Identification Key generation WebService : Installation & Development Guide

Laboratoire Informatique et Systématique

2011

Abstract

Identification keys are widely used by scientists to identify taxa. This new identification key generation WebService generates single-access keys on demand, for single users or research institutions. It receives user input data (using the standard SDD format), accepts several parameters for the key generation (impacting the key topology and representation), and supports several output formats. Furthermore, key generation automation is possible thanks to the WebService architecture.

Contents

1 Introduction **3**

2 WebService installation **3**

2.1 Prerequisites 3

2.2 Installation using the binary packages 3

2.2.1 Deploying the binary packages on Apache Tomcat 4

2.3 Building the project using the source code 4

2.3.1 Importing the Eclipse projects 4

2.3.2 Configuring the project as a Maven project 4

2.3.3 Configuring the projects' `CLASSPATH` 5

2.3.4 Building the projects 5

3 Querying an instance of the WebService **5**

3.1 PHP clients 5

3.1.1 S.O.A.P. client example 6

3.1.2 R.E.S.T. client example 6

3.2 Java clients 7

4 WebService configuration **7**

4.1 Post-deployment configuration 7

4.2 Access restriction 7

4.3 Generated keys automated deletion 8

4.3.1 Configuring the deletion schedule 8

4.3.2 Configuring the deletion delay 8

List of Figures

1 Usage Scenarios 3
2 Deploying a *.war binary package in Apache Tomcat 9
3 Importing the source code projects into Eclipse 10
4 Configuring the project as a Maven project 10
5 Building a project 10

1 Introduction

As a part of the ViBRANT project, the WebService is integrated in the Scratchpads biodiversity networking tool, with an embedded client component. Hence, a Scratchpads user can use the WebService directly and transparently from his Scratchpads instance. It is also possible for anyone to develop his own client component in order to call the WebService directly. The whole WebService and its source code are freely available, thus allowing large institutions to deploy it on their own network and adapt it to their specific needs.

There are two main usage scenarios of this WebService :

I A client component is integrated within the Scratchpads[1] biodiversity networking tool, making the WebService available to Scratchpads user transparently.

II The WebService can be called directly from a WebService client. We provide barebones clients written in Java and PHP, for the 2 communication protocols supported by our WebService; S.O.A.P. [2] and R.E.S.T. [3]

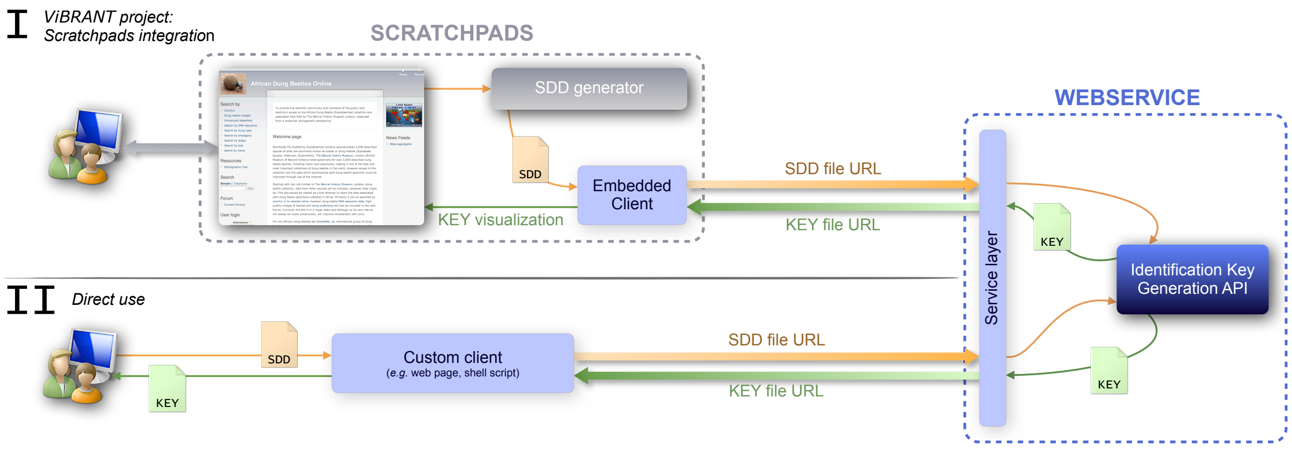


Figure 1: Usage Scenarios

2 WebService installation

2.1 Prerequisites

The WebService is coded in the Java programming language, using the J2EE framework, thus the host machine needs to have a recent JDK (v. ≥ 6) installed, as well as a J2EE web application server (*e.g.* Apache Tomcat, WebSphere or WebLogic). The Java Virtual Machine is freely available on the web[4], and several J2EE web application servers (such as Apache Tomcat) are freely available.

2.2 Installation using the binary packages

The WebServices are available in 2 binary packages :

- `IK_WS_REST-x.x`[5]`.war` : the R.E.S.T. protocol WebService

[1] http://scratchpads.eu/
[2] http://en.wikipedia.org/wiki/SOAP
[3] http://en.wikipedia.org/wiki/Representational_state_transfer
[4] http://www.java.com/en/download/index.jsp
[5] the `x-x` represents the version number

- IK_WS_SOAP-x.x.war : the S.O.A.P. protocol WebService

The two packages can be installed simultaneously on the same web application server. The binary packages are standard war files, and can be deployed on any J2EE web application server using the standard deployment procedure.

2.2.1 Deploying the binary packages on Apache Tomcat

Deploying a *.war file is pretty straightforward :

1. Go to your tomcat server manager page : `http://yourserver:8080/manager/html`
2. Click on the *Choose file...* button next to the *Select WAR file to upload* at the bottom of the page.
3. Browse your filesystem to find the *.war file you want to deploy.
4. Click on the *Deploy* Button.

(*cf.* figure 2 on page 9)

2.3 Building the project using the source code

The source code of this project is freely available on our server [6]. It is packaged as a zip file that contains 3 Eclipse projects :

- `IdentificationKeyAPI` : the project that contains the API.
- `IK_WS_REST` : the project that contains the R.E.S.T. WebService.
- `IK_WS_SOAP` : the project that contains the S.O.A.P. WebService.

Both WebService projects depend on the API to compile properly, *i.e.* they require the presence of a *.jar package assembled from the API project in their `/WebContent/WEB-INF/lib` directory. The projects' dependencies are managed with the Maven utility, thus to build the 3 projects, you will need to have a Maven plugin[7] installed in your Eclipse environment.

2.3.1 Importing the Eclipse projects

To import the projects into Eclipse :

1. In the *File* menu, select the *Import...* option.
2. Select the *General/Existing Projects into Workspace* import procedure.
3. Using the *Select archive file* dialog, select the IK_WS.zip file, select the 3 project, and click *Finish*.

(*cf.* figure 3 on page 10)

2.3.2 Configuring the project as a Maven project

After importing the projects, you will need to convert them to Maven projects in order to manage the dependencies. To do so, right-click on each project, and select the *Configure → Convert to Maven project* option. Once it's done, Maven will download automatically the required dependencies (*cf.* figure 4 on page 10).

[6] http://www.identificationkey.fr/index.php/downloads
[7] http://eclipse.org/m2e/

2.3.3 Configuring the projects' `CLASSPATH`

API project Since the API project contains `properties`, `css` and `js` files, you need to make sure that these file types are included in the project's `CLASSPATH`. To do so, in Eclipse, right-click on the project in the *`Package Explorer`*[8] and select *`Properties`*, this will open a new preference window. In this window, click on *`Java Build Path`* in the left sidebar and select the *`Source`* tab. Select the *`Included`* parameter, and ensure that it contains the following inclusion patterns :

- `**/*.java`
- `**/*.properties`
- `**/*.css`
- `**/*.js`

If it does not, edit it.

WebService projects Both WebService project contain `properties` files, so they only need the `**/*.properties` inclusion pattern.

2.3.4 Building the projects

1. To build the API project, simply right-click the project in Eclipse, and select *`Run as`* → *`Maven package`*, *`Run as`* → *`Maven install`* or *`Run as`* → *`Maven assembly`*. This will generate a *.jar file in the *`target`* folder of the project.
2. Copy the *.jar file containing the API in the `/WebContent/WEB-INF/lib` folder of the WebService project you want to build.
3. The Sigar Library files[9] also need to be added manually in the `/WebContent/WEB-INF/lib` folder of the project
4. Right-click the project in Eclipse, and select *`Run as`* → *`Maven package`*, *`Run as`* → *`Maven install`* or *`Run as`* → *`Maven assembly`*. This will generate a *.war file for the WebService project.

You now have a *.war file that can be deployed on your web application server (*cf.* section 2.2).

3 Querying an instance of the WebService

The WebService can be queried from a Scratchpads instance, but it can also be queried from any WebService client. Here are code samples that can be used to develop your own WebService client, written in `PHP`, for both the S.O.A.P. and the R.E.S.T. protocol. Equivalent examples are also written in `Java`, but are available separately, as Eclipse projects.

3.1 PHP clients

The source code of the two `PHP` clients are available on our server[10]

[8]You can also use the *`Project Explorer`* or the *`Navigator`* view
[9]available here : `http://www.hyperic.com/products/sigar`
[10]`http://www.identificationkey.fr/index.php/downloads`

3.1.1 S.O.A.P. client example

```php
<?php
$wsdl='http://WebService_hostname:8080/IK_WS_SOAP-1.0/identificationKey?wsdl';

$options  = array('compression'=>true, 'exceptions'=>false, 'trace'=>true);
$client = new SoapClient($wsdl, $options);

$param->sddURL = "http://yourServerURL/inputFileSDD.xml";
$param->format = "pdf";
$param->representation = "tree";
$param->fewStatesCharacterFirst = "yes";
$param->mergeCharacterStatesIfSameDiscimination = "no";
$param->pruning = "no";
$param->verbosity = "h";
$param->scoreMethod = "xper";

$res = $client->createIdentificationKey($param);


if(is_soap_fault($res)){
  echo 'fault : ';
  var_dump($client->__getLastRequest());
  var_dump($client->__getLastRequestHeaders());
}else {
  //echo '<pre>'.var_export($res, true).'</pre>';
  //echo $client->__getLastResponse();
  echo $res->identificationKeyResponse;
}

?>
```

3.1.2 R.E.S.T. client example

```php
<?php
$service_url = 'http://WebService_hostname:8080/IK_WS_REST-1.0/identificationKey';
$curl = curl_init($service_url);
// Request parameters
$sddURL = 'http://www.infosyslab.fr/vibrant/project/test/Cichorieae-fullSDD.xml';
$format = 'html';
$representation = "tree";
$fewStatesCharacterFirst= "no";
$mergeCharacterStatesIfSameDiscimination = "no";
$pruning = "no";
$verbosity = "h";
$scoreMethod = "xper";
// urlencode and concatenate the POST arguments
$curl_post_data = 'sddURL='.urlencode($sddURL).'&format='.$format.'&representation='.
    $representation.'&fewStatesCharacterFirst='.$fewStatesCharacterFirst.'&
    mergeCharacterStatesIfSameDiscimination='.
    $mergeCharacterStatesIfSameDiscimination.'&pruning='.$pruning.'&verbosity='.
    $verbosity.'&scoreMethod='.$scoreMethod;
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
curl_setopt($curl, CURLOPT_POST, true);
curl_setopt($curl, CURLOPT_POSTFIELDS, $curl_post_data);
$curl_response = curl_exec($curl);
curl_close($curl);
echo $curl_response;
?>
```

3.2 Java clients

The two Eclipse projects that contain the java clients are available on our server [11]. The mechanism to import these project is similar to the import mechanism used with the WebService projects (*cf.* section 2.3.1 and figure 3, page 10). After the import, you must configure the projects as Maven projects (*cf.* section 2.3.2 and figure 4, page 10).

4 WebService configuration

4.1 Post-deployment configuration

Once the WebService is successfully deployed on your web application server, you must stop the server, put a new file named `confOverridable.properties` in the `/WEB-INF/classes/main/resources` directory[12] of the web application you just deployed. This file contains some parameters used by the WebService. The minimum content of this file is as follows :

```
# --------------------------------------------------------------- #
# The two parameters below determine where the files created by the
# WebService will be located on the host server file system.
# For generatedKeyFiles.prefix, if webapps/ doesn't work, you
# may have to put the absolute  path to the webapps folder
#       e.g. /var/lib/tomcat6/webapps
generatedKeyFiles.prefix = webapps/
generatedKeyFiles.folder = generatedKeyFiles/
# --------------------------------------------------------------- #
# the property below sets the delay (in seconds) after which any
# file generated by the WebService is deleted
# 2592000 is the number of second for 30 days.
generatedKeyFiles.delete.period = 2592000
# --------------------------------------------------------------- #
host = http://yourServerHost:8080/
# --------------------------------------------------------------- #
# the following property may contain any message you want, that
# indicates who created the identification key (e.g. your institution)
message.createdBy = created by *placeholder*
# --------------------------------------------------------------- #
# the following property contains the email address of the webservice
   administrator
email.webmaster = webmaster@yourdomain.com
```

After that, restart your web application server, the WebServices should be functional.

4.2 Access restriction

If you re-deploy the identification key generation WebService on your own network, you may wish to restrict the range of IP addresses that will be able to query the WebService. If you deployed the WebService using Apache Tomcat, you can use IP-based restrictions by adding the following code snippet to the `$CATALINA_HOME/conf/server.xml` file :

```
<Context path="/ApplicationName" docBase="$CATALINA_HOME/webapps/
   ApplicationName" workDir="$CATALINA_HOME/webapps/ApplicationName">
```

[11] http://www.identificationkey.fr/index.php/downloads
[12] If this directory doesn't exist, you'll have to create it yourself

```
  <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow=
     "192.168.*.*,10.0.*.*" deny="*"/>
</Context>
```

By modifying the *allow* and *deny* values of the *Valve* tag, you can specify which IP range should be allowed to query the service, and which IP range should be denied to query the webservice. Keep in mind that in case of conflict between *allow* and *deny*, the *allow* parameter overrides the *deny* parameter.

4.3 Generated keys automated deletion

The identification keys generated by the WebService are stored inside the web application server directory (in `webapps/generatedKeyFiles` by default). To avoid hard drive saturation, there is an automated deletion mechanism implemented in the WebService : every night, at 4:18 am, the WebService deletes every file in `webapps/generatedKeyFiles` that is more than 30 days old. You can modify the deletion schedule, as well as the delay after which a file is deleted

4.3.1 Configuring the deletion schedule

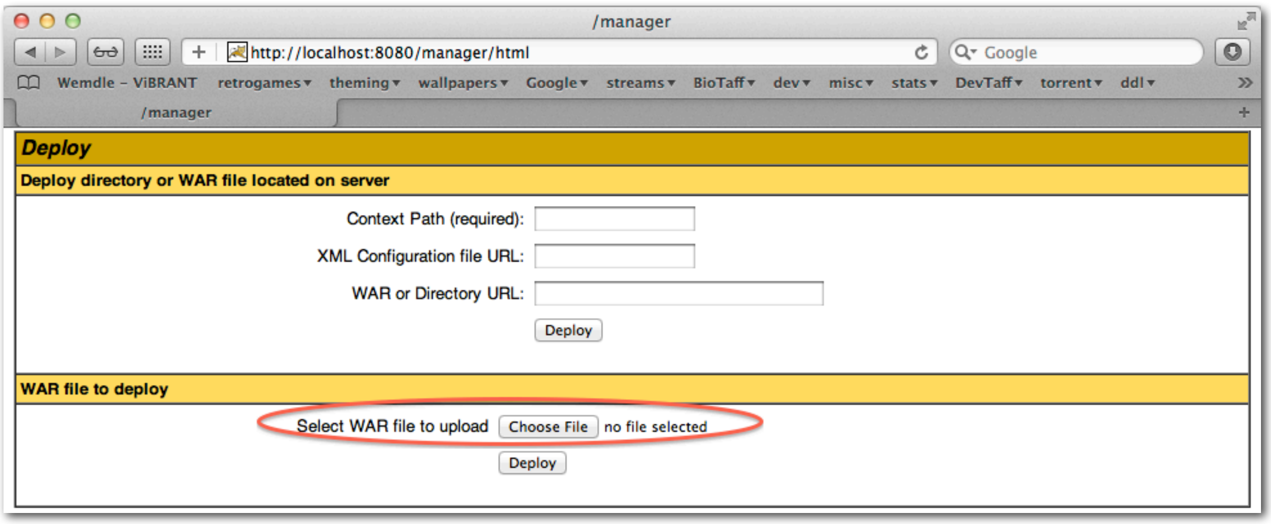
To modify the deletion schedule, you need to edit the `WEB-INF/web.xml` file of the WebService, and modify the text located in the `<param-value>` node in the following web.xml snippet :

```
<servlet>
    <display-name>Scheduler Servlet</display-name>
    <servlet-name>SchedulerServlet</servlet-name>
    <servlet-class>SchedulerServlet</servlet-class>
    <init-param>
        <param-name>cronSchedule</param-name>
        <param-value>18 4 * * ?</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
```

The schedule is passed as a cron schedule (`http://en.wikipedia.org/wiki/Cron`)

4.3.2 Configuring the deletion delay

To modify the file deletion delay, *i.e.* the time lapse after which a file is considered to be too old and is due for deletion (30 days by default), you need to specify a new delay in `confOverridable.properties` (*cf.* part 4.1 on page 7), by setting a new value (in seconds) to the `generatedKeyFiles.delete.period` property.


/manager

http://localhost:8080/manager/html

Deploy

Deploy directory or WAR file located on server

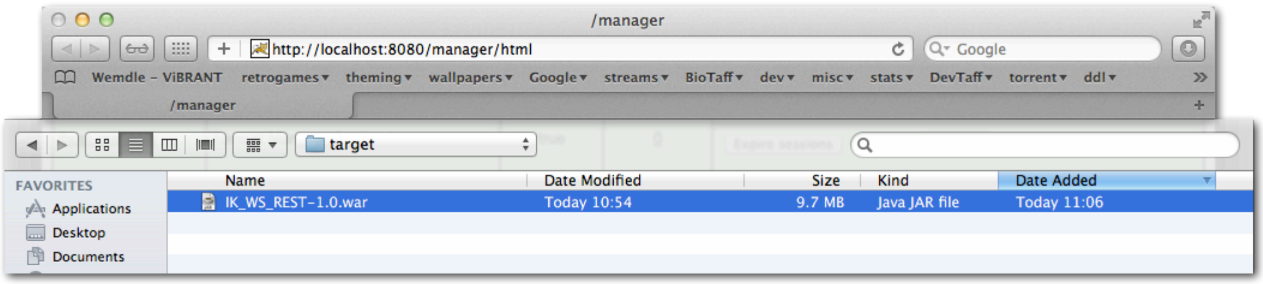
Context Path (required):	
XML Configuration file URL:	
WAR or Directory URL:	

Deploy

WAR file to deploy

Select WAR file to upload Choose File no file selected

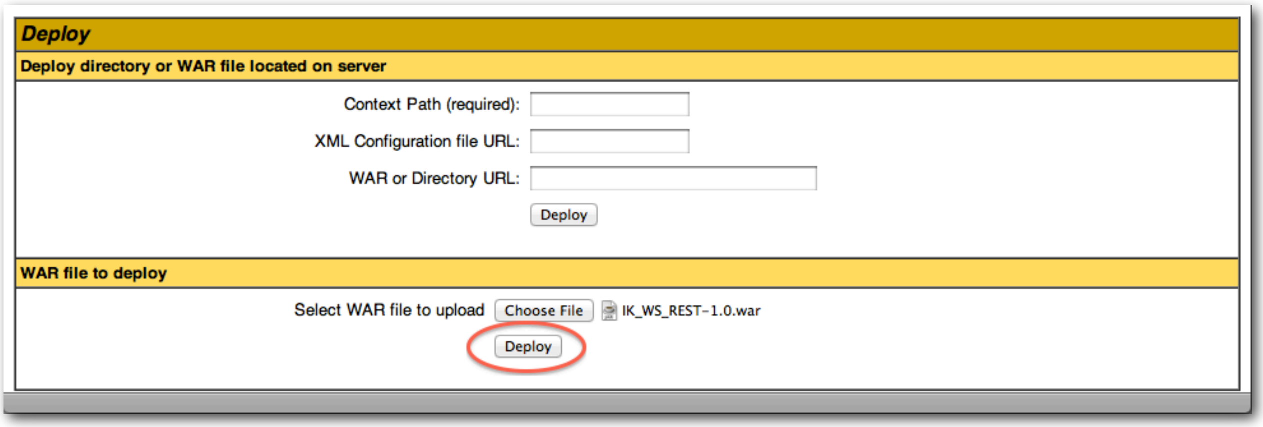
Deploy


/manager

http://localhost:8080/manager/html

target

Name	Date Modified	Size	Kind	Date Added
IK_WS_REST-1.0.war	Today 10:54	9.7 MB	Java JAR file	Today 11:06


Deploy

Deploy directory or WAR file located on server

Context Path (required):	
XML Configuration file URL:	
WAR or Directory URL:	

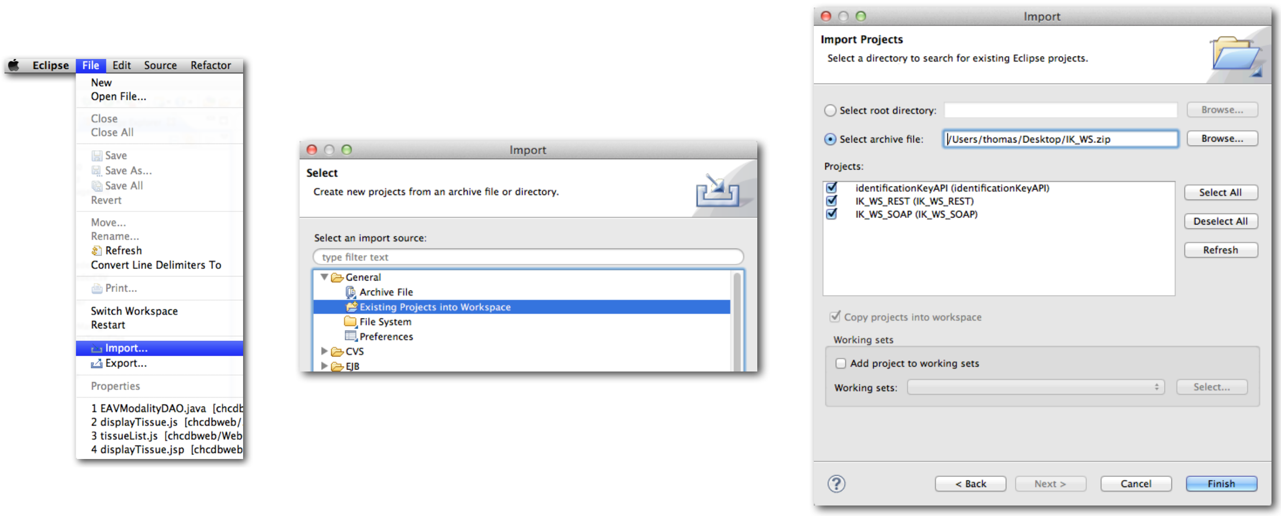
Deploy

WAR file to deploy

Select WAR file to upload Choose File IK_WS_REST-1.0.war

Deploy

Figure 2: Deploying a *.war binary package in Apache Tomcat



Select archive file: /Users/thomas/Desktop/IK_WS.zip

Projects:
- ☑ identificationKeyAPI (identificationKeyAPI)
- ☑ IK_WS_REST (IK_WS_REST)
- ☑ IK_WS_SOAP (IK_WS_SOAP)

Figure 3: Importing the source code projects into Eclipse

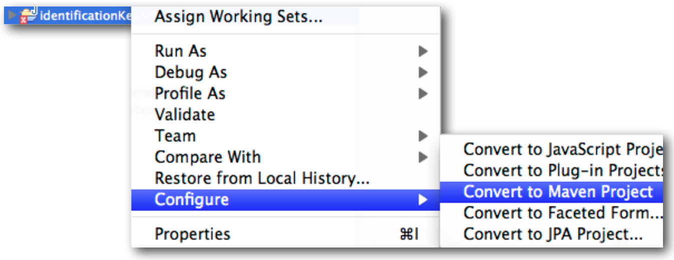

identificationKe
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
Configure
Properties ⌘I
Convert to JavaScript Proje
Convert to Plug-in Projects
Convert to Maven Project
Convert to Faceted Form...
Convert to JPA Project...

Figure 4: Configuring the project as a Maven project

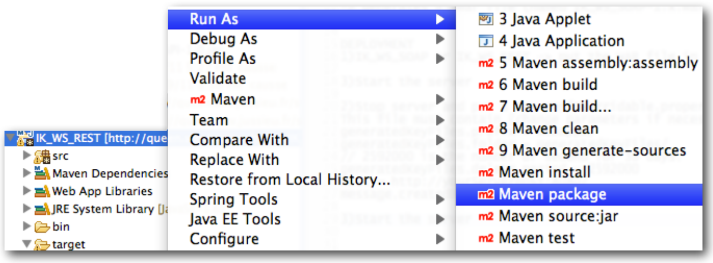


Figure 5: Building a project