Deploying J2EE Applications on SAP NetWeaver

SAP Platform Ecosystem

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help → General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options.
	Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Table of Contents

Introduction

Different projects require different deployment scenarios. SAP Web AS 6.40 provides several ways of generating, assembling, and deploying J2EE applications and application components to the J2EE Engine. This guide briefly introduces the basic tools and techniques to help you decide which of them may fit your needs.

Many companies already have established efficient deployment environments using Apache Ant. Particularly with regard to an, for some cases, slightly oversized JDI there is no urge for these companies to radically change their process.

Hence we will have a look at less documented ways of generating, assembling and deploying J2EE applications and application components in automated deployment environments using Apache Ant.

 *There is no official support for the introduced approaches as to the deployment will be extensively refactored in SAP Web AS releases higher than 6.40. ANT Task for deployment will then be shipped right away and J2EE 1.4 compliancy will support the use of third party deployment tools as postulated in the J2EE Application Deployment Specification (JSR 88).*

Objectives

By the end of this reading, you will:

- Know which deployment techniques and tools are provided by the SAP Web AS 6.40.
- Be able to decide which tools and techniques are best suited for your requirements.
- Be able to integrate SAP Web AS 6.40 deployment functionality with your Apache Ant driven build and deployment process.

Tools and Techniques

There are several ways to deploy your software on the J2EE Engine. Depending on your requirements, SAP Web AS 6.40 supports various deployment scenarios and furthermore assists you with tools providing generation and assembly support.

Software Deployment Manager

If you develop with the SAP NetWeaver Developer Studio, you already have been using the SDM. The SAP NetWeaver Developer Studio calls the SDM directly to deploy your applications to your local system. With the SDM you can also manage and deploy Software Deployment Archives (SDAs) and Software Component Archives (SCAs). For more information on the SDM refer to the official SAP online documentation (http://help.sap.com): Software Deployment Manager.

Deploy Tool

If you do not use the SAP NetWeaver Developer Studio for development but want a powerful and GUI based tool to generate, assemble and deploy J2EE applications and application components, the Deploy Tool is what you are looking for. It allows you to generate J2EE archive components from existing class files and deployment descriptors and to assemble them in an application EAR. With the Deploy Tool you can edit the deployment properties of an EAR or an archive component and deploy it on the specified J2EE Engine. See Deploy Tool on SAP http://help.sap.com for details.

Script Files

A more flexible way of setting up a customized assembly and deployment scenario is to use script files. The DeployManager API allows you to pass XML-descriptor files directly. Thus by simply launching a script file you are able to generate J2EE components, assemble and deploy a J2EE application. Especially in large scale projects where you have to create, assemble and deploy your applications numerous times the possibility of using script files comes in handy. The SAP Web AS 6.40 already ships with these script files. The DTDs for the specific XML-descriptor files are available on SAP Help. See Script Files to learn more. See Deployment: Putting It All Together on http://help.sap.com for more detailed information.

Telnet

In remote deployment scenarios you can use telnet to generate, assemble and deploy J2EE applications and application components. See the example for using Telnet via an Apache Ant target in the J2EE Migration Guide.

Example 1 - Integrating SAP Web AS 6.40 Deployment with Apache Ant

In J2EE development automated Apache Ant based build and deployment solutions are widespread. Developing on SAP web AS 6.40 you do not necessarily have to miss lightweight and flexible deployment using Apache Ant.

To keep it simple, the following two examples will focus on the deployment only. The other functionalities like generating and assembling components and applications work the same way and after having finished this reading, you will be able to easily implement them on your own.

Prerequisites

Systems, Installed Applications, and Authorizations

- SAP WebAS – Java 6.40
- Apache Ant available on your system
- An Ear file – The Application you want to deploy

Combining the `DeployManager` API with Apache Ant provides a rapidly implemented, automation-ready deployment solution. The `DeployManager` API is the Deploy Tools underlying API the script file solution mentioned above is also using. All the deployment parameters you usually would define via the Deploy Tool GUI are passed by a XML file.

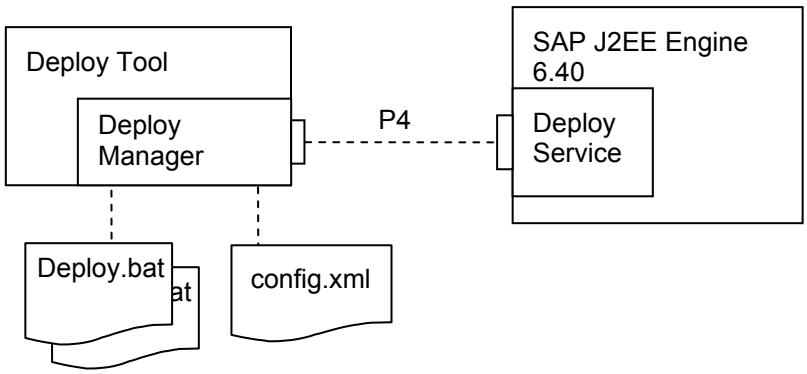

Figure 1: Deploy Manager and Script Files

Since with this approach you have the full functionality of the Deploy Tool, it enables you not only to deploy an application Ear file to SAP Web AS 6.40 but also to generate J2EE components and to assemble your J2EE applications.

So the idea is simple: why not changing the script files to Ant targets (Figure 2):

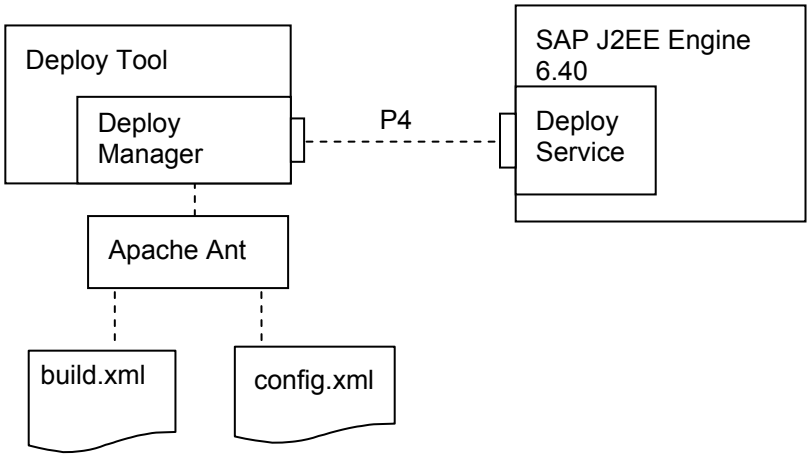

Figure 2: Ant Integration

Procedure

Defining Deployment Parameters in a XML File

First of all you have to define all application and environment specific parameters in an XML file (e.g. `deploy-manager-config.xml`). This file will be passed to the `DeployManager`.

In our example we only want to deploy a single Ear file. For now you can use the following slightly shortened example file. As you might already have seen, the DTD postulates some more elements. For more information on each element see deploy-manager-config.dtd in the Reference Manual.

So simply fill in your parameters and proceed writing the build.xml.

deploy-manager-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE deploy-manager-config [
<!ELEMENT deploy-manager-config (version-id, project-dir?, deployable-object,
element*, login-info, user-role-mapping?, server-components-xml?, library-info?,
supports, deployment-properties?, repl-var-properties?, log-file?)>
<!ELEMENT version-id (#PCDATA)>
<!ELEMENT project-dir (#PCDATA)>
<!ELEMENT deployable-object (ear-file|j2ee-module)>
<!ATTLIST deployable-object action-type (deploy|update) #REQUIRED>
<!ELEMENT ear-file (ear-path, display-name?, altdd*)>
<!ELEMENT ear-path (#PCDATA)>
<!ELEMENT display-name (#PCDATA)>
<!ELEMENT j2ee-module (archive-path, altdd*)>
<!ELEMENT archive-path (#PCDATA)>
<!ATTLIST j2ee-module container (appclient|connector|EJB_Container|servlet_jsp)
```

```
#REQUIRED>
<!ELEMENT element (entry-name, context-root?, altdd*)>
<!ELEMENT context-root (#PCDATA)>
<!ELEMENT altdd (pathname, entry-name)>
<!ATTLIST altdd use-alternative (yes|no) #REQUIRED>
<!ELEMENT pathname (#PCDATA)>
<!ELEMENT entry-name (#PCDATA)>
<!ELEMENT login-info (host, port, transport-protocol*, user-name, user-password)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ELEMENT transport-protocol (#PCDATA)>
<!ELEMENT user-name (#PCDATA)>
<!ELEMENT user-password (#PCDATA|psw-file)*>
<!ELEMENT psw-file (#PCDATA)>
<!ELEMENT user-role-mapping (role-name+)>
<!ELEMENT role-name (#PCDATA | mapping)*>
<!ELEMENT mapping (name, type)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT server-components-xml (#PCDATA)>
<!ELEMENT library-info (library*, reference*)>
<!ELEMENT library (lib-name, lib-jar+)>
<!ELEMENT lib-name (#PCDATA)>
<!ELEMENT lib-jar (#PCDATA)>
<!ATTLIST lib-jar server-map-name CDATA #REQUIRED>
<!ELEMENT reference (from-loader, to-loader+)>
<!ELEMENT from-loader (#PCDATA)>
<!ELEMENT to-loader (#PCDATA)>
<!ELEMENT supports (support+)>
<!ELEMENT support (#PCDATA)>
<!ELEMENT repl-var-properties (property*)>
<!ELEMENT deployment-properties (property*)>
<!ELEMENT property (#PCDATA)>
<!ELEMENT log-file (#PCDATA)>
]>

  <deploy-manager-config>
    <version-id>
    </version-id>
    <project-dir>
       <Path_to_your_project_dir>
    </project-dir>
    <deployable-object action-type="deploy">
      <ear-file>
```

```
        <ear-path>
          <your_Ear_name>.ear
        </ear-path>
      </ear-file>
    </deployable-object>
    <login-info>
      <host>
        <hostname>
      </host>
      <port>
        50004
      </port>
      <transport-protocol>
        None
      </transport-protocol>
      <user-name>
        Administrator
      </user-name>
      <user-password>
        <your_password>
      </user-password>
    </login-info>
    <user-role-mapping>
      <role-name>
        unknown role
      </role-name>
    </user-role-mapping>
    <library-info>
    </library-info>
    <supports>
      <support>
        p4
      </support>
    </supports>
    <deployment-properties>
      <property>
      </property>
    </deployment-properties>
    <repl-var-properties>
    </repl-var-properties>
    <log-file>
      deployer_log.txt
    </log-file>
  </deploy-manager-config>
```

Writing the build.xml

Finally you write the `build.xml` with the corresponding Apache Ant Target.

Create properties for all absolute paths:

Define the absolute Path to your `deploy-manager-config.xml`. The two other paths defined in the properties `deploy.base.dir` and `deploy.lib.dir` point to the `deploying` folder, respectively its `lib` subfolder, which is shipped with your SAP Web AS 6.40. The `deploying` folder contains all the required libraries for generation, assembly and deployment of J2EE applications and components.

Create target “deploy”:

The target runs `com.sap.engine.offline.OfflineToolStart` out of `launcher.jar` which has to be defined in the classpath element. The following arguments are passed as start parameters via the `arg` element:

`com.sap.engine.deploy.manager.DeployManagerImpl` - Name of the executing class of the DeployManager API

`;.\lib` – Locates the `lib` folder

`${deploy.manager.config}` – Points to `deploy-manager-config.xml` when expanded.


Note, that you have to keep the correct order of these parameters!


build.xml

```
<project name="Ant Deployment" default="deploy" basedir=".">
   <description>
      Deploy J2EE Applications to SAP Web AS
   </description>
   <!--
      (1)Properties
      Define absolute paths to deploying dir and
      deploy-manager-config.xml as properties
   -->
   <property
      name="deploy.base.dir"
      location="<...>\usr\sap\<SID>\<instance_number>\j2ee\deploying"/>
   <property
      name="deploy.lib.dir"
      location="${deploy.base.dir}\lib"/>
   <property
      name="deploy.manager.config"
      location="<path to deploy-manager-config.xml>"/>

   <!-- (2)The target
      runs com.sap.engine.offline.OfflineToolStart out of
      launcher.jar and passes the deploy-manager-config.xml
      as a start parameter.
```

```
    -->
    <target name="deploy">
        <java
            dir="${deploy.base.dir}"
            classname="com.sap.engine.offline.OfflineToolStart"
            fork="yes">

            <arg value="com.sap.engine.deploy.manager.DeployManagerImpl"/>
            <arg value=";.\lib"/>
            <arg value="${deploy.manager.config}"/>

            <classpath>
                <pathelement location="${deploy.lib.dir}/launcher.jar"/>
            </classpath>
        </java>
    </target>

</project>
```

Run Apache Ant

So, all that had to be done has been done. Make sure your Ear file is located corresponding to your path definition in `deploy-manager-config.xml` and that `deploy-manager-config.xml` is located corresponding to your definition in `build.xml`.

Finally run the `build.xml` and see how fast your Ear file is deployed on SAP Web AS 6.40.

 *Note that your application is not yet started.*

Example 2 – Programmatic Deployment

In the above example you got to know the simplest approach of using Apache Ant in your SAP Web AS 6.40 deployment environment. This example shows you how to programmatically configure your deployment by implementing your own Deployment Ant Task. This advancement bears some benefits compared with Example 1. Such are an improved remote deploying functionality and the advantages of direct programmatic access to the `DeployManager` API.

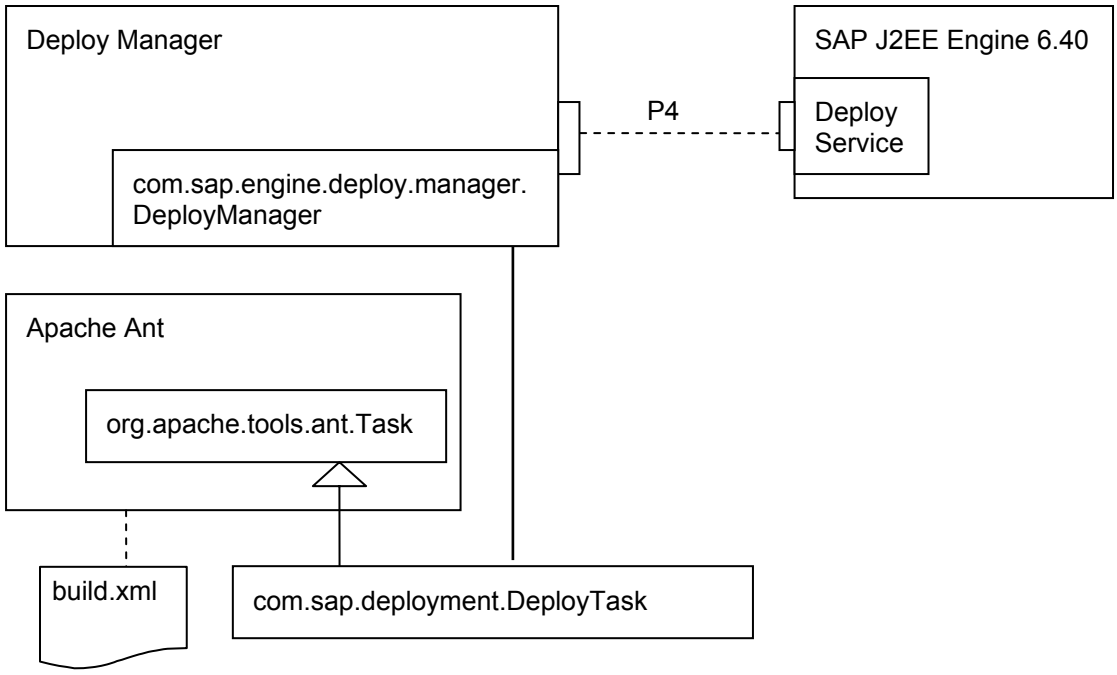

Figure 3: Deploy Task Implementation

To assure improved remote deployment ability, the DeployManager libraries are copied from the file structure of your SAP Web AS 6.40 installation. You might remember that in Example 1 there where dependencies to the `DeployManager`s Libraries under `\usr\sap\<SID>\<instance_number>\j2ee\deploying`. Hence your deployment now can be executed from a remote box without a SAP Web AS 6.40 installation.

An alternative is to reference the DeployManager libraries via the ***`sapmnt share`*** *defined on \usr\sap\. In this case make sure, that the appropriate user rights are defined for the user executing the Ant task.*

Prerequisites

- An Ear file – The Application you want to deploy
- Or download ant_deployment_example.zip. The file contains the example source codes and a simple J2EE application so you can run the example right away.
- SAP Web AS 6.40 is up and running
- Apache Ant available on your system (example tested with Apache Ant 1.6.5)

Writing a Simple Ant Task

To integrate SAP Web AS 6.40 deployment functionality into your own deployment environment, simply extend `org.apache.tools.ant.Task` (Figure 3). The Deploy Managers functionality is achieved by aggregation of `com.sap.engine.deploy.manager.DeployManager`. Once you instantiated your own `DeployManager` object you pass all the deployment parameters to the `DeployManager` in your source code. Ant also allows you to pass your parameters via the `build.xml` script.

DeployTask.java

The overridden method `execute()` of example class `DeployTaskExample.java` instantiates the `DeployManager` implementation and calls its methods passing the deployment parameters. Finally the application is deployed and started.

To keep this example sample it only introduces the `deploy()` and `startApplication()` method. Note that `DeployManager` offers more useful methods for a customized build and deployment environment.

```
package com.sap.deploy.example;

import org.apache.tools.ant.BuildException;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

import com.sap.engine.boot.SystemProperties;
import com.sap.engine.deploy.manager.DeployManager;
import com.sap.engine.deploy.manager.DeployManagerImpl;
import com.sap.engine.deploy.manager.LoginInfo;
import com.sap.engine.frame.core.thread.Task;

public class DeployTaskExample extends Task {

    //deploy properties loaded from file
    private String appName = "";
    private String vendorName = "";
    private String remoteHost = "";
    private String userName = "";
    private String userPassword = "";
    private int p4port;

    private String[] support = new String[] { "p4" };

    // Parameters passed from build.xml
    private String logLocation = null;
    private String earFile = null;
    private String deployPropertyLocation = null;

    /**
     * Deploy an application EAR to SAP Java Engine 6.40
     * and start the application
     */
    public void execute() throws BuildException {

        //To be sure that the tool even runs in environments with specific
        //classloading situations. As recommended in SAP XML Toolkit
        //recommendations.
        ClassLoader oldLoader = null;

        try {

            loadDeployProperties();

            oldLoader = Thread.currentThread().getContextClassLoader();
            Thread.currentThread().setContextClassLoader(
                this.getClass().getClassLoader());

            System.out.println(
                "Starting deployment of application "
```

```
                + appName
                + "\nEAR: "
                + earFile);

        SystemProperties.setProperty(
            "javax.xml.parsers.DocumentBuilderFactory",
            "com.sap.engine.lib.jaxp.DocumentBuilderFactoryImpl");
        SystemProperties.setProperty(
            "javax.xml.transform.TransformerFactory",
            "com.sap.engine.lib.jaxp.TransformerFactoryImpl");

        LoginInfo info = new LoginInfo();
        info.setUserName(userName);
        info.setUserPassword(userPassword);
        info.setRemoteHost(remoteHost);
        info.setRemotePort(p4port);

        DeployManager dm = new DeployManagerImpl();
        dm.setEar(earFile);
        dm.setApplicationName(appName);
        dm.setLog(logLocation);
        dm.setLoginInfo(info);
        dm.setSupport(support);

        dm.deploy();

        dm.startApplication(vendorName + "/" + appName);

        System.out.println(
        "Application " + appName + " is deployed and started.");

    } catch (Exception e) {

        throw new BuildException(
            "\nThe Application "
                + appName
                + " cannot be deployed because:\n"
                + e.toString());
    } finally {

        Thread.currentThread().setContextClassLoader(oldLoader);

    }

}

/*
 * Load deploy properties from deploy properties
 */
public void loadDeployProperties() throws IOException {

    Properties deployProperties = new Properties();
    FileInputStream fis = new FileInputStream(deployPropertyLocation);
    deployProperties.load(fis);

    appName = deployProperties.getProperty("application.name");
    vendorName = deployProperties.getProperty("application.vendor");
    remoteHost = deployProperties.getProperty("deploy.remoteHost");
    userName = deployProperties.getProperty("deploy.userName");
    userPassword = deployProperties.getProperty("deploy.userPassword");
    p4port = Integer.parseInt(deployProperties.getProperty("deploy.p4port"));

    fis.close();

}
```

```
    /*
     * Setter methods to recieve additional parameters
     * from build.xml
     */
    public void setLogLocation(String string) {
        logLocation = string;
    }

    public void setEarFile(String string) {
        earFile = string;
    }

    public void setDeployProperties(String string) {
        deployPropertyLocation = string;
    }

}
```

Writing the build.xml

In build.xml make shure that all needed libraries and the `DeployTask.class` are available in the classpath. By defining the Task Definition you make `DeployTask` executable by an Ant target. The `deploy_to_sap` target in the example `build.xml` passes deployment parameters to the specific setter methods in `DeployTask` and then executes `execute()` in `DeployTask`.

build.xml

```
<project name="DeployTask" default="deploy_to_sap" basedir=".">

<description>
    Deploys ear file to SAP Web AS 6.40
</description>

<!-- set the classpath -->
    <path id="classpath">
        <pathelement location="bin"/>
        <fileset dir="lib">
            <include name="**/*.jar"/>
        </fileset>
    </path>

<!-- Targets -->
    <target name="clean">
        <delete dir="bin"/>
    </target>

    <target name="build" depends="clean">
        <mkdir dir="bin"/>
        <javac srcdir="src" destdir="bin">
            <classpath refid="classpath" />
```

```
        </javac>
    </target>

    <target name="declare" depends="build">
        <taskdef name="deploy"
                 classname="com.sap.deploy.example.DeployTaskExample">
            <classpath refid="classpath"/>
        </taskdef>
    </target>

    <target name="deploy_to_sap" depends="declare">
        <deploy
            earFile="${basedir}/SampleApplication/DeployDummy.ear"
            logLocation="${basedir}/SampleApplication/deploy_log.txt"
            deployProperties="${basedir}/deploy.properties"/>
    </target>

</project>
```

Downloading and Running the Example Project

For a quick start simply download and run the example project. You may import it into your SAP NetWeaver Developer Studio workspace or, assumed a current Ant version is available on your System, run it anywhere.

Running the Example in SAP NetWeaver Developer Studio

The NetWeaver Developer Studio (in short: NWDS) ships with an Ant implementation (Version 1.5.3 in SP11) and also provides the corresponding Ant View. To check out the example follow these steps:

Extract the downloaded archive

To import the project into your NWDS workspace choose `file` in the NWDS main menu. Then choose `import` and follow the wizard's instructions.

To extend Ant classes they have to be available in the build path. Under `poject>properties>Java Build Path>Libraries` edit the location of `ant.jar` to where the NWDS' plugin folder exists on your system (e.g. `c:/programs/sap/jdt/eclipse/plugins` on windows boxes).

In NWDS open the Ant View and add the projects `build.xml`. To do so right click into the empty ant view and choose `Add Buildfile`.

- To make all required libraries available on ant runtime add all the libraries from `lib` folder to ants classpath: Right click the `DeployTask` ant file in Ant View and choose `Properties`…. Select the `classpath` tab and add all the libs in the projects `lib` folder to the classpath entries.
- Click `OK`.
- Change deployment parameters to your preferences in `deploy.properties`.
- Run the target `deploy_to_sap` by double clicking it's representation in ant view. The sample application is deployed to your SAP Web AS 6.40 instance.

Running the Example outside the NWDS

Automated deployment environments often don't provide a full SAP Web AS installation including the NWDS. To give you an impression of how easy it is to migrate deployment functionality out of the SAP Web AS 6.40 environment this example is runnable on any system assumed Apache Ant is available. The example is tested with Apache Ant version 1.6.5.

- Make sure `ANT_HOME` is set correctly in your system classpath.
- Download the example and extract the archive.
- Change to the example's home directory
- Change deployment parameters to your preferences in `deploy.properties`
- Go to a prompt and run ant with the `-lib` option to make the libraries in `lib` folder available at runtime:
- Type `ant -lib lib`

About the Author

Helge Martin is an expert in J2EE development and application migration on SAP Web AS 6.40. He has been recently working in the Market Development Engineering Team of SAP Platform Ecosystem.