Java EE 6 Hands-On Lab
Enterprise JavaBeans (EJB) 3.1 & JAX-RS 1.1

Alexis Moussine-Pouchkine
Oracle Corporation

Table of Contents

Introduction

Back in 2006, Java EE 5 had really three themes: Ease of Development, Ease of Development and Ease of development! The new Java EE Platform version 6 (released late 2009) builds on these major improvements and offers substantial enhancements to existing technologies (new features, simplifications) but also adds some new capabilities such as BeanValidation, JAX-RS (RESTful Web Services), and the new and exciting Context and Dependency Injection (CDI) specification.

This hands-on lab (HOL) is intended to give you a taste of how simple yet powerful the platform has become. It covers EJB 3.1 and JAX-RS 1.1. The other two labs cover JPA 2.0, CDI 1.0, Servlet 3.0 and JSF 2.0.

Setup Requirements

The source code for this Hands-On Lab is available in the form of a NetBeans project available directly from the virtual image this lab is using. It could easily be turned into a maven project to ease the portability across several development tools and environments.

NetBeans 6.9.1 is installed as part of the virtual machine and is probably the best out-of-the-box experience for learning Java EE 6. Eclipse 3.6 (helios), the Oracle Enterprise Pack for Eclipse (OEPE, which contains the GlassFish 3.x plugin), or even IntelliJ are all good alternate options.

This lab requires a **Java EE 6 Web Profile** container. We'll use **GlassFish 3.0.1** for this lab as this is both the reference implementation for the Java EE platform as well as a production-quality open source. The code developed here should obviously work on other Java EE 6 containers when they are made available.

NetBeans 6.9.1 is available from http://netbeans.org/downloads. The “Java” or “Full” downloads will ship with everything you need, including the GlassFish application server. A standalone version of GlassFish 3.0.1 is available from : http://glassfish.org/downloads/3.0.1-final.html

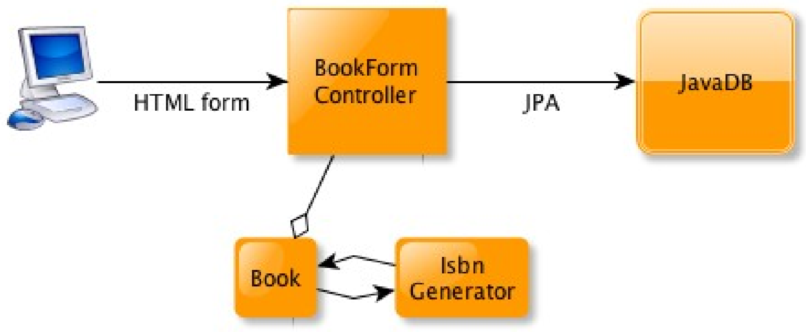
A browser such as **Firefox** and a command-line HTTP client (**wget**, **cURL**) are also required and installed.

Exercise 1 – Enterprise JavaBeans (EJB) 3.1

This part of the lab will walk you through the basic features and benefits of Enterprise JavaBeans in their current form (version 3.1), from simpler packaging in WAR archives, to implicit transactional behavior for leaner code, to singletons, and easy in-container testing.

To get started we'll use here a small web application to populate a books database using a simple HTML form, a servlet extracting the form parameters and storing a new book entity in a JavaDB database using JPA. Note that `BookFormController.java` is using Servlet 3.0's new `@WebServlet` annotation which defines and maps the servlet, thus making `web.xml` optional.

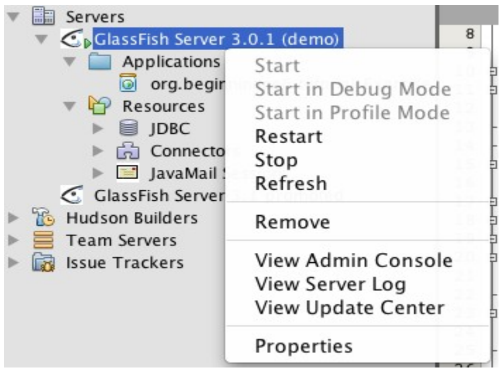
`Book.java` defines a JPA Entity and `IsbnGenerator.java` is a helper **singleton EJB** (new in EJB 3.1) used by the servlet to set the books ISBN number. Note that this EJB is also annotated with `@Startup` to have the container create a single instance of the component at startup of the application.



The JPA engine is configured using `META-INF/persistence.xml` with a persistence unit called "`BookPU`" which uses "`jdbc/__default`" as the DataSource (defined by default in GlassFish), and EclipseLink as the provider (again, the default in GlassFish).

Start off by running the project (right-click > "Run") and exercise the application by entering a few books.

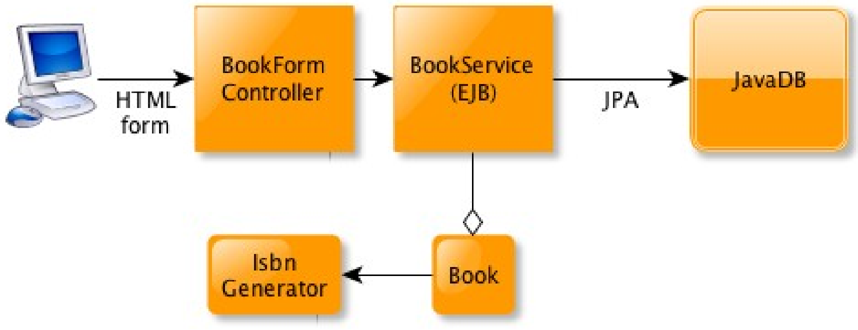
You can control the web context URL using the runtime project properties. Right from NetBeans, GlassFish can be controlled (started, stopped, applications deployed, ...) in the “Servers” section of the “Services” tab :



Use the “`View Server Log`” option to observe application logging messages as well as for troubleshooting information. The raw log file is located in :
`$GLASSFISH_HOME/glassfish/domains/domain1/logs/server.log`

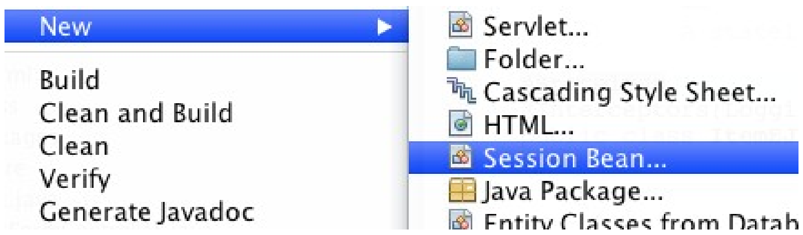
Since servlets are not thread-safe, the code in the servlet needs to acquire a JPA `EntityManager` using a `EntityManagerFactory` and explicitly control the transactional behavior using `EntityTransaction`.

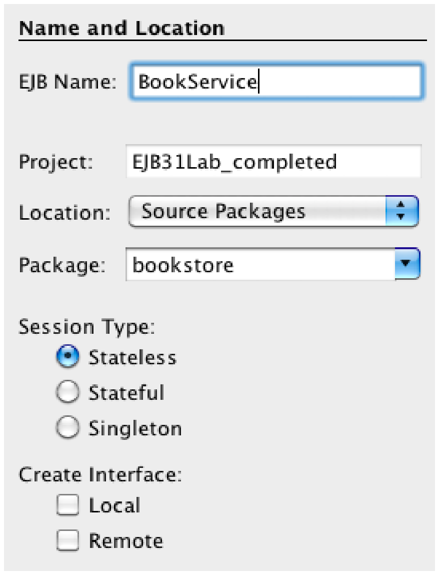
To offer better separation of concerns we'll introduce an EJB. The servlet retains its web front-end responsibility of gathering the user input while the transactional EJB component will now offer a clean persistence interaction removing the boiler-plate transactional code the servlet had to deal with. This is what our application architecture will look like with the new EJB component :



Create a stateless EJB

Right-click on the project, select "New" and select "New **Session Bean**...". Name the EJB "`BookService`" and use "`bookstore`" as the package name. Make sure you keep the other defaults :





Name and Location

EJB Name: BookService

Project: EJB31Lab_completed

Location: Source Packages

Package: bookstore

Session Type:
- [✓ Stateless]
- Stateful
- Singleton

Create Interface:
- Local
- Remote

Before editing the generated code, note the following new features offered by EJB 3.1 :

- you've just added an **EJB in a Web Application** (will be deployed as a WAR), removing the EJB-JAR packaging requirement.
- the EJB is now a single file with no interface, neither local nor remote. This is called the **no-interface view**.

As you can see, the overall packaging is simplified and number of artifacts significantly lowered. The EJB code is not only cleaner than having it inlined in the servlet, it is also **simpler** :

- it uses two injections to get access to the EntityManager and to the IsbnGenerator
- it makes calls the the EntityManager without worrying about the transaction context as EJB methods are implicitly all transactional

Here is the complete source for `BookService.java` :

```
package bookstore;

import java.util.List;
import javax.ejb.*;
import javax.persistence.*;

@Stateless
public class BookService {
    @PersistenceContext
    private EntityManager em;

    @EJB
    private IsbnGenerator isbnGenerator;

    public Book createBook(Book book) {
        book.setIsbn(isbnGenerator.generateIsbn());
        em.persist(book);
        return book;
    }

    public List<Book> findAllBooks() {
        return em.createNamedQuery("findAllBooks").getResultList();
    }

    public List<Book> findAllJavaBooks() {
        return em.createNamedQuery("findAllJavaBooks").getResultList();
    }
}
```

With all those refactorings, you will probably find the “Fix Imports” feature to be useful (right-click on the source code to get to it or check the keyboard binding in the troubleshooting section).

Refactor to servlet to use the EJB

First you need to get a hold of the EJB.

For that **replace** the injection of the `IsbnGenerator` with an injection of the newly created **`BookService`** EJB :

```
public class BookFormController extends HttpServlet {
    // inject a reference to the book service
    @EJB
    BookService bookService;
```

Similarly **remove** the call to the IsbnGenerator :

~~book.setIsbn(isbnService.generateIsbn());~~

and make a simple **call to the book service** :

```
    // delegate the database access to the EJB
    bookService.createBook(book);
```

adjust the calls to the named queries :

~~List<Book> books = em.createNamedQuery("findAllBooks").getResultList();~~

```
  List<Book> books = bookService.findAllBooks();
```

~~books = em.createNamedQuery("findAllJavaBooks").getResultList();~~

```
  books = bookService.findAllJavaBooks();
```

Finally, **remove** the following obsolete code :

```
~~// Gets an entity manager and a transaction~~
~~EntityManagerFactory emf = Persistence.createEntityManagerFactory("BookPU");~~
~~EntityManager em = emf.createEntityManager();~~
~~EntityTransaction tx = em.getTransaction();~~
~~// Persists the book to the database~~
~~tx.begin();~~
~~em.persist(book);~~
~~tx.commit();~~
 ...
~~em.close();~~
~~emf.close();~~
```

You can now run the application again to verify that you didn't break anything while refactoring the code. For that, you simply need to refresh the welcome page in the browser since all the changes were incrementally deployed to the server (see the troubleshooting section for details on this).

All this refactoring makes the code more readable and more maintainable.

Testing your EJB

As a last part for this EJB section of the lab, we'll test the EJBs using the new and standard **`EJBContainer`** API. This new feature allows for in-container testing of EJB's from a Java SE environment (JUnit, Maven, …).

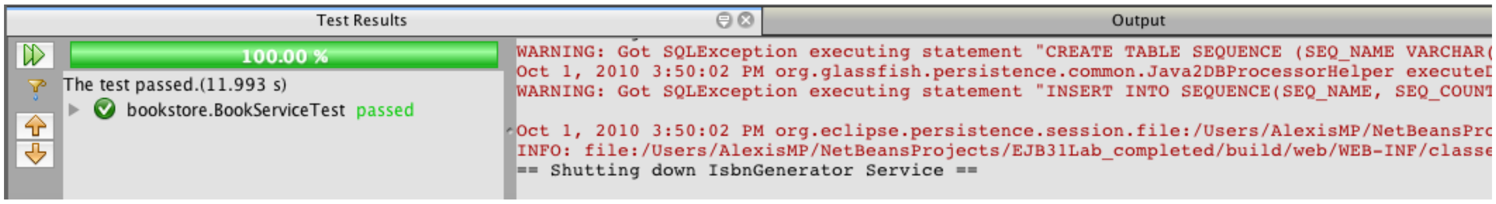
A sample JUnit test using **`EJBContainer`** is provided for you in the "`Test Packages`" section of the project, see **`BookServiceTest.java`**. The `@BeforeClass` method initializes the EJB container using the `EJBContainer.createEJBContainer()` no-argument factory and by deriving the JNDI context. These two attributes will be recycled across multiple tests in this class.

The test itself simply grabs a reference to the EJB using JNDI :

```
BookService bookService = (BookService)
        ctx.lookup("java:global/classes/BookService");
```

before it goes off to test the behavior of the component. Note that the JNDI name used here is a Global JNDI name which has been standardized across containers, making the above test fully portable.

To run the test, simply select "`Test`" on a right-click on the project node. You should see something similar to this output window :



The test passed.(11.993 s)
bookstore.BookServiceTest passed

```
WARNING: Got SQLException executing statement "CREATE TABLE SEQUENCE (SEQ_NAME VARCHAR(
Oct 1, 2010 3:50:02 PM org.glassfish.persistence.common.Java2DBProcessorHelper executeD
WARNING: Got SQLException executing statement "INSERT INTO SEQUENCE(SEQ_NAME, SEQ_COUNT

Oct 1, 2010 3:50:02 PM org.eclipse.persistence.session.file:/Users/AlexisMP/NetBeansPro
INFO: file:/Users/AlexisMP/NetBeansProjects/EJB31Lab_completed/build/web/WEB-INF/classe
== Shutting down IsbnGenerator Service ==
```

This concludes the EJB part of this lab. Other new EJB 3.1 features include the `@Schedule` annotation (a declarative access to the older timer API), as well as a simple to implement asynchronous behavior which can considered as be a replacement to JMS if all you need is asynchrony (and not other qualities such a guarantied delivery or persistent message storing).

Exercise 2 – JAX-RS

Turning the EJB into a REST service

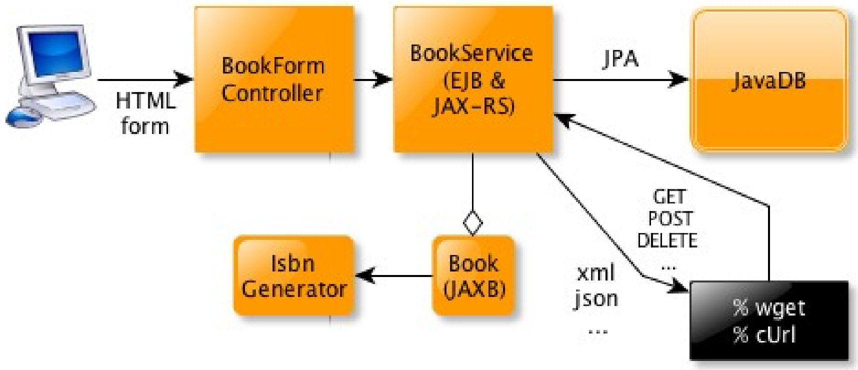
JAX-RS is a simple and elegant yet powerful technology to develop RESTful Web Services. The simplest form is a POJO decorated with a **`@javax.ws.rs.Path`** annotation. In our case we will leverage one of the new features of JAX-RS 1.1 which is to combine the EJB and JAX-RS functionalities (version 1.0 of JAX-RS had been released earlier but was not yet part of the Java EE platform). For more advanced users, a JAX-RS resource can also be a ManagedBean (JSR 250) or a CDI bean (`@ApplicationScoped` for instance). Jersey is the name of the JAX-RS reference implementation used here and is part of the GlassFish application server, so there is no runtime configuration required.

This part of the lab (the NetBeans project is called `JAXRSLab`) picks up where part 1 (EJB) ended with only a new straightforward `getBook(Long bookKey)` method added to the `BookService` :

```
public Book getBook(Long bookKey) {
    Book theBook = em.find(Book.class, bookKey);
    return theBook;
}
```

and an **explicit `web.xml`** deployment descriptor (in `Web Pages > WEB-INF`) with a Jersey servlet defined and mapped in **code you'll need to un-comment**. Part 1 took advantage of the new Servlet 3.0 feature that made `web.xml` optional with the introduction of the `@WebServlet` annotation.

This is what our application architecture will look like once we've added RESTful capabilities to our existing BookService :



Exposing a Java object as a RESTful resource

In `BookService.java`, add the following `@Path` annotation to promote the class to a JAX-RS resource (and adjust your imports with `Ctrl+Shift-I` to include `javax.ws.rs.Path`) :

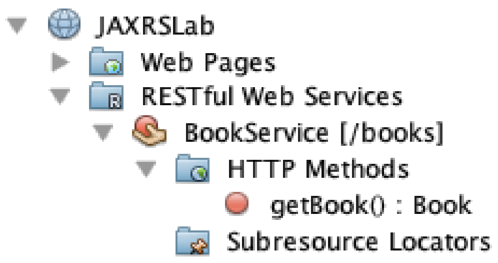
```
@Stateless
@Interceptors
@Path("/books") // resource available from the /books path
public class BookService {
```

and modify its `getBook()` method to use the following JAX-RS annotations (make sure **all** the imports use classes prefixed with `javax.ws.rs`) :

```
@GET // HTTP's GET verb/operation
@Path("{bookKey}") // specializes the path with a parameter
@Produces(MediaType.APPLICATION_XML) // mime-type
public Book getBook(@PathParam("bookKey") Long bookKey) {
```

The bookKey variable will be initialized with the parameter extracted from the requested URL (`123` for instance with URL `http://localhost:8080/.../items/123`).

The above code changes will trigger a new “RESTful Web Services” node in the NetBeans project to appear :

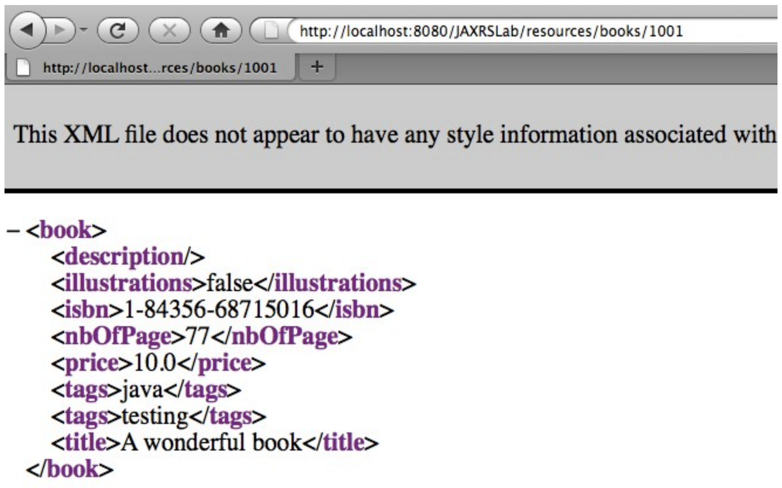


Finally, place an **`@XmlRootElement` annotation** on the `Book` class (alongside the `@Entity` and `@NamedQueries` annotations). This will cause JAXB to be responsible for doing all the heavy-lifting in producing and consuming json and XML representations served by Jersey.

Accessing the RESTful resources

Now let's try this all out and **run the project**. Once the welcome page (the book form) appears in the browser and enter a new book, you'll notice that the servlet now generates a **hyperlink to each entity** (small change in `BookFormController.java` if you care to look at it).

Click on any of these links and observe the XML representation of the resource served (the URL should be in the form of `http://localhost:8080/JAXRSLab/resources/books/1001`) :

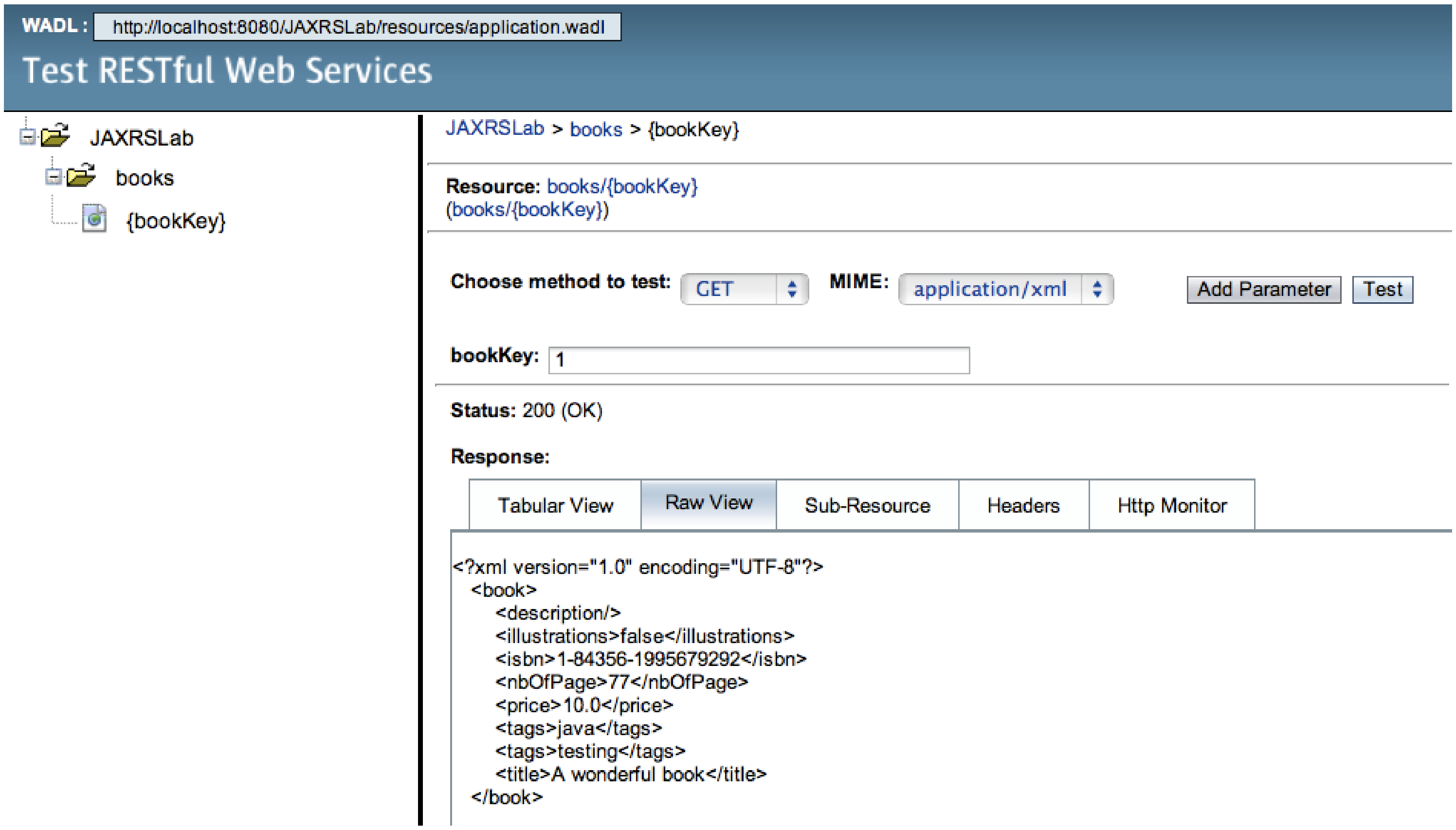

http://localhost:8080/JAXRSLab/resources/books/1001
http://localhost...rces/books/1001

This XML file does not appear to have any style information associated with

```
– <book>
    <description/>
    <illustrations>false</illustrations>
    <isbn>1-84356-68715016</isbn>
    <nbOfPage>77</nbOfPage>
    <price>10.0</price>
    <tags>java</tags>
    <tags>testing</tags>
    <title>A wonderful book</title>
  </book>
```

The `/resources` path is defined in the Jersey servlet mapping definition in `web.xml` (if you experience 404 errors, you may have missed the step above about un-commenting the Jersey servlet definition and mapping). A later part of the exercise will show you how to go without this `web.xml` deployment descriptor.

Note also that browsers such as safari or chrome will show unformatted XML. Use "view HTML source" to see the formatted version. As an alternative, you can use NetBeans' "Test RESTful Web Services" feature (right-click on the "RESTful Web Services" node) :


WADL : http://localhost:8080/JAXRSLab/resources/application.wadl
Test RESTful Web Services
JAXRSLab
books
{bookKey}
JAXRSLab > books > {bookKey}
Add Parameter
Test
Tabular View
Raw View
Sub-Resource
Headers
Http Monitor

Resource: books/{bookKey} (books/{bookKey})

Choose method to test: GET MIME: application/xml

bookKey: 1

Status: 200 (OK)

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <book>
     <description/>
     <illustrations>false</illustrations>
     <isbn>1-84356-1995679292</isbn>
     <nbOfPage>77</nbOfPage>
     <price>10.0</price>
     <tags>java</tags>
     <tags>testing</tags>
     <title>A wonderful book</title>
  </book>
```

For the time being, whatever the MIME type, the answer will always contain an XML representation. Also note that GET is the only method implemented in this lab (others will cause `HTTP 405 – Method not allowed` errors).

Support multiple media types and use a command-line client

Using a browser to access a RESTful resource is of limited use, so we'll move to using a command-line HTTP client to illustrate further JAX-RS features.

In `BookService.java`, modify the `@Produces` annotation to offer json as an additional output format (this is the only thing required to do, courtesy of JAXB and the earlier `@XmlRootElement` annotation) :

```
@Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
```

Saving the source file will cause the application to redeploy one more time. Now you can open a terminal window and use the cURL tool to emit the following requests from the command line :

1. `curl http://localhost:8080/JAXRSLab/resources/books/1001`
2. `curl -H "Accept:application/json" http://localhost:8080/JAXRSLab/resources/books/1001`
3. `curl -H "Accept:application/xml" http://localhost:8080/JAXRSLab/resources/books/1001`

- The first command will return an XML representation of book Id 1001 (adjust to your own book IDs)
- The second command uses an HTTP "`Accept:`" header and will return a JSON representation of that same book.
- The third command explicitly switches back to an XML representation.

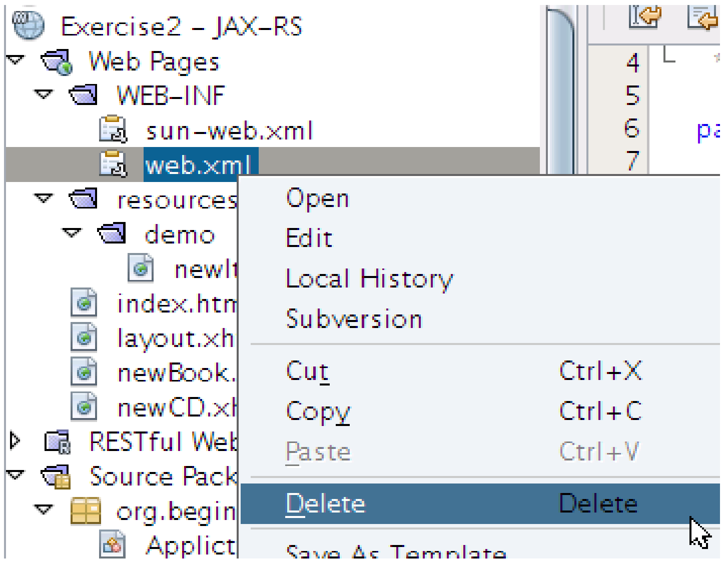
```
% curl -H "Accept:application/json" http://localhost:8080/JAXRSLab/resources/books/1001
{"description":"","illustrations":"false","isbn":"1-84356-
68715016","nbOfPage":"77","price":"10.0","tags":["java","testing"],"title":"A wonderful
book"}

% curl -H "Accept:application/xml" http://localhost:8080/JAXRSLab/resources/books/1001
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><book><description></description><illustrations>false</illustrations><isbn>1-84356-
68715016</isbn><nbOfPage>77</nbOfPage><price>10.0</price><tags>java</tags><tags>testing</tag
s><title>A wonderful book</title></book>
```

Remove `web.xml`

Finally, as promised, to get rid of the `web.xml` configuration file, you can use a new feature of JAX-RS 1.1 which is to create the following empty class in the `bookstore` package :

```
@javax.ws.rs.ApplicationPath("resources")
public class ApplicationConfig extends
                     javax.ws.rs.core.Application {
```

This maps the JAX-RS servlet (Jersey in our case) to the `"resources"` path and `web.xml` can be deleted altogether :



You can now run the application one more time to verify that everything is style working as previously since this is merely a refactoring to remove XML configuration code.

Conclusion

Hopefully this hands-on lab will have you curious enough to go download GlassFish and try out Java EE 6 for yourself. Start developing Java EE 6 applications today!

Make sure you also check out the other two labs on JSF 2.0, CDI 1.0, JPA 2.0 and Servlet 3.0.

See you also on http://beginningee6.kenai.com/ for a more complete set of labs of new Java EE 6 features.

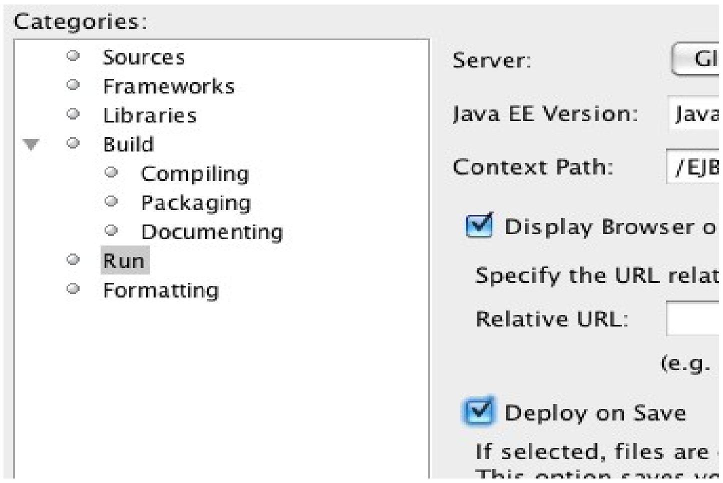
Troubleshooting

Do I need to recompile and redeploy every time I make a change?

No! One of the productivity features of the combined NetBeans+GlassFish toolset is that, after the initial application deployment (`Run` or `Deploy`), simply saving any file such as static HTML/CSS or any Java source (servlet, EJB, POJO, …) triggers an incremental compile by the IDE and a fast redeploy to the GlassFish application server. All you have to do after saving the file is reload the page in the browser. Try it! In this entire lab, the round-trip modification should take a fraction of a second.

In addition, if you develop stateful applications with `HTTPSession`, the GlassFish application server will preserve the content of the session across redeploys.

This default “Deploy on Save” behavior in NetBeans can be modified on a per-application basis using the “Properties” window of a project :



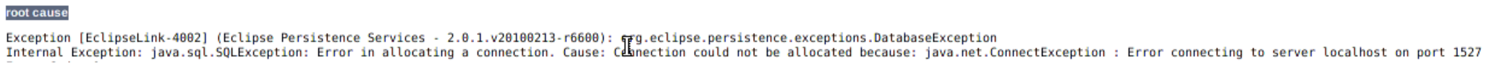
Server: Gl
Java EE Version: Java
Context Path: /EJB
✓ Display Browser o
Specify the URL relat
Relative URL:
(e.g.
✓ Deploy on Save
If selected, files are
This option saves yo

The same features are available when using Eclipse.

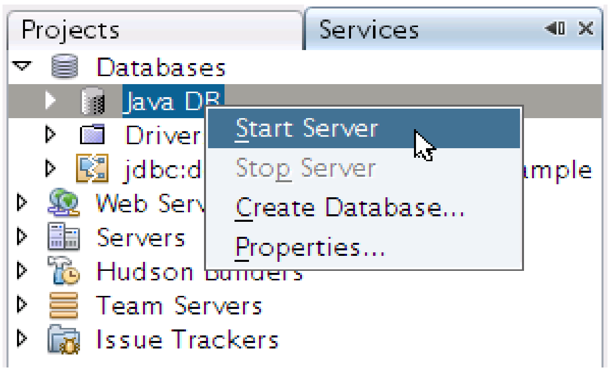
Can I use the IDE debugger?

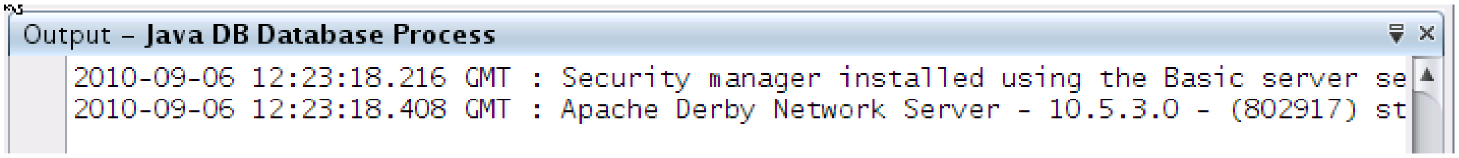
Yes, simply place a breakpoint and right-click on the project to select “`Debug`”. If necessary, the GlassFish application server will be restarted in debug mode for you.

Error connection to server localhost on port 1527



```
root cause

Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.0.1.v20100213-r6600): org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: java.sql.SQLException: Error in allocating a connection. Cause: Connection could not be allocated because: java.net.ConnectException : Error connecting to server localhost on port 1527
```



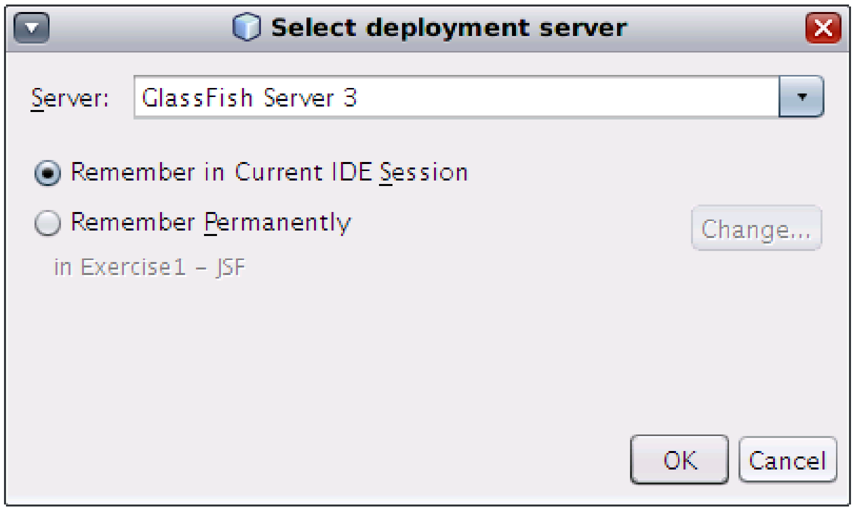


```
2010-09-06 12:23:18.216 GMT : Security manager installed using the Basic server se
2010-09-06 12:23:18.408 GMT : Apache Derby Network Server - 10.5.3.0 - (802917) st
```

Port 1527 is the database (JavaDB/Derby) default port. If the database isn't already started locally, you'll need to start it straight from the NetBeans IDE using `Services > Databases > Java DB > Start Server` as shown in the following screenshots.

Select deployment server

On the first run of each exercise you may be asked to select which server instance you want to deploy to. You are presented with a list of servers defined in the NetBeans IDE. For this lab, there should be only one choice. For faster development turnaround you are encouraged to ask the IDE to remember that choice.



Server: GlassFish Server 3

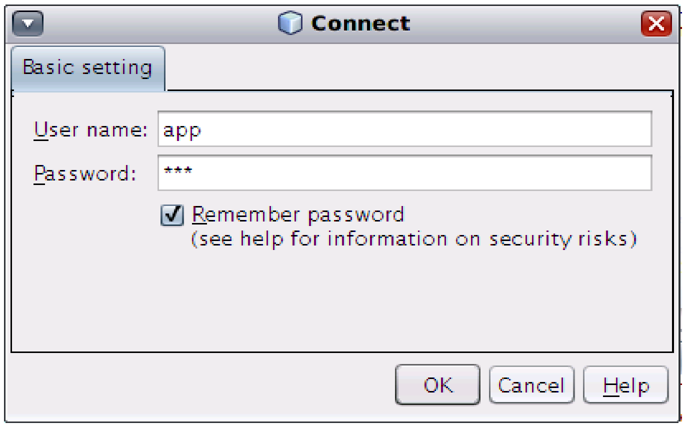
(●) Remember in Current IDE Session

( ) Remember Permanently

in Exercise1 – JSF

Database access credentials

This lab uses a default database hosted in a local JavaDB database. If you're requested a user/password to navigate through the JDBC connection from the NetBeans IDE (Services tab > Database), user “APP” and password “APP” should get you in.



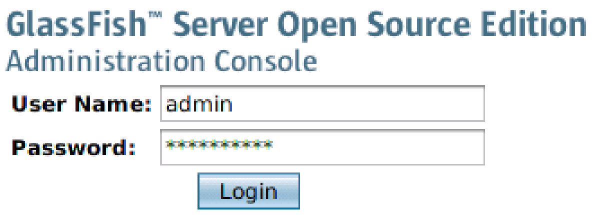
User name: app

Password: ***

[✓] Remember password
(see help for information on security risks)

Accessing the Application Server (GlassFish) Admin Console

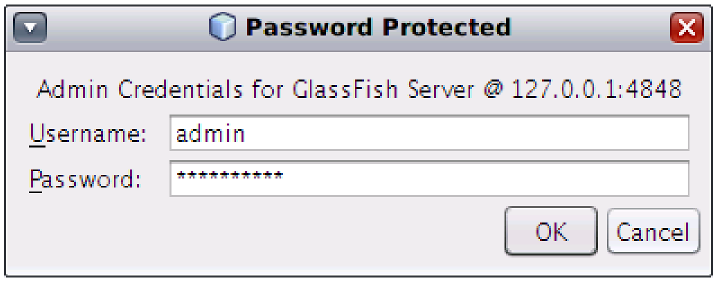
If you need to log into the GlassFish web administration console (available from the IDE or directly from `http://localhost:4848`) the default user is “`admin`” and the default password is “`adminadmin`”.



User Name: admin
Password: ***********

Admin Credentials for GlassFish Server

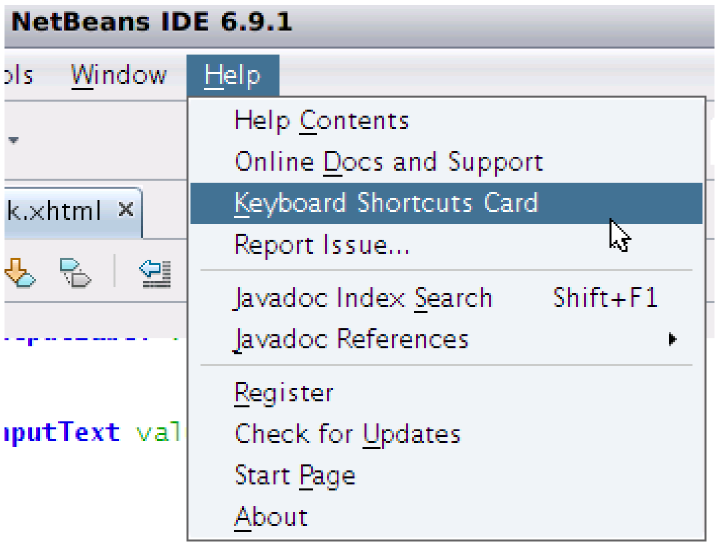
If the IDE asks you for admin credentials for the GlassFish application server (it may happen once when you start using NetBeans), you'll need type in the following: default user is “`admin`” and the default password is “`adminadmin`”.



Admin Credentials for GlassFish Server @ 127.0.0.1:4848
Username: admin
Password: ***********

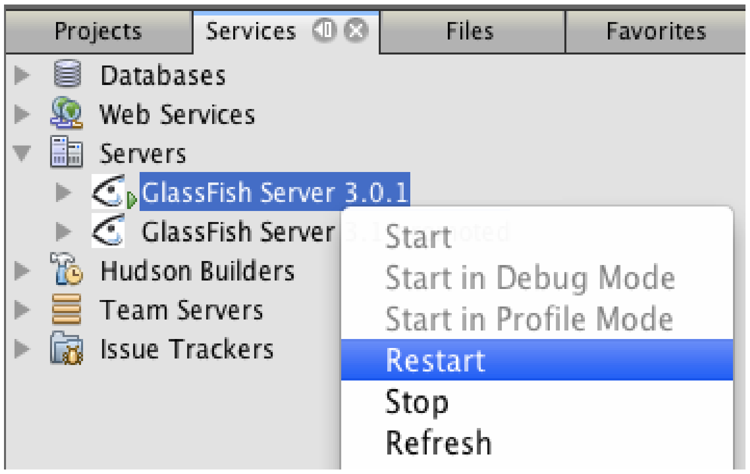
Where can I find a list of shortcuts for the NetBeans IDE?

Simply go to `Help > Keyboard Shortcuts Card`. Shortcuts can be customized in the IDE preferences and defaults vary on different platforms (Mac vs. Windows vs. *nix).



Ok, so GlassFish is behaving really bad and I'd like to restart it, how do I do it?

From the NetBeans IDE (Services Tab), simply right-click on the GlassFish Server icon and select “Restart”. Note that restarting the server isn't necessary under normal conditions.



Ok, get me to the solution!

The completed code for all three exercises is available in similarly named projects :

- `EJB31Lab_completed`
- `JAXRSLab_completed`

To get to the code, simply select File > Open Project and navigate to the appropriate project folders (located by default in the home directory). Note that these projects as configured to be deployed on a different web context than the original lab projects (using a `”_complete”` suffix).