8. Intermediate - How to write my own binding component

{scrollbar}

How to write my own binding component

ATTENTION!

This tutorial page is a work in progress and therefore NOT finished. So check back later for updates.

Prerequisites

- Maven 2.0.7 or higher
  - If you have never used Maven previously the Maven Getting Started Guide explains some valuable concepts surrounding Maven
- ServiceMix 3.2.1 or higher
- A broadband internet connection so Maven can automatically download dependencies

General

This tutorial will explain to you how to write your own binding components. What you learn here is also enabling you to write own service engines because the difference between a binding component and a service engine is of pure logical nature. While a binding component is translating data from outside bus to normalized messages used inside the bus and vice versa a service engine is only working inside the bus with normalized messages.

Should I really create my own binding component?

Before beginning this tutorial, please take the time to read the FAQ entry titled "Should I Create My Own JBI Components?". It is very important that you understand the reason for developing a JBI binding component and this FAQ entry will explain this to you.

A very brief introduction to Java Business Integration (**JBI**)

The Java Business Integration (JBI) spec provides a standards-based, service-oriented approach to application integration through the use of an abstract messaging model, without reference to a particular protocol or wire encoding. JBI introduces the concepts of Binding Components (BCs), Service Engines (SEs) to Service Units (SUs) and Service Assemblies (SAs) to define an architecture for vendor-neutral pluggable components. The purpose of this architecture is to provide standards-based interoperability amongst components/services.

JBI components can be thought of as the *smallest applications* or *services* accessible in a service-oriented architecture. Each service has a very specific purpose and therefore a narrow scope and set of functionality. Components come in two flavors:

- Service Engines (**SE**)
- Binding Components (**BC**).

SU*s must be packaged into a *SA to be deployed to the JBI container. A **SA** is a *complete application* consisting of one or more services. By comparison, this is similar to the way that WAR files must be packaged inside of an EAR file to be deployed to a J2EE container.

See also the page providing information on working with service units

Below some quick definitions which are dominant throughout the **JBI** spec:

- **Component Architecture**
  - **Binding Components** - Components that provide or consume services via some sort of communications protocol or other remoting technology
  - **Service Engines** - Components that supply or consume services locally (within the JBI container)

The difference between binding components (*BC*s) and service engines (*SE*s) is definitely subtle and is not denoted by the JBI APIs. In fact, the only real true difference between the two is in the `jbi.xml` descriptor in the packaging. What it really boils down to is the fact that *BC*s are used to do integration with a service outside the bus and *SE*s are services that are deployed to and solely contained within the bus. Hopefully the JBI 2.0 spec will provide more distinction.

- **Component Packaging**
  - **Service Units** - Packaging for an individual service that allows deployment to the JBI container; similar to a WAR file from J2EE
  - **Service Assemblies** - Packaging for groups of *SU*s for deployment to the JBI container; similar to an EAR file from J2EE

This tutorial focuses on both component architecture and component packaging. For further information and details on JBI, see the following:

- The JBI spec
- The JBI section of the User's Guide
- The JBIforSOI article

- The ServiceMix as an enterprise service bus JavaWorld article

Getting started with a BC

This tutorial will explain to you how to create a binding component for the SNMP protocol. If you don't know what this protocol is about, then see for example the SNMP Wikipedia entry for further details. As a first step we will only create a snmp polling service. Once you understand how to do this, you will be able to go further and create also a sender service but this will be not done inside this tutorial for now.

Now let's move on to creating the Maven projects for the SNMP binding component.

Creating a Maven project for the JBI BC

The focus of this section is on the creation of a JBI binding component. For this task, a Maven archetype will be used to create a Maven project skeleton to house the component. Maven archetypes are templates for Maven projects that jumpstart project creation via the automation of repetitive tasks by following standard conventions. The result of using an archetype to create a Maven project is a directory structure, a Maven POM file and, depending on the archetype being used, sometimes Java objects and JUnit tests.

Below are the steps to follow for creating the directory structure and project. All instructions are laid out to take place on a Unix command-line.

1) Create a Maven project named `servicemix-snmp` and switch to that directory:

$ mvn archetype:create \ -DarchetypeGroupId=org.apache.servicemix.tooling \ -DarchetypeArtifactId=servicemix-project-root \ -DarchetypeVersion=3.2.2 \ -DgroupId=org.apache.servicemix.tutorial.snmp \ -DartifactId=servicemix-snmp-tutorial $ cd servicemix-snmp-tutorial

2. Check the project Maven configuration file

Maven created a project folder for you and already setup a project file called pom.xml. As this is not a beginner tutorial I won't explain the file here. Open up the *pom.xml* file in your favorite editor and look at the content if everything is like you want it to be.

$ nano pom.xml

You will recognize, that the project file is setup for your purposes already. You normally don't need to change anything in there for the moment.

3) Create the binding component sub project

Use the servicemix-binding-component Maven archetype to generate a Maven project for the component.

To create the BC, execute the following command on the command-line:

$ mvn archetype:create \ -DarchetypeGroupId=org.apache.servicemix.tooling \ -DarchetypeArtifactId=servicemix-binding-component \ -DarchetypeVersion=3.2.2 \ -DgroupId=org.apache.servicemix.tutorial.snmp \ -DartifactId=snmp-binding

The command above will create a directory named `snmp-binding` that houses a Maven project for the JBI binding component being created here. The name of the directory is taken from the `artifactId` parameter.

Now switch into the snmp-binding sub folder and open the pom.xml in your editor of choice. We will cleanup the file before proceeding to program code.

Do the following things:

- we already defined repositories in project root pom, so you can safely discard the repository entries in the BC's pom.xml
- fill in a proper name for the binding component in the name tag

ToDo-List

- ~~when to create a binding component?~~
- ~~outline the different jbi packaging units (bc, se, su, sa)~~
- ~~construct a real use case for the bc (for example a snmp poll service for grabbing snmp values of a network device like printer)~~
- start by setting up the folder structure and the root pom -> watch out for not working maven:create and the incomplete BC archetype
- define as much as possible inside the root pom (maybe done when the SU is created)
- create the binding component
- describe the pom content
- detailed description of the key concepts
- describe the base classes of the new bc and their role
- a note on the annotations which control the elements in the su's xbean.xml
- a note on different MEPs to support (or not)
- doing a consumer endpoint (poller)
- doing a test case for the bc
- testing the bc
- creating a service unit for the bc
- describe how to configure the pom.xml
- describe how to setup the xbean.xml
- create the service assembly
- configure the sa to package the su
- deployment of the sa -> bc is still missing -> describe dependency resolving mechanism
- deployment of the bc -> see the SA now deploying
- see result of the polling in console window
- for experienced users:

- doing a provider endpoint (left over for the experienced reader to implement)
- describe marshaler logic as it is used in nearly every SE / BC of smx
- implement a marshaler (for more experienced readers, this will also affect the BC to provide such possibility)
- add a file-sender to write snmp poll results to a file and wire it to the snmp poller
- deployment and testing
- give links etc. for further reading and for looking at other BC's code (snippets)

An overview of INLINE

using Maven-based tooling and archetypes to develop a snmp binding component

Goals of the document

This tutorial provides an easy and convenient way for a new user to learn about:

- using Maven to develop JBI binding components
- using Maven to develop JBI service units and service assemblies
- using Maven to create Eclipse projects
- using xbean.xml files to configure routes and services in ServiceMix
- writing your own binding component

After finishing this tutorial you have a snmp binding component ready to poll devices. Feel free to play around with it and adding improvements.

Contents

true

Start this tutorial

{scrollbar}