package.
- *Action descriptor*. An XML descriptor file that defines specific actions for deploying installable units or configuring software in a particular hosting environment. An *action* is simply a task that needs to be performed in a hosting environment.
- *Media descriptor*. An XML descriptor file that identifies the media location of one or more action descriptors or deployable payload files for a software package. A media descriptor is optional. When repackaging software, the media descriptor provides the new media locations for files that have moved.

Payload files (application-defined)

The files in a software package that are deployable; that is, the application files. These are the files that Deployment Engine actually *installs* in the hosting environment. JAR files, ZIP archives, RPM packages, and configuration files are among the valid types of payload files. The actual data content of these payload files is immaterial to Deployment Engine.

Software deployment program (application-defined)

An interactive or silent “installation” program that uses Deployment Engine APIs to first deploy Deployment Engine (if needed) by running a bootstrap program and then implement software change requests in one or more hosting environments. The software deployment program constructs a software change request and passes it to Deployment Engine for processing. This processing then initiates the changes to one or more hosting environments. Developers can code the software deployment program themselves or use tooling to help generate it. Note that, differently from an installation program, a software deployment program not only installs but also configures, updates, or removes software, thereby handling any type of software deployment described in this book.

Deployment Engine interface

Java application programming interfaces (APIs) that the developers of software deployment programs use to generate the change requests that Deployment Engine requires to initiate changes to one or more hosting environments.

Dependency checker

The Deployment Engine component that determines whether or not dependencies are met before installing software in a hosting environment. A *dependency* is a requirement that one installable unit has on another installable unit or managed resource (see page 38) to ensure that they interoperate correctly.

The dependency checker performs *dependency checking* by using data from the following sources to determine whether the dependencies are met:

- The IU deployment descriptor
- The installation database
- The touchpoint for the target hosting environment

The dependency checker also makes sure that the dependencies of other deployed installable units registered in the installation database are not violated. This function is sometimes referred to as *integrity checking*.

Change manager

The Deployment Engine component that reads the information in a deployment descriptor and then coordinates the change request to be implemented across hosting environments.

Installation database

The Deployment Engine component that retains the information about, and the IU deployment descriptor for, each installable unit instance deployed on the local computer. The installation database also contains a *relationship registry* that saves information about the relationships and interdependencies among deployed installable units. The installation database provided with Deployment Engine is IBM Cloudscape™. Deployment Engine administrators can manage the installation database by using the commands described in the book *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*.

Web service interface

Represents all Web service communications between the Deployment Engine components and the hosting environments.

Hosting environments

Any environments where Deployment Engine can deploy software (for example, the target operating system or the target J2EE server). Deployment Engine communicates with a hosting environment through its touchpoint.

Hosted resources (hosting-environment resource)

Any resources that are present in the hosting environment and accessible through a touchpoint.

Touchpoints (hosting-environment resource)

Web services that interface with hosting environments; for example, with an *operating system hosting environment* or with a *WebSphere hosting environment*. A hosting environment has one touchpoint. The touchpoint enables management operations—in this case, operations or processes related to software deployment—to be performed on the hosting environment and on any of its hosted resources.

More about the *operating system touchpoint*: An operating system touchpoint uses *scanners* and *collectors* (collectively called *sensors*) on the target computer to gather information about it. For example, the touchpoint can scan for an application that was installed outside of Deployment Engine, which might satisfy a new application's software requirements. Or the touchpoint can scan for a Deployment Engine-installed application so that Deployment Engine can update it.

The operating system touchpoint can also read deployment actions that originate from an action descriptor in a software package and then use its *effectors* to process the actions in the hosting environment. Actions include creating or removing directories, installing or removing application files, and updating registries, configuration properties, environment variables, paths, and the like.

How do I implement Deployment Engine?

The following sections explain the basic approaches available to software package developers who are implementing Deployment Engine in their applications.

Software deployment models

You can follow one of two software deployment models to make your application Deployment Engine–compliant. These models, which describe several alternatives for developing an application package, are described in the sections that follow.

Wrappered software deployment

Wrappered software deployment is an entry-level Deployment Engine deployment. Using this model, you make your application Deployment Engine–compliant by designing a Deployment Engine–based application package that reuses your current installation program and application files without modifying them. This model requires the least amount of development effort.

In short, you enclose your current installation program and payload files in a single Deployment Engine–compliant software package that you create for your application. The existing installation program is “wrapped” in the IU deployment descriptor for the software package (see “Deployment descriptors” on page 15).

In the simplest case, the IU deployment descriptor includes one root IU with one smallest IU that defines one action descriptor (see “Action descriptors” on page 45). The action descriptor includes one action, and that action calls your current installation program “as is” and installs your application files. Your case might be more complex, but the general idea is to provide an IU deployment descriptor with the a minimum number of elements needed to wrap your existing application.

Some benefits of a wrappered Deployment Engine–based software deployment:

- You retain the original packaging structure and installation program for your application. You can use your Deployment Engine–compliant software package (with its wrapped installation program) in environments where Deployment Engine is the preferred software deployment mechanism. Or you can use your original packaging structure and installation program in environments that have not adopted Deployment Engine.
- Your application gets registered in a Deployment Engine database, where other applications can be made aware if it.
- Your software deployment can take advantage of the Deployment Engine dependency and integrity checking functions.

Fully enabled software deployment

Fully enabled software deployment is a fully implemented Deployment Engine deployment. Using this model, you make your application Deployment Engine–compliant by designing a compliant application package from the start. Among the things typically included in the application package are one or more Deployment Engine–compliant software packages, with their descriptors and payload files, and a software deployment program to install the files. This model requires more development effort—sometimes substantially more than the wrappered software deployment model.

For an existing application, this could mean decomposing and then recomposing the basic elements of your current application into Deployment Engine–compliant

software packages. For a new application, this means designing and creating software packages to be Deployment Engine–compliant from the ground up.

Unlike wrapped software deployment where the original software packaging and installation program of an existing application is preserved, fully enabled software deployment yields a Deployment Engine–based design that, when implemented, becomes your sole deployment mechanism.

Some benefits of a fully enabled Deployment Engine–based software deployment:

- You only have to maintain one software deployment program.
- You could have less software deployment code to maintain than you would without using Deployment Engine, because most of the code belongs to Deployment Engine.
- Your application gets registered in a Deployment Engine database, where other applications can be made aware of it.
- Your software deployment can take advantage of the Deployment Engine dependency and integrity checking functions, as well as the ability of Deployment Engine to initiate installation and configuration actions in a number of operating system and application environments.

Software deployment methods

A software package is a collection of files that includes the application files to be deployed plus the instructions and data that Deployment Engine requires to deploy the application files. Deployment Engine and a software deployment program use this software package to install and configure all or part of an application in its target environments.

An application can be made up of one or more software packages. The software packages and their files must have a specific structure and each software package must comply with designated file-naming conventions.

You can deploy a single software package, as you might with a standalone application, or you can deploy multiple software packages together, as with a solution or suite. When deploying multiple software packages, you can deploy them hierarchically, in a software package tree (the preferred method), or separately, in a hybrid manner:

Single software package

A single software package is the most elementary deployment method. The software package is processed using a single change request.

Software package tree

A *software package tree* is a hierarchy of software packages that make up a single application or solution, and that are processed together as one software deployment. A software package includes one, and only one, IU deployment descriptor (see “Deployment descriptors” on page 15).

In a software package tree, the IU deployment descriptor of one software package (the *parent software package*) references the IU deployment descriptor of one or more other software packages (*child software packages*).

You accomplish this deployment method by defining in the IU deployment descriptor of parent software packages some contained IUs, contained container IUs, requisites, or a combination of these elements. These

elements are specifically designed to reference (and therefore associate hierarchically) another IU deployment descriptor whose software package is a child software package.

Software package set

A *software package set* is a collection of software packages (or software package trees) that make up a single application, solution, or suite, which, for whatever reason, cannot be organized into one unified software package tree.

You might have to use this implementation if, as part of your software deployment, you have to first install some prerequisite software before installing your application. Rather than deploy the software packages together as a tree, your software deployment program must call separate change requests in order to deploy each software package in the proper order. (Such software deployment programs are sometimes referred to as *hybrid* installation programs.)

In this case, your software deployment program must do some handling that Deployment Engine would otherwise do if you were using one unified software package tree. For example, your software deployment program might have to handle decisions about what order to install certain software packages in.

Software package types

Software deployment places application payload files in a hosting environment. The payload files are part of a software package which also includes the descriptors that Deployment Engine requires to deploy the payload files.

A deployable application has at least one, and often more than one, software package. But software packages have different functions. Their function depends on whether the payload files to be deployed are part of the initial version of the application or, alternatively, are some form of application maintenance. Recall that maintenance-related software packages represent one or more application updates in the form of fixes, incremental updates, or full updates.

Therefore, a software package always represents one of the following (either in total or in part):

- The original version of an application
- An full application update
- An incremental application update
- An application fix

As a result, software packages are identified as one of these corresponding types:

Base A software package that contains the original version of an application, called the *base application*. Use a base software package to deploy an application for the first time. A software deployment that installs an application for the first time is referred to as a *fresh installation*.

Full update

A software package that contains a major upgrade to an application, such as a manufacturing refresh. There are two ways to deploy a full update software package: as a fresh installation or as an upgrade.

The software deployment program queries Deployment Engine to determine whether the application is already present in the hosting

environment. If the application is already present, the software deployment program can instruct Deployment Engine to deploy the software package as an upgrade. If the application is not present, the software deployment program instructs Deployment Engine to deploy the software package as a fresh installation. Either deployment yields the same result: a full update of the application to the latest version. A full update software package increases the version number (version, release, modification, or level number) of the application that it updates.

Incremental update

A software package that contains an application upgrade, such as a refresh pack or fix pack. Unlike a full update software package, you cannot deploy an incremental update software package as a fresh installation, because the software package contains software updates only. Instead, you can only deploy it as an upgrade. Use an incremental update software package to upgrade a currently deployed application. An incremental update software package increases the version number (version, release, modification, or level number) of the application that it updates.

Fix A software package that contains critical software changes or corrections (such as an interim fix or test fix) that need to be deployed *sooner* than the next full update or incremental update, which, as a rule, eventually incorporates the same changes provided by the fix. A fix software package is used to deploy only the changed or corrective software and apply it to a currently deployed application.

A fix software package has no impact on the version number of the application that it fixes.

Specific XML elements in the IU deployment descriptor are used to identify these software package types. The **identity** or **fixIdentity** elements for the root IU characterize the type of software package overall. As a rule, each subordinate unit—any installable unit or configuration unit that can be defined within another installable unit—must have identity information for the software package type that matches its root IU. (A full update, however, is the exception; its installable units can have one of two software package types—base or full update.)

In terms of descriptor processing, then, Deployment Engine determines software package types at the installable unit level, in order to deploy the installable units of the software package correctly and efficiently. Ultimately this means that payload files are either installed anew, installed over existing files, or sometimes not installed at all, depending on the software package types associated with their corresponding installable units. (For details on how to define a software package type using XML elements in the IU deployment descriptor, see the related document, *IBM Autonomic Computing: Installable Unit Deployment Descriptor Specification*.)

Application packages

An *application package* is all the software packaged as part of the application. It includes all the software packages for the application. It always includes Deployment Engine itself, if Deployment Engine is not, or might not, be part of the operating environment where the application will be deployed. An application package can also include other things, such as a Java runtime environment or a software deployment program that a user runs in order to deploy the software packages.

The software deployment program developer must prepare an application package that typically contains the following things:

- All the software packages for the application. The software packages can be either base, full update, incremental update, or fix software packages, as described in “Software package types” on page 12.
- The Deployment Engine run-time environment. To enable users or administrators to deploy your application, the run-time environment must first be installed in the operating environment. However, its inclusion in the application package is not a requirement when the appropriate version of Deployment Engine is known to be present in the operating environment.
- A software deployment program. To deploy your application, a user runs this program in concert with the Deployment Engine run-time environment.
- Any other necessary software. For example, an application might require its own Java runtime environment (JRE).

After the application package has been assembled, tested, and delivered, an administrator or user must deploy the application.

In the operating environment of the target computer that will host the application, an administrator or user typically:

1. Accesses the application package
2. Runs the software deployment program
3. Supplies any inputs required by the program and initiates the software deployment

Unless already present, the software deployment program installs Deployment Engine before anything else. Deployment Engine then uses the information and instructions from the software deployment program, provided in the form of change requests, to deploy the software packages for the application on a target computer. When deployment completes, Deployment Engine registers all the successfully deployed installable units in its installation database.

The contents of a software package

A *software package* is a collection of files that includes the application files to be deployed, called *payload files*, plus some descriptors. *Descriptor* is a general term for the three types of XML documents used by Deployment Engine for software change management:

- Deployment descriptors
- Action descriptors
- Media descriptors

A software package has one deployment descriptor and at least one action descriptor. A media descriptor is typical but optional.

Deployment Engine and a software deployment program work with the contents of one or more software packages when deploying and configuring an application in target hosting environments. Each software package and its descriptors must conform to the file-naming conventions provided by Deployment Engine.

Each descriptor in a software package is an XML document that you structure according to Deployment Engine schemas. Descriptors provide XML elements for defining installable units and configuration units, of which there are several types.

The sections that follow describe the payload files; the types of descriptors, installable units, and configuration units; and their purpose in the software package.

Payload files

Payload files are the files in a software package that are deployable; that is, the application files. These are the files that Deployment Engine actually *installs* in the hosting environment. Payload files can be executable files, binary files, library files, and so on. The payload may also include nonfunctional files such as readme files, installation instructions, and other informational files. JAR files, ZIP archives, and RPM packages are valid delivery mechanisms for payload files.

An XML **files** element that you supply in the IU or CU deployment descriptor instructs Deployment Engine where to find the payload files. These file locations can be superseded by new locations provided in a media descriptor, which is sometimes used when an application is repackaged or for other special circumstances. The actual data content of the payload files does not matter to Deployment Engine.

Sometimes temporary files, files that are neither descriptors nor payload files (like JVM files), are used by the software deployment program during deployment. These files are not specified in any descriptor. They are included in the application package to make it self-contained, so that it can work without any additional prerequisites.

Deployment descriptors

Deployment descriptor is a general term for an XML document that contains instructions and data that Deployment Engine requires to deploy or configure an application.

Deployment Engine works with the following deployment descriptors, which are described later in this section:

- Installable unit deployment descriptors
- Configuration unit deployment descriptors

In general, a deployment descriptor defines the deployment capabilities and dependencies for a given software package, or defines a single configuration task. Deployment descriptors include some XML elements for the installable units to be deployed, as well as for the units that configure them.

Installable unit deployment descriptors

An *installable unit deployment descriptor*, or *IU deployment descriptor*, is an XML document that defines the content and deployment characteristics of a software package. The IU deployment descriptor is provided as input to the components of the Deployment Engine run-time environment.

The IU deployment descriptor includes XML elements that represent basic installable software entities, called *installable units* (or *IUs*), and optionally, basic entities of configuration, called *configuration units* (or *CUs*). The IU deployment descriptor also includes other elements and attributes that describe the deployment capabilities and dependencies of the software package.

A software package contains exactly one IU deployment descriptor, but can reference others. An application that has multiple software packages must have multiple IU deployment descriptors—one for each software package.

An IU deployment descriptor can refer to additional IU deployment descriptors in other software packages (see the sections that describe the two types of contained installable units on pages 23 and 24). In this way you can “link” IU deployment descriptors together in order to, for example, reuse the contents of existing software packages or integrate individual applications into solutions or suites. This linkage of software packages is hierarchical, and the linked packages are referred to as a software package tree, which is described on page 11.

IU deployment descriptors help reduce the complexity of packaging, deploying, and maintaining complex software solutions and help decrease failures that result from dependency mismatches.

The following installable units and configuration units can be defined by XML elements in the IU deployment descriptor:

- “Root installable units” on page 18
- “Smallest installable units” on page 20
- “Container installable units” on page 22
- “Contained installable units” on page 23
- “Contained container installable units” on page 24
- “Solution modules” on page 25
- “Smallest configuration units” on page 29

Figure 2 on page 18 shows a sample IU deployment descriptor. It includes one smallest installable unit (the **SIU** element).

Configuration unit deployment descriptors

A *configuration unit deployment descriptor*, or *CU deployment descriptor*, is an XML document that defines a single configuration task. This task can include multiple steps. The CU deployment descriptor is provided as input to the components of the Deployment Engine run-time environment.

The CU deployment descriptor includes XML elements for configuration units, the basic entities of configuration. The CU deployment descriptor also includes other elements and attributes that describe the application, managed resource (see page 38), or hosting environment to be configured.

There can be more than one CU deployment descriptor in a software package. For example, there can be one CU deployment descriptor for adding a database user and password, another for setting the port, and yet another for changing the trace level.

Only previously installed applications, managed resources, and hosting environments can be configured using a CU deployment descriptor. The recommended and safest use of this descriptor, however, is to perform follow-on configuration or setup of an application that you previously deployed. Unlike the initial configuration of an application that occurs one time, during software deployment, follow-on configurations that use the CU deployment descriptor are repeatable.

The following configuration units can be defined by XML elements in the CU deployment descriptor:

- “Root configuration units” on page 28
- “Smallest configuration units” on page 29

Installable units

Installable unit, or *IU*, is a general term for the entities in an IU deployment descriptor that describe or represent the installable parts of an application. Using Deployment Engine, you can deploy various types of installable units to a hosting environment in order to create new capabilities in that environment.

One type of installable unit, called the root installable unit, or *root IU*, is not deployed but acts as a container for defining all the other types, or *subordinate units*. *Subordinate unit* is a general term for any installable unit or configuration unit that can be defined within another installable unit.

To define any type of installable unit, you specify a set of XML elements and attributes in the IU deployment descriptor. This deployment descriptor is an XML document that you create in accordance with the supplied Deployment Engine schemas.

In an IU deployment descriptor, you include XML elements that define one root installable unit and one or more of its subordinate units. Briefly, these units do the following things (note that some units are designed to target their activities to the same hosting environment, while others target their activities to the hosting environments specified by their subordinate units—in other words, to one *or more* hosting environments):

- **Root installable unit (root IU).** Defines deployment data about the software package, each subordinate unit of the root IU, and the media location where Deployment Engine can find the descriptors and payload files for the software package.
- **Smallest installable unit (smallest IU).** References the action descriptors that describe how to deploy a particular group of payload files in the same hosting environment.
- **Container installable unit (container IU).** Encapsulates some combination of installable units, configuration units, or other container IUs whose payload files are to be deployed in the same hosting environment.
- **Contained installable unit (contained IU).** References the root IU of another IU deployment descriptor, causing its software to be deployed on behalf of the current IU deployment descriptor into one or more hosting environments.
- **Contained container installable unit (contained container IU).** References the root IU of another IU deployment descriptor, causing its software to be deployed on behalf of the current IU deployment descriptor into a single hosting environment.
- **Solution module.** Encapsulates some combination of installable units, configuration units, or other solution modules whose payload files are to be deployed in one or more hosting environments. Each encapsulated unit in the solution module targets its payload files to its own hosting environment or environments, which might be different from the other encapsulated units.

Note: Although the term *solution module* has no *IU* in its name, a solution module is considered to be an installable unit because it is an entity in an IU deployment descriptor that describes or represents some installable part of an application.

- **Smallest configuration unit (smallest CU).** Defines the action descriptors that describe how to initially configure a newly installed software package in a single hosting environment.

Within the IU deployment descriptor of a single software package, a simple application might be wholly defined using a root IU with a single, smallest IU. Complex applications are more likely to be defined using multiple software packages, or using a single software package whose root IU includes various types of installable units. Multiple installable units provide greater flexibility for software deployment. For example, they permit certain application features to be deployed independently from one another. Some installable units might even require other installable units as corequisites (for example, to install feature B, you are also required to install feature A).

For convenience, you can organize the XML elements for the various installable units hierarchically in the IU deployment descriptor. The valid hierarchical structures are described in “Unit hierarchies” on page 31. A Deployment Engine schema governs how, and in what combinations, these various installable units can be organized in an IU deployment descriptor in order to deploy software.

Root installable units: Every IU deployment descriptor must have one, and only one, root installable unit. A *root installable unit*, or *root IU*, defines the software package to be deployed.

The root installable unit is the top-level XML element in an IU deployment descriptor. Every other descriptor element, including elements for the installable units and configuration units, is included in the root IU. The XML elements in the root IU define the following kinds of information:

- The identity (name, UUID, and version) of the application
- Required or optional subordinate units (that is, installable units and smallest configuration units)
- Features and installation groups (optional content) of the application
- The XML schema (location and version) that this root IU conforms to
- Definitions of any global variables used during software deployment
- Target hosting environments
- The media location of the action descriptors associated with the subordinate units
- The media location of the payload files associated with the subordinate units

Note: For a complete list of the XML elements and attributes of a root IU, see the related document, *IBM Autonomic Computing: Installable Unit Deployment Descriptor Specification*.

Figure 2 shows a sample IU deployment descriptor. The **iudd:rootIU** element is the root IU. The root IU contains all the other XML elements in the IU deployment descriptor. The root IU in this sample includes a smallest IU, represented (under the **installableUnit** element) by the **SIU** element:

```
<iudd:rootIU
xmlns:iudd="http://www.ibm.com/namespaces/autonomic/solutioninstall/IUDD"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:OSRT="http://www.ibm.com/namespaces/autonomic/OS_componentTypes"
```

Figure 2. Sample IU deployment descriptor with mandatory root IU and one smallest IU (Part 1 of 3)

```

xmlns:J2EERT="http://www.ibm.com/namespaces/autonomic/J2EE_RT"
xmlns:RDBRT="http://www.ibm.com/namespaces/autonomic/RDB_RT"
xsi:schemaLocation="http://www.ibm.com/namespaces/autonomic/solutioninstall/IUDD iudd.xsd"

IUName="JRE_15_Sample_RootIU">
  <identity>
    <name>JRE 1.5 Sample RootIU</name>
    <UUID>991974510ba426fe1f53841402352114</UUID>
    <full>
      <upgradeBase>
        <minVersion>1.1.0</minVersion>
      </upgradeBase>
    </full>
    <version>1.5.0</version>
  </identity>

  <selectableContent>
    <installableUnit targetRef="tOS">
      <SIU IUName="JRE_15_IU" hostingEnvType="OSRT:Operating_System">
        <identity>
          <name>JRE 1.5 IU</name>
          <UUID>991974510ba426fa1f53841402351125</UUID>
          <version>1.5.0</version>
        </identity>
        <requirements>
          <requirement name="Creation_Req" operations="Create">
            <alternative name="JRE_Not_Installed">
              <checkItem checkIdRef="Windows_Check"/>
              <inlineCheck testValue="false">
                <software checkId="JRE_15_SW_Check">
                  <name pattern="true">Java 2 Runtime Environment, SE v1.5.0</name>
                </software>
              </inlineCheck>
            </alternative>
          </requirement>
          <requirement name="Deletion_Req" operations="Delete">
            <alternative name="JRE_Installed_SI">
              <checkItem checkIdRef="Windows_Check"/>
              <inlineCheck>
                <iu checkId="JRE_15_IU_Check">
                  <UUID>991974510ba426fa1f53841402351125</UUID>
                  <name>JRE 1.5 IU</name>
                </iu>
              </inlineCheck>
            </alternative>
          </requirement>
        </requirements>
        <unit>
          <installArtifacts>
            <installArtifact undoable="true">
              <fileIdRef>JRE15InstallArtifact</fileIdRef>
            </installArtifact>
          </installArtifacts>
        </unit>
      </SIU>
    </installableUnit>
  </selectableContent>
  <features>
    <feature featureID="JRE_15_Feature" required="true">

```

Figure 2. Sample IU deployment descriptor with mandatory root IU and one smallest IU (Part 2 of 3)

```

    <identity>
      <name>JRE 1.5 Feature</name>
    </identity>
    <IUNameRef>JRE_15_IU</IUNameRef>
  </feature>
</features>

<rootInfo>
  <schemaVersion>1.2.1</schemaVersion>
  <build>42</build>
  <size>1</size>
</rootInfo>

<topology>
  <target id="tOS" type="OSRT:Operating_System">
    <checks>
      <property checkId="Windows_Check">
        <propertyName>OSType</propertyName>
        <pattern>Windows.*</pattern>
      </property>
    </checks>
  </target>
</topology>

<files>
  <file id="JRE_15_Source">
    <pathname>../FILES/jre-1_5_0-beta2-windows-i586.exe</pathname>
    <length>0</length>
    <checksum type="MD5">abc</checksum>
  </file>
  <file id="JRE15InstallArtifact">
    <pathname>JRE15InstallArtifact.xml</pathname>
    <length>0</length>
    <checksum type="MD5">abc</checksum>
  </file>
</files>

</iudd:rootIU>

```

Figure 2. Sample IU deployment descriptor with mandatory root IU and one smallest IU (Part 3 of 3)

Think of a root IU as the sole, top-level node in a hierarchy of multiple subordinate units. This hierarchy includes the subordinate units as leaf nodes, and sometimes as intermediate nodes. Therefore the subordinate units are always nested XML elements under the root installable unit. In a hierarchy of installable units, the following subordinate units can be defined as child or descendent nodes of the root installable unit that includes them:

- Smallest installable unit
- Container installable unit
- Contained installable unit
- Contained container installable unit
- Solution module
- Smallest configuration unit

Figure 18 on page 33 illustrates a hierarchy of installable units.

Smallest installable units: A *smallest installable unit*, or *smallest IU*, is the most basic installable unit. The purpose of a smallest installable unit is to deploy all or part of an application to a single hosting environment. A smallest installable unit

identifies action descriptors that describe how to deploy its related payload files. A smallest installable unit delivers the capability of the installable unit itself and its associated action descriptors.

You specify a smallest installable unit in an IU deployment descriptor, either as part of a root installable unit, a container installable unit, or a solution module.

Figure 3 shows an IU deployment descriptor whose root IU references one smallest installable unit. This IU deployment descriptor represents the most elementary form of software deployment. The software package to be deployed, plus the media location of all the action descriptor and payload files, are defined in the root IU. The action descriptors themselves are defined in the smallest installable unit.

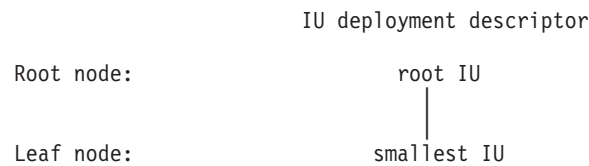


Figure 3. An IU deployment descriptor with mandatory root IU and one smallest IU

Figure 4 shows the structure of a smallest IU represented another way:

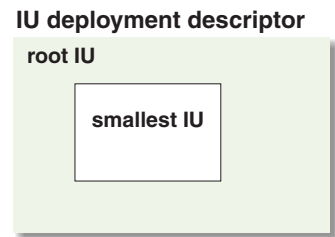


Figure 4. An IU deployment descriptor with mandatory root IU and one smallest IU

The smallest IU contains XML elements that define the following kinds of information:

- The identity (name, UUID, and version) of the smallest IU
- Constraints for the smallest IU
- Definitions of any variables used during software deployment
- Definitions of checks that compare actual values of a property against a defined value in the IU deployment descriptor
- Requirements of the smallest IU
- References to the action descriptors for the smallest IU

Figure 5 on page 22 shows an XML fragment from an IU deployment descriptor. The XML fragment represents a smallest IU. In an IU deployment descriptor, the smallest IU (the **SIU** element) is always a nested element within the top-level **iudd:rootIU** element. In Figure 5, the **SIU** element is highlighted in bold type:

```

<iudd:rootIU ...
...
<installableUnit>
  <SIU IUName="name" hostingEnvType="type">
    <identity>
      <name>human-readable_name</name>
      <UUID>identifier</UUID>
      <version>version</version>
    </identity>
    <constraints>...</constraints>
    <variables>...</variables>
    <checks>...</checks>
    <requirements>...</requirements>
    <unit>
      <installArtifacts>...</installArtifacts>
    </unit>
  </SIU>
</installableUnit>
...
<\iudd:rootIU>

```

Figure 5. An XML fragment representing a smallest IU in an IU deployment descriptor

Container installable units: A *container installable unit*, or *container IU*, is an installable unit that encapsulates some combination of other installable and configuration units whose payload files are to be deployed in a single hosting environment. The encapsulated units can include smallest IUs, smallest CUs, contained container IUs, or other container IUs.

The purpose of a container installable unit is to encapsulate a group of installable units that need to be deployed together on the same hosting environment. Because all the units are targeted to the same place, you can specify the target hosting environment as an attribute of the container IU rather than as attributes of each encapsulated unit. (If you need to target unit payload files to *different* hosting environments, you should define a solution module, not a container IU, to encapsulate the units with different targets.)

Figure 6 shows an IU deployment descriptor whose root IU includes a container IU. The container IU encapsulates a smallest IU and another container IU whose payload files are to be deployed in the same hosting environment:

IU deployment descriptor

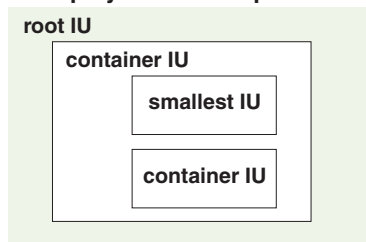


Figure 6. An IU deployment descriptor that includes a mandatory root IU and a container IU with encapsulated installable units

The container IU contains XML elements that define the following kinds of information:

- The identity (name, UUID, and version) of the container IU

- Definitions of any variables used during software deployment
- Subordinate units of the container IU
- Constraints for the container IU
- Definitions of checks that compare actual values of a property against a defined value in the IU deployment descriptor
- Requirements of the container IU

Figure 7 shows an XML fragment from an IU deployment descriptor. The XML fragment represents a container IU. In an IU deployment descriptor, the container IU (the **CIU** element) is always a nested element within the top-level **iudd:rootIU** element. In the figure, the container IU encapsulates two smallest IUs and another container IU. The **CIU** element is highlighted in bold type:

```

<iudd:rootIU ...
:
:
<installableUnit>
  <CIU IUName="name" hostingEnvType="type">
    <identity>...</identity>
    <variables>...</variables>
    <installableUnit>
      <SIU ...>

      :

      </SIU>
    <installableUnit>
    <installableUnit>
      <SIU ...>

      :

      </SIU>
    <installableUnit>
    <installableUnit>
      <CIU ...>

      :

      </CIU>
    <installableUnit>
  </CIU>
</installableUnit>
:
:
</iudd:rootIU>

```

Figure 7. Sample container IU in an IU deployment descriptor

Contained installable units: A *contained installable unit*, or *contained IU*, is an installable unit that references the root IU of another IU deployment descriptor. The purpose of the contained installable unit is efficiency. The contained installable unit reuses another installable unit deployment descriptor. The contained installable unit causes the installable units in the referenced root installable unit to be deployed as if they were part of the current root installable unit. The installable units in the referenced root installable unit are deployed to one or more targeted hosting environments.

Figure 8 on page 24 shows an IU deployment descriptor whose root IU includes a contained IU that references another root IU. The payload files of the referenced root IU are deployed in more than one hosting environment:

IU deployment descriptor

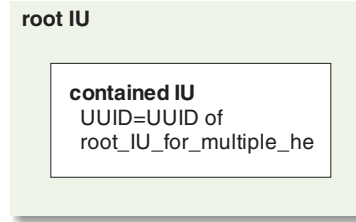


Figure 8. An IU deployment descriptor that includes a mandatory root IU and a contained IU. The contained IU references the UUID of another root IU whose payload files are deployed in multiple hosting environments.

The contained IU contains XML elements that define the following kinds of information:

- The name of the contained IU
- The identity (UUID, version, and file ID reference) of the referenced root IU

Figure 9 shows an XML fragment from an IU deployment descriptor. The XML fragment represents a contained IU. In an IU deployment descriptor, the contained IU (the **containedIU** element) is always a nested element within the top-level **iudd:rootIU** element. In the figure, the contained IU references another root IU. The **containedIU** element is highlighted in bold type:

```
<iudd:rootIU ...  
:  
:  
  <installableUnit targetRef="tOS">  
    <containedIU IUName="Component_IU">  
      <UUID>991974510ba426fe1f53841402350020</UUID>  
      <version>1.2</version>  
      <fileIdRef>ComponentInstall</fileIdRef>  
    </containedIU>  
  </installableUnit>  
:  
:  
</iudd:rootIU>
```

Figure 9. Sample contained IU in an IU deployment descriptor

Contained container installable units: A *contained container installable unit*, or *contained container IU*, is an installable unit that references the root IU of another IU deployment descriptor. Like a contained IU, the purpose of the contained container IU is efficiency. The contained container IU reuses another installable unit deployment descriptor. The contained container installable unit causes the installable units in the referenced root installable unit to be deployed as if they were part of the current root installable unit. However, unlike a contained IU, which deploys the installable units to one or more hosting environments, a contained container IU can deploy the installable units only to a single hosting environment.

Figure 10 on page 25 shows an IU deployment descriptor whose root IU includes a contained container IU that references another root IU. The payload files of the referenced root IU are deployed in only one hosting environment:

IU deployment descriptor

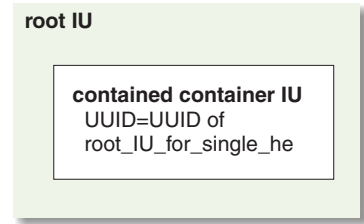


Figure 10. An IU deployment descriptor that includes a mandatory root IU and a contained container IU. The contained container IU references the UUID of another root IU whose payload files are deployed in a single hosting environment.

The contained container IU contains XML elements that define the following kinds of information:

- The name of the contained container IU
- The identity (UUID, version, and file ID reference) of the referenced root IU

Figure 11 shows an XML fragment from an IU deployment descriptor. The XML fragment represents a contained container IU. In an IU deployment descriptor, the contained container IU (the **containedCIU** element) is always a nested element within the top-level **iudd:rootIU** element. In the figure, the contained container IU references another root IU. The **containedCIU** element is highlighted in bold type:

```
<iudd:rootIU ...  
:  
  <installableUnit targetRef="tOS">  
    <containedCIU IUName="Component_IU">  
      <UUID>991974510ba426fe1f53841402350020</UUID>  
      <version>1.2</version>  
      <fileIdRef>ComponentInstall</fileIdRef>  
    </containedCIU>  
  </installableUnit>  
:  
:  
</iudd:rootIU>
```

Figure 11. Sample contained container IU in an IU deployment descriptor

Solution modules: A *solution module* is an installable entity that encapsulates some combination of other installable and configuration units whose payload files are to be deployed in one or more hosting environments. The encapsulated units can include any of the following units:

- Smallest IUs
- Container IUs
- Contained IUs
- Contained container IUs
- Other solution modules
- Smallest CUs

Each encapsulated unit in the solution module targets its payload files to its own hosting environment or environments, which might be different from the other encapsulated units. (If you need to deploy unit payload files to the *same* hosting environment, you should define a container IU, not a solution module, to encapsulate the units whose targets are the same.) Therefore the purpose of a

solution module is to encapsulate some units in the IU deployment descriptor that need to be deployed at the same time to target hosting environments that are different from one another.

Note: Although the term *solution module* has no *IU* in its name, a solution module is considered to be an installable unit because it is an entity in an IU deployment descriptor that describes or represents some installable part of an application.

Figure 12 shows an IU deployment descriptor whose root IU defines a solution module that encapsulates two container IUs.

IU deployment descriptor

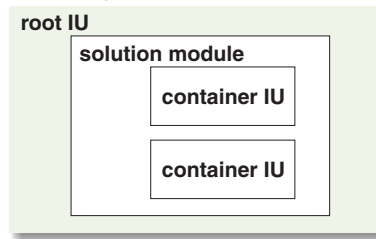


Figure 12. An IU deployment descriptor that includes a mandatory root IU and a solution module with two container IUs

Figure 13 shows an IU deployment descriptor whose root IU also defines a solution module. This solution module includes a container IU, whose units are all targeted to the same hosting environment, and another solution module, whose units are individually targeted to their own hosting environment.

IU deployment descriptor

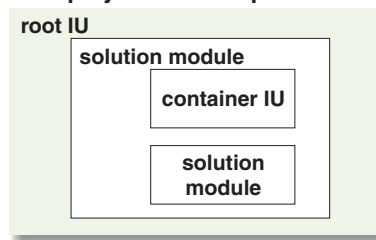


Figure 13. An IU deployment descriptor whose root IU defines a solution module with a single-target container IU and a multitarget solution module

The solution module contains XML elements that define the following kinds of information:

- The identity (name, UUID, and version) of the solution module
- Definitions of any variables used during software deployment
- Subordinate units of the solution module

Figure 14 on page 27 shows an XML fragment from an IU deployment descriptor. The XML fragment represents a solution module. In an IU deployment descriptor, the solution module (the **solutionModule** element) is always a nested element within the top-level **iudd:rootIU** element. In the figure, the solution module encapsulates a single, smallest IU. The **solutionModule** element is highlighted in bold type:

```

<iudd:rootIU ...
:
:
<installableUnit targetRef="tOS">
  <solutionModule IUName="SMD_Wrapper">
    <identity>
      <name>SMD Test Wrapper</name>
      <UUID>991974510ba426fe1f53841402350007</UUID>
      <version>1.2.0</version>
    </identity>
    <installableUnit targetRef="tOS">
      <SIU IUName="SMD_IU" hostingEnvType="OSRT:Operating_System">
        <identity>
          <name>SMD Test</name>
          <UUID>991974510ba426fe1f53841402350008</UUID>
          <version>1.2.0</version>
        </identity>
        <unit>
          <installArtifacts>
            <installArtifact>
              <fileIdRef>SMDInstallActions</fileIdRef>
            </installArtifact>
            <uninstallArtifact>
              <fileIdRef>SMDUninstallActions</fileIdRef>
            </uninstallArtifact>
          </installArtifacts>
        </unit>
      </SIU>
    </installableUnit>
  </solutionModule>
</installableUnit>
:
:
<\iudd:rootIU>

```

Figure 14. Sample solution module in an IU deployment descriptor

Configuration units

Configuration unit, or *CU*, is a general term for the entities in an IU or CU deployment descriptor that describe or define the hosting environments of the software package to be configured and the steps required to configure that software package.

In a CU deployment descriptor, one type of configuration unit, called the root configuration unit (root CU), includes no configuration steps but acts as a container for defining the other type, the smallest configuration unit (smallest CU).

To define either type of configuration unit, you specify a set of XML elements and attributes in the deployment descriptor. This deployment descriptor is an XML document that you create in accordance with supplied Deployment Engine schemas.

In a CU deployment descriptor, you include XML elements that define one root configuration unit and one or more smallest configuration units. Briefly, these units do the following things:

- **Root configuration unit (root CU).** Defines the hosting environments of the software package to be configured and the overall configuration task to be accomplished by the smallest configuration units.

- **Smallest configuration unit (smallest CU).** Defines the action descriptors that describe how to initially configure a newly installed software package, or perform follow-on configurations, in a single hosting environment.

When used within a *CU* deployment descriptor, configuration units change the current configuration of a previously deployed software package. When used within a *IU* deployment descriptor, configuration units perform any needed initial configuration of the subordinate units for the following:

- The current root IU
- A root IU that is referenced by a contained IU, a contained container IU or a requisite of the current root IU

The XML elements for the two types of configuration units can be organized hierarchically in an IU or CU deployment descriptor. The valid hierarchical structures for configuration units are described in “Unit hierarchies” on page 31. A Deployment Engine schema governs how, and in what combinations, the configuration units can be organized in a deployment descriptor in order to configure software.

Root and smallest configuration units are described in the sections that follow.

Root configuration units: Every CU deployment descriptor must have one, and only one, root configuration unit. A *root configuration unit*, or *root CU*, defines the hosting environments of the software package to be configured and the overall configuration task to be accomplished by the smallest configuration units. It also acts as a container for all the smallest configuration units in the CU deployment descriptor.

The root configuration unit is the top-level XML element in a CU deployment descriptor. Every smallest configuration unit of a CU deployment descriptor is included in the root CU. The XML elements in the root CU define the following kinds of information:

- The identity (name, UUID) of the configuration task
- Smallest configuration units
- The XML schema (location and version) that this root CU conforms to
- Target hosting environments
- The media location of the action descriptors associated with the smallest configuration units
- The media location of the payload files associated with the smallest configuration units

Figure 15 on page 29 shows a sample CU deployment descriptor. The **iudd:rootCU** element is the root CU. The root CU contains all the other XML elements in the CU deployment descriptor. The root CU in this sample includes a smallest CU, represented (under the **configurationUnit** element) by the **SCU** element:

```

<iudd:rootCU
  xmlns:iudd="http://www.ibm.com/namespaces/autonomic/solutioninstall/IUDD"
  xmlns:OSRT="http://www.ibm.com/namespaces/autonomic/OS_componentTypes"
  CUName="CURoot" targetRef="tOS">

  <identity>
    <name>CURoot</name>
    <UUID>EEFFDDCCBBAA00998877665544332211</UUID>
  </identity>

  <configurationUnit>
    <SCU CUName="SCU_A">
      <unit>
        <configArtifacts>
          <configArtifact>
            <fileIdRef>configArtifact_SCU_A</fileIdRef>
          </configArtifact>
        </configArtifacts>
      </unit>
    </SCU>
  </configurationUnit>
  <rootInfo>
    <schemaVersion>1.2.1</schemaVersion>
    <build>123456</build>
  </rootInfo>

  <topology>
    <target id="tOS" type="OSRT:Operating_System" />
  </topology>

  <files>
    <file id="configArtifact_SCU_A">
      <pathname>scuAconfigArtifact.xml</pathname>
      <length>0</length>
      <checksum>0</checksum>
    </file>
    <file id="SCUFileA">
      <pathname>../FILES/rootcu_scu_A.txt</pathname>
      <length>1</length>
      <checksum />
    </file>
  </files>

</iudd:rootCU>

```

Figure 15. Sample CU deployment descriptor with mandatory root CU and one smallest CU

The root configuration unit is the sole, top-level node in a hierarchy with smallest configuration units as its leaf nodes. In the CU deployment descriptor, all of the smallest configuration units are nested XML elements under the root configuration unit

A hierarchy of configuration units is illustrated in “Configuration unit hierarchies” on page 33.

Smallest configuration units: A *smallest configuration unit*, or *smallest CU*, is a basic unit for setting up or customizing newly or previously installed software package for a particular use or environment. A smallest configuration unit references action descriptors that describe how to configure the software package. The software being configured is always associated with a single hosting environment.

Deployment Engine interfaces only with the touchpoint of a hosting environment. The touchpoint, in turn, reads the actions which originate in the action descriptor that is defined by the smallest configuration unit, and configures the hosting environment or one of its hosted resources accordingly (for details, see page 9).

You specify a smallest configuration unit in a deployment descriptor. In a CU deployment descriptor, the smallest configuration unit is specified as an element of the root configuration unit. In an IU deployment descriptor, the smallest configuration unit is specified as an element of a root installable unit, container installable unit, or solution module.

Used within a *IU* deployment descriptor, smallest configuration units initially configure the software package that the IU deployment descriptor is deploying. An IU deployment descriptor is processed one time only. You define smallest configuration units in the IU deployment descriptor when you need to configure some of its installable units either during installation or migration.

In the IU deployment descriptor, you can define smallest configuration units to configure subordinate installable units for the following root IUs:

- The root IU of the deployment descriptor itself.
- The root IU of another IU deployment descriptor. This is an external root IU that is referenced by a contained IU, a contained container IU, or a requisite in the current deployment descriptor.

Used within a *CU* deployment descriptor, smallest configuration units change the current configuration of previously deployed applications. Typically, you would use the CU deployment descriptor to perform one or more follow-on configurations to an application that you deployed sometime in the past. A CU deployment descriptor can be processed repeatedly, as needed, to configure or reconfigure the application or another target hosted resource in a particular hosting environment.

A smallest configuration unit consists of a sequence of configuration steps, like a script. Unlike a smallest installable unit, the smallest configuration unit does *not* represent installable software and does *not* have an install-uninstall life cycle. Rather, a smallest configuration unit is used in the CU deployment descriptor to apply required configuration settings to a previously deployed application after the fact—that is, after any initial configuration. A smallest configuration unit has the capability to do the same for other hosted resources. These resources can be hosted resources that are required by applications using Deployment Engine, even though the resources were not created as part of a Deployment Engine-based deployment. However, the recommended and safest use is for performing follow-on configurations of a software package that belongs to your own application.

Figure 16 on page 31 shows a CU deployment descriptor whose root CU references one smallest configuration unit. This CU deployment descriptor represents the most elementary form of software configuration. The configuration task, plus the media location of all the action descriptor and payload files, are defined in the root CU. The action descriptors themselves are defined in the smallest configuration unit.

CU deployment descriptor

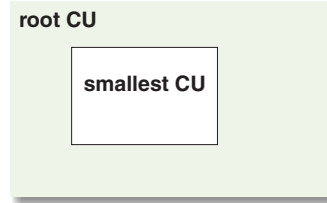


Figure 16. A CU deployment descriptor with mandatory root CU and one smallest CU

For a picture that shows the various ways a smallest configuration unit might appear when organized hierarchically in an IU deployment descriptor, see Figure 18 on page 33.

As noted, a smallest configuration unit can be an XML element in either a CU or IU deployment descriptor. The XML elements and attributes of the smallest configuration unit define the following kinds of information:

- The identity (name) of the smallest configuration unit
- Definitions of any variables used during software deployment
- Definitions of checks that compare actual values of a property against a defined value in the IU deployment descriptor
- Requirements of the container IU
- A reference to the action descriptor for the smallest configuration unit

Figure 17 shows an XML fragment from a CU deployment descriptor. The XML fragment represents a smallest configuration unit. In a CU deployment descriptor, the smallest configuration unit (the **SCU** element) is always a nested element within the top-level **iudd:rootCU** element. In Figure 17, the **SCU** element is highlighted in bold type:

```
<iudd:rootCU ...  
:  
<configurationUnit>  
  <SCU CUName="SCU_A">  
    <unit>  
      <configArtifacts>  
        <configArtifact>  
          <fileIdRef>configArtifact_SCU_A</fileIdRef>  
        </configArtifact>  
      </configArtifacts>  
    </unit>  
  </SCU>  
</configurationUnit>  
:  
</iudd:rootCU>
```

Figure 17. An XML fragment representing a smallest CU in a CU deployment descriptor

Unit hierarchies

Multiple installable units, when packaged together, must be organized hierarchically, or nested, in the deployment descriptor under the root IU. The same is true for multiple configuration units, except that they can also be nested under a root CU.

Installable unit hierarchies: An *installable unit hierarchy*, or *IU hierarchy*, is a structure of installable units, configuration units, or both, subordinated under one top-level root IU, that together function as a tree. You create an installable unit hierarchy whenever you develop an IU deployment descriptor in accordance with the Deployment Engine schema.

In the IU deployment descriptor, installable units and configuration units are subordinate to the root IU, and must be represented as nested XML elements under the root IU. These subordinate units are used by Deployment Engine to deploy a software package. Any unit in the tree (except a container IU) can be associated with its own action descriptor and payload files.

The basic units of the tree must be organized as follows:

- One root node, a root IU
- One *or more* levels of the following intermediate nodes (intermediate nodes always have nodes both above them and below them in the hierarchy):
 - A solution module
 - A container IU
- One or more of the following leaf nodes:
 - A smallest IU
 - A smallest CU
 - A contained IU
 - A contained container IU

Think of the root IU as the sole, top-level node in a hierarchy of multiple subordinate units. This hierarchy must include other units as leaf nodes, and sometimes as intermediate nodes. Either way, the hierarchy always includes a root installable unit as the topmost, or root, node. In the IU deployment descriptor, all other installable or configuration units are nested XML elements under the root installable unit.

Figure 18 on page 33 shows a hierarchy of installable units with four node levels. The figure illustrates a root node, two levels that contain intermediate nodes (there can be one or more levels that contain intermediate nodes) and three levels that contain leaf nodes.

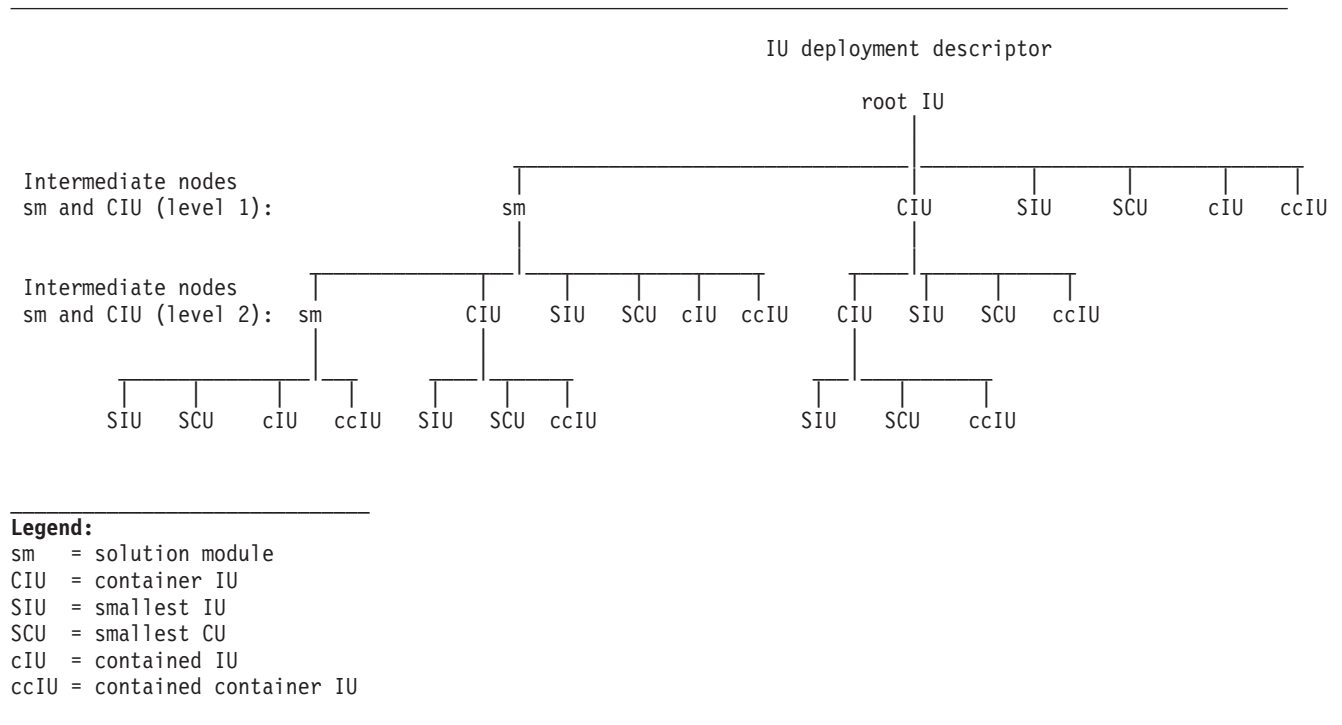


Figure 18. A hierarchy of installable units in an IU deployment descriptor

Configuration unit hierarchies: A *configuration unit hierarchy*, or *CU hierarchy*, is a structure of smallest configuration units, subordinated under one top-level root CU, that together function as a tree. You create a configuration unit hierarchy whenever you develop a CU deployment descriptor in accordance with the Deployment Engine schema.

In the CU deployment descriptor, smallest configuration units are subordinate to the root CU, and must be represented as nested XML elements under the root CU. The subordinate units, or smallest configuration units, are used by Deployment Engine to configure a previously installed software package.

Figure 19 shows a hierarchy of configuration units with one root node and two leaf nodes. Note that in a configuration unit hierarchy there are no intermediate nodes:

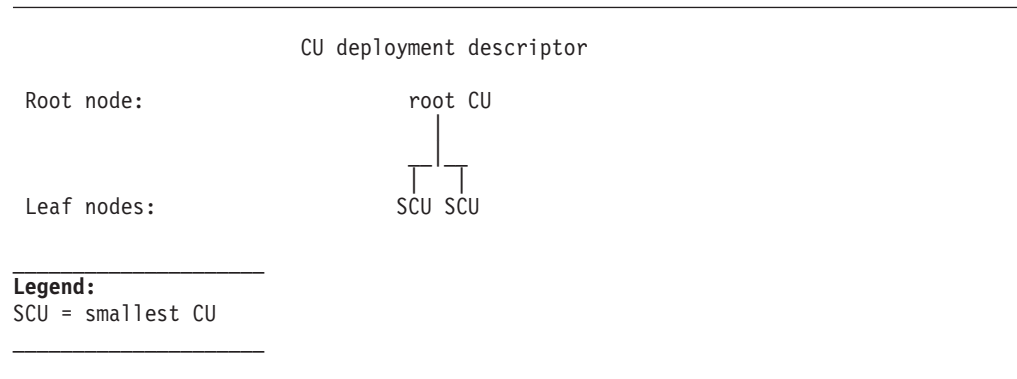


Figure 19. A hierarchy of configuration units in a CU deployment descriptor

Optional content

Base content indicates which installable units to deploy, independently of what selections the user makes at deployment time. Base content, in other words, is the application content that is always meant to be deployed. *Optional content*, on the other hand, indicates which installable units to deploy based on what a user does select when given some choices. Optional content, then, refers to installable units that are not required to be deployed as the core part of an application but instead can be deployed at the user's option; for example, some additional application functionality, samples, or documentation.

There are two types of optional content: *features*, and preselected sets of features called *installation groups*. If appropriate for your application, you can define features, installation groups, or both in the IU deployment descriptor of your software packages. Each feature that you define can specify one or more installable units for inclusion within that feature. Potential installable units that might be included in features are defined as *selectable content* in separate section of the IU deployment descriptor. The installable units that are actually deployed to the hosting environment are determined by which features and installation groups the user chooses during the software deployment process, and which installable units those features and installation groups include from the selectable content.

You identify the installable units that you want to make available for inclusion by features under **installableUnit** elements that are nested under a **selectableContent** element of the IU deployment descriptor. The features that can include those installable units are defined by their own separate elements elsewhere in the deployment descriptor (see “Features” on page 35 and “Installation groups” on page 36 for details).

Figure 20 shows an XML fragment from an IU deployment descriptor. The XML fragment represents two contained IUs defined as selectable content. A feature can include either or both of these contained IUs simply by referencing them. In an IU deployment descriptor, the selectable content (the **selectableContent** element) is always a nested element within the top-level **iudd:rootIU** element. In Figure 20, the **selectableContent** element is highlighted in bold type:

```
<iudd:rootIU
:
:
  <selectableContent>
    <installableUnit>
      <containedIU IUName=contained1>
        <fileIdRef>app1</fileIdRef>
      :
    :
  :
```

Figure 20. An XML fragment representing selectable content in an IU deployment descriptor (Part 1 of 2)

```

        </containedIU>
      </installableUnit>
    <installableUnit>
      <containedIU IUName=contained2>
        <fileIdRef>app2</fileIdRef>
      </containedIU>
    </installableUnit>
  </selectableContent>
</iudd:rootIU>

```

Figure 20. An XML fragment representing selectable content in an IU deployment descriptor (Part 2 of 2)

Features: A *feature* is an optional, separately deployable set of installable units that represent some specific functionality of a larger software application. A feature is a type of optional content that your application can offer as an installation choice to users at software deployment time. Samples, language packs, documentation, or even applications in a suite are considered features. Features provide the mechanism that enables a user to select some optional installable units in the root installable unit to deploy. Offering features as part of your application is not required.

The following things are notable about features:

- Features are pointers to selectable content.
- As such, features map to installable units. For example:
 - Feature A maps to IU_1.
 - Feature B maps to IU_2, IU_3.
 - Feature C maps to IU_4.
- Features can contain other features.
 - Feature E:
 - Feature E1 maps to IU_5.
 - Feature E2 maps to Package Y, Feature S.
- Features can have selection rules based on what the user selects:
 - If the user selects this feature, another feature is automatically selected
 - If the user selects this feature, another feature is automatically deselected
 - If the user deselects this feature, another feature is automatically selected
 - If the user deselects this feature, another feature is automatically deselected
- Features can reference the features of other software packages.
 - Feature D maps to Package X Feature Q.
 - Feature E2 maps to Package Y, Feature S.

Features contain XML elements that define the following kinds of information:

- The name of the feature—a unique name that can optionally take the value of the `displayName` attribute
- A human-readable descriptive name for the feature (the value of the `displayName` attribute)

- Nested subfeatures
- Installable units that are selected by an IUNameRef attribute from a top-level installable unit defined as optional content of the root IU
- References to features in another software package referred to by a contained IU, contained container IU, or requisite
- Optional selection rules (each selection rule identifies the name of a different feature within the root installable unit, where the specified feature is the subject of one of the selection rules noted earlier in this section)

Figure 21 shows an XML fragment from a IU deployment descriptor. The XML fragment represents two features. In an IU deployment descriptor, the features appear within the **features** element, which is always a nested element within the top-level **iudd:rootIU** element. In Figure 21, the **features** element is highlighted in bold type:

```

<iudd:rootIU ...
:
:
<features>
  <feature featureID="BFeature">
    <identity>
      <name>Feature B</name>
    </identity>
    <IUNameRef>MyWorld</IUNameRef>
    <selectionRules>
      <deselectIfSelected>BFeaturePlus</deselectIfSelected>
    </selectionRules>
  </feature>
  <feature featureID="BFeaturePlus">
    <identity>
      <name>Feature B Plus</name>
    </identity>
    <IUNameRef>MyWorld</IUNameRef>
    <IUNameRef>HelloWorld</IUNameRef>
    <selectionRules>
      <deselectIfSelected>BFeature</deselectIfSelected>
    </selectionRules>
  </feature>
</features>
:
:
</iudd:rootIU>

```

Figure 21. An XML fragment representing features in a IU deployment descriptor

Installation groups: An *installation group* is a set of application features that are preselected for the user. An installation group is a type of optional content that your application can offer users at software deployment time. If a user selects the installation group, all the features in the installation group are preselected for deployment. Offering installation groups as part of your application is not required.

An installation group can provide a set of features based on user roles (developer, administrator, general user), usage criteria (typical, compact, full, custom), or any other criteria deemed relevant or beneficial to application users.

The following things are notable about installation groups:

- Installation groups provide alternatives to users.

- One installation group should be selected by default.
- One installation group cannot contain another installation group.

The same feature can appear in one or more installation groups:

- InstallationGroup_1
 - Feature A
 - Feature B
 - Feature C
- InstallationGroup_2
 - Feature B
 - Feature C
 - Feature D
 - Feature E
 - Feature E1
 - Feature E2

Installation groups contain XML elements that define the following kinds of information:

- The name of the installation group
- An optional, human-readable description
- A list naming the features that should be selected for installation if the group is selected

Figure 22 shows an XML fragment from a IU deployment descriptor. The XML fragment represents three installation groups. In an IU deployment descriptor, the installation groups appear within the **groups** element, which is always a nested element within the top-level **iudd:rootIU** element. In Figure 22, the **groups** element is highlighted in bold type:

```

<iudd:rootIU ...
:
:
<groups>
  <group>
    <groupName>Group1</groupName>
    <feature featureIDRef="Feature_Code_A" selection="selected"/>
    <feature featureIDRef="Feature_Code_B" selection="selected"/>
    <feature featureIDRef="Feature_Documentation_B" selection="selected"/>
  </group>
  <group>
    <groupName>Group2</groupName>
    <feature featureIDRef="Feature_Code_A" selection="selected"/>
    <feature featureIDRef="Feature_Code_B" selection="selected"/>
  </group>

```

Figure 22. An XML fragment representing installation groups in a IU deployment descriptor (Part 1 of 2)

```

    <group>
      <groupName>Group3</groupName>
      <feature featureIDRef="Feature_Documentation_B" selection="selected"/>
    </group>
    <default>Group1</default>
  </groups>
  :
  :
</iudd:rootIU>

```

Figure 22. An XML fragment representing installation groups in a IU deployment descriptor (Part 2 of 2)

Features and installation groups determine which installable units to deploy, but they do not determine the order in which the installable units should be deployed. Deployment Engine determines the deployment sequence based on the hierarchy of the installable units in the deployment descriptor.

Managed resources

A *managed resource* is an entity that exists in the run-time environment of an IT system and that can be managed.

Deployment Engine typically views a managed resource as one of the following things:

- The hosting environment (for example, a target operating system)
- An entity in the hosting environment where Deployment Engine ultimately deploys the software (for example, a target file system)
- The deployed software itself (when installed in a hosting environment)
- Software (already installed in a hosting environment) that is used in conjunction with the software to be deployed; for example, installed prerequisite or corequisite software

The topology (see page 38) is the part of the IU deployment descriptor that defines the managed resources that are the logical targets to use, for example, when deploying installable units, checking variables for property queries, or checking dependencies.

Backing resources

A *backing resource* is a physical, managed resource in a hosting environment that is represented in the Deployment Engine installation database by the database entry for an installable unit. There is a one-to-one correspondence between the backing resource and the database entry for the installable unit. Deployment Engine monitors the backing resource to ensure that nothing other than Deployment Engine itself modifies it. The installable unit and the backing resource it represents are linked by a Deploys relationship (see page 61) stored in the relationship registry of the installation database.

In the IU deployment descriptor, a **backing_resource** element nested within the **identity** element of the installable unit allows the touchpoint for the hosting environment to locate the physical backing resource in the hosting environment. The **backing_resource** element does this by referring to a specific hosted resource, called a *resulting resource*, within a topology in the IU deployment descriptor.

Topologies

A *topology* is an XML element in a deployment descriptor that defines, by means of a collection of target definitions, the required hardware and software environment

of the application to be deployed as well as the hosted resources that are created as a result of that deployment. (Such resulting hosted resources are considered to be *backing resources*, because they physically *back up* the installable units that get registered in the Deployment Engine installation database as a result of the application deployment.)

Each target in a topology typically defines one managed resource that represents a part of the topology that the application needs in order to function properly. A target might define a managed resource where installable units can be deployed, where configuration units must be processed, or where one or more checks (for example, a property check) should be performed. For an installable unit that uses (refers to) a particular target, the target can optionally define the resulting hosted resource that will exist in the hosting environment after an installable unit that uses the target is deployed.

In the deployment descriptor, XML elements for installable units, configuration units, or checks include a reference to whichever topology target is pertinent to them. XML elements for installable units or configuration units that do not reference or inherit a target are assumed to have *multiple* targets.

Figure 23 shows two XML fragments within an IU deployment descriptor. The XML fragments represent an installable unit, FOO_TABLE, plus a topology and one of its targets, tDB2. The installable unit refers to its target using the `targetRef="tDB2"` attribute. The installable unit specifies that it has a backing resource, FooTbl, as denoted by the **backing_resource** element. The topology target, tDB2, includes a **hostedResources** element. This element defines the resulting hosted resource, FooTbl, that will exist in the hosting environment after an installable unit that refers to the target—in this case installable unit FOO_TABLE—is deployed.

```
<iudd:rootIU ...
:
:
<installableUnit targetRef="tDB2">
  <SIU IUName="FOO_TABLE">
    <identity>
      <name>Foo Table</name>
      <UUID>991974510ba426fe1f53841402350014</UUID>
      <description>
        <defaultLineText key="ST_01">
          Database table for the Foo Business App
        </defaultLineText>
      </description>
      <version>1.0.0</version>
      <backing_resource>FooTbl</backing_resource>
    </identity>
    <unit>
      <installArtifacts>
        <installArtifact undoable="false">
          <fileIdRef>TableInstall</fileIdRef>
          <type>DDL</type>
        </installArtifact>
      </installArtifacts>
    </unit>
  </SIU>
</installableUnit>
```

Figure 23. Installable unit representing a resulting hosted resource in an IU deployment descriptor (Part 1 of 2)

```

:
:
<topology>
  <target type="RDBRT:IBMDB2UDB" id="tDB2">
    <hostedResources>
      <resulting id="FooTbl1" type="RDBRT:RDB_Table">
        <name>FooTable</name>
      </resulting>
    </hostedResources>
    <selectionRequirements>
      <requirement name="dr1">
        <alternative name="alt_1">
          <inlineCheck>
            <property checkId="IsFooDataBase">
              <propertyName>name</propertyName>
              <value>Foo Database</value>
            </property>
          </inlineCheck>
        </alternative>
      </requirement>
    </selectionRequirements>
  </target>
:
:
</topology>
:
<\iudd:rootIU>

```

Figure 23. Installable unit representing a resulting hosted resource in an IU deployment descriptor (Part 2 of 2)

Requisites

Requisites is an XML element in a deployment descriptor that defines, by means of one or more nested **referencedIU** elements, any prerequisites that are needed by the application to be deployed. The “application” in this case can be a base, full update, incremental update, or fix software package. The needed prerequisites are sometimes referred to as *requisite packages*, because they represent additional software packages that are needed by, and provided with, the software package to be deployed. A requisite package can be either a base, full update, or incremental update software package. A requisite package cannot be a fix software package.

The requisite packages are included, together with the software package that needs them, in a software package tree. In the tree, the dependent software package can include a check in its deployment descriptor that looks for the needed prerequisite in the target hosting environment. In the event the check determines that the needed prerequisite cannot be found in the hosting environment (or you do not want to use the software instance that *is* found), the requisite package can readily be deployed from the software package tree.

Recall that a software package tree is a hierarchy of software packages that make up a single application, solution, or suite. The hierarchy of software packages is usually processed together, as one change request and one software deployment. In a software package tree, the IU deployment descriptor of one, top-level software package (the parent software package) references the IU deployment descriptor of one or more other software packages (the child software packages), which are either requisites, contained IUs, or contained container IUs.

In the case of contained IUs (or contained container IUs), these are integral parts of the software package tree and must be processed together with the parent software package. The parent is always responsible for the installation, maintenance, and removal of all its software packages, including any required software supplied as child software packages. Other applications on the computer cannot share the required software without retaining the entire software package tree—since its parent and child software packages are fully integrated and cannot be separated. Deployment Engine cannot remove the parent software if another application is sharing any of its child software.

As an alternative to using contained IUs (or contained container IUs), a software package tree can use requisites to deploy a child software package. With requisites, however, the child software package is deployed only when needed, and once it is deployed, it can be shared and the parent software package has no further obligation to maintain or remove it. In fact, the child software package is detached from the parent, such that even if the parent is eventually removed, the child remains behind and must be removed independently.

The software package developer can add to the deployment descriptor a requisites section that identifies needed software that the application can deploy, use as a prerequisite, and share without having to maintain it in the future. The requisites section consists of one or more nested **referencedIU** elements—one element for each prerequisite. They behave similarly to **containedIU** elements but additionally include a **resultingResource** element. This element is linked to a required resource in the hosted resources section of the deployment descriptor's topology. The **resultingResource** element indicates that the requisite package, if installed, will satisfy the hosted resource requirement and become a resulting resource whose name and version match the values indicated by the **required** element of the hosted resources section.

Figure 24 shows two XML fragments within an IU deployment descriptor. The XML fragments represent a topology section with a targeted hosted resource that requires Tomcat application server, and a requisites section with a **referencedIU** element that includes a **resultingResource** element linked to Tomcat, indicating that, if the referenced IU is installed, will satisfy the hosted resource's application server requirement.

```
<iudd:rootIU ...  
:  
:  
  <topology>  
    <target id="tOperatingSystem" type="OSRT:Operating_System">  
      <hostedResources>  
        <required id="tomcat_resource" type="Software">  
          <name>Tomcat</name>  
          <version>4.0</version>  
        </required>  
      </hostedResources>  
    </target>  
  </topology>  
:  
:
```

*Figure 24. An IU deployment descriptor whose topology includes a hosted resource that specifies a prerequisite. The deployable, requisite package that represents the prerequisite is indicated in a **referencedIU** element. (Part 1 of 2)*

```

    <requisites>
      <referencedIU IUName="Tomcat">
        <UUID>100074510ba426fe1f53841402350001</UUID>
        <version>4.0</version>
        <fileIdRef>ref_to_ref_IUDD</fileIdRef>
        <resultingResource resourceRef ="tomcat_resource">
      </referencedIU>
    </requisites>
  :
</iudd:rootIU>

```

*Figure 24. An IU deployment descriptor whose topology includes a hosted resource that specifies a prerequisite. The deployable, requisite package that represents the prerequisite is indicated in a **referencedIU** element. (Part 2 of 2)*

Figure note: The **UUID** and **version** elements under the **referencedIU** element are included so that it is possible to determine from looking at the top-level (parent) software package what requisite (child) package is being referred to. The **UUID** and **version** elements are required.

The requisite package specified in the **referencedIU** element can be used either to install the prerequisite or to update an existing instance of the prerequisite. Either way, the requisite package will be deployed before the parent software package. When multiple requisite packages must be deployed, they will be processed in the order they appear in the requisites section of the IU deployment descriptor.

Which units are most appropriate for my IU deployment descriptor?

To make your application Deployment Engine–compliant, you must choose a wrappered or fully enabled approach to your software deployment, as described in “Software deployment models” on page 10.

If you choose a wrappered software deployment, then you could design a simpler IU deployment descriptor with a minimum number of elements and installable units that together act as a wrapper for your current installation program.

If you choose a fully enabled software deployment, as you might if you are designing a new application or you have decided to rework the installation program of an existing application, then your IU deployment descriptor design is probably more complex.

The sections that follow offer some suggestions to help you design and organize your IU deployment descriptor.

Root IU and smallest IUs: Every IU deployment descriptor must have a root IU. The root IU sits at the top of the IU hierarchy and subordinates any other units in the descriptor, including configuration units.

Every IU deployment descriptor must also have one or more smallest IUs that are the leaf nodes of the IU hierarchy. These smallest IUs are associated with action descriptors that actually deploy the software to one or more hosting environments.

IUs that reference other applications (optional): If you are providing a solution, a suite, or need to include some prerequisite software with your application, you can add contained IUs or contained container IUs as additional leaf nodes in your

descriptor. Use these installable units when you want to deploy the software from another IU deployment descriptor as part of your own application deployment.

Contained IUs and contained container IUs are functionally the same. They cause the software represented by another IU deployment descriptor (that is, another root IU) to be deployed as if it were part of the current IU deployment descriptor (your application). The only difference between a contained IU and contained container IU is that a contained container IU deploys all of its referenced software (all the subordinate installable units from the other IU deployment descriptor) to one target hosting environment. A contained IU deploys its referenced software to whatever hosting environment each individual IU from the other IU deployment descriptor specifically targets. So use a contained container IU when the target of your deployment is a single hosting environment and use a contained IU when the target may be multiple hosting environments.

IUs that organize other IUs (optional): You must also decide if any intermediate nodes are required for your application. In this case you would be adding container IUs or solution modules to organize your leaf nodes (see Figure 18 on page 33). Container IUs and solution modules encapsulate other units in your IU deployment descriptor that must be acted upon together (deployed together, configured together, removed together) or targeted to the same hosting environment.

Container IUs and solution modules are functionally the same. The only difference between them is that the former deploys all of its subordinate installable units to the same hosting environment and the latter deploys its subordinate installable units to whatever hosting environment each individual installable unit specifically targets—which may be more than one hosting environment. So use a container IU when the target of its encapsulated units is the same hosting environment and use a solution module when the encapsulated units have target hosting environments that are different from one another.

Smallest CUs that initiate one-time configuration (optional): Include one or more smallest CUs as leaf nodes if any installable units in your application require one-time configuration. The action descriptor associated with the smallest CU can initially configure either a newly installed base software package or a previously installed update software package in a single hosting environment. The processing of a smallest CU can be subject to a certain set of conditions. Therefore design your IU hierarchy accordingly, so that a smallest CU is at the same node level as installable units to be processed in the same hosting environment or under the same conditions. If necessary, you can create intermediate nodes with container IUs and solution modules that serve as organizing entities for encapsulating a smallest CU and related installable units. If the intermediate nodes already exist, add the smallest CU as a leaf node so that it will be processed along with the other leaf or leaves of the same intermediate node.

With a simpler configuration, you might need just one smallest CU in the IU deployment descriptor to configure your entire application. Just make the smallest CU a leaf node of the root IU. To configure another application that your application is deploying, associate a smallest CU with the contained IU that references the other application by adding the smallest CU under the same intermediate node as the contained IU. The smallest CU can then provide one-time configuration of the referenced application. (The valid locations for smallest configuration units in an IU hierarchical structure is shown in Figure 18 on page 33.)

Optional content (optional): Decide whether you want to include any optional content. Optional content consists of features or installation groups whose deployment depends on what the user selects at install time. For installable units that you want to include as selectable content for features, enclose their XML elements within a **selectableContent** element in the IU deployment descriptor. All other installable units are considered base content. Base content is deployed regardless of what features or installation groups the user selects.

If you include features, you need to create a **features** element in your descriptor. Under this element you can nest individual **feature** elements, each of which maps to all the applicable installable units that comprise the feature (the installable units must be included in your selectable content). The feature uses an **IUNameRef** element to reference each installable unit in the selectable content. If a feature includes many installable units, you could have a long list of references. You might want to organize the feature's installable units under an intermediate node in your IU deployment descriptor, and then just reference the intermediate node with the **IUNameRef** element. This is illustrated in Figure 25, where the referenced intermediate node would be a solution module. By referencing the intermediate node, all the installable units below it in the IU hierarchy will be included in the feature:

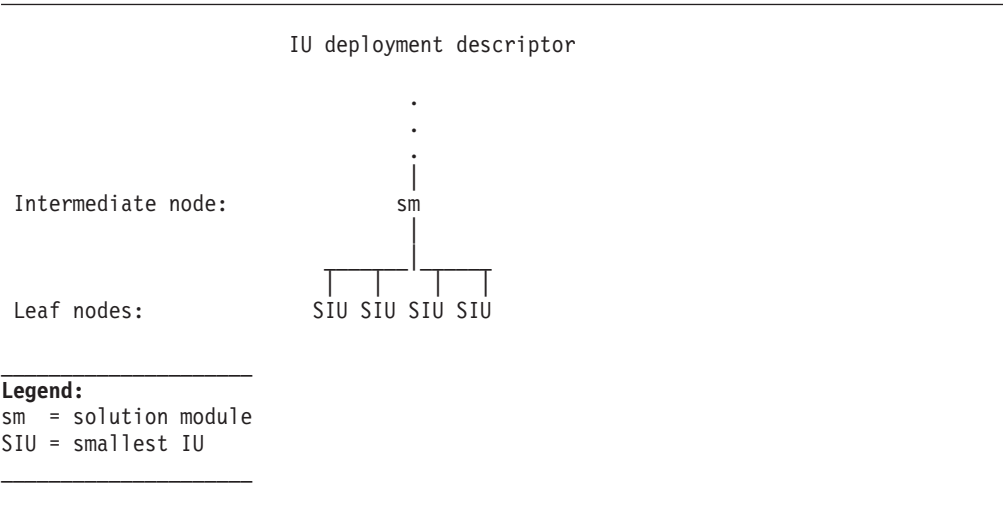


Figure 25. A solution module can be used to organize some smallest IUs associated with a feature

Each **feature** element also provides a feature name and any displayable text, selection rules, or other information that describes the feature.

If you include installation groups, you need to create a **groups** element in your descriptor with nested **group** elements that map to the appropriate features you defined. Each **group** element also provides a group name and any other information relevant to the group.

Summary of IU deployment descriptor entities and their uses: To help you decide which units are most appropriate to implement in your IU deployment descriptor, review the information in Table 1 on page 45. The table summarizes how the various entities in the IU deployment descriptor differ from one another, their primary uses, whether they serve as containers or leaves in an IU hierarchy, and the target environments they are applicable to. Refer to Figure 18 on page 33 for a picture that shows how these entities can be combined within an IU hierarchy. For more information on how to code an IU deployment descriptor, and

for additional elements and attributes that can be included in a descriptor, see the related document, *IBM Autonomic Computing: Installable Unit Deployment Descriptor Specification*.

Table 1. Using entities in the IU deployment descriptor

Entity type	Function and uses	Node type	Target environment for this installable entity or its subordinate units
Root IU	Mandatory installable unit; encapsulates all other unit types.	Top-level container	Multiple hosting environments
Smallest IU	Use as the simplest wrapper for deploying all or some of an application's payload files to one target environment. Use to "wrap" an existing installation program in a single action descriptor to make the program Deployment Engine-compliant without further modifications.	Leaf	Single hosting environment
Smallest CU	Use to do any necessary one-time configuration for the application that is being deployed. Because you need one smallest CU for each target environment, you might want to use a container IU to process a smallest CU together with the IUs in the container.	Leaf	Single hosting environment
Container IU	Use to encapsulate some combination of installable units, configuration units, or other container IUs whose payload files have the same target. A container IU confines its encapsulated units to one target that is usually different than the target of the units outside the container IU.	Container	Single hosting environment
Contained IU	Use to reference the root IU of another IU deployment descriptor and deploy that descriptor's software as part of your own deployment.	Leaf	Multiple hosting environments
Contained container IU	Use to reference the root IU of another IU deployment descriptor and deploy that descriptor's software as part of your own deployment.	Leaf	Single hosting environment
Solution module	Use to encapsulate some combination of installable units, configuration units, or other solution modules whose payload files each have their own target.	Container	Multiple environments
Optional content	Use if you plan to define features, installation groups, or both. In addition, selectable content identifies the installable units that you want to include in your features. Such features can then be included in installation groups.	N/A	N/A
Features	Use to define as optional content a separately deployable set of installable units that represent some specific functionality of the larger software application; the user has the option of whether or not to install the feature.	N/A	N/A
Installation groups	Use to define as optional content a set of application features that are already selected for the user. The set of features can be based on user roles (developer, administrator, user), usage criteria (typical, compact, full, custom), or any other criteria deemed relevant or beneficial to your application users.	N/A	N/A

Action descriptors

An *action descriptor* is an XML document whose XML elements define a series of actions to be processed by the touchpoint of a particular hosting environment. Often there are many action descriptors in a single software package.

In an action descriptor, an *action* is the XML representation of a task that needs to be processed by the touchpoint. An action descriptor often contains a number of actions to be performed, like a script. Creating or removing directories, installing or removing application files, and updating registries, configuration properties, environment variables, and paths are all types of actions that can be found in an action descriptor.

The action descriptor defines each action to be performed in order to accomplish a particular change management operation. Actions are specific to a type of hosting environment. As a result, actions are documented in the appropriate touchpoint guide for the hosting environment. For example:

- The book *Solution Install for Autonomic Computing: Operating System Touchpoint Guide and Reference* describes the operating system touchpoint and its available actions
- The book *Solution Install for Autonomic Computing: WebSphere Touchpoint Guide and Reference* describes the WebSphere touchpoint and its available actions

Actions are defined in their own descriptors in order to be external from the deployment descriptor. (The action descriptors are, however, referenced in the deployment descriptor.) By being external, action descriptors separate their defined actions from the other installable unit or configuration unit dependency information found in the deployment descriptor. This separation enables the deployment descriptor to be interpreted by the Deployment Engine components and the actions to be interpreted by the appropriate hosting environment—more specifically, by a touchpoint. Touchpoint owners can define new actions without affecting the current deployment descriptors of applications that already use the touchpoint.

In a *deployment* descriptor, each smallest IU or smallest CU lists the action descriptors that are associated with it. One action descriptor is listed for each change management operation (described on page 54) that the smallest IU or smallest CU is subject to. Only an associated action descriptor can change or configure software in a hosting environment.

Action descriptors that effect real changes in the hosting environment are either life cycle–related action descriptors (Install, InitialConfig, Migrate, Uninstall) or the Configure action descriptor, which does not affect life cycle states (see “Software life cycle” on page 51 and “Life cycle states” on page 52 for details about life cycles and states). Yet another kind of action descriptor is used to perform custom checks on behalf of certain objects in the deployment descriptor—mainly installable units and topologies. The supported action descriptors are described as follows:

Life cycle–related action descriptors

Action descriptors of this kind are applied to a hosting environment in order to perform software life cycle–related actions on a smallest IU. Each smallest IU in an IU deployment descriptor references one or more of these action descriptors:

- Install action descriptor
An *Install action descriptor* contains actions that carry out a Create or Update change management operation:
 - A Create change management operation installs software package or feature files in a hosting environment.
 - An Update change management operation installs maintenance files (fixes, incremental updates, full updates) in a hosting environment.

For file installation purposes, an Install action descriptor can reference both payload files bundled in the software package media and files already deployed on the local computer.

An Install action descriptor processed in reverse equates to an Undo change management operation. An Undo change management operation removes applied maintenance files (fixes, incremental updates, full updates) from a hosting environment and restores previous states.

In the absence of an Uninstall action descriptor, an Install action descriptor is processed in reverse. This equates to a Delete change management operation. A Delete change management operation removes software package or feature files from a hosting environment.

- InitialConfig action descriptor

An *InitialConfig action descriptor* contains actions that carry out an InitialConfig change management operation. An InitialConfig change management operation performs the initial, one-time setup of a software package during software deployment. For first-time setups, an InitialConfig action descriptor can only reference files already deployed on the local computer.

- Migrate action descriptor

A *Migrate action descriptor* contains actions that carry out a Migrate change management operation. A Migrate change management operation performs the initial, one-time configuration of software immediately following an update. For the one-time configuration of updated software, a Migrate action descriptor can only reference files already deployed on the local computer.

- Uninstall action descriptor

An *Uninstall action descriptor* contains actions that carry out a Delete change management operation. A Delete change management operation removes software package or feature files from a hosting environment. For file removal purposes, an Uninstall action descriptor can only reference files already deployed on the local computer.

When these life cycle–related action descriptors are used by a smallest IU in some combination, they are called an *action descriptor set*. Each smallest IU in an IU deployment descriptor references a life cycle–related action descriptor or action descriptor set to indicate which action descriptors can be used with the smallest IU (and therefore which change management operations can be performed on it).

Configure action descriptor

A *Configure action descriptor* is used to process smallest CUs. Smallest CUs can be found in a CU deployment descriptor or in an IU deployment descriptor.

A Configure action descriptor in a CU deployment descriptor contains actions that carry out a Configure change management operation. A Configure change management operation performs follow-on (repeatable) configuration of software. This follow-on configuration does not affect the life cycle state of the software it configures. Each smallest CU in a CU deployment descriptor references a Configure action descriptor to indicate that the action descriptor can be used to process the smallest CU.

A Configure action descriptor in an IU deployment descriptor contains actions that carry out InitialConfig and Migrate change management operations:

- An InitialConfig change management operation performs the initial, one-time setup of a software package during software deployment
- A Migrate change management operation performs the initial, one-time setup of software immediately following an update

Custom-check action descriptors

Action descriptors of this kind are applied to a hosting environment in order to determine if the environment satisfies some specified requirements. A *Custom-check action descriptor* is any action descriptor created to perform custom dependency checking. It is especially designed to check one or more dependencies that are not defined in the Deployment Engine schema (see page 57 for information about the defined checks). Actions in a Custom-check action descriptor usually require the processing of some custom code that the developer provides to perform the checks.

Unlike life cycle–related or Configure action descriptors, which apply only to smallest IUs or CUs and are intended to effect changes in the hosting environment, the Custom-check action descriptor can apply to any subordinate unit of a root IU or root CU and do not change the hosting environment. Almost any object in the deployment descriptor that might have its own unique requirements could make use of a Custom-check action descriptor. In other words, a smallest IU could require a custom check, but so could a topology, a contained IU, or any other type of installable unit.

Figure 26 shows an XML fragment from an IU deployment descriptor whose smallest IU lists four different action descriptors—an action descriptor set—that can be used with it. The action descriptors are life cycle–related, so the list of action descriptors is defined in an **installArtifacts** element. The **installArtifacts** element is defined in the **unit** element. (The media location of each action descriptor is specified elsewhere in the IU deployment descriptor.)

```
<unit>
  <installArtifacts>
    <installArtifact>
      <fileIdRef>Install_action_descriptor</fileIdRef>
    </installArtifact>
    <initialConfigArtifact>
      <fileIdRef>InitialConfig_action_descriptor</fileIdRef>
    </initialConfigArtifact>
    <uninstallArtifact>
      <fileIdRef>Uninstall_action_descriptor</fileIdRef>
    </uninstallArtifact>
    <migrateArtifact>
      <fileIdRef>Migrate_action_descriptor</fileIdRef>
    </migrateArtifact>
  </installArtifacts>
</unit>
```

Figure 26. An XML fragment from an IU deployment descriptor that lists the action descriptor set associated with a smallest IU

Figure 27 on page 49, on the other hand, shows a sample *action* descriptor. The **action:artifact** element contains all the other XML elements in the action descriptor. The action descriptor in this sample includes several installation actions, the **addDirectory** and **addFile** elements, that are defined as an action group.

```

<action:artifact xmlns:iudd="http://www.ibm.com/namespaces/autonomic/solutioninstall/IUDD"
xmlns:osac="http://www.ibm.com/namespaces/autonomic/solutioninstall/OsActions"
xmlns:action="http://www.ibm.com/namespaces/autonomic/solutioninstall/action"
xmlns:base="http://www.ibm.com/namespaces/autonomic/solutioninstall/BASE"
xmlns:command="http://www.ibm.com/namespaces/autonomic/solutioninstall/command"
xmlns:rtype="http://www.ibm.com/namespaces/autonomic/solutioninstall/ResourceTypes"
xmlns:sigt="http://www.ibm.com/namespaces/autonomic/solutioninstall/Signatures"
xmlns:vsr="http://www.ibm.com/namespaces/autonomic/common/version"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/namespaces/autonomic/solutioninstall -
/OsActions osActions.xsd">
  <artifactType>Install</artifactType>
  <artifactSchemaVersion>1.2.1</artifactSchemaVersion>
  <variables>
    <variable name="INSTALL_LOC">
      <parameter defaultValue="siu_test"/>
    </variable>
    <variable name="INSTALL_DIR">
      <parameter defaultValue="$(INSTALL_BASE)/$(INSTALL_LOC)" />
    </variable>
  </variables>
  <builtInVariables>
    <variable xsi:type="osac:OsBuiltInVariable" name="INSTALL_BASE">
      <javaPropertyValue javaPropName="bvt.workdir" />
    </variable>
  </builtInVariables>
  <actionGroup xsi:type="osac:OsActionGroup">
    <actions>
      <addDirectory actionId="AddDir">
        <directory>
          <location>$(INSTALL_BASE)</location>
          <name>$(INSTALL_LOC)</name>
        </directory>
      </addDirectory>
      <addFile actionId="AddFile">
        <file>
          <location>$(INSTALL_DIR)</location>
          <name>siu_test.txt</name>
          <source>
            <fileIdRef>SIUFile1</fileIdRef>
          </source>
        </file>
      </addFile>
    </actions>
  </actionGroup>
</action:artifact>

```

Figure 27. Sample installation action descriptor with addDirectory and addFile actions

Keep in mind that the name of the action descriptor must be referenced in the **unit** element of the deployment descriptor. Also, the location of each action descriptor needs to be specified in a **file** element in the deployment descriptor. See Figure 2 on page 18 for examples of these elements in an IU the deployment descriptor.

Media descriptor

A *media descriptor* is an optional XML document whose elements identify the media location of one or more of the descriptors or deployable payload files in a software package. A software package can have only one media descriptor.

When the location of a file is supplied in a media descriptor, it overrides the original location of that file as defined in the deployment descriptor. If a file location changes due to repackaging, a media descriptor can be used to indicate

the new location without having to update the deployment descriptor. In addition, a media descriptor has the capability to describe file locations that the deployment descriptor cannot; for example, when the application's files—or even individual large files—must be packaged for delivery on multiple volumes of removable media such as CDs, or in archive files. For these locations, a media descriptor must be used.

A media descriptor is optional because the IU or CU deployment descriptor for a software package already identifies the media location of its own files. Thus when no media descriptor is present, all files specified in the deployment descriptor are assumed to have paths relative to the location of the deployment descriptor in the package. However, using a media descriptor to separate the media location from the deployment descriptor often has advantages.

For instance, using media descriptors, a common IU deployment descriptor can be reused in different software packages, even though their media locations differ. In this case, each software package has its own media descriptor that describes its own particular binding information for the action descriptors and deployable payload files. The binding information in each media descriptor overrides the media location supplied in the common IU deployment descriptor.

An example of this is an application that has several alternative media locations. Perhaps the application can be installed from one or more CDs, from a network file server, from the Internet, or from some other media. In this case you can create multiple software packages that have the same IU deployment descriptor but a different media descriptor for specifying each alternative media location.

Another possibility: if you are reorganizing the file locations for a collection of applications being pulled together into a suite or solution, you can create media descriptors that provide the new media locations for these application files by overriding the original locations specified in the various IU deployment descriptors of each application in the suite. Deployment Engine detects the presence of the media descriptor and uses its location information, rather than the locations specified in the IU deployment descriptor, to find the files.

Sometimes use of a media descriptor is mandatory; for example, if your application image is provided on CD-ROM media and is large enough to require multiple CDs. Because all your descriptors should be packaged on the first CD if possible (as a rule, application content is packaged in the order it is used during software deployment), some applications might not have enough room to include the payload files on the same CD. Whenever additional volumes of removable media—in this example, additional CDs—are needed, you must use a media descriptor. Similarly, a media descriptor is required when placing application content in ZIP files or other extractable archive files, because the media descriptor has capabilities that a deployment descriptor does not.

In addition to using a media descriptor for an application that spans several volumes of removable media, such as several CDs, a media descriptor must also be used when any one file is so large that it needs to be segmented and its file segments packaged on separate media volumes. In Deployment Engine, a file that has been segmented in this way is referred to as a *multivolume file*. A multivolume file is still addressable by means of a single file reference. When a software deployment program or other component in the Deployment Engine run-time environment accesses a multivolume file, the file appears exactly the same (within physical limitations such as the time and interaction required to effect disc-swapping) as if it had been read contiguously from a single volume.

A software package can include no more than one media descriptor. If the software package is aggregated into another software package, any media descriptor in the aggregated software package is used by the aggregating software package.

The media descriptor defines the following information:

- The path name of the deployment descriptor relative to the media descriptor
- A default logical source that can be applied to files that are not explicitly bound in the media descriptor
- The physical location of the action descriptors or deployable payload files in the software package
- The location of all the file segments that make up a multivolume file
- Any overrides to file paths for specific files, a function that can be used to map common files that are shared by multiple installable units

Figure 28 shows an XML fragment from a media descriptor. The **media:binding** element contains all the other XML elements in the media descriptor. The media descriptor in this sample includes a **deploymentDescriptor** element that specifies the physical path of its associated deployment descriptor (relative to the location of the media descriptor), a **defaultLogicalSource** element that specifies the default physical location of any files whose locations are not specified in their own a **fileSource** element, and one or more **fileSource** elements that specify the file ID and location of a particular action descriptor or deployable payload file.

```
<media:binding ...>
  <deploymentDescriptor>

  :
  </deploymentDescriptor>
  <defaultLogicalSource>

  :
  </defaultLogicalSource>
  <fileSource>

  :
    <pathname>... </pathname>
  </fileSource>
  <fileSource>... </fileSource>
</media:binding>
```

Figure 28. Sample XML from a media descriptor

Software life cycle

A *software life cycle* is an end-to-end series of states that characterize the condition of a software entity in a hosting environment.

An installable unit has a software life cycle. Its life cycle begins when the installable unit is deployed and ends when it is removed. Throughout the life cycle of an installable unit, change request results can affect its state. Deployment Engine saves relevant state information about installable units in its installation database. States used by Deployment Engine include Created, Usable, and Updated, as described in “Life cycle states” on page 52.

A configuration unit does not have a software life cycle because it is not a software entity. A configuration unit is used to apply configuration settings to a software entity.

Life cycle states

Because only installable units have a software life cycle, only installable units have states. *Life cycle states* are phases in a life cycle. Throughout the life cycle of an installable unit, its current state can be affected by the change management operations that result from change request processing. (Table 2 on page 55 describes the various types of change requests and their corresponding change management operations.) Deployment Engine saves relevant state information about installable units in its installation database.

Deployment Engine supports the following life cycle states for installable units:

Created state

The state of a newly deployed installable unit following a Create operation and prior to any required InitialConfig operation. An InitialConfig operation applies the first-time configuration to an installable unit that is already in the Created state. If no InitialConfig operation is required, the newly deployed installable unit directly enters the Usable state following the Create operation. Otherwise, it remains in the Created state until an InitialConfig operation brings it to the Usable state.

Updated state

The state of an installable unit following an Update operation and prior to any required Migrate operation. A Migrate operation applies configuration to an installable unit that is already in the Updated state. If no Migrate operation is required, the updated installable unit directly enters the Usable state following the Update operation. Otherwise, it remains in the Updated state until a Migrate operation brings it to the Usable state.

Note: An Update operation, if undoable, does not overwrite or delete the original installable unit that is being updated. The payload files of the original installable unit are saved by the touchpoint in a local repository, and the records of the original installable unit are retained in the installation database. Thus all the original files and data are preserved, in the event the update should need to be undone and the original installable unit restored.

Usable state

The state of the following installable units:

- A newly deployed installable unit after its initial configuration
- A newly deployed installable unit that requires no subsequent initial configuration
- A newly updated installable unit after its migration
- A newly updated installable unit that requires no subsequent migration

Change requests

A *change request* is an object that is constructed by a software deployment program and passed to Deployment Engine to request some kind of software change operation. A change request can be associated either with software deployment or with follow-on configuration of the deployed software. Some commands (see the command “manageIU” on page 86) refer to a change request as a *life cycle operation*.

Deployment Engine processes the following kinds of change requests:

Configure change request

A request to perform a follow-on software configuration operation supplied in a change request. Configure in this sense is independent from a one-time initial configuration.

Create change request

A request to place or install software package files in a hosting environment. This process can include additional actions like adding directories, updating registries, setting properties, and updating paths.

Create Feature change request

A request to place or install feature files in a hosting environment. This process can include additional actions like adding directories, updating registries, setting properties, and updating paths.

Delete change request

A request to remove a software package from a hosting environment. Delete is the reverse of Create. The Delete process can include additional actions like removing directories, updating registries, setting properties, and restoring objects.

Delete Feature change request

A request to remove an applied feature from a hosting environment. Delete Feature is the reverse of Create Feature. The Delete Feature process can include additional actions like removing directories, updating registries, setting properties, and restoring objects.

Initial Configure change request

A request to perform the initial, one-time setup of a software package following a Create change request or of a feature following a Create Feature change request.

Migrate change request

A request to perform a software configuration operation following an Update change request (migrate applies to full and incremental updates, but not to fixes).

Reapply Update change request

A request to reapply a previous incremental update or fix to an application. This change request becomes necessary when an incremental update or fix is applied to a base software package, and then one or more features are later added that generate an UpdatesNeedToBeReapplied warning. The Reapply Update process updates the newly added features. Software fixes and incremental updates are types of maintenance.

Undo change request

A request to remove applied maintenance files (fixes and incremental updates only) from a hosting environment and restore previous states. Undo is the reverse of Update. The Undo process can include additional actions like removing directories, updating registries, setting properties, and restoring objects.

Update change request

A request to place or install maintenance files in a hosting environment. This process can include additional actions like creating directories and updating registries, properties files, and paths. Software fixes, incremental updates, and full updates are types of maintenance.

Change management operations

A *change management operation* is an object that the change manager component of Deployment Engine constructs from an incoming change request. The change request identifies a deployment descriptor for the application that needs to be changed. Change manager constructs one change management operation for each installable unit and configuration unit that it finds in the deployment descriptor, regardless of whether the unit requires a change or processing in the hosting environment. (Only smallest IUs and smallest CUs can actually be changed or processed in the hosting environment; the change management operations for the other units are required in order to update information about them in the installation database.) These change management operations collectively form a *change plan*. Change manager builds the change plan to either deploy installable units or to process configuration units.

There are several types of change management operations. They have a direct correspondence to the types of change requests described in “Change requests” on page 52. Consider this scenario: A software deployment program issues a change request. The change request addresses a particular IU deployment descriptor. Each smallest IU in that IU deployment descriptor references action descriptors that indicate to change manager which types of change management operations the smallest IU is subject to. Change manager can implement a change plan only for the smallest IUs that are subject to the change management operation that is compatible with the change request.

In other words, an Initial Configure change request is applicable only to smallest IUs that are subject to InitialConfig operations, a Migrate change request is applicable only to smallest IUs that are subject to Migrate operations, and so on. Thus, if the original change request is an Initial Configure request, the change manager component examines the IU deployment descriptor specified in the Initial Configure change request and identifies all smallest IUs subject to an InitialConfig operation. Each of those smallest IUs has its own dedicated action descriptor for InitialConfig operations. Therefore, when change manager has completed all its checks and is ready to process the smallest IUs, it locates each of their action descriptors for InitialConfig operations and continues on with the deployment.

Note: Unlike an InitialConfig operation, which does not necessarily apply to every smallest IU in a deployment descriptor, a Create operation does apply to every smallest IU—otherwise a smallest IU could never be deployed. Therefore every smallest IU in the deployment descriptor must reference an action descriptor that indicates to change manager that the smallest IU is subject to a Create operation.

Table 2 on page 55 and Table 3 on page 56 show the change request types and which change management operations and action descriptors are compatible with them. Table 2 on page 55 shows only the action descriptors that can be used on installable units. Table 3 on page 56 shows only the action descriptors that can be used on configuration units.

Table 2. The correspondence between change requests, change management operations, and action descriptors for IUs, by function

Change request type	Change management operation type	Action descriptor that can be used on IUs	Function
Create	Create	Install	Place or install software package files in a hosting environment.
Create Feature	Create	Install	Place or install feature files in a hosting environment.
Delete	Delete	Uninstall ¹	Remove a software package from a hosting environment.
Delete Feature	Delete	Uninstall ¹	Remove an applied feature from a hosting environment.
Initial Configure	InitialConfig ²	InitialConfig	Perform the initial, one-time setup of a software package during software deployment.
Migrate	Migrate ³	Migrate	Perform initial, one-time configuration to software immediately following an update.
Undo	Undo	Install ⁴	Remove applied maintenance files (fixes, incremental updates, full updates) from a hosting environment and restore previous states.
Update	Create	Install	Place or install software package files in a hosting environment.
Update	Update	Install	Place or install maintenance files (fixes, incremental updates, full updates) in a hosting environment.
Reapply Update	Update	Install	Place or install maintenance files (fixes and incremental updates only) in a hosting environment.

¹ If no Uninstall action descriptor is specified, Deployment Engine processes the Install action descriptor in reverse to remove the software package, applied feature, or applied maintenance.

² InitialConfig differs from Configure (described in Table 3 on page 56). InitialConfig (like Migrate) changes the life cycle state of the software it configures. InitialConfig changes the state from Created state to Usable state. Configure, on the other hand, acts upon software *already in* Usable state, and the state of that software remains unaltered. Configure is also repeatable, not just a initial, one-time setup like InitialConfig.

³ Migrate differs from Configure (Configure is described in Table 3 on page 56). Migrate (like InitialConfig) changes the life cycle state of the software it configures. Migrate changes the state from Updated state to Usable state. Configure, on the other hand, acts upon software *already in* Usable state, and the state of that software remains unaltered.

⁴ Processed in reverse.

Table 3 on page 56 shows the change request types and which change management operations and action descriptors used by configuration units are compatible with the change requests:

Table 3. The correspondence between change requests, change management operations, and action descriptors for CUs, by function

Change request type	Change management operation type	Action descriptor that can be used on CUs	Function
Configure	Configure ¹	Configure	Perform follow-on (repeatable) software configuration.
Initial Configure	InitialConfig	Configure	Perform the initial, one-time setup of a software package during software deployment.
Migrate	Migrate	Configure	Perform initial, one-time configuration to updated software.

¹ Configure differs from Migrate and InitialConfig (InitialConfig is described in Table 2 on page 55). Migrate and InitialConfig both change the life cycle state of the software they configure, bringing their states (from Created or Updated) to Usable state. Configure, on the other hand, acts upon software *already in* Usable state, and the state of that software remains unaltered. Configure is also repeatable, not just a initial, one-time setup like InitialConfig.

Dependencies

A *dependency* is a requirement that an entity in a deployment descriptor has on an installable unit or on a managed resource to ensure that they interoperate correctly. The deployment descriptor entities that can have dependencies defined for them are installable units, configuration units, and topologies. A dependency might relate to a prerequisite or corequisite of one of these entities, or to an exrequisite whose presence in the hosting environment could cause interoperability problems.

In the deployment descriptor, a check can be performed on behalf of an installable unit, configuration unit, or topology to determine whether its dependency on things like computer disk space, processor types, processing capacity, resource properties, or other installable units or hosted resources is satisfied. *Dependency checker* is the Deployment Engine run-time component that determines, before installing any software in a hosting environment, whether or not dependencies are met.

Before deploying any installable unit or processing any configuration unit, the dependency checker component performs *dependency checking* by using data from the following sources to determine whether the dependencies are met:

- Dependency information defined in the deployment descriptor
- Persisted information in the installation database
- Properties of the target hosting environment

Dependency checker provides methods for determining whether dependencies for change management operations on a specific installable unit can be met.

Dependency checker performs the following functions:

- Parses the deployment descriptor
- Evaluates check items
- Evaluates dependencies among software packages

Dependency checker also makes sure that the dependencies of other installable units registered in the installation database are not violated. This function is sometimes referred to as *integrity checking*.

Output from dependency checker is used by the change manager component. If dependency checker runs before change manager is invoked, the results of the check is passed to change manager. If change manager runs first, then it invokes dependency checker during the change management operation.

Checks

Dependency checker provides the functionality for performing all the supported checks. A *check* is an XML element in an IU or CU deployment descriptor that defines some property criteria that the installation database or hosting environment must meet.

For example, you could provide a check to determine if the target hosting environment is suitable for deploying the installable units defined in the IU deployment descriptor. A check includes a condition or expression that, when tested against a targeted property in a hosting environment, evaluates to either true or false.

A check can be referenced in multiple places in a deployment descriptor. Any installable unit, configuration unit, or topology with its own dependencies can define checks that Deployment Engine can use for dependency checking. Checks are processed independently from one another by Deployment Engine—except consumption checks, whose requirements are processed cumulatively.

An example check follows. This particular XML fragment shows the deployment descriptor elements and attributes that represent a *capacity check*:

```
<capacity checkId="ProcessorSpeedCheck" type="minimum">
  <propertyName>processor/currentClockSpeed</propertyName>
  <value>400</value>
</capacity>
```

Types of dependencies and their corresponding checks

Recall that a dependency is a requirement that an entity in a deployment descriptor has on an installable unit or on a managed resource to ensure that they interoperate correctly. The following types of dependencies can be defined as a check in a deployment descriptor:

Capacity

A dependency that an entity in a deployment descriptor has on the processing capacity of a hosting environment.

A *capacity check* is used to determine if the local computer of the hosting environment meets the minimum or maximum *processing capacity* required by the dependent entity (compare with *consumption check*, which deals with *resource capacity*). The capacity check targets a specific property of the hosting environment (for example, the property that indicates the processor speed of the local computer) along with the threshold value that the targeted property must satisfy.

Consumption

A dependency that an entity in a deployment descriptor has on the amount of a resource that is available or consumable in a hosting environment.

A *consumption check* is used to determine if a resource on the local computer of the hosting environment has the minimum *resource capacity* required by the dependent entity (compare with *capacity check*, which deals with *processing* capacity). The consumption check targets a specific property of the hosting environment (for example, the property that indicates the amount of available disk space or memory on the local computer) along with the threshold value that the targeted property must satisfy.

The value of the property in the hosting environment must be equal to or greater than the amount of resource required by all the installable units to be deployed. Consumption checks are the only checks that are processed cumulatively. The consumption check sums the requirements for each installable unit in the deployment descriptor and then tests that sum against the value of the target property in the hosting environment to determine if the dependency is met.

Custom

A user-defined dependency that an entity in a deployment descriptor has on a hosting environment. The dependency is defined by some customized code.

A *custom check* is applied to a hosting environment in order to determine if that environment satisfies one or more requirements that are not currently defined by a specific check in the Deployment Engine schema.

Custom checks involve the processing of a Custom-check action descriptor (see page 48) that is specifically designed for custom dependency checking. Actions defined in the Custom-check action descriptor usually require the processing of some custom code provided to perform the unique checks. The Custom-check action descriptor is referenced from the installable unit deployment descriptor.

Hosted resource

A dependency that an entity in a deployment descriptor has on the availability of a hosted resource in a hosting environment. The hosted resource is typically a prerequisite or corequisite application that was deployed by means other than Deployment Engine.

A *hosted resource check* is used to determine if the hosted resource that is needed by the dependent entity is present in, and known to, the hosting environment (compare with *installable unit check*, which deals with software deployed by Deployment Engine). Unlike an installable unit check, which queries the installation database to determine if the software (the installable unit, in this case) is present, a hosted resource check queries a touchpoint in the hosting environment. The target application is identified by its name, minimum and maximum version, or some combination of these attributes.

Installable unit

A dependency that an entity in a deployment descriptor has on the availability of an installable unit that was deployed by Deployment Engine in a hosting environment. The installable unit typically belongs to a prerequisite or corequisite application that was deployed by Deployment Engine.

A *installable unit check* is used to determine if an installable unit that is needed by the dependent entity is present in the hosting environment. Unlike a hosted resource check (or a software check), which queries a touchpoint in the hosting environment to determine if the software (the hosted resource, in this case) is present, an installable unit check queries

the Deployment Engine installation database. The installable unit check targets the UUID or name of the installable unit, and can additionally target other characteristics such as minimum or maximum version and features that are deployed as part of the installable unit.

Property

A dependency that an entity in a deployment descriptor has on the value of a property defined by a particular managed resource. The managed resource can be a hosted resource such as a file system, or a hosting environment such as an operating system.

A *property check* is used to determine if the managed resource has a property whose name and value matches the name and value defined in the property check of the dependent entity. The value is expressed as a String value.

Relationship

A dependency that a managed resource in a deployment descriptor—in particular, a target managed resource that is identified in a topology—has on its relationship with another managed resource. This dependency is sometimes needed to ensure that the correct hosted resource is matched to the correct hosting environment (or vice versa) prior to any software update or configuration. The dependency requires that a Hosts relationship exists between the hosting environment and the hosted resource, to ensure that these two managed resources are paired correctly. (This is a Hosts relationship known only to a hosting environment touchpoint, and is *not* the same as a Hosts relationship maintained in the relationship registry of the Deployment Engine database.)

A *relationship check* is used during topology resolution to correctly match a hosted resource and a hosting environment. Either of these managed resources can be the target of the software update or configuration. Before deploying or configuring any software associated with one of these targets, a relationship check can determine if the correct target has been found.

This check is useful on a computer where multiple hosting environments are present (for example, multiple J2EE server hosting environments, such as WebSphere Application Server) and the target managed resource is:

- A hosting environment (a specific WebSphere Application Server) that currently hosts a particular hosted resource (a J2EE application), *or*
- A hosted resource (a J2EE application) that resides in a particular hosting environment (a particular WebSphere Application Server)

Software

A dependency that an entity in a deployment descriptor has on the availability of some specific software in an operating system hosting environment. The software is typically a prerequisite or corequisite application that was deployed by means other than Deployment Engine.

A *software check* is used to determine if the software that is needed by the dependent entity is present in, and known to, the operating system hosting environment (compare with *installable unit check*, which deals with software that was deployed by Deployment Engine). Unlike an installable unit check, which queries the installation database to determine if the software (the installable unit, in this case) is present, a software check queries a touchpoint in the operating system hosting environment. The target software is identified by its name, minimum and maximum version, or some combination of these attributes.

Note: The software check is a deprecated check. A hosted resource check is the recommended check to use.

Version

A dependency that an entity in a deployment descriptor has on the version of a particular managed resource. The managed resource can be a hosted resource such as an application, or a hosting environment such as an operating system.

A *version check* is used to determine if the version of a managed resource equals the minimum or maximum version number (or falls within that range of minimum or maximum numbers) required by the dependent entity. A version check is similar to a property check, but with a version check the value of the property is always version-related.

Checks verify very specific types of information, such as *is DB2 Version 9.1 present?* or *is 1 GB of RAM available?* There are various ways to organize checks. When multiple alternatives might satisfy a dependency, checks can be organized into requirements, as shown in Figure 29 (note that the **checkItem** elements shown in the figure must point to actual checks located elsewhere in the descriptor):

```
<iudd:rootIU ...  
:  
:  
  <requirement ...>  
    <alternative ...>  
      <checkItem .../>  
      <checkItem .../>  
    </alternative>  
    <alternative ...>  
      <checkItem .../>  
      <checkItem .../>  
    </alternative>  
  </requirement>  
:  
:  
</iudd:rootIU>
```

Figure 29. XML fragment from an IU deployment descriptor that illustrates a requirement consisting of two alternatives and their associated checks

For example, a product might support several databases, such as DB2 Universal Database[™] and Oracle Database. A check can look for only one of these databases, either DB2 or Oracle. But you can organize checks into a requirement (in this case, a database requirement) that can be satisfied by one of multiple alternatives.

A requirement includes a set of alternatives that in turn consist of some checks. The requirement is met if any one of its alternatives is met. In the database example, DB2 Universal Database and Oracle Database would be the two alternatives for the database requirement. If either database is found, the requirement is satisfied.

Since each alternative can include one or more checks, the alternatives in the database example could then specify checks for acceptable database versions. A check for DB2 Universal Database, for example, could look for version 9.1. If DB2 9.1 is found, the check is satisfied.

The requirement itself is met if any one of its alternatives is met. If one check in an alternative fails, the whole alternative fails. Only one alternative needs to succeed to satisfy the original requirement, but if *all* of the alternatives fail, the requirement itself fails.

Some checks are going to have different values on different types of resources. Operating systems are a typical example. An application might require 512 MB of RAM on one particular operating system and 1 GB of RAM on another. As before, you can use requirements with alternatives to address these multiple dependencies.

You can put your requirements within installable unit elements, configuration unit elements, or topology elements in a deployment descriptor. If you put them in a topology, the requirements are used to select the target hosting environment where your application will be deployed. If you put them in an installable unit or configuration unit element, the requirements are used to determine if the IU or CU can be deployed in the hosting environment that was resolved by the topology processing. In this case the requirement is processed only for the already-selected hosting environment. If the requirement is not met, the deployment fails.

So, if you want your requirements to be processed early, before the hosting environment is chosen, they should be part of the topology. If you want your requirements to be processed later—for example, when the requirement is only for an optional, user-selectable feature that might never be installed—then they should be part of an installable unit or configuration unit, in this case a unit used by the feature.

Relationships

A *relationship* is an association that one software entity in a hosting environment has with another; for example, a relationship that a host has with a hosted resource, that one installable unit has with another, that a feature has with an installable unit.

For Deployment Engine-enabled applications, Deployment Engine saves relationship information in its installation database and uses this information in order to correctly install or remove software updates, fixes, maintenance, service packs, and the like, without adversely affecting other deployed software. Relationships are established automatically during Create, Update, InitialConfig, Migrate, and Undo operations (see page 63), which register the relationships in the relationship registry of the installation database.

Relationship types

Deployment Engine supports the following types of relationships defined in the `relationships.xsd` schema: Deploys, Federates, Fixes, HasComponents, Hosts, Supersedes, and Uses. (The `relationships.xsd` schema also defines other relationship types that Deployment Engine does not currently support, such as HasMember, ImplementedBy, and Virtualizes.)

When initially deploying a software entity such as an installable unit, Deployment Engine registers the relationships of that entity in the relationship registry of the installation database, where Deployment Engine maintains the relationship information for future reference. Deployment Engine can register the following relationship types:

Deploys relationship

Indicates a direct relationship that an installable unit has with a managed

resource (in particular, a backing resource) in its hosting environment. For example, when an installable unit with a backing resource specification is deployed to specific hosting environment, that installable unit has a Deploys relationship with its backing resource.

Federates relationship

Indicates that an installable unit can be shared. *Federates* indicates a parent-child relationship among software entities where one parent (a feature) allows another parent (another feature) to share its child (an installable unit). Specifically, when a feature is deployed, the feature has a Federates relationship with its child installable units only (not all descendents). Because of this Federates relationship, multiple features can contain the same installable unit and each be a parent of that installable unit.

A feature can establish a Federates relationship with installable units that are already deployed. If the feature requires an installable unit that has not been deployed, the unit can be deployed together with the feature that requires it. During uninstallation of a feature, a federated installable unit can be removed with the feature that federates it, provided no other deployed feature is in a Federates relationship with it.

Fixes relationship

Indicates that an installable unit in a fix software package has changed or corrected another installable unit (that is not itself a fix), while leaving its version, release, modification, and level unaltered. When an installable unit in a fix software package is deployed, the installable unit has a Fixes relationship with the installable unit it changes or corrects.

HasComponents relationship

Indicates that a feature or installable unit cannot be shared. *HasComponents* indicates a parent-child relationship among software entities where the parent is the sole parent of its child and allows no other parent to be a parent of that child.

When deployed, the parent entity has a HasComponents relationship with its direct children only (not all descendents), as follows:

- An installable unit in an IU hierarchy has a HasComponents relationship with each of its child installable units, except when the child installable unit is a contained IU or a contained container IU.
- Although a contained IU or a contained container IU cannot have a HasComponents relationship, its parent installable unit *does* have a HasComponents relationship with the root IU referenced by the contained IU or a contained container IU.
- A root IU has a HasComponents relationship with each of its child features.
- A feature has a HasComponents relationship with each of its child features.

Hosts relationship

Indicates that a particular hosting environment is a container for, or presides over, a hosted resource or an installable unit deployed in that hosting environment. For example, when an installable unit is deployed to a specific hosting environment, that hosting environment has a Hosts relationship with the installable unit.

Supersedes relationship

Indicates that a new installable unit replaces all or part of another

installable unit. A Supersedes relationship occurs when you replace all or part of an installable unit by means of an incremental update. In this case, an installable unit in the incremental update software package has a Supersedes relationship with the installable unit that it replaces.

A Supersedes relationship also occurs when you deploy a new fix software package that includes all of a previously deployed fix software package. In this case, any installable unit in the new fix software package that replaces an installable unit in the previously deployed fix software package has a Supersedes relationship with that replaced installable unit.

Uses relationship

Indicates that one installable unit depends on another installable unit. For example, a deployed installable unit has a Uses relationship with each of its prerequisite and corequisite installable units. A Uses relationship can be established either with installable units that are already deployed or with installable units deployed as part of a requisite package.

Management of relationships

Table 4 lists change management operations and their handling of relationship types in the relationship registry of the installation database.

Table 4. Change management operation handling of relationship types

Change management operation	Handling of relationship types	Relationship types handled
Create	A Create operation places or installs software package or feature files in a hosting environment and automatically registers the required relationship types for the installed units in the relationship registry of the installation database.	Deploys Federates HasComponents Hosts Uses
Update	An Update operation upgrades a previously deployed installable unit. If, during the update, the Update operation deploys any new installable units, it also automatically registers relationship types for the units in the relationship registry of the installation database.	Deploys Federates Fixes HasComponents Hosts Supersedes Uses
Undo	The Undo operation removes applied maintenance files (fixes, incremental updates) from a hosting environment, reinstates all previous relationship types that were present before that maintenance was first applied, and deletes any relationship that was added when that maintenance was first applied.	Deploys Federates Fixes HasComponents Hosts Supersedes Uses
Delete	The Delete operation removes software package or feature files from a hosting environment and deletes the relationship types for the installable units it removes.	Deploys Federates Fixes HasComponents Hosts Supersedes Uses
InitialConfig	The InitialConfig operation performs the initial, one-time setup of a software package after a Create operation.	Uses
Configure	The Configure operation performs follow-on software configuration but does not automatically register relationship types.	
Migrate	The Migrate operation performs the initial, one-time configuration of software immediately following an Update operation.	Uses

Relationships and integrity checking

When the dependency checker component verifies requirements and the change manager component builds a step-by-step change plan, dependency checker evaluates the impact that the requested change might have on previously deployed software. Dependency checker examines relationship and constraint information in the installation database to ensure that deploying an installable unit will not violate the dependencies defined by its associated installable units. For each relationship or constraint, dependency checker evaluates the requested change to ensure that both software entities in the relationship can support the change and that the change does not violate any constraints.

Variables

A *variable* is an XML element in an IU deployment descriptor that represents a value which can be subsequently obtained or determined at deployment time, from user input, an expression, a property query, or other sources. The software package developer names and defines variables in the IU deployment descriptor, and later, during software deployment, Deployment Engine evaluates the variables and substitutes any symbolic references to them with their ascertained values.

Variable types

One or more variables, of various types, can each be defined by nesting them under a **variables** element within an installable unit section of the deployment descriptor. Each nested variable is indicated with a **variable** element. Each **variable** element identifies one named variable whose evaluation might be required in order to deploy the installable unit that includes it. Any **variable** element must include at least one subelement for one of the following supported variable types:

parameter variable

Represents a value that is obtained from user input, either by means of a graphical user interface or a response file, or is obtained from the parameter map of an installable unit that aggregates (encapsulates) the installable unit containing the parameter variable. A parameter variable can include a default value by specifying a `defaultValue` attribute. The parameter variable is defined by a **parameter** element:

```
<variables>
  <variable name="installToAllUsers">
    <parameter defaultValue="true"/>
  </variable>
</variables>
```

In this example, if no overriding value is provided by a user or parameter map, the variable `installToAllUsers` resolves to the default value of `true`.

derived variable

Represents a value that is derived from one or more variable expressions. Typically multiple expressions are used, and each one must be qualified by specifying a condition. If the conditions of more than one expression can be satisfied at a time, a priority attribute must be specified for each expression. The derived variable is defined by a **derivedVariable** element:

```
<variables>
  <variable name="install_root">
    <derivedVariable>
      <expression condition="$ (Windows_Check)">C:\Program Files</expression>
    </derivedVariable>
  </variable>
</variables>
```

```

        <expression condition="$(Linux_Check)"/>/usr/opt</expression>
    </derivedVariable>
</variable>
</variables>

```

In this example, the variable `install_root` is set to the value `C:\Program Files` or to the value `/usr/opt`, depending on the conditions defined in the two variable expressions. Here, each condition is either true or false, depending on the boolean result of a check. The Windows check is true only if the hosting environment is a Windows operating system; the Linux check is true only if the hosting environment is a Linux operating system. Because these two checks cannot simultaneously be true for the same hosting environment, there is no need to specify a priority.

query property variable

Represents a value that results from a query against a target or a required hosted resource. The query property variable must include a property name attribute that specifies the property to be queried. An optional target reference attribute can be specified if the query is to be performed elsewhere than on the target hosting environment of the installable unit containing the query property variable. In spite of its name, the optional target reference attribute, `targetRef`, can reference *either* a topology target or a required hosted resource. The query property variable is defined by a **queryProperty** element:

```

<variables>
  <variable name="PROGRAM_FILES_LOC">
    <queryProperty property="programFilesLocation" targetRef="t0S" />
  </variable>
</variables>

```

In this example, a query property variable is defined to obtain the location of operating system program files. The example assumes that the topology target `t0S` defines a hosting environment that exposes the needed location by means of the `programFilesLocation` property.

query IU discriminant variable

Represents a value that results from a query against an instance of an installable unit in the Deployment Engine installation database, yielding the instance's discriminant as the value. A *discriminant* is any unique identifier that is used to distinguish multiple instances of the same application or the same installable unit from one another. Deployment Engine uses an installable unit check (described on page 58) to determine which installable unit instance to query for the value. An `iuCheckRef` attribute is required that refers to the check ID of an installable unit check (the check is defined elsewhere in the IU deployment descriptor). The query IU discriminant variable is defined by a **queryIUDiscriminant** element:

```

<variables>
  <variable name="Linux_JRE_Home">
    <queryIUDiscriminant iuCheckRef="JRE_for_Linux_check" />
  </variable>
</variables>

```

In this example, the variable `Linux_JRE_Home` is set to the value of the installation location (the discriminant in this case) for the installable unit instance that satisfies the installable unit check `JRE_for_Linux_Check`.

For more information on variables, expressions, and conditions, see the related document *IBM Autonomic Computing: Installable Unit Deployment Descriptor Specification*.

Internal variables

In addition to the variable types described in the previous section, each of which must be explicitly defined in the IU deployment descriptor, the Deployment Engine run-time environment has several internal variables that you should be aware of:

_discriminant variable

A *discriminant* is any unique identifier that is used to distinguish multiple instances of the same software from one another. Even though a discriminant can be any string value, it is a good practice to assign a commonly recognized value for the discriminant, such as “installation location.” By doing so, if the software instance is being installed, the value of the discriminant will indicate the target installation location for the software instance. If the software instance is already installed, the value of the discriminant will indicate the actual hosting environment location of the installed software instance.

During Create operations, Deployment Engine retains a record of each discriminant value together with its associated software instance—for Deployment Engine this means its associated *installable unit instance*—in the installation database. Afterward, Deployment Engine refers to the saved discriminant values for any follow-on change requests.

The Deployment Engine run-time environment must obtain the installable unit's discriminant value prior to processing a Create change request for the installable unit. There are several ways Deployment Engine can obtain the discriminant value initially, at install time:

- From the software deployment program that is deploying the installable unit
- From a Deployment Engine command
- From a variable defined in a deployment descriptor

On Create change requests, Deployment Engine knows the installable unit instance it is deploying but needs the discriminant in order to register that instance with its unique identifier in the installation database. On other types of change requests, which affect already installed and registered instances, Deployment Engine needs the discriminant to determine which installable unit instance in the hosting environment is the correct target of the change request. Given the discriminant of the target installable unit instance, Deployment Engine can identify the correct instance by locating the discriminant in the installation database and obtaining its associated installable unit instance.

On Create change requests, Deployment Engine often depends on software deployment program or user input for the discriminant value, regardless of whether the deployment descriptor already defines its own variable for calculating the installation location. That is because Deployment Engine cannot automatically recognize a developer-named variable for the installation location as the equivalent of a discriminant. But once Deployment Engine receives a value that has been clearly identified as the discriminant—for example, either from a software deployment program or user—Deployment Engine saves the value in its database along with its

associated installable unit instance. During future deployment operations, the discriminant value can then be used to readily identify the correct instance to target the change request to.

Whenever a deployment descriptor defines its own variable for calculating the installation location, and you intend for the discriminant to resolve to that same location, it can be to your advantage to set the discriminant value within the deployment descriptor itself rather than make the software deployment program or user provide it. This can save the software deployment program developer some coding time, ensure that discriminant values are represented consistently when multiple deployment descriptors are involved, reduce user input errors, and the like.

But, as previously mentioned, Deployment Engine does not automatically recognize a developer-named variable as the equivalent of a discriminant. Deployment Engine does, however, provide its own internal variable name, `_discriminant`, that you can use to identify a variable intended to represent the discriminant of any root IU included (or referenced) in a deployment descriptor. On Create operations, Deployment Engine will recognize a derived variable (or another of the variable types described on page 64) named `_discriminant` as the discriminant for the root IU of the descriptor.

Deployment Engine only provides the *name* of the `_discriminant` variable. Using this name, you must still define your own variable in the deployment descriptor by using one of the available variable types described in “Variable types” on page 64. This makes the descriptor responsible to for providing Deployment Engine with the value of the discriminant, which Deployment Engine then uses during software deployment. If you decide to implement the `_discriminant` variable and are preparing a software package tree, be sure to use the name `_discriminant` consistently to represent the discriminant value for the root IU of each deployment descriptor in the tree.

To define the variable `_discriminant`, use a variable type described on page 64. As a typical example, you might define the variable `_discriminant` as a derived variable whose value is obtained from another variable expression:

```
<variables>
  <variable name="_discriminant">
    <derivedVariable>
      <expression>${install_loc}</expression>
    </derivedVariable>
  </variable>
  <variable name="install_loc">
    <parameter defaultValue="C:\my_installation_dir"/>
  </variable>
</variables>
```

In the preceding example, the first variable, `_discriminant`, resolves to the value of the second variable, `install_loc`, which in this case is the installation location.

There is another advantage to using this internal variable. When a software deployment program installs applications, it installs each root IU in the software package to the installation locations indicated in the deployment

descriptor. If you are installing a software package tree, you might want applications in your solution or suite to be installed to different target locations. An application server, for instance, might be targeted to a directory path different from the rest of the suite. If you do not use the `_discriminant` variable in your deployment descriptors, Deployment Engine installs each root IU to its targeted installation location but registers in the installation database the *same* discriminant value, as provided by the software deployment program, for every root IU in the software package tree. Of course this is not desirable or correct, since each installable unit instance should be registered with its correct installation location.

In cases like this, it is advantageous to use `_discriminant` variables in your software package deployment descriptors. That is because when Deployment Engine registers discriminants in the installation database during Create operations, the discriminant values from the deployment descriptors always *supersede* the single discriminant value provided by the software deployment program. Thus, if each descriptor in your software package tree includes a `_discriminant` variable for its root IU, following software deployment the installation database will reflect the correct, intended discriminant value for *each* installable unit processed during the Create operation, rather than the same discriminant value for every installable unit in the tree.

selectedFeatures variable

During software deployment, user input can include some selected application features. Feature selection enables users to tailor the application they are installing to their own needs or personal preferences.

The `selectedFeatures` variable is an internal Deployment Engine variable that represents a space-separated list of *all* selected features for an application that is to be installed. This full-feature list includes, in addition to the (partial) list of user-selected features passed to Deployment Engine from the software deployment program, any related, corequisite features that are auto-selected by Deployment Engine. If you specify `$(selectedFeature)` in the deployment descriptor, the full-feature list contained in the `selectedFeatures` variable can, for example, be passed to an existing, wrapped installation program for its use during software deployment.

Because `selectedFeature` is an internal variable, you cannot use this name for any other variables in your deployment descriptor.

Also, in addition to the variables described in “Variable types” on page 64, all of which must be explicitly defined in the IU deployment descriptor, every check, alternative, and requirement defined in the IU deployment descriptor is considered to be a boolean variable.

Chapter 2. The Deployment Engine run-time environment

When a software package developer initially installs tooling designed to assist with software package development, the tooling in turn installs the Deployment Engine run-time environment into the developer's working (development) environment as part of its own installation. During software package development, you will need to use the Deployment Engine run-time environment to validate the descriptors that you include in the software packages (see the command “validateIUDD” on page 101).

Later, after your Deployment Engine-enabled application is complete and users proceed to install it on their own computers, the Deployment Engine run-time environment must be automatically installed first in the target installation environment. Therefore its inclusion in your application package is required, unless the appropriate version of the Deployment Engine run-time environment is already known to be present in the target installation environment (for example, when deploying maintenance software, the correct version of the Deployment Engine run-time environment may already be present from the *initial* software deployment). Your application's software deployment program should run the Deployment Engine bootstrap program to check for Deployment Engine in the installation environment and install it if it is not already there.

This chapter describes the user modes and privileges associated with the Deployment Engine run-time environment as well as some of the key directories that Deployment Engine creates when its run-time environment is installed. This chapter also explains how to remove Deployment Engine from your development environment.

User mode selection for the run-time environment

At the time the Deployment Engine run-time environment is installed (that is, at application install time), Deployment Engine examines the user's authorities and determines which of two user modes to install Deployment Engine in. The particular user mode is determined prior to Deployment Engine installation and cannot be changed following that installation.

A privileged user—a user with special authorities like root or administrator—gets a mode of Deployment Engine (multiuser mode) that provides some extra functionality. Once a multiuser mode Deployment Engine is installed on the computer, it is available to all users of that computer and includes additional capabilities in the areas of scheduling and database access.

A general user—a user with lesser or no special authorities—gets a mode of Deployment Engine (single-user mode) for personal use only, without the privileged-user extras.

The selection of the user mode is transparent to the person who is installing Deployment Engine. But, because the user mode is authority-based, it does affect who can upgrade the installed run-time environment later, what scheduling and database capabilities are available, and, in some cases, what Deployment Engine commands the user can perform.

You can determine the user mode that your copy of Deployment Engine was installed in by issuing the **de_version** command, which is described in the book *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*. The **de_version** command displays the Deployment Engine user mode as well as the version.

Deployment Engine user modes

When Deployment Engine is transparently installed as part of an application deployment, the authorities of the person performing the installation determine a user mode that is permanently established for the Deployment Engine run-time environment. Selection of the user mode is also transparent to the user who is installing Deployment Engine as part of the new application. During its initial installation, Deployment Engine determines the user mode based on whether the deploying user is a root or nonroot user.

The sections that follow describe the Deployment Engine user modes, what authorities they require, and the subsequent functionality differences for the run-time environment that are available as a result of those authorities.

Multiuser mode

Multiuser mode is an installation condition where Deployment Engine is installed on a computer and is available to all users of that computer for the purpose of deploying other software. Multiuser mode is established whenever the installer of Deployment Engine has the required authorities for their operating system. Deployment Engine refers to a user with these required authorities as a *root user*. Example root users are:

On Windows operating systems

A root user is a user who has the authority to create a service and write to the Program Files directory. For example, any member of the Administrators group is a root user (provided the group's default permissions were not changed).

On UNIX-based operating systems

A root user is a user who has a user ID of zero (UID=0).

On OS/400® operating systems

A root user is a user who has all of the following authorities: *ALLOBJ, *SECADM, *JOBCTL, and *IOSYSCFG.

A system administrator is one typical example of a root user.

A multiuser deployment of the Deployment Engine run-time environment provides some additional functionality that is made possible by the operating system authority of the deploying root user. This functionality includes the use of an operating system service that is registered during the initial deployment to enable future scheduling of file system scans and database backups (see the commands for administering Deployment Engine in the related book, *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*). This functionality also includes the ability for multiple users to simultaneously access the installation database. Only one multiuser deployment of the Deployment Engine run-time environment can reside on the same computer.

Single-user mode

Single-user mode is an installation condition where a private copy of Deployment Engine is installed on a computer for the purpose of deploying other software. This copy of Deployment Engine in single-user mode is meant for the installer's personal use only (although root users can use it also). Single-user mode is established if the installer of Deployment Engine does not have the required root user authorities for their operating system. Deployment Engine refers to such a user as a *nonroot user*. Example nonroot users are:

On Windows operating systems

A nonroot user is a user who does not have all the root user authorities for their operating system.

On UNIX-based operating systems

A nonroot user is a user who does not have all the root user authorities for their operating system.

On OS/400 operating systems

A nonroot user is a user who does not have all the root user authorities for their operating system, but *does have* the following authorities: *RSTOBJ, *STRTCPSVR, *ENDTCPSVR, and *RSTLIB.

A general user or a product administrator are typical examples of a nonroot user.

A single-user deployment of the Deployment Engine run-time environment does not include the additional scheduling and database functionality provided with a multiuser deployment, because the installing nonroot user in this case lacked the operating system authority necessary to provide that functionality. One or more single-user deployments of the Deployment Engine run-time environment can reside on the same computer. However, each user of that computer can deploy only one.

Note: You can have one multiuser and one or more single-user deployments on the same computer, but the software deployment program will not install a Deployment Engine run-time environment in single-user mode if a compatible multiuser deployment already exists.

Database access

The following sections describe the different database access restrictions for Deployment Engine in multiuser mode and in single-user mode.

Access restrictions in multiuser mode

The Deployment Engine installation database can be concurrently accessed. The following database restrictions apply to concurrent access:

- Only one change request is allowed at a time. This restriction exists even if it is different JVMs that are attempting concurrent change requests. If concurrent change requests are attempted, an exception is thrown.
- Any number of dependency checker operations are allowed at any given time. Dependency checker results might be invalidated if the change manager component starts to concurrently process a change request.
- Touchpoints will throw an exception if concurrent action descriptor processing is attempted.

Access restrictions in single-user mode

Because of limitations with the Deployment Engine installation database, there is no concurrent access to the database in single-user mode. Only one user at a time can access Deployment Engine and its related data.

Installed directories

Depending on the Deployment Engine mode detected, Deployment Engine files are written to different directories, as described in the following sections.

Directories for users of Deployment Engine in multiuser mode

For users of Deployment Engine in multiuser mode, which is described in “Deployment Engine user modes” on page 70, the following operating system-specific directories are created by default, unless a different directory is specified at install time:

Installation directory

The default installation directory is one of the following operating system-specific locations:

Windows operating systems

C:\Program Files\ibm\common\acsi

UNIX-based operating systems

/usr/ibm/common/acsi

OS/400 operating systems

/QOpenSys/QIBM/ProdData/acsi

Note: An installation directory different than the default directory might have been specified at the time the Deployment Engine run-time environment was installed. You can determine the location of the directory where Deployment Engine was installed by querying your operating system for the value of the `SI_PATH` environment variable.

Common directory

The common directory is one of the following operating system-specific locations:

Windows operating systems

C:\Program Files\ibm\common\acsi

UNIX-based operating systems

/var/ibm/common/acsi

OS/400 operating systems

/QOpenSys/QIBM/ProdData/acsi

Directories for users of Deployment Engine in single-user mode

For users of Deployment Engine in single-user mode, which is described in “Deployment Engine user modes” on page 70, the following operating system-specific directories are created by default, unless a different directory is specified at install time:

Installation directory

The default installation directory is one of the following operating system-specific locations:

Windows operating systems

C:\Documents and Settings*username*\acsi_*username*

UNIX-based operating systems

/home/*username*/.acsi_*username*

OS/400 operating systems

/home/*username*/.acsi_*username*

Where *username* is the short name of the current operating system user.

Note: An installation directory different than the default directory might have been specified at the time the Deployment Engine run-time environment was installed. You can determine the location of the directory where Deployment Engine was installed by querying your operating system for the value of the SI_PATH environment variable.

Common directory

The common directory is one of the following operating system-specific locations:

Windows operating systems

C:\Documents and Settings*username*\acsi_*username*

UNIX-based operating systems

/home/*username*/.acsi_*username*

OS/400 operating systems

/home/*username*/.acsi_*username*

Where *username* is the short name of the current operating system user.

Environment variables for the installed directories

Deployment Engine creates the following environment variables for important directories. In this book, these variable names are sometimes specified with a prepended dollar sign (\$) to represent the directories when they appear in the paths of fully qualified file or directory names. For example, \$SI_PATH/schema is used to indicate the schema directory, regardless of operating system.

ACU_COMMON

The ACU_COMMON variable is set to the common directory for Deployment Engine files. This directory is not user-configurable. This directory contains the ACUApplication.properties file and the ACULogger.properties file. To determine the location of the common directory for Deployment Engine, just query your operating system for the value of the ACU_COMMON environment variable.

SI_PATH

The SI_PATH variable is set to the directory where Deployment Engine is installed. The installation directory depends on the command options used when the Deployment Engine run-time environment was installed. To determine the location of the installation directory where Deployment Engine was installed, just query your operating system for the value of the SI_PATH environment variable. For additional details about the default installation directories for Deployment Engine on different operating systems, see “Installed directories” on page 72.

Removing Deployment Engine

To remove Deployment Engine from your development environment, perform the following steps:

1. Remove the Deployment Engine code and installation database by invoking the **si_inst** command with the remove (**-r**) option. A force (**-f**) option is also available. These options are described below. The command syntax is as follows:

si_inst -r [-f]

The available options are:

-r Removes the components of Deployment Engine, including its installation database.

In Deployment Engine 1.3, if you have other applications that were deployed using Deployment Engine, and therefore registered in its installation database, your **si_inst -r** command processing will end with the following warning message:

ACUINI0066W Active IU instances exist in the Deployment Engine IU Registry. The uninstall request was not processed. You may use the -f option to force removal of Deployment Engine.

If you are using a version of Deployment Engine earlier than version 1.3 (for example, Solution Install 1.2.1), you will *not* receive the above warning message, and Deployment Engine and its installation database will be deleted, even if other applications were installed by and registered with Deployment Engine. That is because with earlier versions, when you specify the **-r** option, the command is processed with an implied force, as if you specified the command with the **-r** and **-f** options together.

Note that after Deployment Engine is deleted, there is no database information about, or means to track, any applications previously deployed by Deployment Engine. Further, your uninstallation program may not be able to remove these applications, because the program can no longer invoke Deployment Engine. Therefore use the **-r** option with care.

-f

(Option new for Deployment Engine 1.3.) Used together with the **-r** option, the **-f** option forces removal of the Deployment Engine components, even if Deployment Engine-enabled applications are currently registered in the database. Such registered applications indicate that Deployment Engine will be required to uninstall these applications in the future. If you think you will need Deployment Engine to remove these applications later, do not force its removal.

When removing Deployment Engine, any backups of the installation database that are currently located in the default backup directory `$SI_PATH/backupDBs` will *not* be deleted, and you will receive the following warning messages:

ACUINI0077W The backupDBs directory was not removed. Please remove it manually.

ACUINI0027W Deployment Engine uninstallation could not remove all files and directories. Please remove them prior to installing Deployment Engine again. The batch file cannot be found.

You should remove the database backups directory, `$SI_PATH/backupDBs`, manually before any reinstallation of Deployment Engine.

Note: You must use the `si_inst` command to remove the Deployment Engine components. Deployment Engine does not provide an interactive user interface to remove its components.

2. Check the associated logs for any messages related to the removal of Deployment Engine.

If problems occur during the removal of Deployment Engine, several logs are written to the `$SI_PATH/logs/username` directory. `SI_PATH` is an environment variable that represents the top-level directory where Deployment Engine was installed. To determine the location of this installation directory, just query your operating system for the value of the `SI_PATH` environment variable. *Username* is the user ID of the user on the current operating system.

Note that if Deployment Engine removal is successful, all logs in the `$SI_PATH/logs/username` directory are deleted.

When removing Deployment Engine, check for the following logs:

DE_Install.log

This log file contains information related to requests from the Deployment Engine bootstrap program, including uninstallation requests, installation requests, change-password requests, and the like. The name of this log file depends on one of the following installation modes:

- For a multiuser mode Deployment Engine instance, the `DE_Install.log` log file is created.
- For single-user mode Deployment Engine instance, the `username_DE_Install.log` log file is created, where *username* is the user ID of the user on the current operating system.

acu_de.log

This log file contains information related to requests from the Deployment Engine bootstrap program *and* other Deployment Engine functions.

Part 2. Commands

Chapter 3. Command summary

Table 5 summarizes the commands that you can use to test and validate your Deployment Engine-enabled applications.

Table 5. Deployment Engine commands

Command	Purpose	Page
manageIU	A software deployment program “simulator” that can be used to test a software package and its associated life cycle operations to determine if the application deploys as intended.	86
validateIUDD	Validates a deployment, action, or media descriptor in an application's software package.	101

Code page changes might be required for some languages: For Russian, Hungarian, Polish, and Czech languages, use the following code pages to correctly display message output on a Microsoft Windows NT[®] or Microsoft Windows 2000[®] system:

- Code page 1251 for Russian
- Code page 1252 for Hungarian, Polish, and Czech

From a DOS command window you can change the code page by entering one of the following commands, as appropriate for your language:

chcp 1251

or

chcp 1252

Chapter 4. Working with commands

The following sections contain information to be aware of when working with the commands provided by Deployment Engine.

Command authorization

Use of the commands in this chapter depends on the privileges granted to the current user. A root user requires some additional special privileges that a nonroot user does not have. As noted below, even a nonroot user might need some special privileges to issue commands on some operating systems.

Deployment Engine defines a *root user* by operating system, as follows:

On Windows operating systems

A root user is a user who has the authority to create a service and write to the Program Files directory. For example, any member of the Administrators group is a root user (provided the group's default permissions were not changed).

On UNIX-based operating systems

A root user is a user who has a user ID of zero (UID=0).

On OS/400 operating systems

A root user is a user who has all of the following authorities: *ALLOBJ, *SECADM, *JOBCTL, and *IOSYSCFG.

A software package developer or system administrator is typically a root user.

Deployment Engine defines a *nonroot user* by operating system, as follows:

On Windows operating systems

A nonroot user is a user who does not have all the root user authorities for their operating system.

On UNIX-based operating systems

A nonroot user is a user who does not have all the root user authorities for their operating system.

On OS/400 operating systems

A nonroot user is a user who does not have all the root user authorities for their operating system, but *does have* the following authorities: *RSTOBJ, *STRTCPSVR, *ENDTCPSVR, and *RSTLIB.

A general user or product administrator is typically a nonroot user.

The "Authorization" section of each command describes the specific privileges that the command requires.

If you do not know whether the local copy of Deployment Engine was installed in multiuser mode or single-user mode, you can find out by issuing the **de_version** command, described in the book *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*. The **de_version** command displays the Deployment Engine user mode as well as the version.

Locating and running the commands

The Deployment Engine developer command files are installed on the computer in the following location:

`$SI_PATH/bin`

where `$SI_PATH` is the directory where Deployment Engine was installed. To determine the location of this installation directory, you can query your operating system for the value of the `SI_PATH` environment variable. For example, on Windows operating systems, the default value for `SI_PATH` is `C:\Program Files\IBM\Common\ASCI`.

On Windows operating systems, specifying *command_name* (along with any desired command options) from the `$SI_PATH/bin` directory will run the file *command_name.bat* or *command_name.cmd* (for example, specifying `validateIUDD` or `validateIUDD -silent` will run the command file `validateIUDD.cmd`). On UNIX-based and OS/400 operating systems, you must specify *command_name.sh* to run the command file *command_name.sh*.

To run a command from another directory, specify the fully qualified command name (along with any desired command options):

`$SI_PATH/bin/command_name [option_1 option_2 ... option_n]`

Command syntax conventions

The command documentation uses the following special characters to define the command syntax:

- [] Identifies optional options. Options not enclosed in brackets are required.
- ... Indicates that you can specify multiple values for the previous option.
- | Indicates mutually exclusive information. You can use the option to the left of the separator or the option to the right of the separator. You cannot use both options in a single use of the command.
- { } Delimits a set of mutually exclusive options when one of the options is required. If the options are optional, they are enclosed in brackets ([]).
- \ Indicates that the command line wraps to the next line. It is a continuation character.

Command options are listed alphabetically in the “Options” section for each command.

For other conventions used in this book, see page vi.

Case sensitivity in commands

Command names are case sensitive when the operating system is case sensitive (for example, on UNIX and Linux® operating systems). Command names are not case sensitive when the operating system is not case sensitive (for example, on Windows operating systems).

Options are case sensitive.

Most values are not case sensitive. However, path values are case sensitive on an operating system that is case sensitive. For example, /user/path and /user/Path can refer to different coexisting paths on UNIX or Linux operating systems, so these strings should be treated as case-sensitive values on these operating systems. On Windows operating systems, paths are not case sensitive; so these strings can be treated as such on Windows.

Values (such as *discriminant*) that are stored in the Deployment Engine installation database *are* case sensitive.

The following table shows command syntax elements and their case sensitivity:

Table 6. Case sensitivity of developer command syntax elements, by operating system.

Command syntax element	Case sensitive on UNIX and Linux operating systems?	Case sensitive on Windows operating systems?
Command names	Yes	No
Options	Yes	Yes
Values (in general)	No	No
Values (paths and file names)	Yes	No
Values that are compared to strings stored in the Deployment Engine installation database	Yes	Yes

Specifying a software instance uniquely

Usually, fully identifying a software instance that is the result of a deployed software package requires specifying two options, such as a discriminant value and a root IU type identifier value (consisting of a UUID and version). For example, in Table 7, notice that the software instances require a discriminant–root IU type identifier pair to specify the software instance uniquely:

Table 7. Software instances that require both a discriminant and a root IU type identifier for unique identification.

Deployed software instance	Discriminant	Root IU type identifier
Software_instance_1	C:Program Files\abc	11111, v1
Software_instance_2	C:Program Files\abc	22222, v1
Software_instance_3	C:Program Files\xyz	22222, v1

With only a discriminant specification, Deployment Engine cannot distinguish deployed software_instance_1 from software_instance_2, because their discriminant identifiers are the same. However, by also specifying a root IU type identifier in your command, Deployment Engine can locate the correct software instance by its unique combination of discriminant and root IU type identifier (the discriminant in this example is an installation directory path, but it can be any string). Similarly, with only a root IU type identifier specification, Deployment Engine cannot distinguish software_instance_2 from software_instance_3, because their root IU type identifiers are the same. But by additionally specifying a discriminant in your command, Deployment Engine can locate the correct software instance by its unique combination of discriminant and root IU type identifier.

Retrieving return codes

The documentation for each command provides any available return code information in a "Return values" section. To determine whether the command ran successfully, retrieve the return code after running a command. To retrieve the return code, echo the error level environment variable.

For Windows operating systems, run following command:

```
echo %ERRORLEVEL%
```

For Linux and UNIX operating systems, run following command:

```
echo $?
```

For OS/400, run the following command:

```
echo $?
```

Representing strings that include spaces

Strings that are supposed to be treated as a single value, but contain spaces, should be enclosed in quotation marks (for example, "C:\Program Files\..."). In this case, single (' ') and double (" ") quotation mark pairs are equivalent.

Chapter 5. Developer commands

These commands are provided to test an application to be installed, configured, or updated using Deployment Engine, and to validate the application's software package descriptors. You must be a nonroot or root user to use the developer commands in this chapter. (Additional restrictions may be imposed by the computer operating system, its applications, or its system administrator.)

manageIU

A software deployment program “simulator” that can be used to test a software package and its associated life cycle operations to determine if the application deploys as intended.

Note: The **manageIU** command is intended for human, not programmatic, use. The command is intended for software package developers who do not have their own software deployment program to test with, *not* for inclusion in any software deployment program. Because **manageIU** command output can change from release to release and is not backward compatible, the output is not suitable long-term for parsing by other computer programs.

Syntax

```
manageIU -o operation -r discriminant [-i file_name] -p directory | file_name [-u true | false] [-v true | false] [-force file_name] [-rerun file_name | all | features]
```

```
manageIU -o operation -r discriminant [-i file_name] -d rootIUTypeID [-u true | false] [-v true | false] [-force file_name] [-rerun file_name | all | features]
```

```
manageIU -o operation -r discriminant [-i file_name] -Cfg rootIUTypeID [-u true | false] [-v true | false] [-force file_name] [-rerun file_name | all | features]
```

Description

The **manageIU** command behaves like a software deployment program. The command can be used to test the life cycle operations that Deployment Engine performs on a software package by calling many of the Deployment Engine APIs. The command is for software package developers who do not have their own software deployment program to test with. By selecting certain command options and by customizing one or more input files that work in conjunction with the **manageIU** command, you can use the **manageIU** command to determine if your application's software packages deploy as intended.

Command options are case sensitive; option order does not matter.

When you issue the **manageIU** command, specify **manageIU.sh** on UNIX-based and OS/400 operating systems; the operating system will process the **manageIU.sh** command file. On Windows operating systems, specify **manageIU**; in this case, the operating system will process the **manageIU.bat** command file.

Life cycle operations are the equivalent of change requests (such as Configure, Create, Delete, or Undo), which are described in “Change requests” on page 52. The **manageIU** command can perform life cycle operations (operations that are usually initiated by Java API calls in an application's software deployment program) on either new software or on previously deployed software.

Along with the command input that you specify directly in your **manageIU** command options, you can provide additional command input within a customized response file. The name of the response file is specified in the **-i** option of your command. You use the response file to test specific installation group and feature selections, as well as new or alternative values for the parameter variables defined in your application's IU deployment descriptor. Because the **manageIU** command uses a response file to initiate a *silent* software deployment that does not prompt for user input, you must know exactly what content is necessary to include

in your response file in order to accurately test your software package deployment. The **-force** and **-rerun** options can be used to indicate additional input files that the **manageIU** command should process.

All command options and response file options do not necessarily apply to every life cycle operation. The options that you specify in your command, and the options that you include in your **-i** option's response file, must be appropriate for the life cycle operations used to deploy the software packages you are testing. Table 8 shows the valid combinations of command options and life cycle operations. An entry of **Required** indicates that the option is required for the life cycle operation. An entry of **Optional** indicates that the option applies, but is not required, for the life cycle operation. The absence of an entry indicates that the option does not apply to the life cycle operation.

Table 8. Valid combinations of command options and life cycle operations.

Command options	Create operation	Delete operation	Update operation	Configure operation	Undo operation	Create Feature operation	Delete Feature operation	Initial Configure and Migrate operations	Reapply Update operation
-allowResource-RequiredBase true false			Optional						
-Cfg rootIUTypeID								Optional	
-d rootIUTypeID		Optional (and recommended)			Optional (and recommended)		Optional (and recommended)		
-force file_name	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional
-i file_name	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional
-p directory file_name	Required	Optional (but not recommended)	Required	Required	Optional (but not recommended)	Required	Optional (but not recommended)	Optional	Required
-r discriminant	Required	Required	Required	Required (but ignored)	Required	Required	Required	Required	Required
-rerun file_name all features	Optional	Optional	Optional		Optional	Optional	Optional	Optional	Optional
-u true false			Optional						
-v true false	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional

Table 9 shows the valid combinations of response file options and life cycle operations. As with Table 8, an entry of **Required** indicates that the option is required for the life cycle operation. An entry of **Optional** indicates that the option applies, but is not required, for the life cycle operation. The absence of an entry indicates that the option does not apply to the life cycle operation.

Table 9. Valid combinations of response file options and life cycle operations.

Response file options	Create operation	Delete operation	Update operation	Configure operation	Undo operation	Create Feature operation	Delete Feature operation	Initial Configure and Migrate operations	Reapply Update operation
Variable	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional
selectedGroup	Optional								

Table 9. Valid combinations of response file options and life cycle operations. (continued)

Response file options	Create operation	Delete operation	Update operation	Configure operation	Undo operation	Create Feature operation	Delete Feature operation	Initial Configure and Migrate operations	Reapply Update operation
selectedFeature	Optional					Required			
deleteFeature	Required								

When processing the response file, the **manageIU** command first makes sure the set of selected features is valid. Then **manageIU** processes requirements based on the selected features. If the requirements are satisfied, **manageIU** proceeds on to action processing. In the Deployment Engine run-time environment, actions in the action descriptor are processed by the change manager component. During a Create life cycle operations, change manager automatically registers each newly deployed installable unit in the Deployment Engine installation database. This database information can then be referenced and updated by change manager whenever it performs any subsequent, non-Create life cycle operation, such as Delete, on a deployed installable unit.

You should note that, although the **manageIU** command is a useful tool for testing software package deployment, it is not capable of calling every available Deployment Engine API. For example, the **manageIU** command cannot complete the deployment of a software package that requires a reboot or re-login before deployment can continue. So, in this case, you need to create some software deployment code of your own that causes processing to resume after the reboot or re-login and then finishes deploying your software package.

The **manageIU** command duplicates some functions that can be accomplished using the administration commands (documented in the book *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*), but the **manageIU** command and administration commands each include functions that the other does not. So it is best to test using both the **manageIU** command and administration commands, if you are testing administration tasks like installing maintenance or uninstalling applications, features, or maintenance. For example, it is recommended that you also use the **de_instmaint** and **de_uninstmaint** commands to test the installation and uninstallation of maintenance, rather than use the **manageIU** command alone.

Options

-allowResourceRequiredBase=true | false

Indicates whether an existing resource instance can be used as the required base software during an Update life cycle operation. Use this option only if the application to be updated was not originally installed by Deployment Engine (for example, if the software package that you are currently testing is intended to update a previously installed IBM DB2® application that Deployment Engine did not install). This situation is sometimes referred to as a *jump point installation*, because from this point forward the application, which was not formerly maintained by Deployment Engine, will be maintained by Deployment Engine and tracked in its installation database.

Specify true to update an existing resource instance, otherwise specify false. The default value is false. Note that the

-allowResourceRequiredBase option is only valid for the Update life cycle operation, and that you would only use it if testing an update to an

existing resource instance that was not originally installed by Deployment Engine. You would *not* use this option, for example, if you are testing an update to an installable unit instance that Deployment Engine already tracks and maintains.

-Cfg *rootIUTypeID*

Specifies the root IU type identifier of the IU deployment descriptor associated with the software instance to be initially configured or migrated. Although the value for the root IU type identifier is unique for each IU deployment descriptor, it is not unique for each deployed software instance. The root IU type identifier, combined with a discriminant specified with the **-r** option, uniquely identifies the software instance (see “Specifying a software instance uniquely” on page 83 for a detailed explanation). The **-Cfg** option is only valid on Initial Configure and Migrate life cycle operations. If you do not specify the **-Cfg** option, you must specify the **-p** option on Initial Configure and Migrate life cycle operations. The option name **-Cfg** is case sensitive.

-d *rootIUTypeID*

Specifies the root IU type identifier of the IU deployment descriptor associated with the software instance to be deleted or undone. Although the value for the root IU type identifier is unique for each IU deployment descriptor, it is not unique for each deployed software instance. The root IU type identifier, combined with a discriminant specified with the **-r** option, uniquely identifies the software instance (see “Specifying a software instance uniquely” on page 83 for a detailed explanation). The **-d** option is only valid on Undo and Delete (includes Delete Feature) life cycle operations.

-force *file_name*

Specifies the name of a file that contains one or more force options which provide additional input to the **manageIU** command. Use this option when you want the **manageIU** command to force software package processing to go forward despite the fact that unsatisfied requirements, existing Uses relationships (see page 61), backward compatibility constraints, or installed software that is identical to the software you are trying to deploy is generating a failure during your software package deployment. The file type for this file must be `.properties`; for example, `force_options.properties`.

In some circumstances where requirements, relationships, conditions, or other criteria for a safe and satisfactory deployment are violated or not met, you have the option to ignore the failures and force change request processing to go forward anyway. (Change requests, which are described in “Change requests” on page 52, are the equivalent of life cycle operations.) These criteria are put in place to ensure a successful deployment, so, generally speaking, using force processing is not recommended. That said, Deployment Engine does give you some flexibility by providing a way for the **manageIU** command to indicate to change manager that it can ignore whatever failed criteria you include in the force options file.

Figure 30 on page 90 shows the format of some entries that might be used in a force options file for the **manageIU** command. These entries are described in greater detail in the text that follows the figure.

```

# force_options.properties file template for use with manageIU command
# -----
#
# This file is used to force software package processing to go forward despite
# the fact that failed requirements, existing Uses relationships, backward
# compatibility constraints, or installed identical software is blocking your
# software package processing.
#
#
# The following example indicates whether or not you want to ignore one or
# more failed requirements. To ignore a requirement, specify the name of the
# requirement as the value for the options keyword. Delimit multiple
# requirements with commas:
#
options=requirement_name_1,requirement_name_2, ... requirement_name_n
#
#
# The following example indicates whether or not you want to ignore existing
# Uses relationships. For the value of the breakRelationships keyword, specify
# "true" to ignore Uses relationships; otherwise specify "false":
#
breakRelationships=true|false
#
#
# The following example indicates whether or not you want to overwrite
# any identical software that is already deployed. For the value of the
# overwriteBackingResource keyword, specify "true" to overwrite identical
# software; otherwise specify "false":
#
overwriteBackingResource=true|false
#
#
# The following example indicates whether or not you want to ignore backward
# compatibility constraints. For the value of the forceUpdateReferencedIU keyword,
# specify "true" to ignore backward compatibility constraints when directly
# updating a root IU referenced by a contained IU (or contained container IU);
# otherwise specify "false":
#
forceUpdateReferencedIU=true|false

```

Figure 30. A force options file template for the **manageIU** command, showing the format of force option entries.

Ignoring failed requirements. Requirement failures indicate that one or more system requirements of the application being deployed were not satisfied. Each requirement is defined within the **requirements** element of the application's IU deployment descriptor. Every requirement includes a set of alternatives that consist of some checks. The requirement is met if any one of these alternatives is met. If one check in an alternative fails, the whole alternative fails. Only one alternative needs to succeed to satisfy the original requirement, but if *all* of the alternatives fail, the requirement itself fails. To force change request processing despite a failed requirement is risky, because the force processing ignores *all* the failed alternatives. Using the **-force** option allows change request processing to go forward by ignoring the specified requirements.

The following example shows the key-value format for the force option that indicates one or more failed requirements that you want to ignore. Specify the name of the requirement as the value for the **options** keyword. Delimit multiple requirements with commas:

`options=requirement_name_1,requirement_name_2, ... requirement_name_n`

Ignoring existing Uses relationships (ignoring dependent applications).

One of the things integrity checking does is look at existing relationships to see if a dependency might prohibit a change request from going forward. If integrity checking fails for a change request that is *removing* software, and the prohibiting relationship is a *Uses* relationship (where an installable unit in the hosting environment depends on the installable unit being removed by the change request), you can force change request processing to proceed.

This force option does not typically apply to Create change requests, because they do not remove installable units. But it can apply to Delete, Delete Feature, and Undo change requests. Consider a Delete change request that fails because it would remove from the hosting environment an installable unit that is currently being used by another one. These two installable units are linked at present by a Uses relationship in the relationship registry. (For more on Uses relationships, see page 61.)

Though risky, you can use the **-force** option to ignore this relationship. If you do use the **-force** option, the Uses relationship that is ignored will be removed from the installation database and cannot be reestablished. In fact, all Uses relationships between the deleted installable unit and its dependent installable units are removed.

The force allows change request processing to go forward, ignoring all prohibiting Uses relationships. Note that, by forcing the change request in the above Delete example, not only will the target installable unit be deleted and all Uses relationships removed, but following these changes the installable unit that depended on the now-deleted installable unit may no longer function properly. So use this force option with caution.

The following example shows the key-value format for the force option that determines whether or not to ignore existing Uses relationships. For the value of the **breakRelationships** keyword, specify true to ignore Uses relationships; otherwise specify false:

breakRelationships=true | false

Overwriting identical software. Before creating or updating software, Deployment Engine checks its installation database, and if possible, the hosting environment, to make sure that the software or software update is not already deployed. A failure results if Deployment Engine finds software in either place that is identical to the software it plans to deploy.

Deployment Engine can check the hosting environment only if the software to be deployed provides a backing resource specification (BRS) in its IU deployment descriptor. Think of the BRS as a software signature that Deployment Engine can send in a query to a touchpoint. Using the BRS, the touchpoint can find out if the backing resource is already present in the hosting environment and then inform Deployment Engine. If the touchpoint responds that the BRS was found, integrity checking fails due to evidence of identical software in the hosting environment.

This failure can occur only with a Create or Update change request. You can choose to ignore the identical software and allow the change request to

overwrite it with the backing resource by specifying the **-force** option. The option might be used if you know there is a problem with the software currently deployed on the computer, or if the software was deployed outside Deployment Engine and you want to overwrite it with a backing resource that is registered with Deployment Engine. You might also force if you know that even though the touchpoint located the BRS for the software, the software itself does not actually exist in the hosting environment. The force allows change request processing to go forward and the backing resource to overwrite the identical software already present in the hosting environment.

The following example shows the key-value format for the force option that determines whether or not to overwrite any identical software that is already deployed. For the value of the **overwriteBackingResource** keyword, specify **true** to overwrite identical software; otherwise specify **false**:

overwriteBackingResource=true | false

Ignoring backward compatibility constraints. Sometimes, as part of its own deployment, one application deploys a second application. The second application is deployed at a particular version level that meets the needs of the first application. Typically, any future update to the second application is handled as part of an Update change request to the first application. This keeps the versions of both applications compatible with one another.

However, it is possible that a change request attempt could be made directly to the second application, without involving the original application. A direct update that bypasses the original application is called an *independent update*. In this case, the version of the application in the update must be backward compatible with the version that was originally deployed by the first application. If not, integrity checking for the Update change request fails.

In installable unit terms, the second application is a root IU that is referenced by a contained IU (or contained container IU) of the first application. And the independent update is an Update change request targeted directly to that referenced root IU. The new root IU provided in the independent update must declare its backward compatibility with the version specified by the contained IU of the original application to prevent the integrity checking failure.

A backward-compatibility failure indicates that updating the contained IU's referenced root IU could render it unusable by the original application that deployed it. Though risky, you could ignore the potential backward compatibility failure by specifying the **-force** option. This would allow the independent update to occur even though the root IU in the update is not backward compatible with the contained IU that references it. The force allows change request processing to go forward, ignoring the backward compatibility failure. After the force, however, the original application that deployed the overwritten application may no longer be able to use it.

The following example shows the key-value format for the force option that determines whether or not to ignore backward compatibility constraints. For the value of the **forceUpdateReferencedIU** keyword,

specify true to ignore backward compatibility constraints when directly updating a root IU referenced by a contained IU (or contained container IU); otherwise specify false:

forceUpdateReferencedIU=true | false

Figure 31 shows a sample force options file:

```
#####
#Force options
#####
#
#
options=PlatformRequirements
breakRelationships=true
```

Figure 31. Sample force options file for the **manageIU** command.

-i file_name

Specifies the name of a response file that contains deployment descriptor entities that can be tested using the **manageIU** command. This response file is used to test parameter variable values, installation groups, selected features used during a Create (includes Create Feature) life cycle operation, and features to be deleted during a Delete (includes Delete Feature) life cycle operation. The file type for the response file must be .properties; for example, my_application.properties.

Figure 32 shows the format of some entries that might be used in a response file for the **manageIU** command. These entries are described in greater detail in the text that follows the figure.

```
# response_file.properties file template for use with manageIU command
# -----
#
# This file is used to test the value of one or more parameter variables,
# to test an installation group, to test one or more selected features, or
# to test one or more features that are to be deleted.
#
#
# The following example shows the format for a parameter variable,
# installLocation, and its value:
#
Variable#RootIUTypeID[32_character_UUID,version]#installLocation=value
#
#
```

Figure 32. A response file template for the **manageIU** command, showing the format of parameter variable, installation group, and feature entries. (Part 1 of 2)

```
# The following example shows the format for two installation groups,
# Developer and End user (only one installation group can be active--
# or tested--at one time; spaces must be escaped with a backslash):
#
selectedGroup1#RootIUTypeID[32_character_UUID,version]#Developer
# selectedGroup2#RootIUTypeID[32_character_UUID,version]#End\ user
#
#
# The following example shows the format for two selected features,
# Samples and Documentation, to be deployed:
#
selectedFeature1#RootIUTypeID[32_character_UUID,version]#Samples
selectedFeature2#RootIUTypeID[32_character_UUID,version]#Documentation
#
#
# The following example shows the format for two features, Developer
# Toolkit and Dictionary, to be deleted (spaces must be escaped with
# a backslash):
#
deleteFeature1#RootIUTypeID[32_character_UUID,version]#Developer\ Toolkit
deleteFeature2#RootIUTypeID[32_character_UUID,version]#Dictionary
```

Figure 32. A response file template for the **managelU** command, showing the format of parameter variable, installation group, and feature entries. (Part 2 of 2)

Testing the values of parameter variables. Each parameter variable included in the response file must contain the following strings, separated by pound signs (#):

- The string Variable
- The string RootIUTypeID[UUID, version]
- The string parameter_name=value

If you include a parameter in the response file, the parameter must match a parameter that is already defined in the deployment descriptor. However, the *value* that you define for the parameter in the response file will override any default value that is specified for the matching parameter in the deployment descriptor.

The following example shows the key-value format for a variable named installLocation:

```
Variable#RootIUTypeID[32_character_UUID,version]#installLocation=value
```

Testing an installation group. Each installation group included in the response file must contain the following strings, separated by pound signs (#):

- The string selectedGroup appended with a number
- The string RootIUTypeID[UUID, version]
- The string installation_group_name

In the response file, list each installation group from the IU deployment descriptor that you want to test. Only one installation group can be active at a time, so comment out all the installation groups with a pound sign (#) except the active one that you want to test. When specifying an installation group, you do not need to include the features associated with the installation group (because they are "preselected" by the installation group).

The active installation group specified in the response file overrides any default installation group defined in the IU deployment descriptor.

The following example shows the format for two installation groups, Developer (active) and End user (inactive):

```
selectedGroup1#RootIUTypeID[32_character_UUID,version]#Developer
#selectedGroup2#RootIUTypeID[32_character_UUID,version]#End\ user
```

Spaces must be escaped, as with End\ user, above. Because only one installation group can be active at a time, the End user group is commented out.

Note that if you are running the **manageIU** command with a Create life cycle operation specified, and you are using the **-i** option with a selectedGroup entry activated in the response file, you can deselect a particular feature from the selected group using a deleteFeature entry. In this case, the installation group (selectedGroup) will be created, but without the feature specified in the deleteFeature entry.

Testing the installation and removal of features. In the response file, list the feature or feature combination from the IU deployment descriptor that you want to test. You can test feature installation or feature deletion. To test feature installation, specify a deployment descriptor feature that is available during a Create life cycle operation as a selected feature (selectedFeature). Note that the Create life cycle operation is used to both create (freshly install) an application *and* add a feature. To test feature deletion, specify a deployment descriptor feature that is available during a Delete life cycle operation as a delete feature (deleteFeature). The Delete life cycle operation is used to both delete an application *and* remove a feature.

Each selected feature (that is, each feature to be installed) defined in the response file must contain the following strings, separated by pound signs (#):

- The string selectedFeature appended with a number
- The string RootIUTypeID[UUID, version]
- The string *feature_name*

When running the **manageIU** command with a Create life cycle operation specified, make sure any deleteFeature entry in the response file is commented out if you have the same feature activated as a selectedFeature entry. Of the two entries, selectedFeature and deleteFeature, only the last activated entry will be processed.

The following example shows the format for two selected features, Samples and Documentation:

```
selectedFeature1#RootIUTypeID[32_character_UUID,version]#Samples
selectedFeature2#RootIUTypeID[32_character_UUID,version]#Documentation
```

Each delete feature (that is, each feature to be deleted) defined in the response file must contain the following strings, separated by pound signs (#):

- The string deleteFeature appended with a number
- The string RootIUTypeID[UUID, version]
- The string *feature_name*

The following example shows the format for two delete features, the Developer Toolkit and Dictionary:

```
deleteFeature1#RootIUTypeID[32_character_UUID,version]#Developer\ Toolkit
deleteFeature2#RootIUTypeID[32_character_UUID,version]#Dictionary
```

Spaces must be escaped, as with Developer\ Toolkit, above.

Figure 33 shows a sample response file:

```
#####
# Variable Settings
#####

# The installLocation variable is to be populated with the root directory for the installation
# (for example, /usr for UNIX-based operating systems, or #C:/Program Files for Windows operating
# systems).
# The IUDD assigns platform specific defaults. You are not required to set this variable but may
# override the default set in the IUDD.
#
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocation=C:/Program\ Files
#
# The IUDD sets reasonable defaults for these variables.
# Only uncomment the ones you want to change from the default values.
#
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocationFamily=Family
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocationChildren=Children
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocationParents=Parents
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocationDaughter=Daughter
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocationDad=Dad
# Variable#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#installLocationMom=Mom

#####
# Group Selections for Create
#####

# Note that if you are using group selections in this sample, it is not necessary
# to select features in the feature section. If you want to test different combinations,
# just select the features directly and comment out all of the group selections.
#
# selectedGroup1#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Everybody
# selectedGroup2#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Parents
selectedGroup3#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Children
```

Figure 33. Sample response file for the **manageIU** command. (Part 1 of 2)

```
#####
# Feature Selections for Create
#####

# Parents Feature must be selected when initially installing either Mom Feature or Dad
# Feature. Once Mom Feature or Dad Feature is installed, then Parents Feature does not
# need to be chosen for the other. Also, Parents Feature could be installed without
# installing Mom Feature or Dad Feature. Make sure that all of the "deleteFeature" entries
# are commented out, if you have any of the "selectedFeature" entries uncommented.
#
# selectedFeature1#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Parents\ Feature
# selectedFeature2#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Dad\ Feature
# selectedFeature3#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Mom\ Feature
#
# selectedFeature4#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Children\ Feature

#####
# Feature Selections for Delete
#####

# Choosing Parents Feature for delete will automatically delete Dad Feature and Mom Feature, but it
# will leave the root IU. Also, unlike initial create, if you want to delete only Mom Feature or
# Dad Feature, you do not have to select Parents Feature. Make sure all of the "selectedFeature"
# entries are commented out if you have any of the "deleteFeature" entries uncommented.
#
# deleteFeature1#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Parents\ Feature
# deleteFeature2#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Dad\ Feature
# deleteFeature3#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Mom\ Feature
#
# deleteFeature4#RootIUTypeID[991974510ba426fe1f53841402351114,1.2.0]#Children\ Feature
```

Figure 33. Sample response file for the **managelU** command. (Part 2 of 2)

-o *operation*

Specifies the life cycle operation to perform. Life cycle operations are the equivalent of change requests (which are described in “Change requests” on page 52), though the operation names that you specify with the **-o** option do differ slightly from the change request names. Also, the Create life cycle operation (create in the list below) is used to both create (freshly install) an application *and* create features, and the Delete life cycle operation (delete in the list below) is used to both delete an application *and* delete features.

The **-o** option supports the following values for *operation*. These values are not case sensitive:

- apply_updates
- configure
- create
- delete
- init_config
- migrate
- undo
- update

-p *directory | file_name*

Specifies a directory or file name value that Deployment Engine can use to find the software package to be deployed:

directory

Specifies the root directory of the software package to be deployed. The root directory is the directory that contains the /META-INF and /FILES subdirectories.

file_name

Specifies the file name of one of the following files:

- The archive file that contains the software package to be deployed.
- The deployment descriptor of the software package to be deployed.

On Configure life cycle operations, you must use the **-p** option to identify the location of the software package containing the CU deployment descriptor. On Initial Configure and Migrate life cycle operations, you must either use the **-p** option to identify the location of the software package containing the IU deployment descriptor or, alternatively, use the **-Cfg** option to identify the root IU type identifier of the IU deployment descriptor. On Reapply Update, Create (includes Create Feature), and Update life cycle operations, you must use the **-p** option to identify the location of the software package containing the IU deployment descriptor. Do *not* use the **-p** option on Delete (includes Delete Feature) and Undo life cycle operations.

-r *discriminant*

Identifies the particular installed software instance affected by the life cycle operations being tested. If the software instance is affected by a Create life cycle operation, the discriminant might identify the target path where the software instance is to be deployed. If the software instance is affected by any other life cycle operation, the discriminant might identify the current path where the software instance is already deployed.

Note that the discriminant can have *any* string identifier; it does not *have* to be a directory path. Also note that if the `_discriminant` variable described on page 66 is used in a deployment descriptor, the value derived from the `_discriminant` variable supersedes the value supplied with the **-r** option. To determine whether this is advantageous in your situation, refer to “Internal variables” on page 66.

The **-r** option is required, although it is ignored on Configure life cycle operations. Combine this option with the **-Cfg**, **-d**, or **-p** option to uniquely identify a software instance. A root IU type identifier and discriminant specified together identify an installed software instance uniquely (see “Specifying a software instance uniquely” on page 83 for a detailed explanation).

-rerun *file_name* | **all** | **features**

Specifies one of the following values, which indicate some particular action descriptors that should be rerun:

file_name

Reruns the action descriptors for all of the smallest installable units listed in file *file_name*, which provides this list as additional input to the **manageIU** command. The file type for this file must be `.properties`; for example, `rerun.properties`.

If you specify *file_name*, the action descriptors for any selected feature (selectedFeature) listed in the response file used with the **-i** option will also be rerun. If you specify *file_name*, however, you do not *have to* specify the **-i** option.

all Reruns the action descriptors for all installed smallest IUs. If you specify **all**, you *cannot* use the **-i** option.

features Reruns the action descriptors for any selected feature (selectedFeature) listed in the response file that you specified with the **-i** option. If you specify **features**, you *must* use the **-i** option.

Figure 34 shows the format of an entry in a rerun file that might be used with the **managelIU** command.

```
# rerun.properties file for use with manageIU command
# -----
#
# This file is used to rerun the action descriptors for all of the installable
# units listed below.
#
#
# The following example shows the format for a software IU type identifier,
# which identifies a smallest IU to be rerun. Multiple software IU type
# identifiers can be listed:
#
SoftwareIUTypeID[32_character_UUID,version,RootIUTypeID[32_character_UUID,version]]
```

Figure 34. A rerun file template for the **managelIU** command, showing the format of the list of installable units whose action descriptors are to be rerun.

Figure 35 shows a sample rerun file:

```
#####
#Installable units to rerun
#####
#
#
SoftwareIUTypeID[100900510ba426fe1f5377770000111b,1.5,RootIUTypeID[100900510ba426fe1f53777700001111,1.5]]
```

Figure 35. Sample rerun file for the **managelIU** command.

-u true | false

Indicates whether you want to allow the software changed by an Update life cycle operation (specified with the **-o** option) to have an Undo life cycle operation performed on it in the future. This undo option applies to fixes and incremental updates only. It does not apply to full updates. Specify **true** if you want to allow an Undo operation for the update or **false** if you do not. The default value is **false**.

-v true | false

Indicates whether you want the most output possible—so-called "verbose" output—displayed on-screen when running the **managelIU** command; that is, whether you want to display the maximum available processing information, including the display of command syntax and option

descriptions when a command is entered that is not valid. Specify true if you want the maximum information or false if you do not. The default value is false.

Authorization

Root or nonroot user.

See “Command authorization” on page 81 for a description of these users.

See also

When testing using the **manageIU** command, consider the following things:

- Deployment Engine is platform-specific, so be sure to test your software packages on all applicable Windows, UNIX-based, and OS/400 operating systems.
- If possible, test your software packages on single-user mode and multiuser mode Deployment Engine installations, and test for both root and nonroot users.
- Be sure to test all the life cycle operations that are appropriate for your software package using the **manageIU** command. You can double-check some life cycle operations by also using the administration commands **de_instmaint**, **de_uninstmaint**, **de_uninstfeat**, or **de_uninstapp**, described in the book *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*.
- Use whatever techniques are appropriate to validate that your software packages were deployed correctly. For example, use the administration commands **de_lsapp**, **de_lsfeat**, or **de_lsmaint**, described in the book *IBM Autonomic Deployment Engine: Autonomic Deployment Engine for Administrators*, to query the Deployment Engine installation database and verify that the installable units you intended to deploy are registered. In addition, check your target file system to ensure that critical application files were deployed as expected.

validateIUIDD

Validates a deployment, action, or media descriptor in an application's software package.

Syntax

```
validateIUIDD [-silent] [-s schema_directory] file_name
```

```
validateIUIDD -?
```

Description

The **validateIUIDD** command validates a specified descriptor file in a software package for an application. You can validate a specified deployment descriptor, action descriptor, or media descriptor against its associated Deployment Engine schema file and, in the case of the deployment descriptor, Deployment Engine XML semantics.

Command options are not case sensitive; option order does not matter.

When you issue the **validateIUIDD** command, specify **validateIUIDD.sh** on UNIX-based and OS/400 operating systems; the operating system will process the **validateIUIDD.sh** command file. On Windows operating systems, specify **validateIUIDD**; in this case, the operating system will process the **validateIUIDD.cmd** command file.

Options

file_name

Specifies the fully qualified name of a descriptor that you want to validate in the application's software package. When validating an IU deployment descriptor, Deployment Engine will also validate any of the descriptor's requisites (as defined by **referencedIU** elements), contained IUs, or contained container IUs.

-s *schema_directory_path*

Specifies the full path to the schema (*.xsd) file for the specified descriptor. This option is required only if the schema is in a directory different from the default directory where Deployment Engine installed it. The default directory is \$SI_PATH/schema, where \$SI_PATH is the directory where Deployment Engine was installed. When specifying the path, do *not* specify the schema file name. Deployment Engine obtains the schema file name from the descriptor file that you specify in your command.

-silent Suppresses output messages when processing this command.

-? Displays command syntax for the **validateIUIDD** command.

Authorization

Root or nonroot user.

See "Command authorization" on page 81 for a description of these users.

Return values

0 Validation completed without errors.

1 Validation detected one or more errors within the specified descriptor file.

- 2 Could not find the specified descriptor file. Validation could not be performed.
- 3 Could not find schema files. Validation could not be performed.
- 4 Command incorrectly invoked.
- 5 An internal error occurred.

Examples

1. To determine whether deployment descriptor AB_packagedIU.xml in the C: directory is valid, enter the following command:

```
validateIUDD.cmd c:\AB_packagedIU.xml
```

If the deployment descriptor is valid, you should receive the following messages:

```
ACUVI0001I Validating c:\AB_packagedIU.xml...
ACUVI0009I Document is valid
```

If the deployment descriptor is not valid, the messages you receive should include information about the errors:

```
ACUVI0001I Validating c:\AB_packagedIU.xml...
ACUVI0006E Error: [line7,col20] -cvc-pattern-valid: Value '8563047' is not
facet-valid with respect to pattern '[0-9]{27}'.
Error: [line7,col20] -cvc-type.3.1.3: The value '8563047' of element 'UUID'
is not valid.
Error: [line11,col44] -cvc-pattern-valid: Value 'CPU Check' is not facet-valid
with respect to pattern '[a-zA-Z_]+[0-9a-zA-Z_]*'
ACUVI0008I ValidateIUDD completed with 0 warnings and 3 errors.
```

2. To determine whether deployment descriptor AB_packagedIU.xml in the C: directory is valid and not receive messages (like those in the previous example) to the screen, enter the following command:

```
validateIUDD.cmd c:\AB_packagedIU.xml -silent
```

To determine whether the command ran successfully, retrieve the return code after running a command, as described in “Retrieving return codes” on page 84.

Part 3. Problem determination

Chapter 6. Locating the Deployment Engine log files

Deployment Engine logs message and trace information into flat-text log files. These log files can be used for troubleshooting purposes. The locations of the Deployment Engine message and trace logs for components can be found in the ACULogger.properties file.

To locate the Deployment Engine logs:

1. Find the ACULogger.properties file, as described on page 105.
2. Find the logs for components, as described on page 105.

Finding the ACULogger.properties file

To determine the location of the message and trace log files for the Deployment Engine components, you need to look in the The ACULogger.properties file. This properties file can be found in the common directory (referred to as \$ACU_COMMON) for Deployment Engine files.

As described in “Environment variables for the installed directories” on page 73, the ACU_COMMON environment variable can be used to locate the common directory for Deployment Engine files. To determine the location, just query your operating system for the value of the ACU_COMMON environment variable.

After locating the ACULogger.properties file, use the information in the following section to find the log files for messages and traces.

Finding the logs for components

In the sample property-file lines shown below, a **fileName** line specifies the file name for a log file. The subsequent **fileDir** line specifies the path of the directory, *username*, where that log file is located.

To find the message log file for the Deployment Engine component messages, look for the following lines in the \$ACU_COMMON/ACULogger.properties file:

```
acu.message.handler.file.fileName = de_msg.log  
acu.message.handler.file.fileDir = $SI_PATH/logs
```

Note: This line represents the path to the log file directory *username*, where *username* is the user ID of the user on the current operating system. Therefore the log file will actually be located in \$SI_PATH/logs/*username*. (SI_PATH is an environment variable that represents the top-level directory where Deployment Engine was installed. To determine the location of this installation directory, just query your operating system for the value of the SI_PATH environment variable.)

To find the trace log file for the Deployment Engine components, look for the following lines in the \$ACU_COMMON/ACULogger.properties file:

```
acu.trace.handler.file.fileName = de_trace.log  
acu.trace.handler.file.fileDir = $SI_PATH/logs
```

Note: This line represents the path to the log file directory *username*, where *username* is the user ID of the user on the current operating system. Therefore the log file will actually be located in `$SI_PATH/logs/username`. (SI_PATH is an environment variable that represents the top-level directory where Deployment Engine was installed. To determine the location of this installation directory, just query your operating system for the value of the SI_PATH environment variable.)

Chapter 7. Message logging

The sections that follow explain the standard form of Deployment Engine messages and provide help information about the Deployment Engine warning and error messages.

Message identifier

Deployment Engine message identifiers have the following format:

AAABBnnnnC

where the parts of the message are as follows:

AAA The “product” prefix. The prefix for Deployment Engine is **ACU**.

BB The “component” prefix. The “component” prefix for Deployment Engine commands is **EX**. The component prefixes Deployment Engine subcomponents, or internal components, are as follows:

SI Common (indicates messages common to multiple internal components)

DB Installation database component

CM Change manager component

DC Dependency checker component

OS Operating system touchpoint

nnnn A numeric identifier unique within the combination of product and component prefixes.

C The severity code indicator:

I **Informational:** Informational messages provide users with information or feedback about normal events that have occurred or are occurring, or request information from users in cases where the outcome will not be negative, regardless of the response.

Examples:

- The status request is processing.
- The files were successfully transferred.
- Do you want to save your output in file a or in file b?

Note: Informational messages issued by Deployment Engine are not documented in this chapter, as they are complete in themselves and require no further information or explanation. This also applies to informational messages giving the usage of the Deployment Engine administrator commands.

W **Warning:** Warning messages indicate that potentially undesirable conditions have occurred or could occur, but the program can continue. Warning messages often ask users to make decisions before processing continues.

Examples:

- A requested resource is missing. Processing will continue.

- A file already exists with the same name. Do you want to overwrite this file?

E **Error:** Error messages indicate problems that require intervention or correction before the program can continue.

Examples:

- The specified file could not be found.
- You are out of space on the x drive. The file cannot be saved to this drive.

Message text

Every attempt has been made to represent the message text exactly as it appears in the displayed or written message.

Where the system has included variable information in the message text, this variable information is represented by an italicized label, describing the type of information referred to by the variable. For example, if the message text that appears on your screen is:

The error code is 1.

the message text shown in this chapter would be:

The error code is *error_code*.

In this case the label *error_code* tells you that the information that will be inserted into the message by Deployment Engine is the error code for the problem.

Message help

In addition to the message text itself, help information is provided for most warning and error messages. In this book, message help can include the following help topics:

Explanation

This help topic typically expands on the information provided in the message text, more fully explaining the circumstances in which the message was issued.

User Response

This help topic tells the user what to do next. This might include instructions on how to solve a problem or correct an error. Sometimes, particularly with warning messages, there is nothing that the user needs to do. Sometimes the user is directed to where to find more information, or, if necessary, who to see for further assistance.

Message log format

The message log contains a list of messages for the end-users that includes the following fields:

Message entry date

Indicates the year, month, and day that the message entry was generated.

Message entry time

Indicates the time of day that the message entry was generated.

Java class name

Indicates the name of the Java class that generated the message entry.

Method name

Indicates the name of the method that generated the message entry.

Host name

Indicates the fully qualified host name of the computer that is running the instance of Deployment Engine that generated the message entry.

Message identifier

Indicates the identifier of the message entry.

Message text

Indicates the text of the message entry.

Chapter 8. Messages issued by components

This section contains message help information for all of the error and warning messages issued by internal components of Deployment Engine that are referenced by a unique message reference number. The Deployment Engine components that issue these messages are described in “The components of a Deployment Engine environment” on page 6.

The messages are presented in the order listed below. The Deployment Engine common messages are presented first. Then the messages for each Deployment Engine internal component are presented in alphanumeric order, beginning with the change manager subcomponent.

- “Common messages” on page 112
- “Change manager messages” on page 113
- “Dependency checker messages” on page 114
- “Operating system touchpoint messages” on page 115

Note: Informational messages issued by Deployment Engine are not documented in this chapter, because they are complete in themselves and require no further information or explanation. This also applies to informational messages that provide usage information about Deployment Engine commands.

Common messages

The messages in this section are common to more than one Deployment Engine component.

ACUSI0000E A processing error occurred that you cannot resolve by yourself. The error code is *error_code*.

Explanation: This error happened within internal componentry that is not generally accessible. Fixing the problem requires outside assistance.

User response: Make a note of the error code, and contact your support representative for help.

ACUSI0002E The Deployment Engine session is currently busy processing another operation. The new operation cannot be processed.

Explanation: Deployment Engine must process database operations singly to prevent the corruption of its database. Because one such operation was already in progress, the new operation was not processed.

User response: Make sure you are not doing multiple operations that affect the database at the same time. Examples of multiple operations include simultaneous software deployments and a software deployment that occurs concurrently with a database operation like backup or restore. Retry your operation again when no other database-related operations are in progress.

Change manager messages

The messages in this section are issued by the Deployment Engine change manager component.

ACUCM3002W Change management operation *operation* failed for installable unit *installable_unit*.

Explanation: A failure occurred during the processing of an action that subsequently caused the change management operation to fail. This failed change management operation will usually cause the current change request to fail as well.

User response: This warning requires no response, but corrective actions could be necessary if the change request fails. Refer to the user response that is provided for any subsequent error messages.

ACUCM3004W Rollback of change management operation *operation* failed for installable unit *installable_unit*.

Explanation: Because a change request failed, Deployment Engine attempted to roll back each change management operation that was processed for the change request. The purpose of these rollbacks is to return the software to its original state. However, one of the rollbacks failed. Deployment Engine will not attempt to roll back any additional change management operations.

User response: This warning requires no response, but corrective actions will be necessary because the change request failed. Refer to the user response that is provided for any subsequent error messages.

ACUCM3005E Change request type *request_type* failed for software package *package_name*. Deployment Engine successfully rolled back the changes that it made when processing the change request.

Explanation: The change request failed because a change management operation failed. All changed software was restored to its original state.

User response: Resubmit the change request. If the problem persists, contact your support representative for assistance.

ACUCM3006E Change request type *request_type* failed for software package *package_name*. Rollback also failed; Deployment Engine was unable to restore the changes that it made when processing the change request.

Explanation: The change request failed because a change management operation failed. The changed software was not restored to its original state.

User response: Uninstall the software package and resubmit the change request. If the problem persists, contact your support representative for assistance.

Dependency checker messages

The messages in this section are issued by the Deployment Engine dependency checker component.

ACUDC0002W One or more dependencies were not satisfied for software package *package_name*.

Explanation: Prior to software deployment, Deployment Engine can check for different kinds of dependencies, including a computer's processing or resource capacity, the availability of other software, or a property value, relationship, or version associated with some particular software. In most cases an unsatisfied dependency will cause the current change request to fail.

User response: The software deployment program might issue a message that provides details about the dependencies that caused this problem or might prompt for additional information to address the problem. Contact your support representative if you need additional help to identify and fulfill the unsatisfied dependencies so that software deployment can go forward.

ACUDC0003W Deployment Engine found one or more integrity check violations in software package *package_name*.

Explanation: Prior to software installation, Deployment Engine makes sure that the software registered in the installation database can coexist with the software you are trying to install. Prior to software removal, Deployment Engine makes sure that no registered software depends on the software you are trying to remove. This function is referred to as integrity checking. Continuing when integrity check violations are present can adversely affect other software currently in use. Therefore an integrity check violation will usually cause the current change request to fail.

User response: The software deployment program might issue a message that provides details about the integrity check violation that caused this problem or might prompt for additional information to address the problem. Typically you do not want to install or remove any software as long as it continues to violate the Interoperability requirements of other software currently in use. Contact your support representative if you need additional help to 1) understand the integrity check violations or 2) make adjustments to the registered software so that software installation or removal can go forward.

Operating system touchpoint messages

The messages in this section are issued by the operating system touchpoint.

ACUOS0046W A failure occurred during the installation processing of action *action_name*, sequence number *seq_num*. Deployment Engine will ignore the failure and process the next action.

Explanation: This failure is usually associated with an authorization, disk space, or other operating system problem. Deployment Engine received instructions to ignore this failure and continue processing. As a result, this particular failure will not cause the change management operation to fail.

User response: This warning requires no response. However, corrective actions might be required at a later time to ensure the proper functioning of any related software or applications affected by the failed action.

ACUOS0059W A failure occurred during the undo processing of action *action_name*, sequence number *seq_num*. Deployment Engine will ignore the failure and process the next action.

Explanation: This failure is usually associated with an authorization, disk space, or other operating system problem. Deployment Engine received instructions to ignore this failure and continue processing. As a result, this particular failure will not cause the change management operation to fail.

User response: This warning requires no response. However, corrective actions might be required at a later time to ensure the proper functioning of any related software or applications affected by the failed action.

ACUOS0060W A failure occurred during the uninstallation processing of action *action_name*, sequence number *seq_num*. Deployment Engine will ignore the failure and process the next action.

Explanation: This failure is usually associated with an authorization, disk space, or other operating system problem. Deployment Engine received instructions to ignore this failure and continue processing. As a result, this particular failure will not cause the change management operation to fail.

User response: This warning requires no response. However, corrective actions might be required at a later time to ensure the proper functioning of any related software or applications affected by the failed action.

ACUOS0061W A failure occurred during the installation processing of action *action_name*, sequence number *seq_num*. Subsequent actions will not be processed.

Explanation: This failure is usually associated with an authorization, disk space, or other operating system problem. Deployment Engine received instructions to stop processing when this action failed. As a result, Deployment Engine will not process any more actions in this change management operation and the change management operation will probably fail.

User response: This warning requires no response, but corrective actions could be necessary at a later time if the change management operation and its associated change request fail. Refer to the user response that is provided for any subsequent error messages.

ACUOS0062W A failure occurred during the undo processing of action *action_name*, sequence number *seq_num*. Subsequent actions will not be processed.

Explanation: This failure is usually associated with an authorization, disk space, or other operating system problem. Deployment Engine received instructions to stop processing when this action failed. As a result, Deployment Engine will not process any more actions in this change management operation and the change management operation will probably fail.

User response: This warning requires no response, but corrective actions could be necessary at a later time if the change management operation and its associated change request fail. Refer to the user response that is provided for any subsequent error messages.

ACUOS0063W A failure occurred during the uninstallation processing of action *action_name*, sequence number *seq_num*. Subsequent actions will not be processed.

Explanation: This failure is usually associated with an authorization, disk space, or other operating system problem. Deployment Engine received instructions to stop processing when this action failed. As a result, Deployment Engine will not process any more actions in this change management operation and the change management operation will probably fail.

User response: This warning requires no response, but corrective actions could be necessary at a later time if the change management operation and its associated change request fail. Refer to the user response that is provided for any subsequent error messages.

Chapter 9. Trace logging

The trace log file contains information that you can use to troubleshoot problems yourself, or that you can send to IBM Support so that IBM can troubleshoot the problem.

In the `$ACU_COMMON/ACULogger.properties` file, you can set the trace level for all the components in the Deployment Engine operating environment (except the WebSphere touchpoint) using a single setting. (For information about the tracing facility used by the WebSphere touchpoints, see the book *Solution Install for Autonomic Computing: WebSphere Touchpoint Guide and Reference*.) That setting must be one of the following values:

`DEBUG_MIN`

Enables high-level tracing

`DEBUG_MID`

Enables mid-level tracing, including the tracing of method entries and exits

`DEBUG_MAX`

Enables full tracing.

To specify a trace level for all the supported Deployment Engine components, modify the following line in the `ACULogger.properties` file (where *trace_level* is either `DEBUG_MIN`, `DEBUG_MID`, or `DEBUG_MAX`):

```
acu.logger.level=trace_level
```

You can specify a trace level on a component basis for some internal and external components of Deployment Engine that have their own trace loggers. By default, these loggers are configured to use the trace level set for *all* Deployment Engine components, as noted above. However, you do have the option to reconfigure them. The following trace loggers are supported for these specific components:

DC Trace logger for dependency checker processing

CM Trace logger for change manager processing

AP Trace logger for the action processor of the operating system touchpoint; this logger records the overall success or failure of each processed action as the touchpoint progresses through the action descriptor

OSTP Trace logger for the action-specific code of the operating system touchpoint; this logger records the step-by-step processing of each individual action in the action descriptor

PA Trace logger for assisting with performance analysis; this logger logs time stamps at particular points in the trace data, to help with performance analysis

To reconfigure the component trace loggers, modify the following two lines in the `ACULogger.properties` file for each logger that you want to change (where *logger_name* is the `DC`, `CM`, `AP`, `OSTP`, or `PA` trace logger).

```
acu.logger.logger_name.logging=true|false
acu.logger.logger_name.level=trace_level
```

On the first line, a setting of `false` (the default) indicates that the logger's trace level is the same as the trace level set for *all* Deployment Engine components (as

set on the `acu.logger.level`= line, described previously). To assign the logger its own trace level, which may be different from all the other components, 1) specify a value of `true` on the first line, and 2), on the second line, specify the trace level you want for this particular logger (where *trace_level* is either `DEBUG_MIN`, `DEBUG_MID`, or `DEBUG_MAX`):

The trace log contains information to help trace the flow of processing when diagnosing problems. Each trace entry in the log includes the following fields:

Trace entry date

Indicates the year, month, and day that the trace entry was generated.

Trace entry time

Indicates the time of day that the trace entry was generated.

Java class name

Indicates the name of the Java class that generated the trace entry.

Method name

Indicates the name of the method that generated the trace entry.

Host name

Indicates the fully qualified host name of the computer that is running the instance of Deployment Engine that generated the trace entry.

Message identifier

Indicates the identifier of the message for the trace entry.

Message text

Indicates the text of the message for the trace entry.

Chapter 10. Troubleshooting

The following topics provide troubleshooting information to help prevent or resolve problems related to Deployment Engine:

- **Specifying a value for maximum version.**

The `maxVersion` element in a deployment descriptor indicates the *exact* maximum value allowed for the version of the software that is being checked. If the software version is any higher than the exact maximum value, it will fail the version check. For example, if the `maxVersion` value is set to 5.1, any software with a version of 5.1.1 or greater will fail the version check because it exceeds the maximum value of 5.1. In this particular case, to allow software with a version up to 5.1.*n* to pass the version check, set the `maxVersion` value to 5.1.*n*. For example, to allow any software with a version up to 5.1.9999 to pass a version check, set the value of the `maxVersion` element to 5.1.9999.

- **When Uninstall and Undo action descriptors are used.**

When performing an Undo change request on an incremental update or fix software package, the Install action descriptor is processed in reverse. The Uninstall action descriptor is not used during Undo change request processing. The Uninstall action descriptor is used only when a Delete change request is run against the base software package. (See Table 2 on page 55 for more information.)

- **Circular contained IUs yield StackOverflowError.**

A Java error, `java.lang.StackOverflowError`, is reported when Deployment Engine encounters circular contained IUs. For example, a stack overflow error is reported if an IU deployment descriptor has a contained IU that references a second descriptor with a contained IU that references the original descriptor. Make sure that none of the software packages within your application package have IU deployment descriptors whose contained IUs circularly reference one another's IU deployment descriptor. These circular references are not valid and will cause a Java error.

- **Missing targetRef attribute yields NullPointerException.**

A Java exception, `java.lang.NullPointerException`, is reported when one or more smallest IUs or CUs in a deployment descriptor do not reference a topology target. (Topology targets are referenced using the `targetRef` attribute.) For example, a null pointer exception will be reported during a Configure change request when one or more smallest CUs do not reference a topology target and no topology target is referenced by the root CU.

Smallest IUs and CUs must either reference a topology target or inherit the topology target of an ancestor unit in the IU or CU hierarchy. When referencing a topology target using the `targetRef` attribute, set the value of the `targetRef` attribute to the value of the `id` attribute of a target that is defined in the topology section of the IU or CU deployment descriptor.

- **The expected rollback of a change request did not occur.**

There are some common reasons why a rollback might not occur as expected. Before Deployment Engine can attempt a rollback, *all* of the following requirements must be met:

- The change request is a Create, Create Feature, Update, Reapply Update, or Configure request (Delete, Delete Feature, Undo, InitialConfig, and Migrate requests do *not* support rollback).
- The change request is *not* a forced change request.

- Rollback support is enabled by the `ChangeRequest.setRollbackEnabled(boolean)` API (rollback support is enabled by default).
- The error handler is set to `ON_ERROR_ROLLBACK` (or, to attempt rollback on the failed installable unit only, is set to `PAUSE_ROLLBACK_FAILED_IU`).

If any of the above requirements are not met and an error occurs in change request processing, change manager will *not* attempt a rollback.

If, during rollback, only some `xxxArtifact` elements in the IU deployment descriptor are defined with the `undoable` attribute set to `true`, change manager processes the rollback until it encounters the first `xxxArtifact` element that does not have the `undoable` attribute set to `true`. Change manager then stops, registers the current state of the change request, and the change request fails.

- **State of an installable unit changes from Created to Usable without running the InitialConfig action descriptor.**

The state of an installable unit can change from Created to Usable—even without running the required InitialConfig action descriptor—if the condition attribute of the installable unit evaluates to `false` during the processing of the InitialConfig change management operation. When specifying a condition attribute for an installable unit, specify a value that evaluates to either `true` or `false` on *all* change management operations. For example, do not specify a condition that can change over time, such as a required amount of available disk space. Such a condition could evaluate as `true` at the time of a Create operation but as `false` at the time of an InitialConfig operation.

In general, then, the condition of an installable unit should evaluate to the same value on all operations. This will prevent other, similar problems. For example, when performing a full update, Deployment Engine cannot update an installable unit if its condition evaluates to `false` for the full update.

- **InitialConfig action descriptors are ignored for new installable units in update software packages.**

A new installable unit that is introduced in a full or incremental update software package cannot list an InitialConfig action descriptor as one of its valid action descriptors. During an Update change request, Deployment Engine ignores any InitialConfig action descriptor provided for a newly created installable unit, because it expects a follow-on Migrate change management operation with a Migrate action descriptor. To circumvent the problem, provide the InitialConfig information in the Install action descriptor instead.

- **Using JavaCustomAction yields ClassCastException.**

A Java exception, `java.lang.ClassCastException`, is reported by Deployment Engine when processing the action `JavaCustomAction` if the custom class does not implement the `com.ibm.ac.si.ap.action.JavaCustomActionInterface` interface. So make sure the custom class implements this interface.

Part 4. Appendixes

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Open source license notices

This section contains details concerning certain notices IBM must provide to you under its license to certain software code. The relevant notices are provided or referenced below. Please note that any non-English version of the licenses below is unofficial and is provided to you for your convenience only. The English version of the licenses below, provided as part of the English version of this file, is the official version.

Notwithstanding the terms and conditions of any other agreement you may have with IBM or any of its related or affiliated entities (collectively "IBM"), the third party software code identified below are "Excluded Components" and are subject to the terms and conditions of the License Information document accompanying this Program.

Apache Software License, Version 1.1

Copyright © 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."
Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

Apache Software License, Version 2.0

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - b. You must cause any modified files to carry prominent notices stating that You changed the files; and
 - c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

W3C Software Notice and License

<http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C[®] Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION. The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Trademarks

IBM, the IBM logo, DB2, OS/400, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft and Windows NT, and Windows 2000 are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

- action descriptor sets
 - definition 47
- action descriptors
 - actions 46, 47
 - and change management operations 54
 - and root installable units 48
 - and smallest configuration units 46
 - and smallest installable units 46, 48, 54
 - Configure action descriptors 47
 - Custom-check action descriptors 48
 - definition 45
 - identified in an IU deployment descriptor (XML sample) 48
 - in Deployment Engine operating environment 8
 - InitialConfig action descriptors 47
 - Install action descriptors 46
 - life cycle-related 46
 - Migrate action descriptors 47
 - sets 47
 - Uninstall action descriptors 47
 - XML sample 49
- actions
 - definition 46
- ACU_COMMON environment variable 73
- administration commands v, 88
 - de_version command 81
- alternatives 60, 90
- appendix information 123
- application packages
 - definition 13
- applications
 - definition 5
 - testing with manageIU command 86
- artifacts
 - See action descriptors
- authorities
 - multiuser mode 70
 - single-user mode 71

B

- backing resources
 - and Deploys relationship 61
 - and manageIU -force command 91
 - and overwriting identical software 91
 - and resulting resources 38
 - definition 38
- backward-compatible constraints
 - forcing failed checks 92
- base application
 - definition 12
- base content
 - definition 34
- base software packages
 - definition 12
- bold typeface, meaning of vi
- bootstrap program 8

C

- capacity dependencies 56
- case sensitivity in command syntax 82
- change management operations
 - and handling of relationship types 63
 - Configure operation 55, 56
 - Create operation 55
 - definition 54
 - Delete operation 55
 - InitialConfig operation 55
 - Migrate operation 55
 - Undo operation 55
 - Update operation 55
- change manager
 - definition 9
 - in Deployment Engine operating environment 9
 - messages 113
- change plan
 - definition 54
- change requests
 - and life cycle operations 97
 - Configure change request 53
 - Create change request 53
 - Create Feature change request 53
 - definition 52
 - Delete change request 53
 - Delete Feature change request 53
 - Initial Configure change request 53
 - Migrate change request 53
 - Reapply Update change request 53
 - Undo change request 53
 - Update change request 53
- checks
 - and alternatives 60, 90
 - and dependencies 57
 - and requirements 60, 90
 - capacity checks 57
 - consumption checks 57
 - custom checks 58
 - definition 57
 - forcing failed checks 90
 - hosted resource checks 58
 - installable unit checks 58
 - property checks 59
 - relationship checks 59
 - software checks 59
 - types 57
 - version checks 60
- commands 83, 84
 - case sensitivity in 82
 - description 79
 - location of command files 82
 - manageIU command 86
 - summary of developer commands 79
 - validateUDD command 101
- common messages
 - for Deployment Engine components 112
- components
 - application-defined
 - software packages 14
 - Deployment Engine operating environment 6

- conditioned units
 - definition 17
- configuration unit deployment descriptors
 - definition 16
 - in Deployment Engine operating environment 8
 - root configuration units 28
 - smallest configuration units 29
- configuration unit hierarchies
 - definition 33
- configuration units
 - definition 27
 - root configuration units 28
 - smallest configuration units 29
- Configure action descriptors
 - definition 47
- configuring
 - definition 6
- contained container installable units
 - definition 24
 - XML sample 25
- contained installable units
 - definition 23
 - XML sample 24
- container installable units
 - definition 22
 - XML sample 23
- conventions used in this book
 - command syntax 82
 - typeface vi
 - UNIX directory names vii
 - UNIX directory paths vii
 - UNIX environment variables vii
- Created state
 - definition 52
- CUs
 - See* configuration units
- Custom-check action descriptors
 - definition 48

D

- de_version command 81
- dependencies
 - and checks 57
 - capacity checks 57
 - capacity dependency 57
 - consumption checks 57
 - consumption dependency 57
 - custom checks 58
 - customized dependency 58
 - definition 56
 - hosted resource checks 58
 - hosted resource dependency 58
 - installable unit checks 58
 - installable unit dependencies 58
 - property checks 59
 - property dependencies 59
 - relationship checks 59
 - relationship dependencies 59
 - software checks 59
 - software dependencies 59
 - version checks 60
 - version dependency 60
- dependency checker
 - definition 8, 56
 - in Deployment Engine operating environment 8
 - messages 114

- dependency checking
 - and Custom-check action descriptors 48
 - definition 56
- deploying
 - definition 6
- deployment descriptors
 - configuration unit deployment descriptors 16
 - definition 15
 - installable unit deployment descriptors 15
 - types 15
 - validating with validateIUID command 101
- Deployment Engine
 - definition 3
 - installation database 9
 - messages from components 111
 - operating environment components 6
 - previous name (Solution Install) 3
 - run-time components 7
 - what it does 5
- Deployment Engine interface
 - in Deployment Engine operating environment 8
- Deployment Engine run-time environment
 - definition 69
 - in the application package 69
- Deployment Engine-enabled application
 - definition v
- Deploys relationship 61
- derived variable
 - definition 64
- descriptor
 - See* deployment descriptor
- descriptors
 - action descriptors 45
 - definition 14
 - deployment descriptors 15
 - in Deployment Engine operating environment 7
 - media descriptor 49
- discriminant
 - definition 65
 - internal variable 66
 - query IU discriminant variable 65
- discriminant variable
 - definition 66

E

- environment variables 73
 - ACU_COMMON environment variable 73
 - ERRORLEVEL environment variable 84
 - SL_PATH environment variable 73
 - ERRORLEVEL environment variable 84

F

- features
 - definition 35
 - testing with manageIU command 93
- Federates relationship 62
- fix pack
 - in incremental update software package 13
- fix software packages
 - definition 13
- Fixes relationship 62
- force processing
 - with manageIU command 89

- fresh installation
 - definition 12
- full update software packages
 - definition 12
- fully enabled software deployment
 - definition 10

H

- HasComponents relationship 62
- hosted resources
 - definition 9, 58
 - in Deployment Engine operating environment 9
- hosting environments
 - definition 9
 - in Deployment Engine operating environment 9
- Hosts relationship 62
- hybrid software deployment programs
 - definition 12

I

- implementing Deployment Engine
 - single software package 11
 - software deployment methods 11
 - software deployment models 10
 - software package set 12
 - software package tree 11
 - wrapped software deployment 10
- incremental software packages
 - definition 13
- InitialConfig action descriptors
 - definition 47
- Install action descriptors
 - definition 46
- installable unit deployment descriptors
 - contained container installable units 24
 - contained installable units 23
 - container installable units 22
 - definition 15
 - in Deployment Engine operating environment 7
 - root installable units 18
 - smallest installable units 20
 - solution modules 25
 - validating with validateIUDD command 101
 - what units to include 42
- installable unit hierarchies
 - definition 32
- installable units
 - contained container installable units 24
 - contained installable units 23
 - container installable units 22
 - definition 4, 17
 - root installable units 18
 - smallest installable units 20
 - software life cycle 51
 - solution modules 25
 - supported states 51, 52
- installation database
 - access restrictions in multiuser mode 71
 - access restrictions in single-user mode 72
 - database access 71
 - definition 9
 - in Deployment Engine operating environment 9
 - multiuser access 70

- installation groups
 - definition 36
 - testing with manageIU command 93
- instances
 - definition 5
- integrity checking
 - and relationships 64
 - definition 57
 - forcing failed checks 91
- interim fix
 - in fix software package 13
- introduction to Deployment Engine 3
- italic typeface, meaning of vi
- IUs
 - See* installable units

L

- legal notices
 - IBM 123
 - third party 124
- levels, trace 117
- life cycle
 - See* software life cycle
- life cycle operations
 - See also* change requests
 - definition 86
 - option in manageIU command 97
- life cycle states
 - Created state 52
 - definition 52
 - Updated state 52
 - Usable state 52
- logging
 - trace logging 117

M

- maintenance
 - definition 6
- managed resources
 - backing resources 38
 - definition 38
- manageIU command 86
- manufacturing refresh
 - in full update software package 12
- media descriptors
 - and multivolume files 50
 - definition 49
 - in Deployment Engine operating environment 8
- message log fields 108
- messages
 - change manager messages 113
 - common messages
 - for Deployment Engine components 112
 - dependency checker messages 114
 - format
 - message help 108
 - message identifiers 107
 - help information for 111
 - message log fields 108
 - message text 108
 - operating system touchpoint messages 115
- Migrate action descriptors
 - definition 47
- models for software deployment 10

- monospace font, meaning of vi
- multiuser mode
 - and de_version command 81
 - definition 70
- multivolume files
 - definition 50

N

- nonroot user
 - and Deployment Engine in single-user mode 71
 - definition 81
- notices
 - IBM 123
 - third party 124

O

- operating environment of Deployment Engine 6
- operating system touchpoint
 - in Deployment Engine operating environment 9
 - messages 115
- optional content
 - definition 34
 - features 35
 - installation groups 36
 - XML sample 35
- overview of Deployment Engine 3

P

- packaging
 - definition 6
- parameter variable
 - definition 64
 - testing with manageIU command 93
- payload files
 - definition 3, 15
 - in Deployment Engine operating environment 8
- problem determination 105
 - Deployment Engine component messages 111
 - finding the ACULogger.properties files
 - component messages 105
 - message help 108
 - trace logging 117
 - troubleshooting 119
- property dependencies 56

Q

- query IU discriminant variable
 - definition 65
- query property variable
 - definition 65

R

- refresh pack
 - in incremental update software package 13
- relationship types
 - Deploys relationship 61
 - Federates relationship 62
 - Fixes relationship 62
 - HasComponents relationship 62
 - Hosts relationship 62

- relationship types (*continued*)
 - relationships.xsd schema 61
 - Supersedes relationship 62
 - Uses relationship 63
- relationships
 - and integrity checking 64
 - definition 61
 - management of 63
- representing strings that include spaces 84
- requirements 60, 90
- requisite packages
 - See requisites
- requisites
 - definition 40
 - XML sample 42
- rerun processing
 - with manageIU command 98
- resulting resources
 - as backing resource 39
 - definition 38
 - XML sample 40
- return codes, retrieving 84
- root configuration units
 - definition 28
 - XML sample 29
- root CUs
 - See root configuration units
- root installable units
 - definition 18
 - XML sample 20
- root IUs
 - See root installable units
- root user
 - and Deployment Engine in multiuser mode 70
 - definition 81

S

- selectedFeatures variable
 - definition 68
- SL_PATH environment variable 73
- single-user mode
 - and de_version command 81
 - definition 71
- smallest configuration units
 - definition 29
 - XML sample 31
- smallest CUs
 - See
- smallest installable units
 - definition 20
 - XML sample 21, 22
- smallest IUs
 - See smallest installable units
- software dependencies 56
- software deployment
 - methods 11
 - models 10
- software deployment models
 - fully enabled software deployment 10
 - implementing Deployment Engine
 - fully enabled software deployment 10
 - wrapped software deployment 10
- software deployment programs
 - and bootstrap program 8
 - hybrid software deployment program 12
 - in Deployment Engine operating environment 8

- software life cycle
 - definition 51
 - supported states 52
- software package set
 - definition 12
- software package tree
 - and requisites 40
 - definition 11
- software package types
 - base software package 12
 - definition 12
 - fix software package 13
 - full update software package 12
 - incremental update software package 13
- software packages
 - base software package 12
 - definition 3, 14
 - fix software package 13
 - fresh installation 12
 - full update software package 12
 - in Deployment Engine operating environment 7
 - incremental update software package 13
 - testing with manageIU command 86
 - that include fix packs and refresh packs 13
 - that include interim fixes and test fixes 13
 - that include manufacturing refreshes 12
 - types 12
- software states
 - See* life cycle states
- solution modules
 - definition 25
 - XML sample 27
- specifying a software instance uniquely 83
- states
 - See also* life cycle states
 - and software life cycle 51
- subordinate units
 - definition 17
- Supersedes relationship 62

T

- targets
 - definition 38
- terminology, introductory 3, 5
- test fix
 - in fix software package 13
- topologies
 - definition 38
- touchpoints
 - in Deployment Engine operating environment 9
- trace levels 117
- trace loggers
 - AP logger 117
 - CM logger 117
 - DC logger 117
 - OSTP logger 117
 - PA logger 117
 - trace log fields 118
- trace logging
 - description 117
 - setting the trace level 117
 - trace loggers 117
- troubleshooting 119
- typeface conventions vi

U

- Uninstall action descriptors
 - definition 47
- Updated state
 - definition 52
- Usable state
 - definition 52
- user modes
 - multiuser mode 70
 - single-user mode 71
- Uses relationship 63

V

- validateIUDD command 101
- variables
 - _discriminant variable 66
 - boolean variables 68
 - definition 64
 - derived variable 64
 - environment variables 73
 - ERRORLEVEL environment variable 84
 - internal variables 66
 - parameter variable 64, 93
 - query IU discriminant variable 65
 - query property variable 65
 - selectedFeatures variable 68
 - variable types 64

W

- Web service interface
 - in Deployment Engine operating environment 9
- WebSphere touchpoint
 - in Deployment Engine operating environment 9
- wrapped software deployment
 - definition 10



Printed in USA