

Working with EJBs and JBoss

This documentation expects you to...

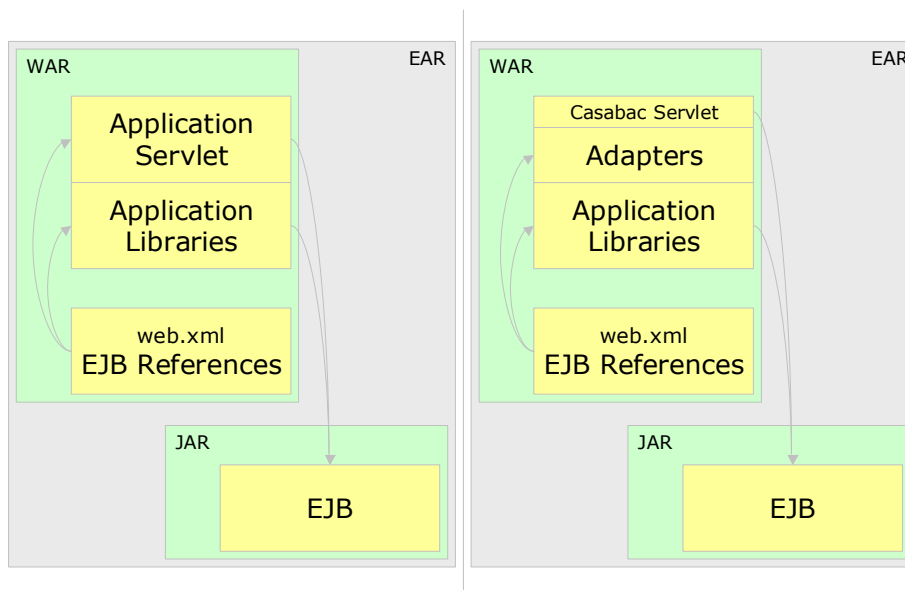
- ...be familiar with the Casabac GUI Server and the development of pages inside the Casabac GUI Server
- ...be familiar with the EJB 1.2 standards
- ...be familiar with JBoss

It is intended to serve as practical example for writing a very simple EJB scenario. In addition you will understand the basics behind and transfer them into your concrete development scenarios.

Use Casabac version 20_casabac_build_20030413 or later.

Architecture

Casabac is from architectural point of view a "just normal" Web Application – there is no difference in developing "straight Servlet/ JSP based applications" and "Casabac applications". As consequence there is also no principal difference between connecting to EJBs in a "straight Servlet/ JSP scenario" and a "Casabac scenario":



The left part of the image shows a "normal" servlet based application: either directly from the servlet or from application libraries belonging to the Web Application EJB objects are referenced and called. The EJB objects need to be named as EJB-reference inside the web application's web.xml descriptor.

The right part of the image shows the "Casabac scenario". Casabac has one central servlet to communicate with the web client. Below the servlet adapters are managed, adapters themselves may access application libraries. From "EJB connection" point of view there is no difference between left and right.

This means: EJBs are referenced the same way in Casabac as you are used to references them from normal servlet:

- Create a JNDI context object.
- Look up the home interface from the context.
- Create a bean through the home interface and work with the bean.
- Place EJB reference information into the web.xml file.

Example

In the following example a very simple stateless session bean is accessed within a Casabac adapter. The example is deployed to a JBoss 3.0.3 application server.

Step 1 – Create the Enterprise Java Bean (JAR)

The name of the EJB is "InfoFacade". It contains a method "getSystemInfo()" that will be called from the adapter later on. The remote interface, bean implementation and home class are shown below.

Remote Interface:

```
package com.casabac.demo.minibean;

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface InfoFacade
    extends EJBObject
{
    public String getSystemInfo() throws RemoteException;
}
```

Bean Implementation:

```
package com.casabac.demo.minibean;

import java.util.Date;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class InfoFacadeBean
    implements SessionBean
{
    public String getSystemInfo()
    {
        return "This is system XYZ, the time is " + (new Date()).toString();
    }
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
}
```

Home Class:

```
package com.casabac.demo.minibean;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface InfoFacadeHome
    extends EJBHome
{
    public InfoFacade create() throws RemoteException, CreateException;
}
```

The EJB descriptor file (/META-INF/ejb-jar.xml) is:

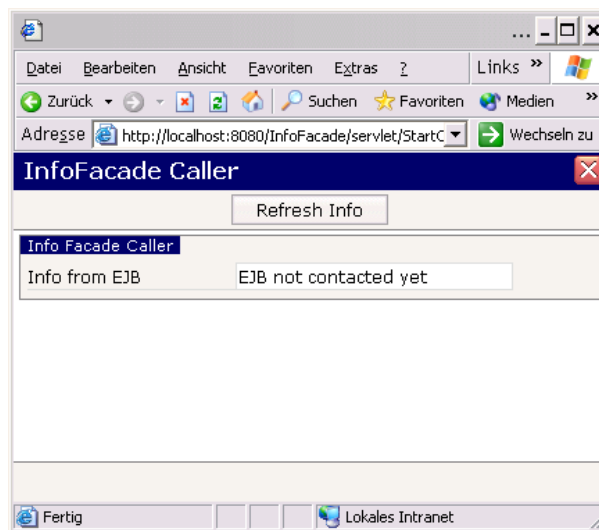
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <description>InfoFacade</description>
  <display-name>InfoFacade</display-name>
  <enterprise-beans>
    <session>
      <display-name>InfoFacade</display-name>
      <ejb-name>InfoFacade</ejb-name>
      <home>com.casabac.demo.minibean.InfoFacadeHome</home>
      <remote>com.casabac.demo.minibean.InfoFacade</remote>
      <ejb-class>com.casabac.demo.minibean.InfoFacadeBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Package everything together to form an EJB jar-File either manually, via some ANT procedure or via a deployment/packaging tool (e.g. the one that comes with the J2EE reference implementation).

Step 2 – Create the Web Application (WAR)

The Web Application is a Casabac installation in which you develop a layout using the layout manager and some classes "just as normal". We assume that the classes are stored in the Casabac project's appclasses/classes directory.

The layout looks the following way:



When pressing the "Refresh Info" button the adapter contacts the bean from the previous chapter. The text that is passed back is displayed in the field.

The layout definition is:

```
<page model="InfoFacadeCallerAdapter">
  <titlebar name="InfoFacade Caller">
  </titlebar>
  <header withdistance="false">
    <button name="Refresh Info" method="onRefreshInfo">
    </button>
  </header>
  <pagebody>
    <rowarea name="Info Facade Caller">
      <itr>
```

```

                <label name="Info from EJB" width="150">
                </label>
                <field valueprop="infoEJB" width="200">
                </field>
            </itr>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>

```

Please pay attention: do not forget to publish the layout inside the Layout Painter – otherwise no corresponding html-page will be generated to be accessed later on.

The adapter class is:

```

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import com.casabac.demo.minibean.InfoFacade;
import com.casabac.demo.minibean.InfoFacadeHome;
import com.casabac.server.Model;

public class InfoFacadeCallerAdapter
    extends Model
{
    String m_infoEJB = "EJB not contacted yet";
    public String getInfoEJB() { return m_infoEJB; }

    public void onRefreshInfo()
    {
        m_infoEJB = "";
        try
        {
            Context jndiContext = new InitialContext();
            Object homeObj = jndiContext.lookup("InfoFacade");
            InfoFacadeHome home = (InfoFacadeHome)
                PortableRemoteObject.narrow(homeObj, InfoFacadeHome.class);
            InfoFacade infoFacade = home.create();
            m_infoEJB = infoFacade.getSystemInfo();
        }
        catch (Throwable t)
        {
            t.printStackTrace();
            m_infoEJB = t.toString();
        }
    }
}

```

Nothing “special” can be seen... this is just the typical way to access a bean. The bean must be references inside the web.xml descriptor file. Please pay attention: when working inside the standard Casabac delivery then the web application is “CasabacGUIServer” and the starting of Tomcat is done via the start script “GUIServer.bat” or “GUIServer.sh”. In this scenario first the web_template.xml is read, some values are replaced and the result is copied as web.xml. Consequence: if you do any changes to the web.xml then these will be overwritten with the next start of the GUI server. Put your changes into web_template.xml as consequence!

The web.xml (web_template.xml) looks as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="CasabacGUIServer">
    <servlet id="Connector">
        ...
    </servlet>
    <servlet id="StartCasabacPage">
        ...
    </servlet>
    <servlet id="StartDynamicPage">
        ...
    </servlet>
    <servlet id="Fileupload">

```

```
...
</servlet>
<servlet-mapping>
...
</servlet-mapping>

<!--
***** EJB references *****
-->
<ejb-ref>
  <ejb-ref-name>InfoFacade</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.casabac.demo.minibean.InfoFacadeHome</home>
  <remote>com.casabac.demo.minibean.InfoFacade</remote>
</ejb-ref>
</web-app>
```

All the "standard XML" of the web.xml is abbreviated with "...". At the end the EJB reference is added.

For deployment in JBoss an extra information must be added – the mapping of the EJB-reference name to a JNDI name. This is done in the file "jboss-web.xml" that is placed into the same directory as the normal web.xml file. The content of the file is:

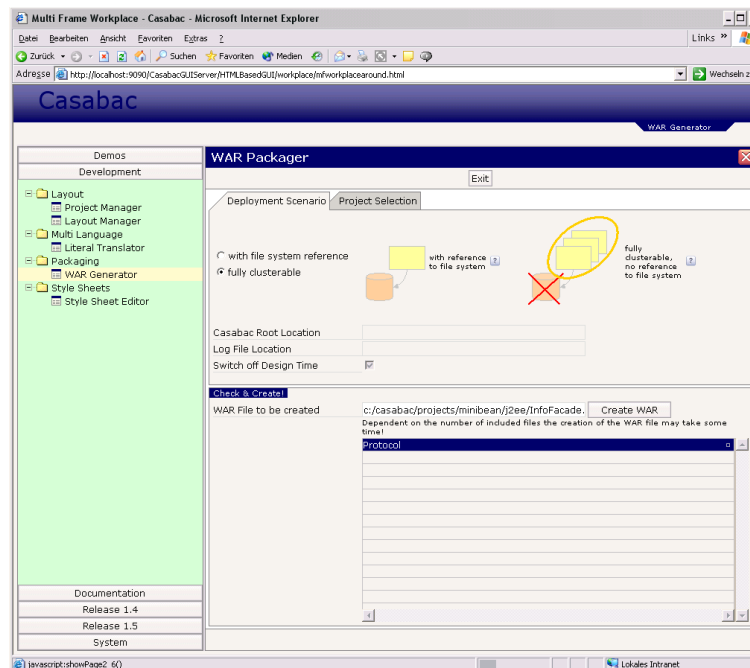
```
<jboss-web>
  <ejb-ref>
    <ejb-ref-name>InfoFacade</ejb-ref-name>
    <jndi-name>InfoFacade</jndi-name>
  </ejb-ref>
</jboss-web>
```

After having...

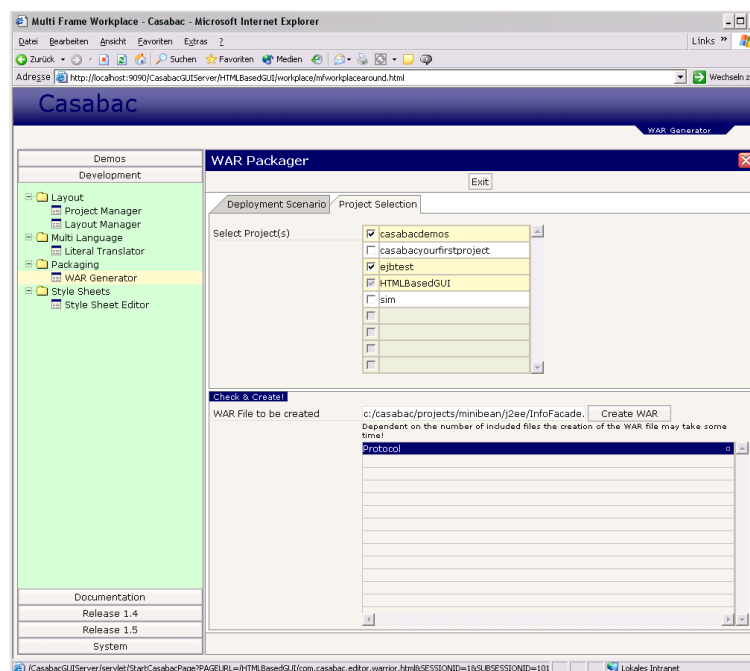
- ...built the layout and published it!
- ...developed the adapter class and compiled it!
- ...appended the EJB references to the web.xml (web_template.xml)
- ...specified the jboss-web.xml deployment file

...you now have to package the web application. This can be done in several ways, e.g. you can zip the whole web application directory into a WAR file. You can also use a certain Casabac tool: the WAR packager. This does all the "ugly stuff" for you.

Call the WAR packager inside the Casabac workplace:



Select the “fully clusterable” mode and specify the output WAR-file. Select the projects to take into the WAR-file inside the second tab. The project you have defined the layout above in and in which you have compile the adapter code should be selected..



After pressing the “Create WAR” button and waiting a while the WAR-file will be generated. This means that the following steps are automatically executed:

- The web.xml file is updated to NOT contain a link to the “home directory” (casabac.root) and not contain a link to the “log directory” (casabac.log).
- The casaconfig.xml is updated to NOT use the class loader of Casabac.

- The compiled classes under <project>/appclasses/classes and the jar files under <project>/appclasses/lib are positioned in the WAR file below WEB-INF/classes and WEB-INF/lib. This means they are reachable for the normal web application class loader by the servlet container.
- All files below WEB-INF, META-INF and below the selected project directories are zipped and the result is stored as WAR file.

Step 3 – Create the Enterprise Application (EAR)

Now all things come together. You have to create an EAR file that contains...

- above's JAR file containing the bean
- above's WAR file containing the web application

...and have to add certain information into the "application.xml" file before packaging everything to form an EAR file. The application.xml file looks as follows:

```
<?xml version="1.0" encoding="Cp1252"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN"
'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>
<application>
  <display-name>InfoFacade</display-name>
  <description>Application description</description>
  <module>
    <web>
      <web-uri>InfoFacade.war</web-uri>
      <context-root>InfoFacade</context-root>
    </web>
  </module>
  <module>
    <ejb>InfoFacade.jar</ejb>
  </module>
</application>
```

The context-root is the one to be used later on when inputting the URL into the browser (in this example: <http://server:port/InfoFacade/...>).

Step 4 – Deploy to JBoss

Deployment is quite simple with JBoss: just copy the EAR file created in the former step into JBoss's server/default/deploy directory. Watch for error messages in JBoss's console.

Step 5 – Test

Access the page via the URL:

<http://server:port/InfoFacade/servlet/StartCasabacPage?PAGEURL=/ejbtest/InfoFacadeCaller.html>.

Development Process

After having successfully managed this first example you should take some time to properly set up your development process:

- Where are your files stored?
- How is generation of archive files done? Manually? Via ANT? Via some batch files? Via a deployment tool?
- How is all this synchronized with your central storage of development objects (e.g. your CVS system)?

Goal should be an infrastructure...

- ...in which the number of manually driven steps is minimized
- ...in which the speed of development and deployment is maximized

...in order to not having too many time burned between changing your adapter code and testing against your EJBs.