## Integrating the Service in the Backbase

1. [Configure Maven](https://my.backbase.com/resources/documentation/portal/5.6.0/devsetup_mavenconfig_integrationservices.html)

The settings.xml file contains the Maven configuration. It is located in:

C:\Documents and Settings\*user\_name*\.m2

The file should contain the following lines:

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

<profiles>

<profile>

<id>backbase</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

<repositories>

<repository>

<id>Backbase Artifact Repository</id>

<url>https://repo.backbase.com/repo/</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>Backbase Artifact Repository</id>

<url>https://repo.backbase.com/repo/</url>

</pluginRepository>

</pluginRepositories>

</profile>

</profiles>

<servers>

<server>

<id>Backbase Artifact Repository</id>

<username>atos\_portal\_p5</username>

<password>atos\_portal\_1654</password>

</server>

</servers>

</settings>

1. The following environment variables set:

|  |  |
| --- | --- |
| Environment Variable | Value |
| JAVA\_HOME | Installation folder of the Java Development Kit. |
| M2\_HOME | Installation folder of Maven. |
| M2 | %M2\_HOME%\bin |
| MAVEN\_OPTS | -Xmx512m -XX:MaxPermSize=256m |
| Path | %JAVA\_HOME%\bin and %M2% |

#### Generate Project

1. **Retrieve archetype**

Execute the following command:

**mvn archetype:generate -DarchetypeArtifactId=integration-distributed-web-archetype -DarchetypeGroupId=com.backbase.portal.archetypes -DarchetypeVersion=5.6.0.0**

In response to a series of command-line prompts, enter the following:

* groupId — a package name like my.company.com.
* artifactId — an identifier for the new Maven project like myportal. Maven will create the project in a folder by the same name.
* version — the version of the project.
* package — a package name like my.company.com. Defaults to the value of groupId.

Finally, confirm the settings. Maven creates the project in the folder defined by artifactId.

Note: - If Maven returns a [WARNING] Unable to get resource message, your system may be missing required certificates.

ii)[**Generate Project for Camel Java DSL**](https://my.backbase.com/resources/documentation/portal/5.6.0/javadev_generateproject.html)

Create a Java class that extends org.apache.camel.builder.RouteBuilder and implements the configure method.

Here follows a Hello World example:

**package** com.acme.services;

**import** org.apache.camel.builder.RouteBuilder;

**public** **class** MyCamelService **extends** RouteBuilder {

    @Override

**public** **void** configure() **throws** Exception {

        from("restlet:/v1/myService")

          .setBody().constant("<html><body><b>Hello World!</b></body></html>")

          .setHeader("Content-Type", constant("text/html"));

    }

}

Take the following in to account:

* Set the correct package path.
* Create folders according to your package in the src folder in the project. For example:src/main/java/com/acme/services. Save your Java file there.
* Routes should start with restlet:/. Everything that follows this will determine the URL to reach this integration service.

## iii) **Register the Java Component**

You need to register the Java component in Spring. Open the backbase-integration-service.xml file located in thesrc/main/resources/META-INF/spring folder in the project. Add the following bean:

<bean name="MyServiceName" class="com.acme.services.MyCamelService" />

Take the following in to account:

* The name can be any string.
* The class should correspond to the Java class you have created above.

#### Generate Project

Open a command shell. Go to the root folder of the project. Run the following command to build the project:

mvn clean install

#### Add dependency

After building the integration service, you can add it to the main project, which is either Portal Services (with Integration Services embedded) or Integration Services stand-alone. Open the POM file of the relevant project:

* Embedded: open the POM file of the Portal Services project
* Stand-alone: open the POM file of the Integration Services stand-alone project

Add the following lines as a child of the dependencies element:

<**dependency**>

    <**groupId**>com.acme.services</**groupId**>

    <**artifactId**>integrationservice</**artifactId**>

    <**version**>1.0-SNAPSHOT</**version**>

</**dependency**>

Make sure you use the same groupId, artifactId and version as in the POM file of the integration service project.

Note :- If you are writing the service inside the portalserver then no need to add the dependency as it will compile and will service will be the part of generated war.

#### Use Integration Services in a Widget

#### Create Dummy Integration Service

## In order for this example to work, you need to develop an integration service that returns a JSON object with account information. The integration service requires a Java class and a Spring XML file to register the Java class. For more information

## Use the following code to create the Java class with a dummy integration service:

**package** com.acme.services;

**import** org.apache.camel.builder.RouteBuilder;

**public** **class** AccountsService **extends** RouteBuilder {

    @Override

**public** **void** configure() **throws** Exception {

        from("restlet:/accounts")

          .setBody().constant("[{\"name\":\"Vladimir\",\"amount\":1480},{\"name\":\"Hannah\",\"amount\":1263},{\"name\":\"Terry\",\"amount\":389},{\"name\":\"Anita\",\"amount\":561}]")

          .setHeader("Content-Type", constant("application/json"));

    }

}

Register the Java class with the following Spring XML code:

<**beans** xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:camel="<http://camel.apache.org/schema/spring>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xsi:schemaLocation="

<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://camel.apache.org/schema/spring> <http://camel.apache.org/schema/spring/camel-spring.xsd>">

    <**bean** name="AccountsService" class="com.acme.services.AccountsService" />

</**beans**>

The following example is a widget definition that retrieves account data from Integration Services and displays it in a list. For the example to work, you will need to create an integration service that returns the account data, see below.

<!DOCTYPE html>

<**html** xmlns="<http://www.w3.org/1999/xhtml>" xmlns:g="<http://www.backbase.com/2008/gadget>" xml:lang="en">

<**head**>

    <**meta** http-equiv="Content-Type" content="text/xml; charset=UTF-8"/>

    <**title**>Integration services example</**title**>

    <**g:preferences**>

        <**g:preference** name="serviceUrl" label="Service Url" type="text" default="$(contextRoot)/services/rest/accounts" viewhint="user"/>

    </**g:preferences**>

    <**script**><![CDATA[

        function loadAccounts(widget) {

            //store the URL of the service as a configurable preference

            var serviceUrl = widget.getPreference('serviceUrl');

            serviceUrl = serviceUrl.replace('$(contextRoot)', b$.portal.config.serverRoot);

            //here we use jQuery to make a simple AJAX call

            $.get(serviceUrl, function(data) {

                //imagine the data returned contains an array of accounts that we want to output to the widget body

                var accountsHtml = data.reduce(function(html, account) {

                    html += '<li>' + account.name + ': ' + account.amount + '</li>';

                    return html;

                }, '');

                $(widget.body).find('.accounts > ul').html(accountsHtml);

            });

        }

    ]]></**script**>

</**head**>

<**body** g:onload="loadAccounts(\_\_WIDGET\_\_)">

    <**div** class="accounts">

        <**ul**></**ul**>

    </**div**>

</**body**>

</**html**>

Note :- in the above code below line constructing the url for the service.

        <**g:preference** name="serviceUrl" label="Service Url" type="text" default="$(contextRoot)/services/rest/accounts" viewhint="user"/>

In this case, service must have like “from("restlet:/accounts")” to call the service as above.

## REST with Java (JAX-RS) using Jersey

## Follow the below link to understand and implement the RESTful API using Jersey

## <http://www.tutorialspoint.com/restful/restful_first_application.htm>

## <http://www.vogella.com/tutorials/REST/article.html>

## <http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/> (Important)

## <http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html>

## <http://javapapers.com/java/restful-web-services-with-java-jax-rs-using-jersey/>