# Packaging Native Libraries Inside SAP NetWeaver AS Java

## Applies to:

SAP NetWeaver Application Server Java (AS Java) – 7.1 and higher

## Summary

The purpose of this tutorial is to teach users (SAP developers and customers) how to deliver native libraries to AS Java in order to use them from Java EE applications

**Authors:** Pavel Genevski, Katya Todorova

**Company:** SAP

**Created on:** 29 October 2010

## Authors Bio

**Pavel Genevski** is a Java/C++ developer with more than 7 years of experience. Throughout his career, he has worked on various projects, ranging from software for microcontrollers to CRM and ERP solutions. Currently he is a Senior Java developer in the Technology Development Group at SAP, with responsibilities in the core server runtime and deployment.

**Katya Todorova** is a software developer. She has been at SAP for 3 years with her expertise mainly in the areas of Componentization, Modularization and Class Loading within SAP NetWeaver.

## Table of Contents

## Problem definition

If you are reading this document you probably have a good reason to use native libraries with all their powers and inconveniences. If in doubt, refer to the JNI specification, section 1.4 “When to use JNI”.

### What is a native library?

The term “native library” comes from the JNI specification and means an operating system (OS) specific library, like a Windows DLL or Linux/Unix shared library (shared object). Java bytecode running inside a Java Virtual Machine (JVM) can access code from native libraries in a vendor neutral way through the Java Native Interface (JNI). JNI provides special means that allow Java and native (e.g. C++) code to interoperate and call each other safely, without violating the Java memory model.

### How can Java bytecode call functions from a native library?

There are two ways to do that, direct use of JNI native methods and dynamic invocation through “shared stubs” (which also relies on JNI but makes it easier for the user):

#### Direct JNI

Using this approach, a developer has to write one native method per function they want to call. The process is described well in various JNI tutorials.

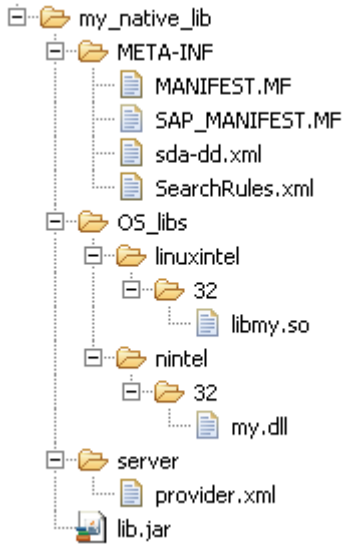
#### Dynamic invocation

With this approach, developers use a home grown or third party library to do the native call dispatching for them. One of the popular libraries for dynamic invocation is JNA (See

Related Content). There is a good tutorial at the JNA website as well that explains how to use it.

## Creating a deployment archive with a library

1. Create a directory named e.g. “my_native_lib” with the following structure:




2. Put the following contents in those files:

- MANIFEST.MF (a single row plus a carriage return)

```
Manifest-Version: 1.0

```

- SAP_MANIFEST.MF

```
Manifest-Version: 1.0

Ext-SDM-SDA-Comp-Version: 1

softwaretype: primary-library

JarSAP-Version: 20040427.1000

JarSAPProcessing-Version: 20040406.1800

deployfile: sda-dd.xml

keyname: MyNativeLib

keyvendor: mycompany.com

keylocation: localhost

keycounter: 2004.10.28.14.24.15

componentelement: <componentelement  name="MyNativeLib"
vendor="mycompany.com" component

 type="DC" subsystem="NO_SUBSYS" location="localhost"
counter="2004.10.28.14.24.15" delt
```

```
 aversion="F"/>

JarSL-Version: 20040702.1400

compress: true
```

**Note:** If you need to modify this file, please conform to these two requirements:
All lines must be no longer than 89 characters. If line content exceeds this number, use a carriage return and a space to continue.
The last line in the file must end in a carriage return.

- sda-dd.xml

```
<?xml version="1.0" ?>

<SDA>

  <SoftwareType>primary-library</SoftwareType>

      <engine-deployment-descriptor version="2.0"/>

</SDA>
```

- SearchRules.xml
  (For further details see SAP Note 850116)

```
<?xml version="1.0"?>

<native-parts>

  <native-part>

    <path platform="ntintel" jvm-bitlength="32" supported="true">

          os_libs/ntintel/32/my.dll

    </path>

    <path platform="linuxintel" jvm-bitlength="32" supported="true">

      os_libs/linuxintel/32/libmy.so

    </path>

  </native-part>

</native-parts>
```

- my.dll – this is the version of your native library for 32 bit Windows®
- libmy.so – the same library for 32 bit Linux
- provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE provider-descriptor SYSTEM "library.provider.dtd">
```

```
<provider-descriptor>
      <display-name>
      MyNativeLib
    </display-name>
      <component-name>
      MyNativeLib
    </component-name>
      <major-version>6</major-version>
      <minor-version>40</minor-version>
      <micro-version>0</micro-version>
      <provider-name>
      sap.com
    </provider-name>
      <references/>
      <jars>
          <jar-name>lib.jar</jar-name>
      </jars>
</provider-descriptor>
```

- lib.jar – here you put the Java classes that expose the functionality of the native library. There are two options:
    - Include an existing JAR file that calls the native library
    - Create a Java class with native methods that call the native library. Use java.lang.System.loadLibrary() inside that class to load the native library. This class must be packaged inside the native library, not inside the application.

3. Package the content of the “my_native_lib” folder in a ZIP archive (my_native_lib.zip).
4. Change the extension of the file to .sda, resulting in my_native_lib.sda.

## Deployment of an archive with native libraries

Use the standard SAP NetWeaver Developer Studio deployment functionality to deploy the SDA file on AS Java (causes an AS Java restart). If deployment is successful your native library will appear under the directory/usr/sap/<SID>/<IN>/j2ee/os_libs of your server installation.

For more information, see Deploying and Undeploying with SAP NetWeaver Developer Studio

## Create a Java EE application that packages the native library

1. Create a standard Java EE application.
   For more information, see Developing Your First Java EE 5 Application
2. Add a runtime reference to the library.
   For more information, see
   Creating runtime references
   Editing Runtime References in the application-j2ee-engine.xml

The application itself must not contain classes with native methods. Put the native methods inside the native library instead.

## Repackaging existing JARs with native libraries

Let's assume that you want to deliver JNA to AS Java. To do so, you have to package its artifacts in an SAP compliant deployment archive (SDA):

1. Take the example SDA file created above as a starting point
2. Find all native libraries inside the JNA JAR (i.e. all the .dll and .so files).
3. Copy them to the respective folders under os_libs. Create additional folders if needed
4. Adjust the SearchRules.xml file accordingly, so that it contains the locations and os/architecture of all libraries inside os_libs
5. Copy the JNA JAR file(s) (without the native libraries) to the root of the archive
6. Adjust provider.xml in order to include the new JAR(s).

## Related Content

JNI Specifications

JNA

## Copyright