

Deployit Reference Manual

Version 3.6.1

Table of Content

Preface	3
Deployment Overview	3
Interacting with Deployit	5
Deployit Glossary	5
Artifacts	5
Command Line Interface (CLI)	6
Control Tasks	6
Configuration Items (CIs)	6
Containers	6
Deployables	6
Deployeds	6
Deploying an Application	7
Deployment Archive (DAR) Format	7
Deployment Package	7
Dictionaries	7
Environment	7
Placeholders	7
Plugin	8
Repository	8
Containment and References	8
Deployed Applications	8
Resource Specifications	8
Security	9
Step	9
Steplist	9
Tag-based Deployments	9
Task	9
Task Recovery	10
Task State	10
Type System	10
Undeploying an Application	10
Unified Deployment Model (UDM)	10
Upgrading an Application	10

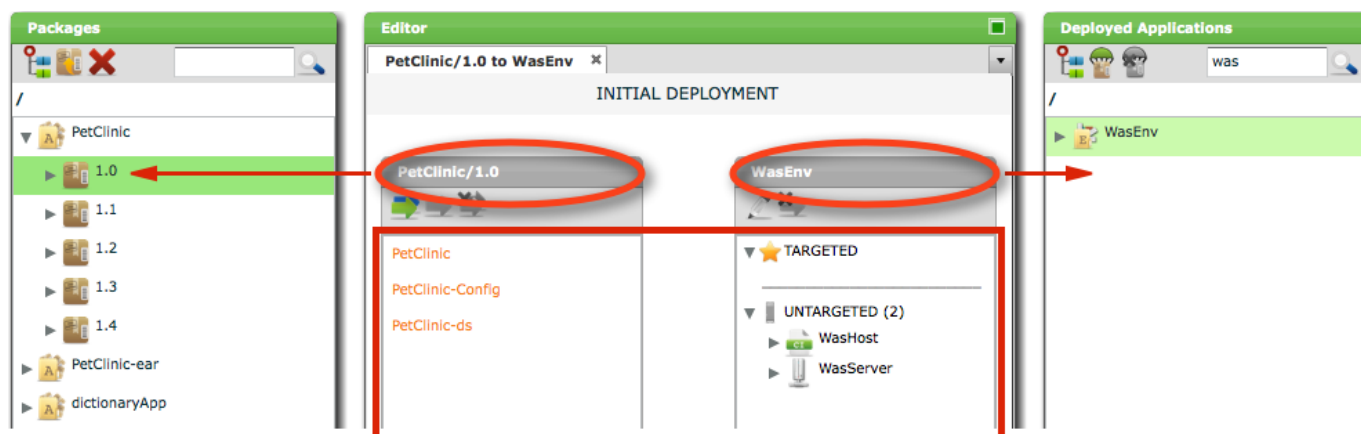
Preface

This manual contains reference information for users of Deployit.

Deployment Overview

Deployit is the first out-of-the-box deployment automation solution that allows non-experts to perform application deployments. A deployment consists of all the actions needed to install, configure and start an application on a target environment.

At a high level, deployments in Deployit are modeled using the *Unified Deployment Model (UDM)*. The following diagram depicts the main concepts in the UDM as they appear in the Deployit GUI:



Deployments are defined by:

- A **Package** containing *what* is to be deployed (shown on the left side of the picture).
- An **Environment** defining *where* the package is to be deployed (shown on the right side of the picture).
- Configuration of the **Deployment** specifying (possibly environment-specific) *customizations* to the package to be deployed (the middle square).

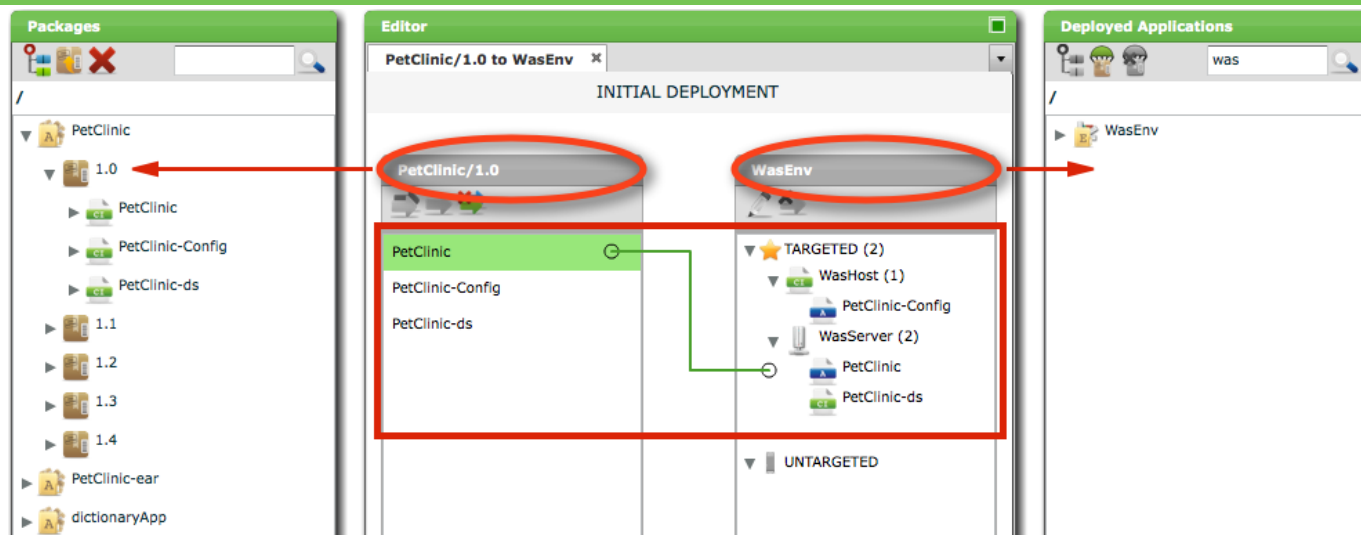
Of course, packages and environments are made up of smaller parts:

- Packages consist of *deployables* or *things that can be deployed*.
- Environments consist of *containers* or *things that can be deployed to*.

Deployables come in two flavors: *artifacts* are physical files (examples are an EAR file, a WAR file or a folder of static HTML) and *resource specifications* are middleware resources that the application needs to run (examples are a queue, a topic or a datasource). These types of deployables are put together in a package.

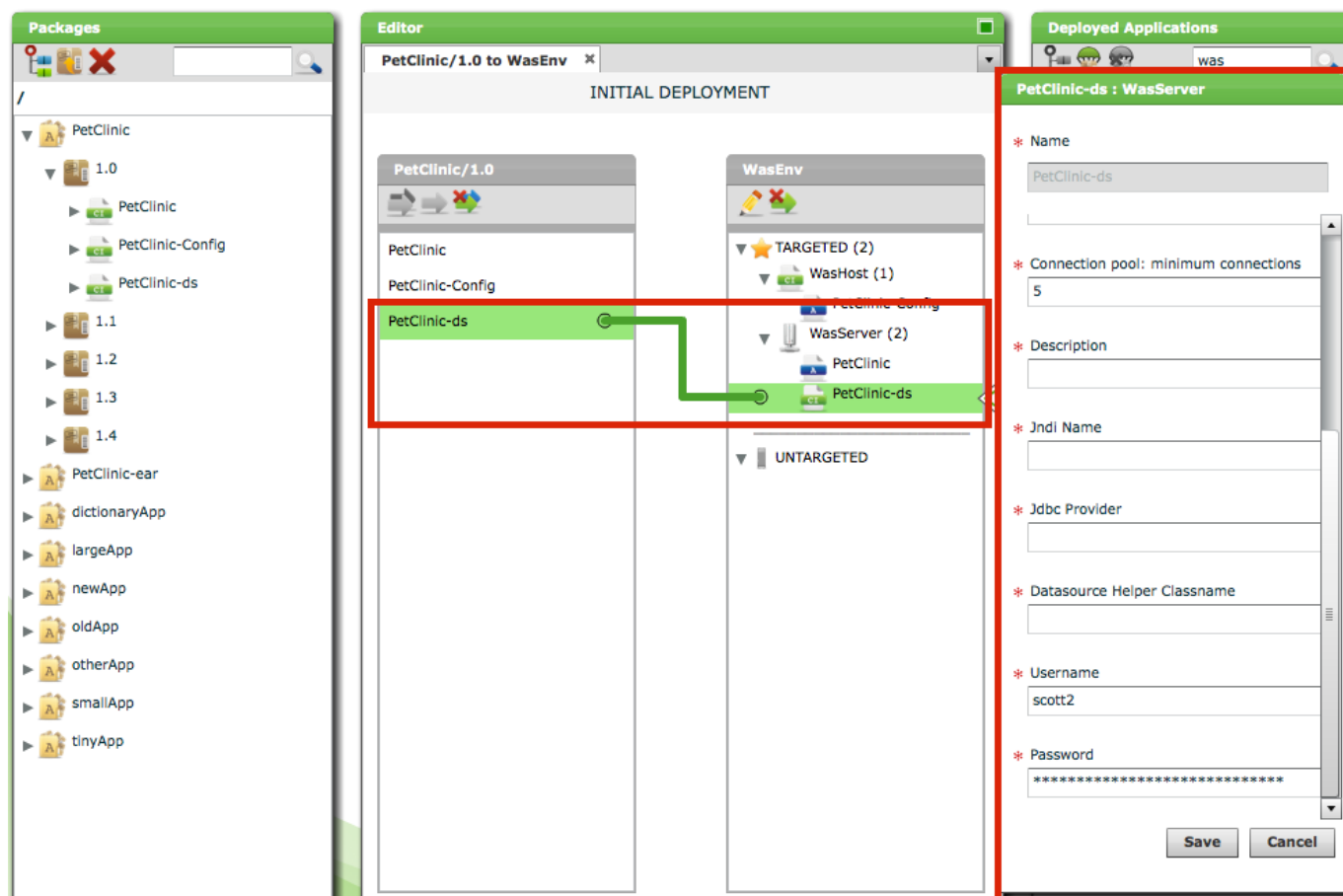
Containers are actual middleware that deployables can be deployed to. Examples of containers are an application server such as Tomcat or WebSphere, a WebSphere node or cell, a host or a database server.

Let's say we have a package that consists of an EAR file (an artifact), a datasource (a resource specification) and some configuration files (artifacts) and we want to deploy this package to an environment containing an application server and a host (both containers). The exact deployment could look something like this:



In the above picture, you can see that the EAR file and the datasource are deployed to the application server and the configuration files are deployed to the host.

As you can see above, the deployment also consists of smaller parts. The combination of a particular deployable and container is called a *deployed*. Deployeds represent *the deployable on the container* and contain customizations for this specific deployable - container combination. For example, the *PetClinic-ds* deployed in the picture represents the datasource from the deployment package as it will be deployed to the *was.Server* container. The deployed allows a number of properties to be specified:



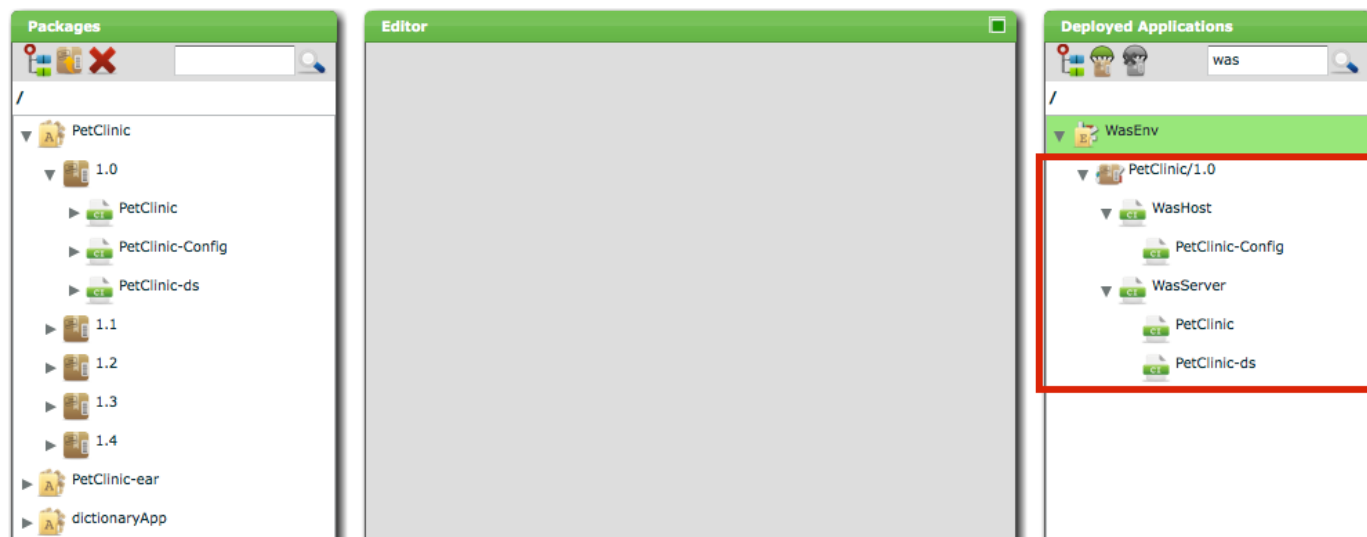
For example, the deployed has a specific username and password that may be different when deploying the same datasource to another server.

Once a deployment is specified and configured using the concepts above (and the *what*, *where* and *customizations* are known), Deployit takes care of the *how* by preparing a list of steps that need to be executed to perform the actual deployment. Each step specifies one action to be taken, such as copying of a file, modifying a configuration file or restarting a server.

When the deployment is started, Deployit creates a *task* to perform the deployment. The task is an independent process running on the Deployit server. The steps are executed sequentially and the deployment is finished successfully when all steps have been executed. If an error occurs during deployment, the deployment stops and an operator is required to intervene.

The result of the deployment is stored in Deployit as a *deployed application* and is shown in the Deployed Application Browser on the right hand side of the GUI. The deployed applications are organised by environment so it is clear where the application is deployed to. It is also possible to see which parts of the deployed package have been deployed to which environment member.

The final result of our sample deployment looks like this:



Interacting with Deployit

There are two ways of interacting with Deployit and each is intended for a specific audience.

Performing tasks interactively is possible using the **Graphical User Interface (GUI)**. The GUI is a Flash application running inside a user's internet browser. After logging in, the user can configure and perform deployments, view and edit the repository and view reports (provided the user's permissions allow him to perform these tasks). In general, tasks that require graphical output or benefit from a graphical overview are performed in the GUI. See the **Deployit Graphical User Interface (GUI)** manual for more information.

The **Command Line Interface (CLI)** is used to automate Deployit tasks. The CLI is a Jython application that the user can access remotely. In addition to configuring and performing deployments, the user can also setup middleware and security permissions in the CLI. In general, the CLI is used to perform administrative tasks or automate deployment tasks. See the **Deployit Command Line Interface (CLI)** manual for more information.

Please note that Deployit's security system is applied to all user actions, regardless of whether they are performed in the GUI or from the CLI.

Deployit Glossary

These are the main concepts in Deployit, in alphabetical order.

Artifacts

Artifacts are files containing application resources such as code or images. The following are examples of artifacts:

- a WAR file.
- an EAR file.
- a folder containing static content such as HTML pages or images.

Command Line Interface (CLI)

The Deployit CLI provides a way to programmatically interact with Deployit. The CLI can be programmed using the **Python** programming language. For more detailed information, see the **Deployit Command Line Interface Manual**.

Control Tasks

Control tasks are actions that can be performed on middleware or middleware resources.

Control tasks are defined on a particular CI type and can be executed on a specific instance of that type. When a control task is invoked, Deployit starts a task that executes the steps associated with the control task. Deployit users can use them to interact directly with the underlying middleware. An example of a control task is starting or stopping of an Apache webserver.

Control tasks can be defined in Java or synthetically (see the **Customization Manual**) or using scripts (see the **Generic Model Plugin Manual**).

Configuration Items (CIs)

A *configuration item* (CI) is a generic term that describes all objects that Deployit keeps track of. Applications, middleware, environments and deployments in Deployit are all represented in Deployit as CIs. A CI has a certain *type* that determines what information it contains and what it can be used for. CIs have properties to describe it and may have relations to other CIs.

Deployit CIs all share one property:

- **id**. This is a unique identifier for this CI that can be used to reference it. The id determines the place of the CI in the *repository*.

Some properties of a CI are mandatory (must be given a value before the CI can be used or stored), others are optional. To determine which properties are available and which are mandatory or optional for a CI, see the **CI Reference** section in this document (or the plugin manual for the plugin that the CI is a part of) or use the help facility in the Deployit CLI.

For example, a CI of type *udm.DeploymentPackage* represents a *deployment package*. It has properties containing its version number. It contains child CIs for the *artifacts* and *resource specifications* it contains and has a link to a parent CI of type *udm.Application* which describes which application the package is a part of.

Containers

Containers are configuration items (CIs) that *deployable* CIs can be deployed to. Containers are grouped together in an *environment*. Examples of containers are a host, WebSphere server or WebLogic cluster.

Deployables

Deployables are configuration items (CIs) that can be deployed to a *container* CI. Deployables are part of a *deployment package*. Deployables come in two forms: *artifacts* (for instance, EAR files) and *specifications* (for instance, a datasource).

Deployeds

Deployeds are configuration items (CIs) that represent *deployable* CIs as it is deployed on a target *container* CI. The deployed CI specifies settings that are relevant for the CI on the container.

For example, a *wls.Ear* deployable is deployed to a *wls.Server* container, resulting in a *wls.EarModule* deployed.

Another example is a *wls.DataSourceSpec* that is deployed to a *wls.Server* container, resulting in a *wls.DataSource* deployed. The *wls.DataSource* is configured with the database username and password that are required to connect to the database from this particular server.

Deployeds go through the following life-cycle:

- the deployed is created on a target container for the first time in an *initial* deployment
- the deployed is upgraded to a new version in an *upgrade* deployment
- the deployed is removed from the target container when it is *undeployed*.

Deploying an Application

This process installs a particular application version (represented by a *deployment package*) on an environment. Deployit copies all necessary files and makes all configuration changes to the target middleware that are necessary for the application to run.

Deployment Archive (DAR) Format

The DAR format is the native format Deployit supports for *deployment packages*. A DAR file is a standard ZIP file with additional metadata information contained in a manifest. For a comprehensive description of the DAR format, see the **Deployit Packaging Manual**.

Deployment Package

In the **Unified Deployment Model**, a particular version of an application (consisting of both artifacts and resource specifications) is contained in a single *deployment package*. The package contains all *deployables* that the application needs. The package is environment-independent and can be deployed to any environment unchanged (see the **Unified Deployment Model**).

Deployit accepts packages in the *Deployment ARchive (DAR)* format.

Dictionaries

A dictionary is a CI that contains environment-specific entries for placeholder resolution. Entries can be added in the GUI or using the CLI. This allows the deployment package to remain environment independent so it can be deployed unchanged to multiple environments.

For an example of using placeholders in CI properties, see the **Deployit Packaging Manual**.

Environment

An *environment* is a grouping of infrastructure items, such as hosts, servers, clusters, etc. Environments can contain any combination of infrastructure items that are used in your situation. An environment is used as the target of a deployment, allowing *deployables* to be mapped to members of the environment.

Placeholders

Placeholders are configurable entries in your application that will be set to an actual value at deployment time. This allows the deployment package to be environment-independent and thus reusable.

There are two types of placeholders. *File* placeholders occur inside of artifacts in the deployment package. Deployit scans packages that it imports for text files and searches these text files for file placeholders. The following items are scanned:

- File-type CIs
- Folder-type CIs
- Archive-type CIs

Property placeholders are used in CI properties by specifying them in the package's manifest.

Deployit recognizes placeholders using the following format:

```
{{ PLACEHOLDER_KEY }}
```

Values for placeholders can be provided manually or filled in from a *dictionary*.

Plugin

A *plugin* is a self-contained piece of functionality that adds capabilities to the Deployit system. Plugins are packaged in a JAR file and installed in Deployit's plugins directory. Plugins can contain:

- functionality to connect to specific middleware
- a new method to connect to a host
- a new importer

Repository

The *repository* is a storage space for all things Deployit knows about. This includes configuration items (CIs), binary files (such as *deployment packages*) and Deployit's security configuration (user accounts and rights). The repository can be stored on disk (default) or in a database (see **Configuring Database Storage** in the **System Administration Manual**).

Each CI in Deployit has an id that uniquely identifies the CI. This id is a path that determines the place of the CI in the repository. For instance, a CI with id "Applications/PetClinic/1.0" will appear in the **PetClinic** subfolder under the **/Applications** root folder.

The repository has a hierarchical layout and a version history. All CIs of all types are stored here. The top-level folders indicate the type of CI stored below it. Depending on the type of CI, the repository stores it under a particular folder:

- Application CIs are stored under the **/Applications** folder.
- Environment and Dictionary CIs are stored under the **/Environments** folder.
- Middleware CIs (representing hosts, servers, etc.) are stored under the **/Infrastructure** folder.

Containment and References

Deployit's repository contains CIs that refer to other CIs. There are two ways in which CIs can refer to each other:

- **Containment.** In this case, one CI *contains* another CI. If the parent CI is removed, so is the child CI. An example of this type of reference is an Environment CI and it's deployed applications.
- **Reference.** In this case, one CI *refers* to another CI. If the referring CI is removed, the referred CI is unchanged. Removing a CI when it is still being referred to is not allowed. An example of this type of reference is an Environment CI and it's middleware. The middleware exists in the **/Infrastructure** folder independently of the environments the middleware is in.

Deployed Applications

A deployed application is the result of deploying a *deployment package* to an *environment*. Deployed applications have a special structure in the repository. While performing the deployment, package members are installed as *deployed items* on individual environment members. In the repository, the *deployed application* CI is stored under the *Environment* node. Each of the *deployed items* are stored under the infrastructure members in the *Infrastructure* node.

So, deployed applications exist in both the **/Environment** as well as **/Infrastructure** folder. This has some consequences for the security setup. See the **Deployit System Administration Manual** for details.

Resource Specifications

Resource specifications are specifications of middleware resources that an application needs to run. The following are examples of these resources:

- a datasource.

- a queue or topic.
- a connection factory.

Security

Deployit supports a fine-grained access control scheme to ensure the security of your middleware and deployments. The security mechanism is based on the concepts of *principals* and *permissions*. A (security) principal is an entity that can be authenticated and that can be assigned rights over resources in Deployit. Principals are assigned certain permissions within the system. These permissions can be either global (that is, they apply to all of Deployit, such as login permission) or relevant for a particular CI or set of CIs (for instance, the permission to read certain CIs in the repository).

The security system uses the same permissions whether the system is accessed via the GUI or the CLI.

For more information about Deployit's security system, see the **System Administration Manual**.

Step

A step is a concrete action to be performed to accomplish a *task*. Steps are contributed by *plug-ins* based on the deployment that is being performed. All steps for a particular deployment are grouped together in a *steplist*. Deployit ships with many step implementations for common actions. Other, middleware-specific steps are contributed by the plugins.

The following are examples of steps:

- copy file /foo/bar to host1, directory /bar/baz.
- install petclinic.ear on the WAS server on was1.
- restart the Apache HTTP server on web1.

Steplist

A steplist is a sequential list of *steps* that are contributed by one or more *plugins* when a deployment is being planned.

Tag-based Deployments

Tag-based deployments make it easier to configure initial deployments by marking which deployables should be mapped to which containers. Deployit can automatically generate the required deployments during deployment configuration.

To perform a deployment using tags, specify tags on the deployables (either in the imported package or by using the repository browser) and containers. When deploying the package to the environment containing the tagged containers, let Deployit automatically generate the deployments. Deployit will map deployables to containers that have at least one tag in common.

An example scenario is deploying a front-end and back-end application to two application servers. By tagging the front-end EAR and front-end application server both with 'FE' and the back-end EAR and back-end server with 'BE', Deployit will automatically create the correct deployments.

Task

A task is an activity in Deployit. When starting a deployment, Deployit will create and start a task. The task contains a list of *steps* that must be executed to successfully complete the task. Deployit will execute each of the steps in turn. When all of the steps are successfully executed, the task itself is successfully executed. If one of the steps fails, the task itself is marked stopped.

Deployit supports the following tasks:

- **deploy application**. This task deploys a package onto an environment.

- **update application.** This task updates an existing deployment of an application.
- **undeploy application.** This task undeploys a package from an environment.
- **discovery.** This task discovers middleware on a host.

Task Recovery

Deployit periodically stores a snapshot of the tasks in the system to be able to recover tasks if the server crashes. If this happens, Deployit reloads the tasks from the recovery file when it restarts. The tasks, deployed item configurations and generated steps will all be recovered. Tasks that were running in Deployit when the server crashed will be put in stopped state so the user can decide whether to rerun or cancel it. Only tasks that have been started will be recovered.

Task State

Deployit allows a user to interact with the task. In addition to starting a task, a user can:

- **stop the task.** Deployit will wait for the currently executing step to finish and will then cleanly stop the task. Note that, due to the nature of some steps, this is not always possible. For example, a step that calls an external script may hang indefinitely.
- **abort the task.** Deployit will attempt to kill the currently executing step. If successful, the aborted step and task are marked failed.
- **cancel the task.** Deployit will remove the task from the system. If the task was executing before, the task will be archived since it may have made changes to the middleware. If the task was pending and never started, it will be removed but not stored.

Type System

Deployit features a configurable type system that allows modification and addition of CI types. This makes it possible to extend your installation of Deployit with new types or change existing types. Types defined in this manner are referred to as *synthetic* types. The type system is configured using XML files called *synthetics.xml*. All files containing synthetics are read when the Deployit server starts and are available in the system afterwards.

Synthetic types are first-class citizens in Deployit and can be used in the same way that the built-in types are used. This means they can be included in deployment packages, used to specify your middleware topology and used to define and execute deployments. Synthetic types can also be edited in the Deployit GUI, including new types and added properties.

For more information about extending Deployit, see the **Customization Manual**.

Undeploying an Application

This process removes a *deployed application* from an *environment*. Deployit stops the application and undeploys all its components from the target middleware.

Unified Deployment Model (UDM)

The UDM is XebiaLabs' model for describing deployments and is used in Deployit. The UDM consists of the following components:

- **Deployment Package.** This is an environment-independent package containing *deployables*, CIs that together form a complete application.
- **Environment.** This is an environment containing *containers* (deployment targets, i.e. servers that applications can be deployed on). An example is a test environment containing a cluster of WebSphere servers.
- **Deployment.** This is the process of configuring and installing a *deployment package* onto a specific *environment*. It results in *deployeds* describing the coupling of a *deployable* and a *container*.

Upgrading an Application

This process replaces an application deployed to an *environment* with another version of the same application. When performing an upgrade, most *deployeds* can be inherited from the initial deployment. Deployit recognizes which artifacts in the deployment package have changed and deploys only the changed artifacts.