

# Deployit Packaging Manual

January, 2011

## Contents

<b>Preface</b>	<b>1</b>
<b>Introduction</b>	<b>1</b>
<b>Packages</b>	<b>1</b>
<b>DAR Format</b>	<b>1</b>
<b>Manifest Format</b>	<b>2</b>
Specifying the Application . . . . .	2
Specifying Artifacts . . . . .	2
Specifying Middleware Resources . . . . .	3
<b>Using Placeholders</b>	<b>3</b>
<b>Making a deployment package</b>	<b>4</b>
<b>Maven Plugin</b>	<b>4</b>

## Preface

This manual describes how to package applications for use in Deployit.

See **Deployit Reference Manual** for background information on Deployit and deployment concepts.

## Introduction

The Deployit deployment automation tool is designed to help you deploy application packages to target middleware. To make it possible to deploy your applications, they must be packaged in a format that Deployit understands. This manual describes the Deployment ARchive (DAR) format and various other topics related to packaging applications.

## Packages

Deployit uses the Unified Deployment Model (UDM) to structure its deployments (see the **Deployit Reference Manual**). In this model, packages are containers to distribute complete applications, including both the application artifacts as well as the middleware resources (datasources, topics, queues) that the application needs to run. When a package is deployed to an environment, you can be sure that everything is in place to run the application successfully.

Also, packages are promoted **unchanged** from development to test to acceptance and finally to production. All environment- or target-dependent configuration is removed from the artifacts and resources and is instead specified when describing how to deploy the package to a specific environment or target. This enables a single artifact to make the entire journey from development to production.

## DAR Format

Out of the box, Deployit supports its own Deployment ARchive (DAR) format for packages. A valid DAR package consists of the following components:

1. A Java JAR formatted file with the extension `.dar`.
2. A manifest file containing a description of the contents of the package.

Valid DAR archives can be produced using standard command line tools, such as the Java `jar` utility or the Maven `package` plugin. Deployit also comes with its own Maven plugin to facilitate packaging. See the section **Using the Maven plugin** below.

## Manifest Format

The manifest file included in a DAR describes the contents of the archive for Deployit. When importing a package, the manifest is used to construct CIs in Deployit's repository based on the contents of the imported package. For background information on the JAR format and manifest, see the JAR file specification.

A valid Deployit manifest starts with the following preamble:

```
Manifest-Version: 1.0
Deployit-Package-Format-Version: 1.2
```

This identifies the Java manifest version and the Deployit package format version.

## Specifying the Application

A deployment package contains a specific version of an application. These entries tell Deployit that the package contains the *AnimalZoo* application version *4.0*:

```
CI-Application: AnimalZoo-ear
CI-Version: 4.0
```

These entries are part of the manifest preamble.

## Specifying Artifacts

Artifacts are represented by files or folders in the deployment package. To import an artifact as a CI in Deployit, use:

```
Name: AnimalZooBE-1.0.ear
CI-Type: Ear
CI-Name: AnimalZooBE
```

The standard manifest **Name** entry must refer to an actual file included in the DAR. The **CI-Type** is the shortname of a CI type that exists in Deployit. This type can be part of standard Deployit or it can be provided by one of the plugins. The **CI-Name** property indicates the name of the CI to be created.

If the **CI-Name** entry is missing, Deployit will create a CI with a name based on the filename (in this case, *AnimalZooBE-1.0.ear*, stripping out the extension).

Similarly, this construct can be used to create a folder CI:

```
Name: sql
CI-Type: SqlFolder
CI-Name: DatabaseScripts
```

It is not possible to set properties on the artifact CI using the manifest.

## Specifying Middleware Resources

Middleware resources are not specified by files in the deployment package. They represent resources that must be created on the target middleware when the application is deployed. Middleware resources are constructed completely based on the information in the manifest. The manifest also specifies values for properties of the CI.

For example, this manifest snippet constructs a *Datasource* CI:

```
Name: petclinicDS
CI-Type: DataSource
CI-driver: com.mysql.jdbc.Driver
CI-url: jdbc:mysql://localhost/petclinic
CI-username: petclinic
CI-password: my$ecret
```

The **Name** entry specifies the name of the CI to be created <sup>1</sup>. The **CI-Type** is the shortname of a CI type that exists in Deployit. This type can be part of standard Deployit or it can be provided by one of the plugins.

**Note:** the names of artifacts in your package must conform to platform requirements. For instance, deploying an *SqlFolder* CI with name “q2>2” cannot be deployed to a Windows host, because “>” may not be part of a file or directory name in Windows.

The other entries, **url**, **username** and **password** refer to properties on the new CI. These properties will get the values specified.

It is also possible to specify a map using the following syntax:

---

<sup>1</sup>In contrast to the manifest specification, the **Name** property in a Deployit manifest does not refer to a physical file present in the DAR in the case of a middleware resource.

```
CI-settings-EntryKey-1: autoCommit
CI-settings-EntryValue-1: true
```

This populates the `settings` map with the name-value pair `autoCommit=true`.

Note that it is also possible to add middleware resources to a package that is already imported in Deployit. See the **Command Line Interface (CLI) Manual** for more information.

## Using Placeholders

When importing a package, certain files in the package are scanned for *placeholders*. Placeholders are configurable entries in your application that will receive an actual value at deployment time. This allows the deployment package to be environment-independent and thus reusable. The value can be dependent on the deployment environment or specific mapping.

For example, if your application uses a properties file to configure logging and needs a different log directory in your development versus your test environment, you could specify the logging directory (using a fictitious syntax) as follows:

```
directory=${LOG_DIRECTORY}
```

When importing the CI that contains the logging configuration file, Deployit will scan file and record the placeholder `LOG_DIRECTORY`. When deploying the deployment package to an environment, you will have to specify which value to use for this placeholder. The file containing the placeholder is then modified as it is deployed to the environment. If you are deploying the CI to multiple targets within the environment, you can also vary the log directory per target.

Deployit scans for and replaces placeholders in files with the following extensions:

- .properties
- .sql
- .xml
- .bat
- .cmd
- .sh
- .txt

The files may be listed as separate artifacts in your manifest or may be part of a folder CI. Binary files (EAR, WAR, JAR, ZIP) are not scanned for files containing placeholders.

## Making a deployment package

Creating a deployment package can be done by hand. If you are using a Unix system and have a directory `myApplication` containing the files your application needs, follow these steps:

- Manually create a `MANIFEST.MF` file in a text editor, specifying the CIs your package will contain. Store it outside of the `myApplication` directory.
- Issue the following commands:

```
cd myApplication
jar cmf ../myApplication.dar ../MANIFEST.MF *
cd ..
```

This results in a deployment archive called `myApplication.dar` in the current directory.

## Maven Plugin

To enable continuous deployment, Deployit can be integrated with the Maven build system. A maven plugin exists that exposes some of Deployit's functionality via maven. Specifically, the plugin supports:

- creating a deployment package containing artifacts from the build
- performing a deployment to a target environment
- undeploying a previously deployed application

For more information, see the Deployit maven plugin documentation.