Redpaper

Martin Keen
Rafael Coutinho
Sylvi Lippmann
Salvatore Sollami
Sundaragopal Venkatraman
Steve Baber
Henry Cui
Craig Fleming

Developing Web Applications using JavaServer Faces

This IBM® Redpaper™ publication introduces the features, benefits, and architecture of JavaServer Faces (JSF), a framework that simplifies building user interfaces for web applications. This paper is intended for web developers interested in JSF. The paper includes an example application that uses JSF, with persistence implemented in the Java Persistence API (JPA).

The paper also demonstrates the support for JSF that is available in IBM Rational® Application Developer for IBM WebSphere® Software V8, the Eclipse 3.6 technology-based platform for building Java Platform, Standard Edition Version 6 (Java SE 6), and Java Platform, Enterprise Edition Version 6 (Java EE 6) applications. Rational Application Developer focuses on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal. Rational Application Developer also provides integrated tools for all development roles, including web developers, Java developers, business analysts, architects, and enterprise programmers.

The paper is organized into the following sections:

- Introduction to JSF
- Developing a web application by using JSF and JPA
- More information

The sample code shown in this paper is available in the `4883codesolution/jsf` folder.

This paper was originally published as a chapter in the IBM Redbooks® publication, *Rational Application Developer for WebSphere Software V8 Programming Guide*, SG24-7835. The full publication includes working examples that show how to develop applications and achieve the benefits of visual and rapid application development. The publication is available at the following website:

http://www.redbooks.ibm.com/abstracts/sg247835.html?Open

Introduction to JSF

The JSF framework allows users to utilize pre-built components and easily create web applications. For example, JSF provides an Input Text, Output Text, and a Data Table component. You can add these components easily to a JSF web page and connect them to the data of your project. JSF technology and the JSF tools provided by Rational Application Developer enable even inexperienced developers to quickly develop web applications.

This section provides an overview of the following aspects of JSF:

- JSF 1.x features and benefits
- JSF 2.0 features and benefits
- JSF 2.0 application architecture
- JSF features in Rational Application Developer

JSF 1.x features and benefits

The JSF 1.*x* specification is defined in *Java Specification Request (JSR) 127: JavaServer Faces*.

The following list describes the key features and benefits of using JSF for web application design and development:

- Standards-based web application framework: JSF technology is the result of the Java Community process. JSF uses the model view controller (MVC) pattern; it addresses the view or presentation layer though user interface (UI) components, and addresses the model through managed beans.
- Event-driven architecture: JSF provides server-side rich UI components that respond to client events.
- UI development:
  - UI components are decoupled from their rendering, which means that they can be extended to use other technologies, such as Wireless Markup Language (WML).
  - JSF allows direct binding of UI components to model data.
  - Developers can use extensive libraries of prebuilt UI components that provide both basic and advanced web functionality. In addition, custom UI components can be created and customized for specific uses.
- Session and object management: JSF manages designated model data objects by handling their initialization, persistence over the request cycle, and cleanup.
- Validation and error feedback: JSF allows direct binding of reusable validators to UI components. The framework also provides a queue mechanism to simplify error and message feedback to the application user. These messages can be associated with specific UI components.
- Globalization: JSF provides tools for the globalization of web applications, including supporting number, currency, time, and date formatting, and the externalization of UI strings.

JSF 2.0 features and benefits

The JSF 2.0 specification is defined in *JSR 314: JavaServer Faces 2.0*. It builds on and extends the features that are available in JavaServer Faces 1.*x*.

This section describes the major features of JSF 2.0:

- Facelet usage
- Built-in Ajax support
- Annotation usage
- Creating templates
- New components
- Custom components

Facelet usage

JSF 2.0 uses *Facelets*, which are XHTML pages instead of JSP pages, as the view layer. Facelets relieve JSF of the restrictions that are imposed by JSP technology. For more information, see “Improving JSF by Dumping JSP” by Hans Bergsten in O’Reilly on Java.com, 9 June 2004:

`http://onjava.com/pub/a/onjava/2004/06/09/jsf.html`

Facelets: Facelets are the standard view decoration language for JSF application development in JSF 2.0. We do not recommend combining Facelets and Faces JSP pages in the same project.

Built-in Ajax support

In JSF V1.*x*, it was possible to use Ajax with JSF, but this capability required additional component libraries. Now with JSF 2.0, a JavaScript library for performing simple Ajax operations is provided. No additional libraries are required. The example application in this paper uses Ajax.

Annotation usage

With JSF 2.0, Java classes can be directly annotated, which eliminates the need to register these classes in `faces-config.xml`. For example, `@ManagedBean` indicates that this class is a Faces Managed Bean. The `@ManagedBean` annotation can be used in place of a managed-bean entry in `faces-config.xml`.

Other annotations, such as `@FacesComponent` and `@FacesRenderer`, are also available.

Creating templates

Facelet pages can be created from templates to allow a more uniform look across your project and to aid with future maintenance. We describe the creation of these templates in “Creating Facelet templates” on page 14.

New components

JSF provides various components for use in your web pages. In addition to the components that were available in JSF 1.*x* (Input Text, Output Text, Data Table, and other components), JSF 2.0 includes two new components that simplify GET navigation: <h:link> and <h:button>.

For more information about components that are available on Facelet pages, see the following website:

`http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tcrtfaceletspgcontent.html`

Custom components

Composite components build on the templating features of Facelets so that you can implement custom components. The advantage to custom components is that you can implement them without any configuration and without any Java code.

More detailed information and examples are available at the following website:

http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html

The steps for creating and customizing custom components with the Rational Application Developer are described in detail in the following information centers:

- http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tjsfover.html
- http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tcrtfaceletcomposite.html

JSF 2.0 application architecture

You can extend the JSF application architecture easily in various ways to suit the requirements of your particular application. You can develop custom components, renderers, validators, and other JSF objects and register them with the JSF run time.

This section highlights the JSF 2.0 application architecture, as shown in Figure 1.

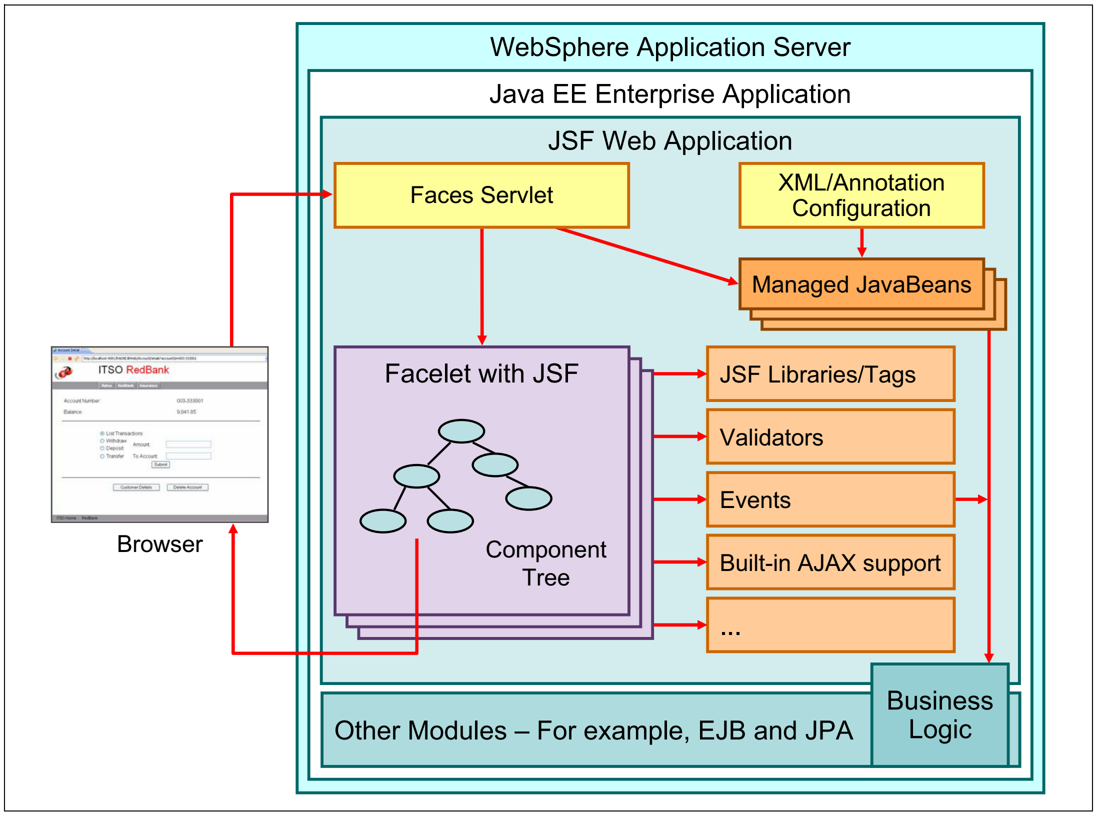


Figure 1 JSF application architecture

The JSF application architecture includes these components:

- Facelet pages: These pages are built from JSF components, where each component is represented by a server-side class.
- Faces servlet: One servlet (`FacesServlet`) controls the execution flow.
- XML/Annotation Configuration: The configuration of validators and Faces managed beans can be defined either with the XML file `faces-config.xml` or by using annotations.
- Tag libraries: The JSF components are implemented in tag libraries.
- Validators: Java classes are used to validate the content of JSF components. For example, user input can be validated according to specific business logic.
- Faces managed beans: JavaBeans are defined in the configuration file to hold the data of JSF components. Faces managed beans represent the data model and are passed between the business logic and user interface.
- Events: Java code is executed in the server for events, such as pushing a button and invoking business logic.
- Ajax: Ajax is supported by JSF 2.0 as a built-in feature.

Figure 2 represents the structure of a simple JSF 2.0 application, in this case, our `RAD8JSFWeb` project.

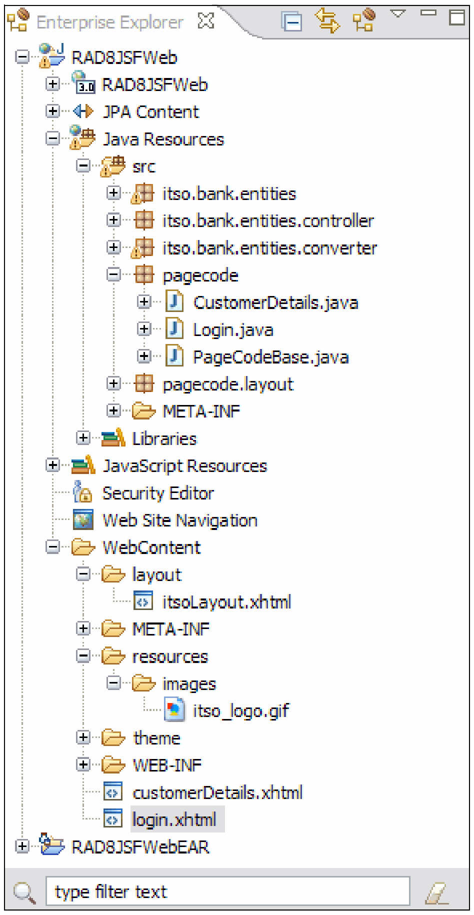


Figure 2   JSF 2.0 application structure within Rational Application Developer

JSF features in Rational Application Developer

Rational Application Developer provides a number of rich features to enhance your usage of the JSF framework:

- JSF Trace
- Integration of third-party JSF tag libraries
- Customizable data templates

JSF Trace

For help debugging your JSF application or reviewing the phases of the JSF lifecycle, you can use JSF Trace. This feature visually displays information about your application at run time, including incoming requests, error messages, and objects in the request, session, and application scopes.

JSF Trace is covered in detail in “Debug and troubleshoot JavaServer Faces applications by using JSFTrace in Rational Application Developer” by Yury Kats in IBM developerWorks®, 24 September 2009:

`http://www.ibm.com/developerworks/rational/library/09/debugjavaserverfacestraceapplicationdeveloper/index.html`

Integration of third-party JSF tag libraries

JSF has a set of standard components for building web pages. These standard components can be supplemented with components from JSF tag libraries that were created by other companies. Rational Application Developer makes it easy to integrate these third-party tag libraries, and even to customize the tooling for these tags.

For more information, see “Faces library definitions for third-party JavaServer Faces controls” by Scott Paxton in IBM developerWorks, 18 June 2009:

`http://www.ibm.com/developerworks/rational/library/09/faceslibrarydefinitionrationalapplicationdeveloper/index.html`

Customizable data templates

Rational Application Developer makes it easy to generate UI components based on the data structures of your project. Simply by adding a data source, such as a Faces managed bean, to the Page Data view and then dragging it onto your web page, you can create controls that are connected to that data source.

For even more control over the controls that are generated on your web page, see “Introduction to JavaServer Faces data templates” by Christie Rice in IBM developerWorks, 25 September 2009:

`http://www.ibm.com/developerworks/rational/library/09/intro_jfs_data_templates_rad/index.html`

Developing a web application by using JSF and JPA

In this section, we describe a web application that is implemented with JavaServer Faces (JSF) and Java Persistence API (JPA). For each Facelet that we create, a managed bean class is generated. For each action in the Facelet, a method in the managed bean class is invoked. In those methods, we use the JPA Manager Beans to retrieve the necessary data.

Rational Application Developer provides tooling to interact directly with the JPA entities without using a session bean. A JPA Manager Bean is created for each JPA entity with methods, such as find and update.

Structure of the JSF web application

The sample application consists of the following pages:

- Login page (login): Validates a customer's unique ID. If the ID is valid, display the customer details page.
- Customer details page (customerDetails): Shows details (title, first name, and last name) of a customer and the associated account balances.

You can find a completed version of the web application in the `c:\4883codesolution\jsf\RAD8JSFWeb.zip` project file.

Setting up the ITSOBANK database

The JPA entities are based on the `ITSOBANK` database. Therefore, we must define a database connection within Rational Application Developer that the mapping tools use to extract schema information from the database.

Creating the Derby database

The `\4883code\database\derby` directory provides command files to define and load the ITSOBANK database in Derby. For the `DerbyCreate.bat`, `DerbyLoad.bat` and `DerbyList.bat` files, you must install WebSphere Application Server in the `C:\IBM\WebSphere\AppServer` folder. You must edit these files to point to your WebSphere Application Server installation directory if you installed the product in a separate folder.

In the `\4883code\database\derby` directory, you can execute the following command files:

- `DerbyCreate.bat` to create the database and table
- `DerbyLoad.bat` to delete the existing data and add records
- `DerbyList.bat` to list the contents of the database

These command files use the SQL statements and helper files that are provided in the following files:

- `itsobank.ddl`: Database and table definition
- `itsobank.sql`: SQL statements to load sample data
- `itsobanklist.sql`: SQL statement to list the sample data
- `tables.bat`: Command file to execute `itsobank.ddl` statements
- `load.bat`: Command file to execute `itsobank.sql` statements
- `list.bat`: Command file to execute `itsobanklist.sql` statements

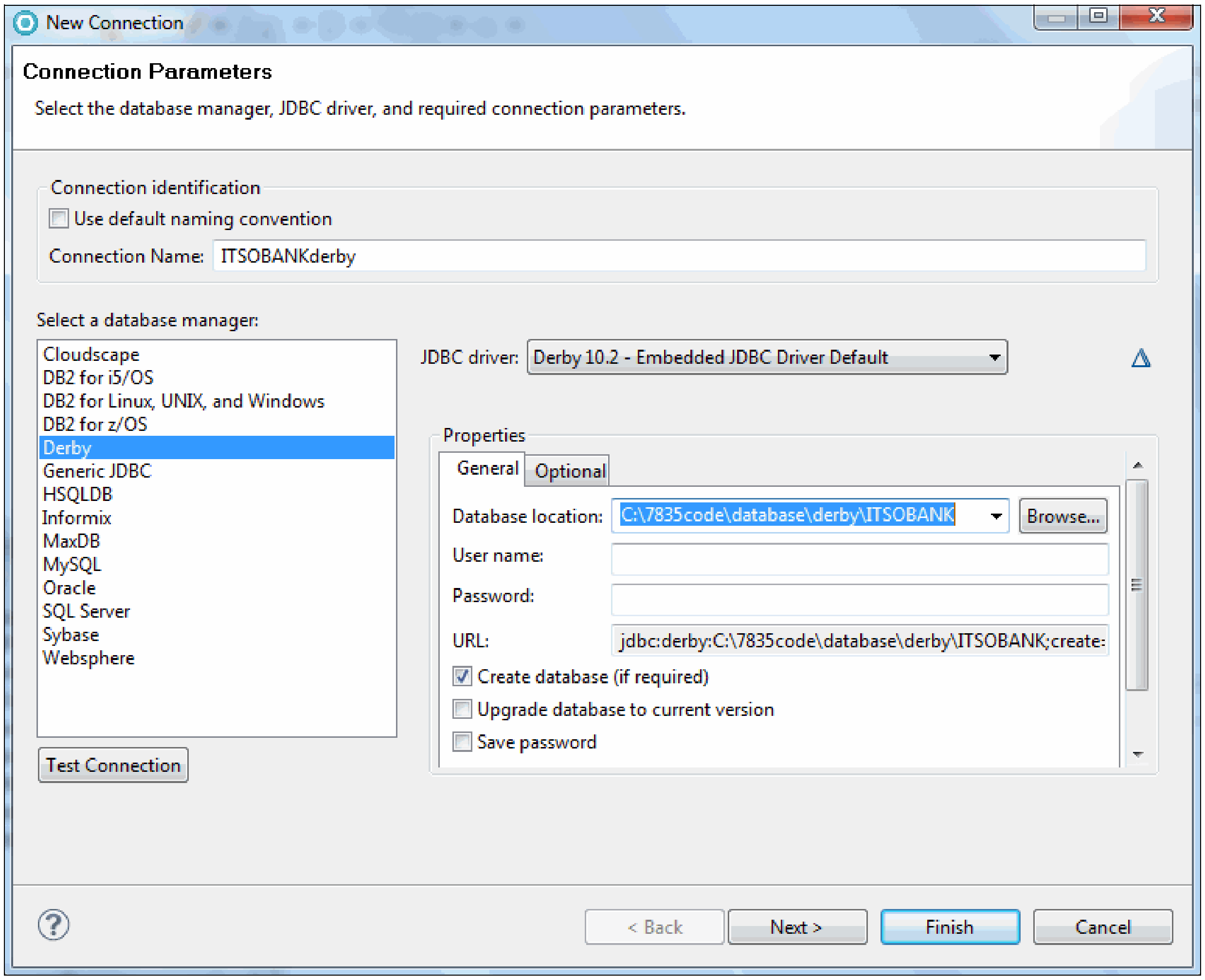
The Derby ITSOBANK database is created in the `\4883code\database\derby\ITSOBANK` directory.

Creating a connection to the ITSOBANK database

To connect to the Derby ITSOBANK database by using the New Database Connection wizard, follow these steps:

1. Because Derby allows only one connection, stop WebSphere Application Server if it is running and if you accessed the ITSOBANK database.
2. Select **Window** → **Open Perspective** → **Other** to open the Data perspective. In the Open Perspective window, select **Data** and click **OK**.

3. In the Data perspective, locate the Data Source Explorer view, which is typically in the lower left in the Data perspective.
4. In the Data Source Explorer, right-click **Database Connections** and select **New**.
5. In the New Connection wizard (Figure 3), follow these steps:
   a. Clear **Use default naming convention**, and for Connection Name, type `ITSOBANKderby`.
   b. For Select a database manager, select **Derby**.
   c. For JDBC driver, select **Derby 10.2 - Embedded JDBC Driver Default**.
   d. For Database location, click **Browse** and locate the `\4883code\database\derby\ITSOBANK` directory.
   e. Leave the User name and Password fields empty; the Derby database does not require authentication.
   f. Select **Create database (if required)**.
   g. Click **Test Connection**. A window opens and shows the "`Connection succeeded`" message. Click **OK** to close the window.
   h. Click **Next**.


New Connection

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☐ Use default naming convention

Connection Name: ITSOBANKderby

Select a database manager:

Cloudscape
DB2 for i5/OS
DB2 for Linux, UNIX, and Windows
DB2 for z/OS
Derby
Generic JDBC
HSQLDB
Informix
MaxDB
MySQL
Oracle
SQL Server
Sybase
Websphere

JDBC driver: Derby 10.2 - Embedded JDBC Driver Default

Properties

General | Optional

Database location: C:\7835code\database\derby\ITSOBANK

User name:

Password:

URL: jdbc:derby:C:\7835code\database\derby\ITSOBANK;create:

☑ Create database (if required)

☐ Upgrade database to current version

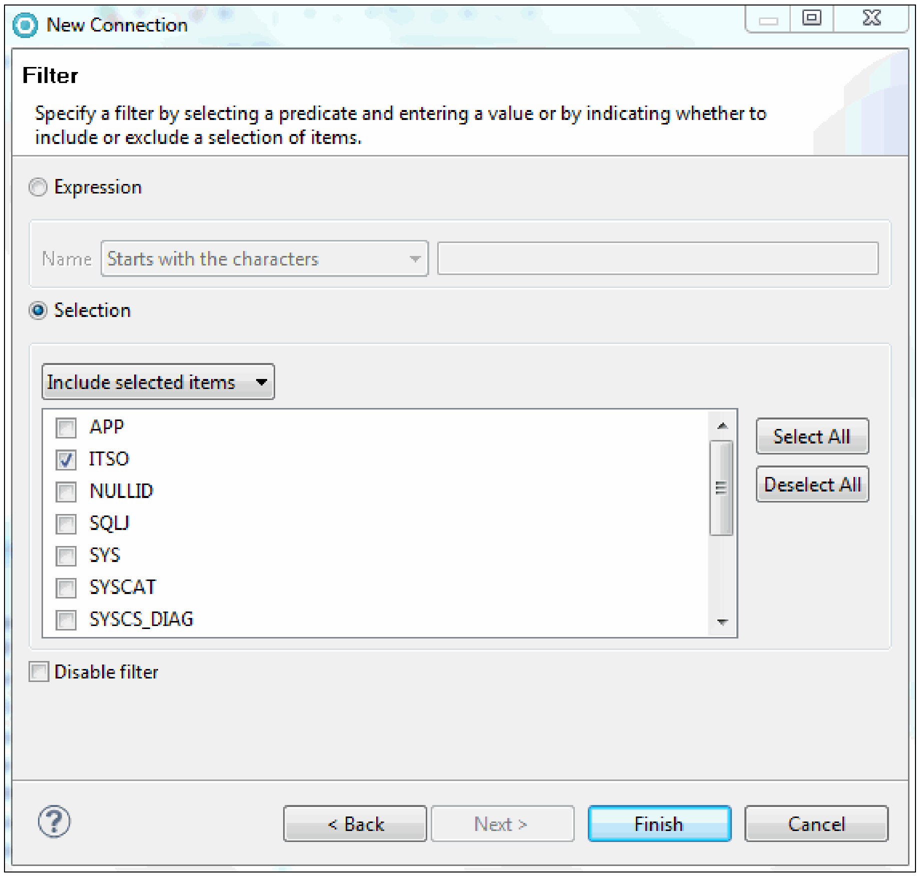
☐ Save password

Figure 3 New Connection: Connection Parameters window

You can use filters to exclude data objects, such as tables, schemas, stored procedures, and user-defined functions, from the view. Only the data objects that match the filter condition are shown.

6. To see the objects in the ITSO schema, in the Filter window (Figure 4), follow these steps:

   a. Clear the **Disable filter** check box.
   b. Select **Selection**.
   c. Select **Include selected items**.
   d. From the schema list, select **ITSO**.
   e. Click **Finish**.



New Connection

Filter

Specify a filter by selecting a predicate and entering a value or by indicating whether to include or exclude a selection of items.

( ) Expression

Name: Starts with the characters

(●) Selection

Include selected items

- [ ] APP
- [✓] ITSO
- [ ] NULLID
- [ ] SQLJ
- [ ] SYS
- [ ] SYSCAT
- [ ] SYSCS_DIAG

[ ] Disable filter

Figure 4   New Connection: Filter window

7. When the connection is displayed in the Data Source Explorer, expand **ITSOBANKderby [Apache Derby 10.5.1.1 ...]** → **ITSOBANK**. The Schemas folder is marked as `[Filtered]`. Only one schema (ITSO) is listed, and the others are filtered (Figure 5).

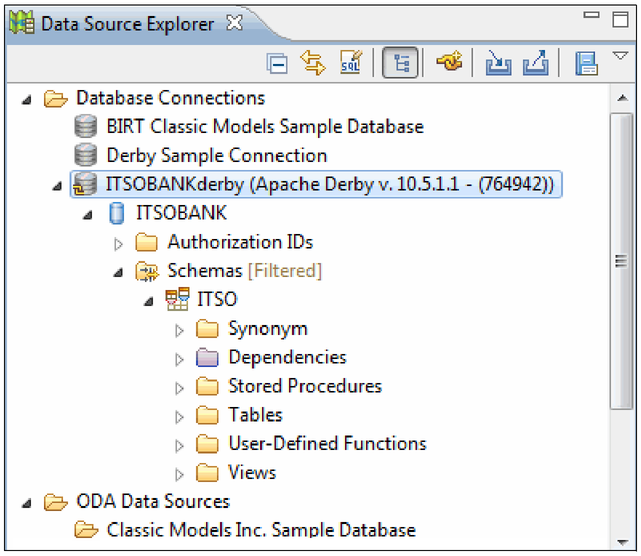


Figure 5 Connection with schema and tables in Data Source Explorer

With the filter framework, you can filter out the tables at a more granular level. Suppose that we want to see only tables that start with the letter A. Follow these steps:

1. Expand the schema **ITSO**, right-click **Tables**, and select **Filter**.
2. In the Filter window, follow these steps:
   a. Clear the **Disable filter** check box.
   b. Select **Expression**.
   c. In the Name section, select **Starts with the characters** and type `A`.
   d. Click **OK**.

   Now you can see only two tables in the Data Source Explorer: ACCOUNT and ACCOUNT_CUSTOMER.
3. To disable the filter, right-click **Tables,** select **Filters**, and select **Disable filter**. Click **OK**.

Configuring the data source

You can choose from one of the following methods to configure the data source:

- Use the WebSphere administrative console.
- Use the WebSphere Enhanced EAR, which stores the configuration in the deployment descriptor and is deployed with the application.

While developing JSF and JPA web applications with Rational Application Developer, the data source is created automatically when you add JPA-managed data to a Facelet file. The data source configuration is added to the EAR deployment descriptor.

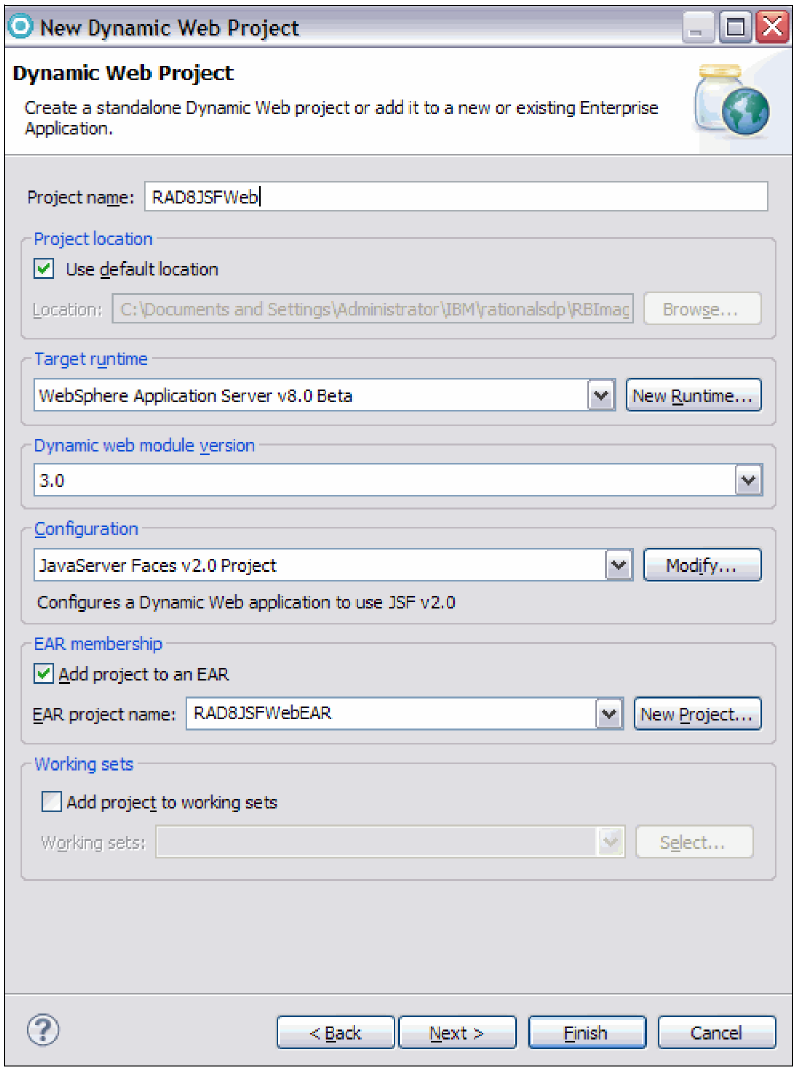
However, if you already defined the `ITSOBANKderby` data source on the server, you might experience problems, because you can have only one active connection to the database. To clear the connection, remove all applications from the server and then restart the server.

Creating the JSF Project

In this section, we describe how to create a dynamic web JSF project.

This application consists of `RAD8JSFWebEAR` (the enterprise application) and `RAD8JSFWeb` (the web application). Follow these steps:

1. In the Enterprise Explorer view, right-click and select **New** → **Project**.
2. In the New Project wizard, select **Web** → **Dynamic Web Project** and click **Next**.
3. In the New Dynamic Web Project wizard, define the project details (Figure 6):
   a. For the Project name, type `RAD8JSFWeb`.
   b. Leave **Use default location** checked.
   c. For the Target Runtime, select **WebSphere Application Server v8.0 Beta**.
   d. Leave the Dynamic web module version at **3.0**.
   e. For the Configuration, select **JavaServer Faces v2.0 Project** and click **Modify**.



Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: RAD8JSFWeb

Project location

[✓] Use default location

Location: C:\Documents and Settings\Administrator\IBM\rationalsdp\RBImag

Target runtime

WebSphere Application Server v8.0 Beta

Dynamic web module version

3.0

Configuration

JavaServer Faces v2.0 Project

Configures a Dynamic Web application to use JSF v2.0

EAR membership

[✓] Add project to an EAR

EAR project name: RAD8JSFWebEAR

Working sets

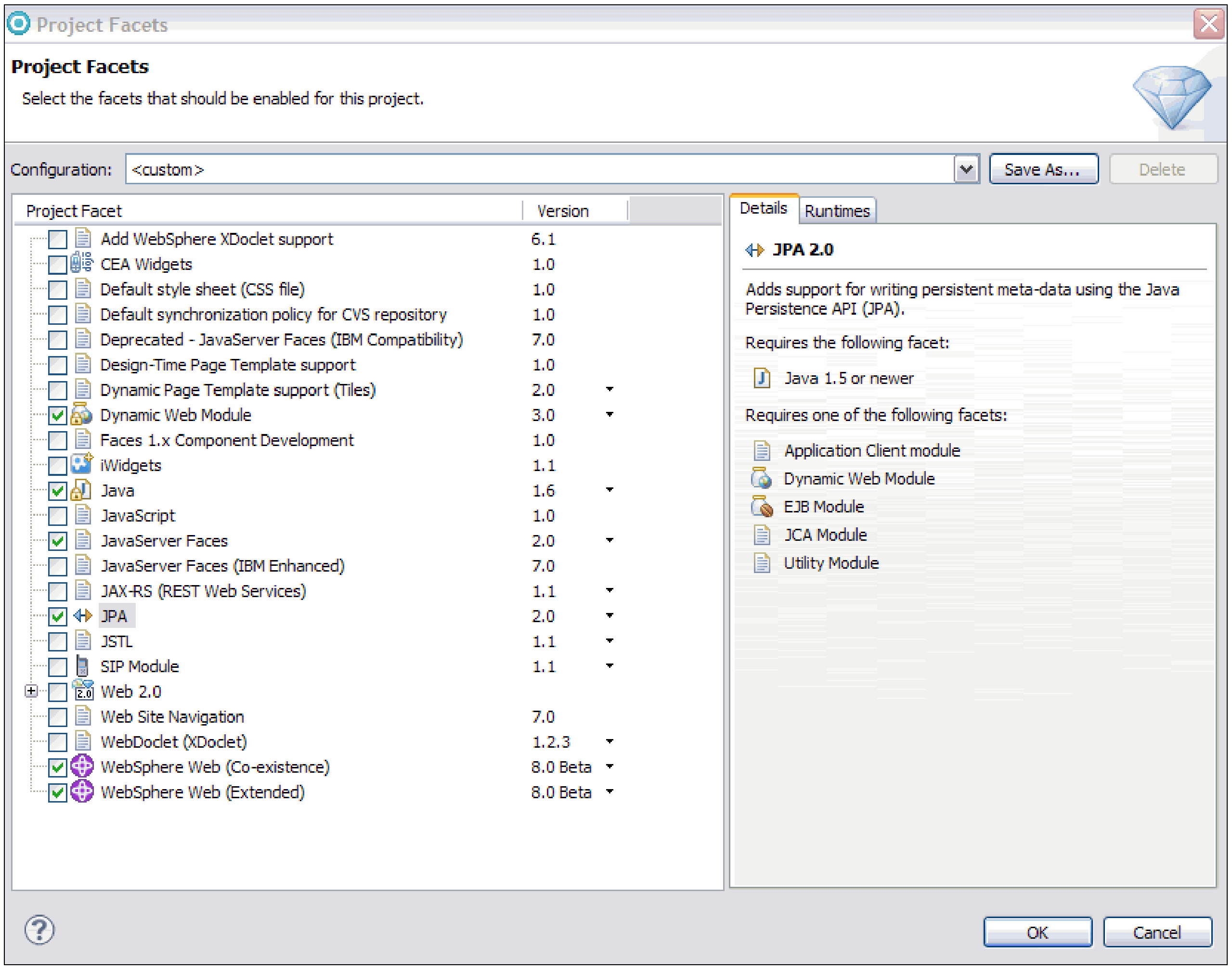
[ ] Add project to working sets

Working sets:

Figure 6 New Dynamic Web Project wizard

f. In the Project Facets window, select the following Project Facets (Figure 7):

- **Dynamic Web Module**: Preselected
- **Java**: Preselected
- **JavaServer Faces**: Preselected
- **JPA**: Select this facet
- **WebSphere Web (Co-existence)**: Preselected
- **WebSphere Web (Extended)**: Preselected


Project Facets

Project Facets

Select the facets that should be enabled for this project.

Configuration: <custom>

Project Facet	Version
[ ] Add WebSphere XDoclet support	6.1
[ ] CEA Widgets	1.0
[ ] Default style sheet (CSS file)	1.0
[ ] Default synchronization policy for CVS repository	1.0
[ ] Deprecated - JavaServer Faces (IBM Compatibility)	7.0
[ ] Design-Time Page Template support	1.0
[ ] Dynamic Page Template support (Tiles)	2.0
[✓] Dynamic Web Module	3.0
[ ] Faces 1.x Component Development	1.0
[ ] iWidgets	1.1
[✓] Java	1.6
[ ] JavaScript	1.0
[✓] JavaServer Faces	2.0
[ ] JavaServer Faces (IBM Enhanced)	7.0
[ ] JAX-RS (REST Web Services)	1.1
[✓] JPA	2.0
[ ] JSTL	1.1
[ ] SIP Module	1.1
[ ] Web 2.0	
[ ] Web Site Navigation	7.0
[ ] WebDoclet (XDoclet)	1.2.3
[✓] WebSphere Web (Co-existence)	8.0 Beta
[✓] WebSphere Web (Extended)	8.0 Beta

Details

JPA 2.0

Adds support for writing persistent meta-data using the Java Persistence API (JPA).

Requires the following facet:

Java 1.5 or newer

Requires one of the following facets:

Application Client module
Dynamic Web Module
EJB Module
JCA Module
Utility Module

Figure 7 Web project facets for JSF

g. Click **OK**. The value of Configuration in the Dynamic Web Project window changes to `<custom>`.

h. Click **Next**.

4. Accept the source folder `src` and click **Next**.
5. Select **Generate web.xml deployment descriptor** and click **Next**.
6. In the JPA Facet page, complete these steps:

a. Leave **RAD JPA 2.0 Platform** selected for the Platform.

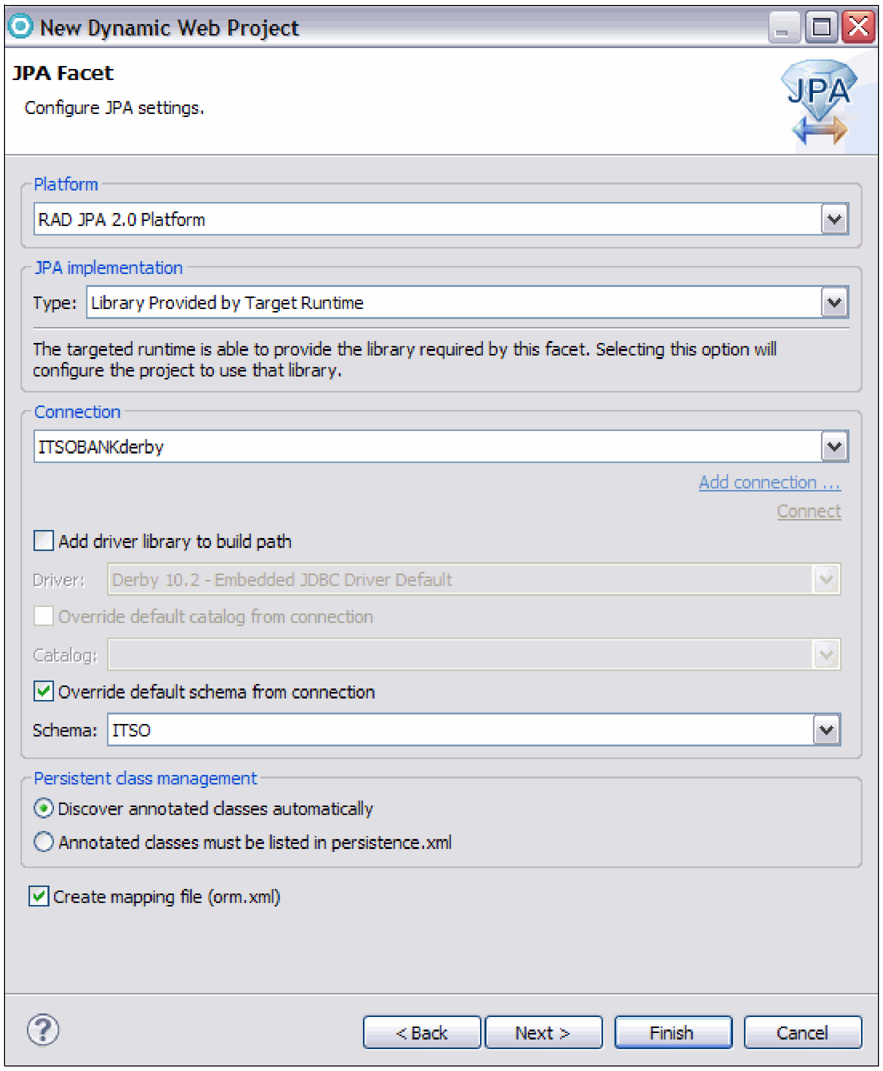
b. Leave **Library Provided Target Runtime** selected for the Type.

c. For the connection, select the **ITSOBANKderby** that was defined before. Complete these tasks:
   - Click **Connect** if the connection is not active.
   - In case you did not create the connection to the `ITSOBANK` database yet, click **New Connection** to define it.

d. Select **Override default schema from connection** and select **ITSO**.

e. For Persistence class management, select **Discover annotated classes automatically**.

f. Select **Create mapping file (orm.xml)**. See Figure 8.



Platform: RAD JPA 2.0 Platform

JPA implementation
Type: Library Provided by Target Runtime

The targeted runtime is able to provide the library required by this facet. Selecting this option will configure the project to use that library.

Connection: ITSOBANKderby

☐ Add driver library to build path
Driver: Derby 10.2 - Embedded JDBC Driver Default
☐ Override default catalog from connection
Catalog:
☑ Override default schema from connection
Schema: ITSO

Persistent class management
● Discover annotated classes automatically
○ Annotated classes must be listed in persistence.xml

☑ Create mapping file (orm.xml)

Figure 8 JPA Facet settings

g. Click **Next**.

7. In the JSF Capabilities page, leave **Default JSF Implementation** selected.
8. Click **Finish**. The project is created for you.
9. Switch to the **Web perspective** when prompted.
10. **Close** the Technology Quickstarts.

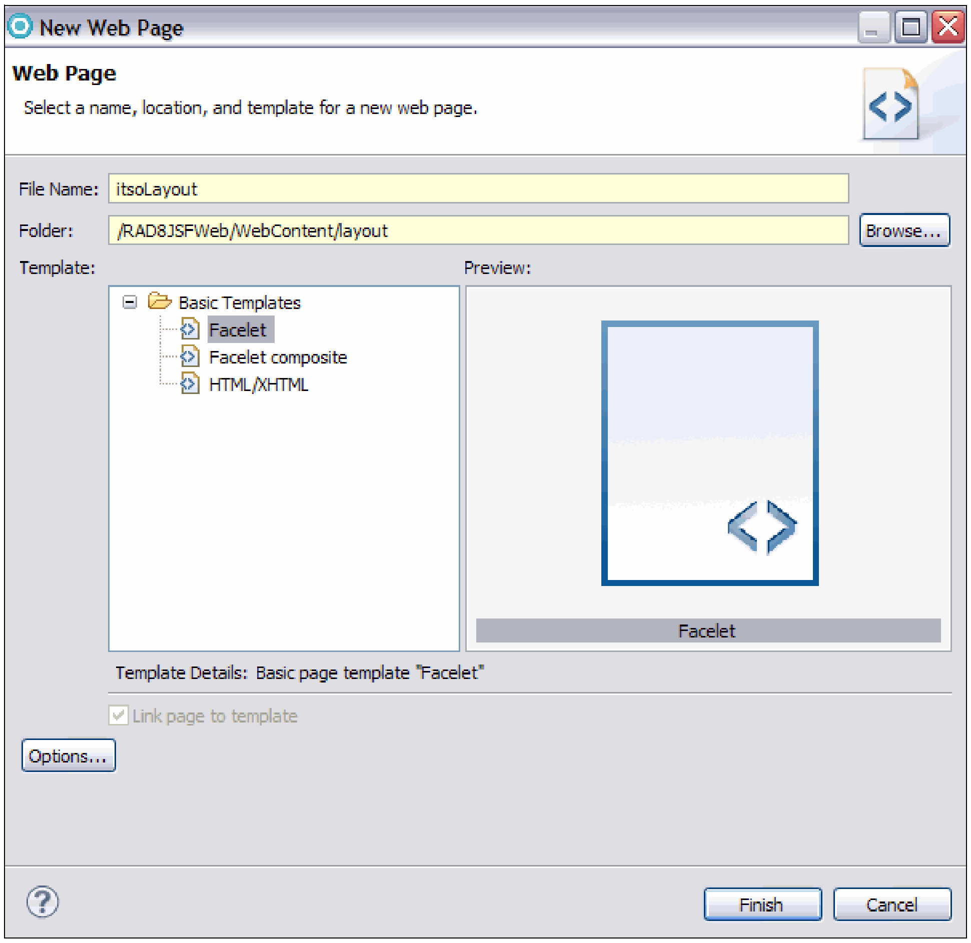
Creating Facelet templates

Before we create the Facelet web pages, `login` and `customerDetails`, we define the template that these pages use. This template helps to provide a consistent format to the application by creating universal header and footer sections.

Creating the template

Follow these steps to create the template:

1. In the Enterprise Explorer, expand **RAD8JSFWeb** and select the folder **WebContent**.
2. Right-click folder **WebContent**. Select **New** → **Folder** and type `layout` as the name.
3. Right-click folder **layout** and select **New** → **Web Page**.
4. In the New Web Page window, specify the details for this web page (Figure 9):
   a. Type `itsoLayout` for the File Name.
   b. Select **Facelet** as the type of Template.



Select a name, location, and template for a new web page.

File Name: itsoLayout

Folder: /RAD8JSFWeb/WebContent/layout

Template:

Template Details: Basic page template "Facelet"

☑ Link page to template

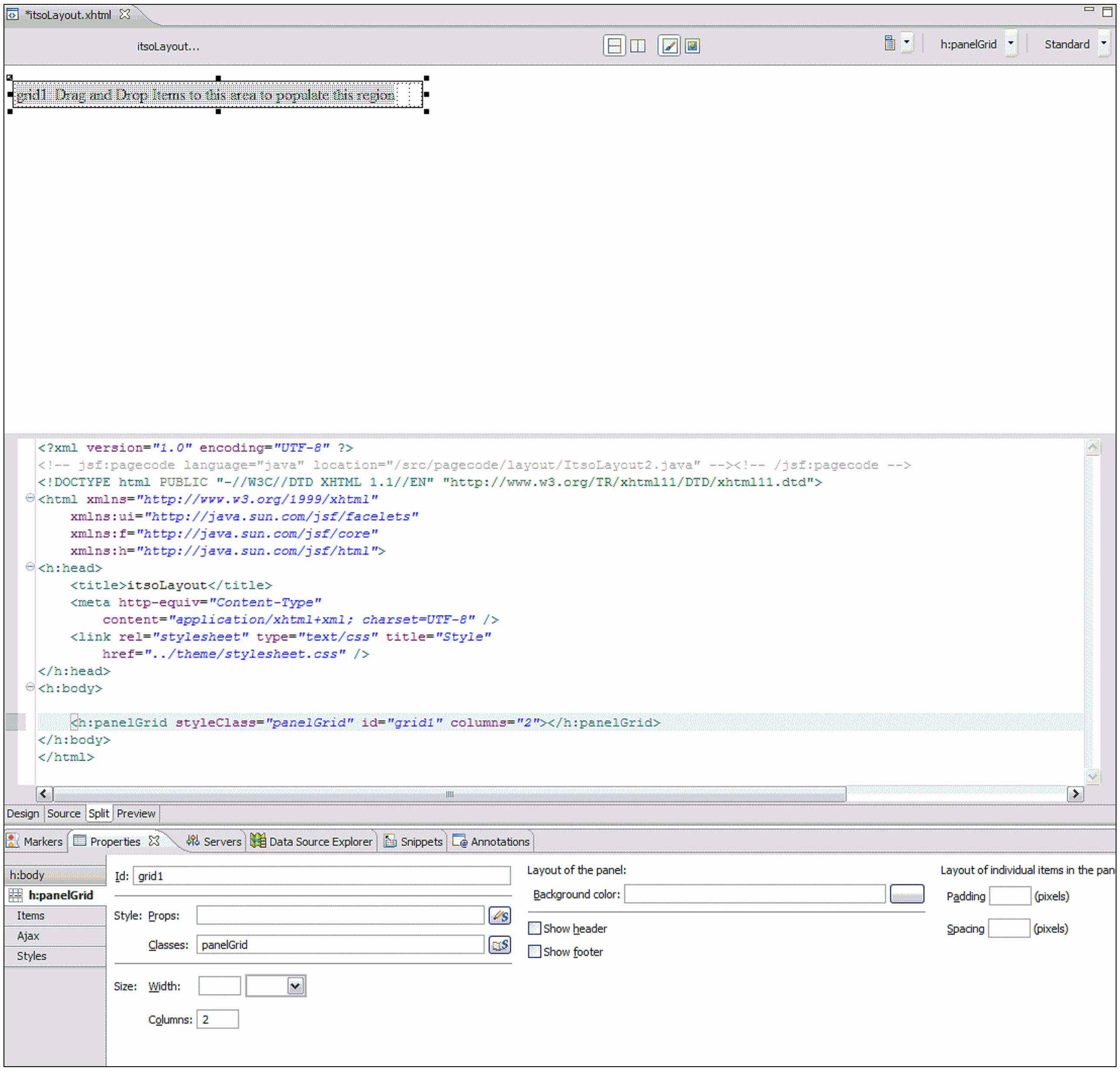
Figure 9 Create itsoLayout

5. Click **Finish** and the `itsoLayout.xhtml` file opens.

Creating the header

First, we create the header for our template:

1. In the Palette view, go to the Standard Faces Components drawer and select **Panel - Grid**.
2. Drag this Insert tag to the `itsoLayout` page (Figure 10):
   a. In the Design view, you see a visualization of the new tag with the text: `grid1: Drag and Drop Items to this area to populate this region.`
   b. Click this text and open the **Properties** view.
   c. The **h:panelGrid** tab is selected in the Properties view. Enter `2` for the number of Columns.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- jsf:pagecode language="java" location="/src/pagecode/layout/ItsoLayout2.java" --><!-- /jsf:pagecode -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
<h:head>
    <title>itsoLayout</title>
    <meta http-equiv="Content-Type"
        content="application/xhtml+xml; charset=UTF-8" />
    <link rel="stylesheet" type="text/css" title="Style"
        href="../theme/stylesheet.css" />
</h:head>
<h:body>

    <h:panelGrid styleClass="panelGrid" id="grid1" columns="2"></h:panelGrid>
</h:body>
</html>
```

Id: grid1
Style: Props:
Classes: panelGrid
Size: Width:
Columns: 2

Layout of the panel:
Background color:
Show header
Show footer

Layout of individual items in the pan
Padding (pixels)
Spacing (pixels)

Figure 10 Properties view for h:panelGrid

3. We import the logo that we use in the page header. In the Enterprise Explorer, right-click **WebContent** under `RAD8JSFWeb` and select **New** → **Folder**.

4. Enter `resources` for the folder name and click **Finish**.
5. We create a folder called `images` inside the resources folder. Right-click the new **resources** folder and select **New** → **Folder**.
6. Enter `images` for the folder name and click **Finish**.
7. Now that the folders are created, we can import the logo. Right-click **images** and select **Import**. Click **General** → **File System**. Click **Next**.
8. Locate **itso_logo.gif** on your file, which is in `4883code/jsf`. Browse to the location of this file. Click **Finish**.
9. The file is imported (Figure 11).

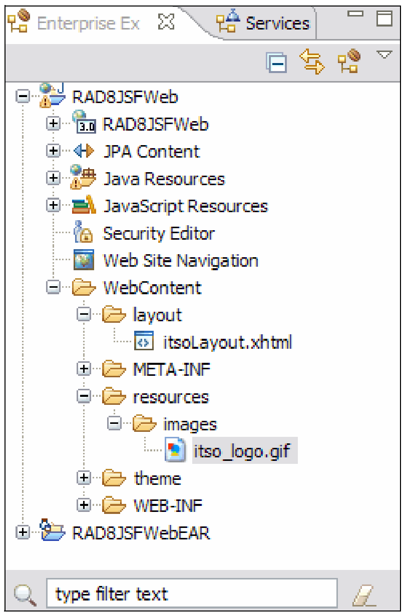
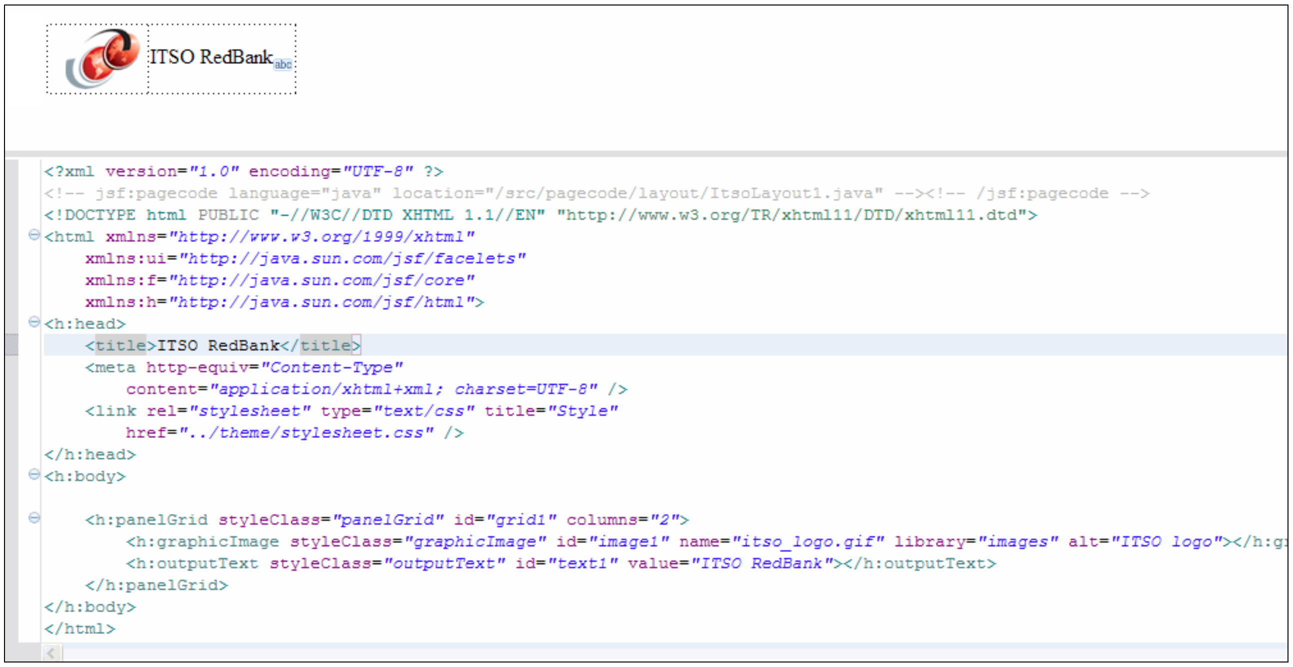


Figure 11 itso_logo imported into the project

10. Drag an image from the Standard Faces components drawer of the Palette, dropping it on top of the text **grid1**.
11. Go to the **Properties** view. The **h:graphicImage** tab is selected. Click **Browse** next to Name. In the Select an image resource dialog window, select **itso_logo.gif** from under **images** and click **OK**. The logo is visible on your page.
12. Switch to the **Accessibility** tab of the Properties view, which is located under **h:graphicImage**. Enter `ITSO logo` for the Alternate Text.
13. Drag an Output from the Palette and drop it on the right side of the logo.
14. Go to the **Properties** view. The **h:outputText** tab is selected. Type `ITSO RedBank` for the Value.
15. Look at the source for `itsoLayout`. Find the title tag. Change the text from itsoHeader to `ITSO RedBank`.

    We created the header for our page that shows the logo of the bank and a title, as shown in Figure 12 on page 17.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- jsf:pagecode language="java" location="/src/pagecode/layout/ItsoLayout1.java" --><!-- /jsf:pagecode -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
<h:head>
    <title>ITSO RedBank</title>
    <meta http-equiv="Content-Type"
        content="application/xhtml+xml; charset=UTF-8" />
    <link rel="stylesheet" type="text/css" title="Style"
        href="../theme/stylesheet.css" />
</h:head>
<h:body>

    <h:panelGrid styleClass="panelGrid" id="grid1" columns="2">
        <h:graphicImage styleClass="graphicImage" id="image1" name="itso_logo.gif" library="images" alt="ITSO logo"></h:g
        <h:outputText styleClass="outputText" id="text1" value="ITSO RedBank"></h:outputText>
    </h:panelGrid>
</h:body>
</html>
```

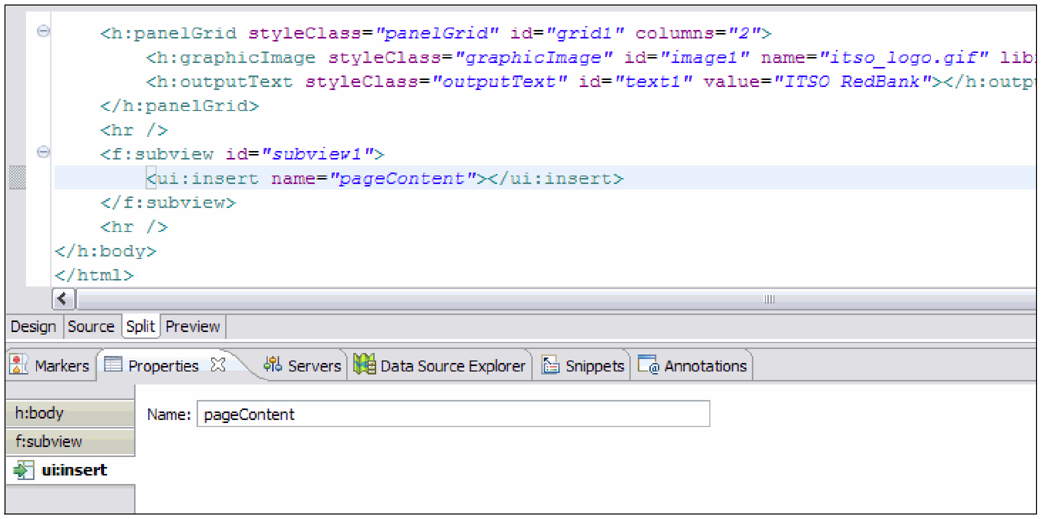
Figure 12   itsoLayout with header

16. Save the page.

Creating the content area

We create a content area to hold the content of our `login` and `customerDetails` pages. Complete these steps to create the content area:

1. Expand the **HTML Tags** drawer in the Palette. Drag a **Horizontal Rule** to the page and drop it beneath the Panel Grid.
2. Expand the **Facelet Tags** drawer. Drag an **Insert** to the page and drop it beneath the Horizontal Rule.
3. Go back to the **HTML Tags** drawer. Drag a second **Horizontal Rule** and drop it at the bottom of the page.
4. Click the text **Drop controls here for content area** “**< no name >**” and go to the **Properties** view. The **ui:insert** tab is selected. Enter `pageContent` as the Name, as shown in Figure 13 on page 18.



```
    <h:panelGrid styleClass="panelGrid" id="grid1" columns="2">
        <h:graphicImage styleClass="graphicImage" id="image1" name="itso_logo.gif" lib
        <h:outputText styleClass="outputText" id="text1" value="ITSO RedBank"></h:outp
    </h:panelGrid>
    <hr />
    <f:subview id="subview1">
        <ui:insert name="pageContent"></ui:insert>
    </f:subview>
    <hr />
</h:body>
</html>
```

Design Source Split Preview
Markers Properties Servers Data Source Explorer Snippets Annotations
h:body
f:subview
ui:insert

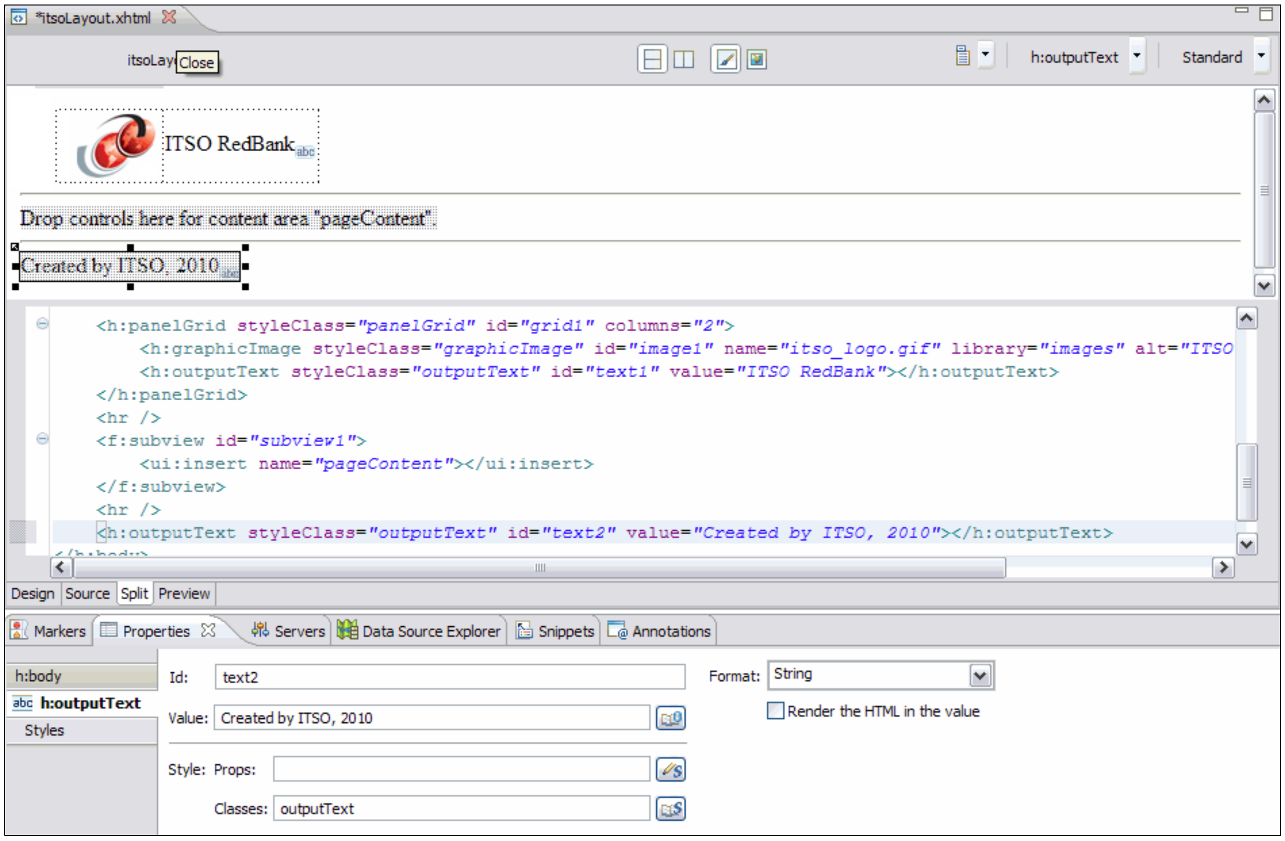
Name: pageContent

Figure 13 itsoLayout with content area

Creating the footer

Complete these steps to create the footer:

1. We create a footer for the page, which consists of a simple text string. Drag an **Output** from the Standard Faces Components drawer of the Palette and drop it at the bottom of the page.
2. Go to the **Properties** view. Enter `Created by ITSO, 2010` for the Value (Figure 14).



*itsoLayout.xhtml
itsoLay Close
h:outputText
Standard
ITSO RedBank
Drop controls here for content area "pageContent".
Created by ITSO, 2010

```
    <h:panelGrid styleClass="panelGrid" id="grid1" columns="2">
        <h:graphicImage styleClass="graphicImage" id="image1" name="itso_logo.gif" library="images" alt="ITSO
        <h:outputText styleClass="outputText" id="text1" value="ITSO RedBank"></h:outputText>
    </h:panelGrid>
    <hr />
    <f:subview id="subview1">
        <ui:insert name="pageContent"></ui:insert>
    </f:subview>
    <hr />
    <h:outputText styleClass="outputText" id="text2" value="Created by ITSO, 2010"></h:outputText>
```

Design Source Split Preview
Markers Properties Servers Data Source Explorer Snippets Annotations
h:body
h:outputText
Styles

Id: text2
Format: String
Value: Created by ITSO, 2010
Render the HTML in the value
Style: Props:
Classes: outputText

Figure 14 itsoLayout with footer

3. Save the page.

Creating Facelets

In this section, we create the `login.xhtml` and `customerDetails.xhtml` Facelets based on our layout template, `itsoLayout.xhtml`:

1. In the Enterprise Explorer, expand **RAD8JSFWeb** and select the folder **WebContent**.
2. Right-click **WebContent** and select **New** → **Web Page**. Complete these tasks:
   a. Type `login` for the File Name.
   b. Select your defined template **itsoLayout.xhtml** in the folder layout under **MyTemplates** (Figure 15).

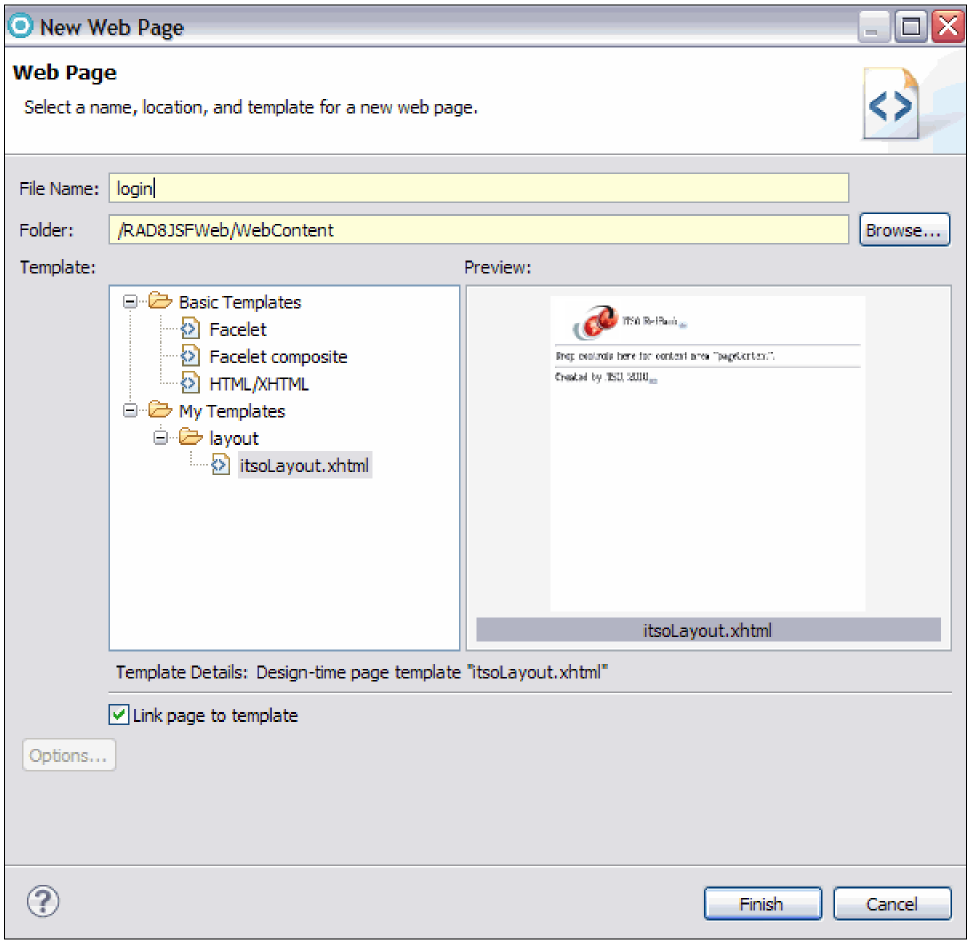


Figure 15 Create login.xhtml based on a template

3. Click **Finish**. The `login.xhtml` file opens.
4. Use the same steps to create the `customerDetails.xhtml` Facelet.

Creating JPA Manager Beans

In this section, we create JPA Manager Beans and JPA entities for the ITSOBANK database.

Creating entities

Complete these steps to create the entities:

1. If the server is running and is connected to the ITSOBANK database, stop the server.
2. Open **customerDetails.xhtml** and go to the **Page Data** view.
3. Expand **JPA**. Right-click **JPA Manager Beans** and select **New** → **JPA Manager Bean** (Figure 16).

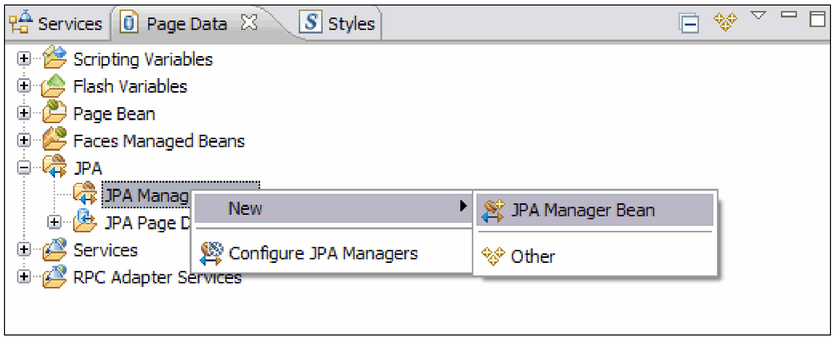
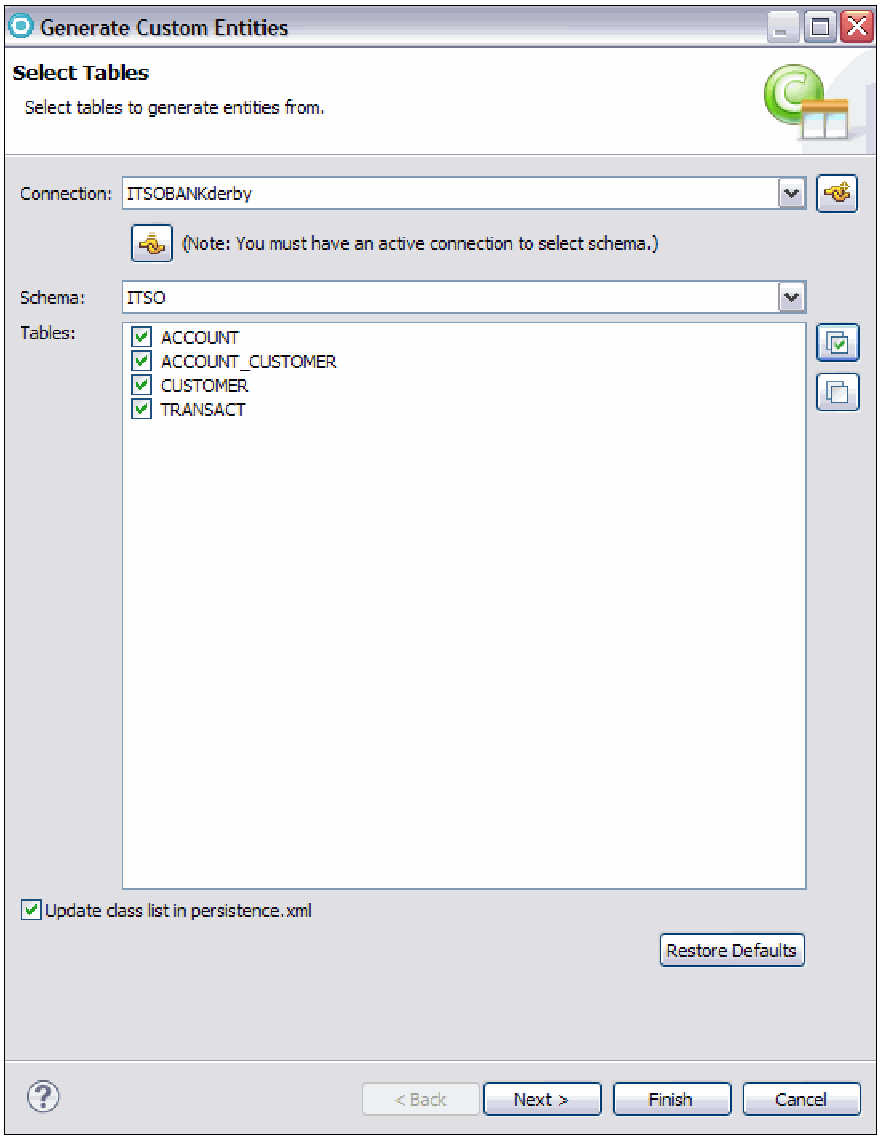


Figure 16 Creating a JPA Manager Bean from the Page Data view

4. In the JPA Manager Bean Wizard, click **Create New JPA Entities**.

Important: To retrieve records from the relational database, we require a connection. We use the ITSOBANKderby connection that was created in "Creating the JSF Project" on page 11.

5. In the Generate Custom Entities wizard, define the connection, schema, and tables (Figure 17 on page 21):
   a. For the Connection, select **ITSOBANKderby**.
   b. For the Schema, select **ITSO** (click **Connect** if you do not see this schema listed).
   c. Click the **Select All** icon to select the four tables.
   d. Select **Update class list in persistence.xml** so that the generated classes are added to the file.



Generate Custom Entities

Select Tables

Select tables to generate entities from.

Connection: ITSOBANKderby

(Note: You must have an active connection to select schema.)

Schema: ITSO

Tables:

- [x] ACCOUNT
- [x] ACCOUNT_CUSTOMER
- [x] CUSTOMER
- [x] TRANSACT

- [x] Update class list in persistence.xml

Figure 17 Select tables

e. Click **Next**.

6. Click **Next** on the Table Associations page.
7. In the Customize Default Entity Generation page, define the Table mapping and the package name (Figure 18 on page 22):

   a. For the Table mapping definition, for the Key generator, select **none**.

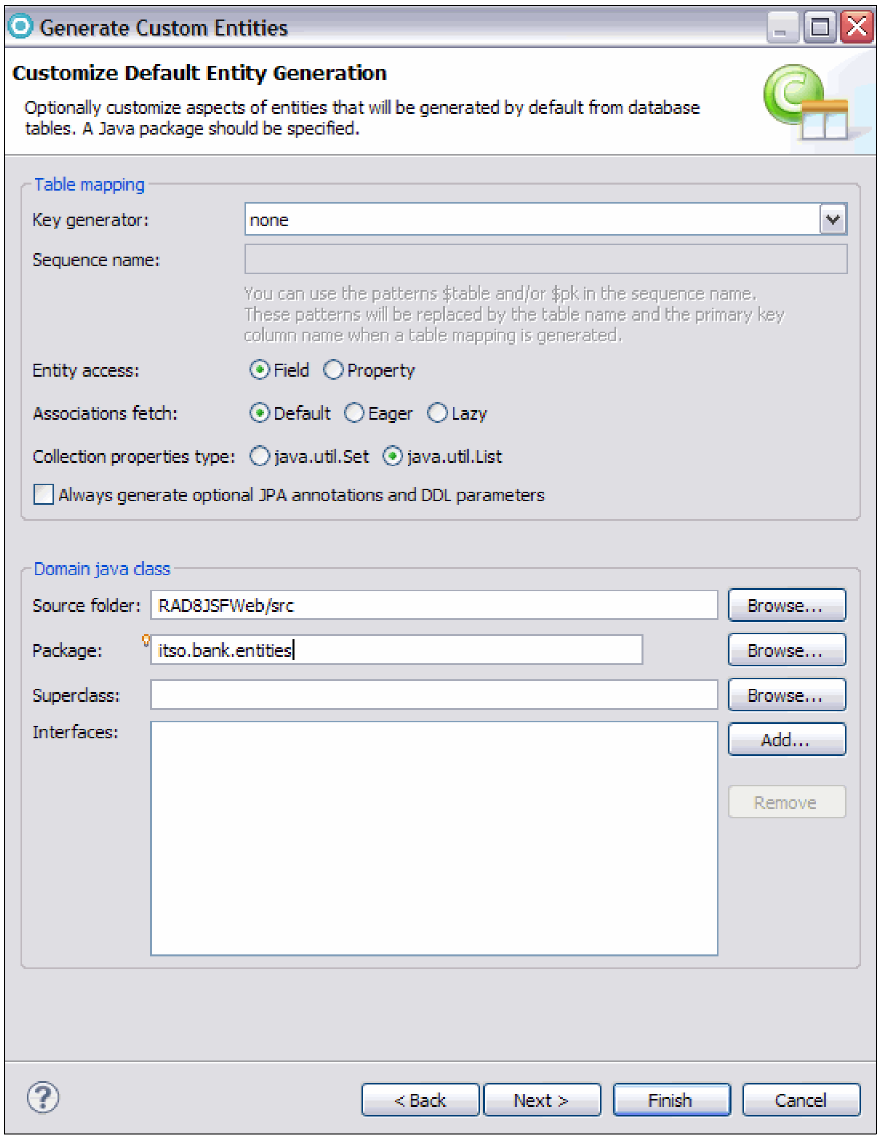
   b. For the Entity access, select **Field**.

   c. For the Associations fetch, select **Default**.

   d. For the Collection properties type, select **java.util.List**.

   e. Clear **Always generate optional JPA annotations and DDL parameters**.

   f. For the Package, type `itso.bank.entities`.


Generate Custom Entities

Customize Default Entity Generation

Optionally customize aspects of entities that will be generated by default from database tables. A Java package should be specified.

Table mapping

Key generator: none

Sequence name:

You can use the patterns $table and/or $pk in the sequence name. These patterns will be replaced by the table name and the primary key column name when a table mapping is generated.

Entity access: [✓ Field] Property

Associations fetch: [✓ Default] Eager Lazy

Collection properties type: java.util.Set [✓ java.util.List]

☐ Always generate optional JPA annotations and DDL parameters

Domain java class

Source folder: RAD8JSFWeb/src

Package: itso.bank.entities

Superclass:

Interfaces:

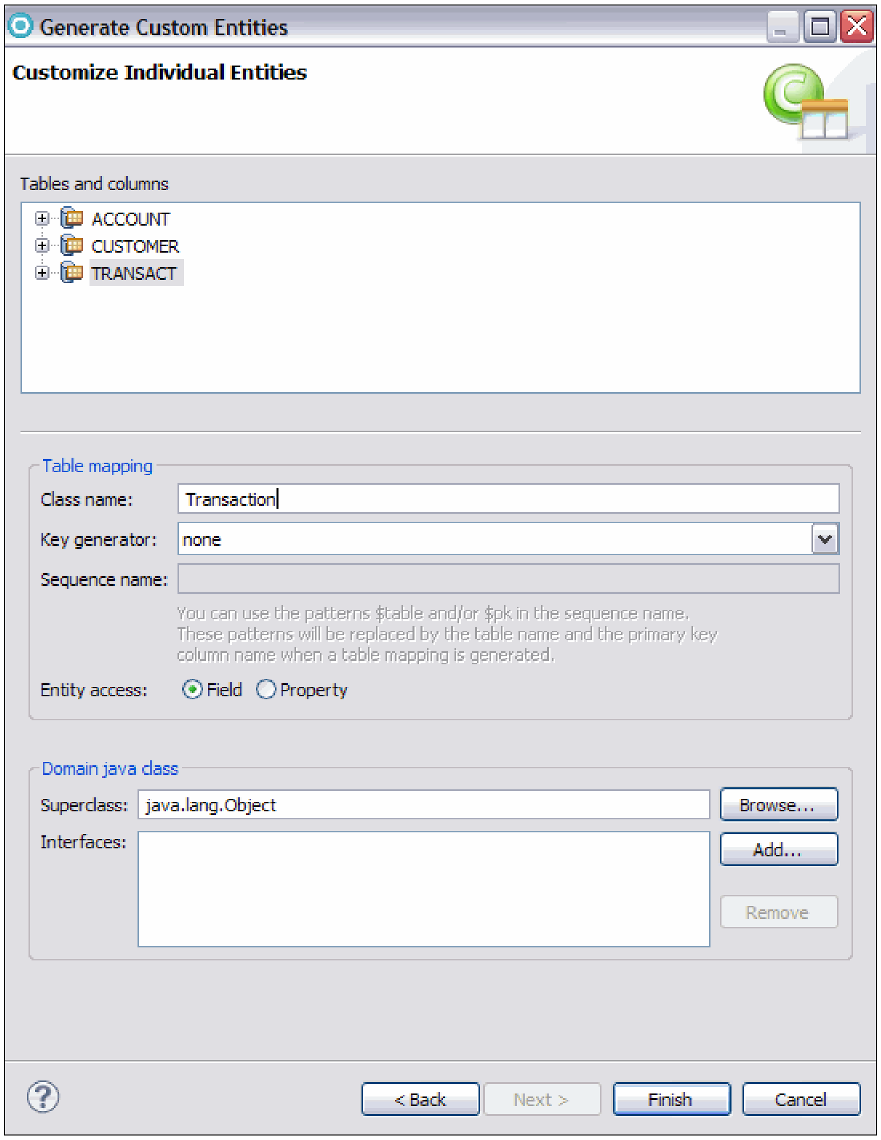
Figure 18 Customize Default Entity Generation

g. Click **Next**.

8. In the Generate Custom Entities: Customize Individual Entities window, define the class names (Figure 19):

   a. Select **TRANSACT** in the Tables and columns pane.

   b. The default class name is Transact. Change the class name to `Transaction`.


Generate Custom Entities

Customize Individual Entities

Tables and columns
- ACCOUNT
- CUSTOMER
- TRANSACT

Table mapping

Class name: Transaction

Key generator: none

Sequence name:

You can use the patterns $table and/or $pk in the sequence name. These patterns will be replaced by the table name and the primary key column name when a table mapping is generated.

Entity access: [✓ Field] Property

Domain java class

Superclass: java.lang.Object

Interfaces:

Figure 19   Customize Individual Entities

9. Click **Finish**.

Editing the Customer entity

Complete these steps to edit the Customer entity:

1. In the JPA Manager Bean Wizard window, the Account, Customer, and Transaction entities are displayed.
2. Select the **Customer** entity and click **Edit Selected Entities**.
3. In the Tasks page, go through the tasks, and make the following selections:
   a. For the Primary key, ensure that **ssn** is selected.
   b. For the Relationships, ensure that **Account** is selected.
   c. For Named Queries, click **Add**.
   d. In the Add Named Query dialog box, ensure that **getCustomer** is the Named Query Name and then click **OK**.

e. For Concurrency Control, ensure that **No Concurrency Control** is selected.
f. Leave all check boxes clear for the Other task.
g. Select **Automatically set up initial values for JDBC Deployment**.
h. Click **Finish** to return to the JPA Manager Bean Wizard window.

4. Click **Next** in the JPA Manager Bean Wizard window.
5. In the Tasks window, click the **Other** task.
6. Select **I want the container to inject the persistence unit into my beans** and check **Generate JSF Converter** for target entity.
7. Click **Finish**.

Editing the Account entity

We add additional JPA managers and we create a JPA Manager for Accounts:

1. In the Page Data view on `customerDetails.xhtml`, right-click **JPA Manager Beans** and select **Configure JPA Manager Beans**.
2. In the JPA Manager Bean wizard, click **Create new JPA Manager**.
3. Select **Account** and then click **Edit Selected Entities**.
4. Click **Named Queries**.
5. Click **Add**.
6. On the Add Named Query dialog window, change the name to `getAccountBySSN` and change the query statement to `select a from Account a, in(a.customers) c where c.ssn =:ssn order by a.id`. This code snippet is available in the `4883code/jsf/getAccount.txt` file.
7. Click **OK**, **Finish**, and then **Finish** again.
8. Click **Cancel**.
9. The AccountManager bean is now created for you.

Creating JPA page data

Now we add JPA page data so that the JSF components can interact with the JPA data. We want a single customer record and a list of accounts.

Customer record

Follow these steps:

1. Open `customerDetails.xhtml`. In the Page Data view, right-click **JPA** and select **New → JPA Page Data**.
2. In the Add JPA data to page dialog window, select **CustomerManager** and select **Retrieve a single record**.
3. Click **Next** twice.
4. On the Set Filter Values page, change the primary key value to `#{sessionScope.customerId}`. Click **Finish**.

Account list

Follow these steps:

1. In the Page Data view, right-click **JPA and select New → JPA Page Data**.
2. In the Add JPA data to page dialog window, select **AccountManager** and select **Retrieve a list of data**.
3. Click **Next** twice.
4. On the Set Filter Values page, change the primary key value to `#{sessionScope.customerId}`. Click **Finish**.
5. Save the page.

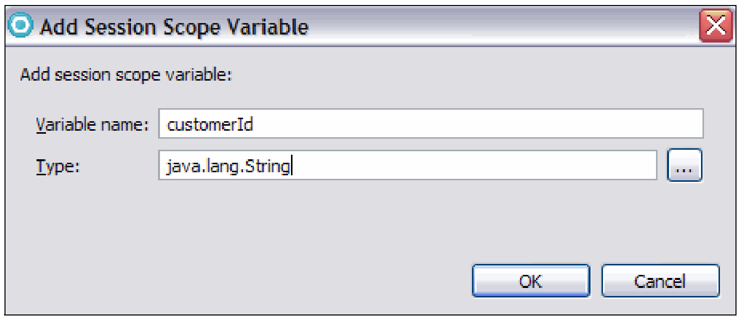
Editing the login page

Next, we complete our `login` page. We add UI components, simple validation, and navigation to go from `login.xhtml` to `customerDetails.xhtml`.

Adding UI components

Instead of adding each UI component individually, we define data for the customer ID and have Rational Application Developer generate the necessary UI for us:

1. Open **login.xhtml**. Open the **Page Data** view. Right-click **Scripting Variables** and select **New → Session scope variable**.
2. In the Add Session Scope Variable dialog window, enter `customerId` for the Variable name and `java.lang.String` for the Type (Figure 20).


Add Session Scope Variable

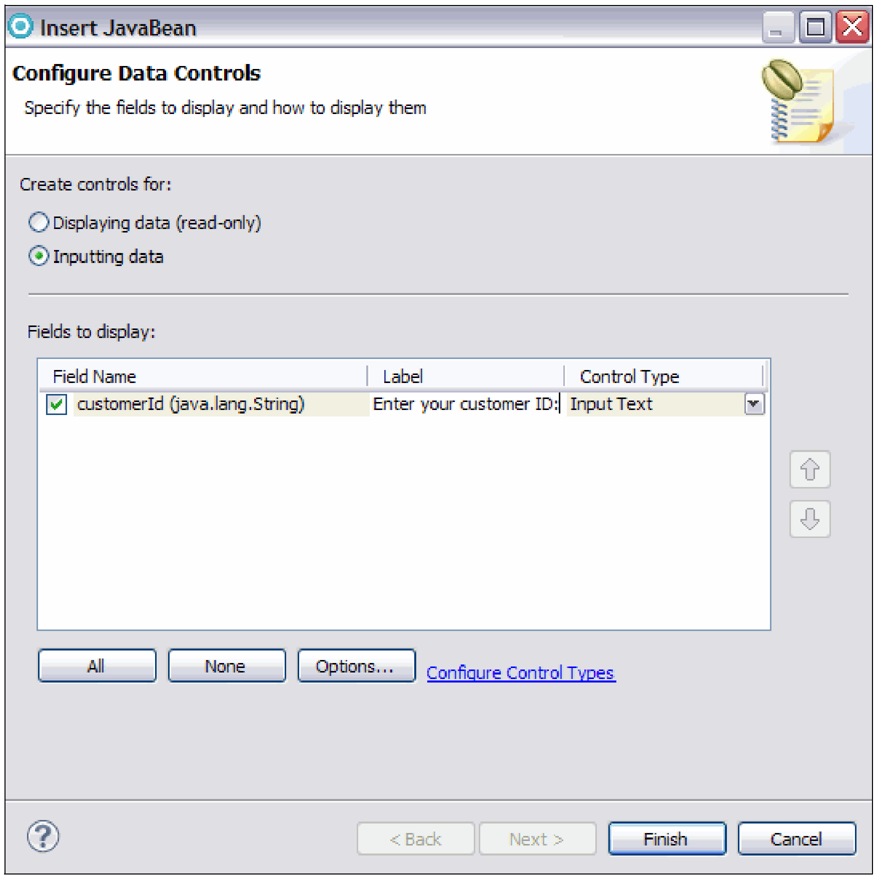
Add session scope variable:

Variable name: customerId

Type: java.lang.String

Figure 20 Add Session Scope Variable

3. Click **OK.**
4. In the Page Data view, expand **Scripting Variables → sessionScope**. Click **customerId** and drag it to the page and drop it on top of "Drop controls here".
5. In the Insert JavaBean wizard, select **Inputting data**.
6. Change the label from CustomerId to `Enter your customer ID:` (Figure 21 on page 26).



Create controls for:

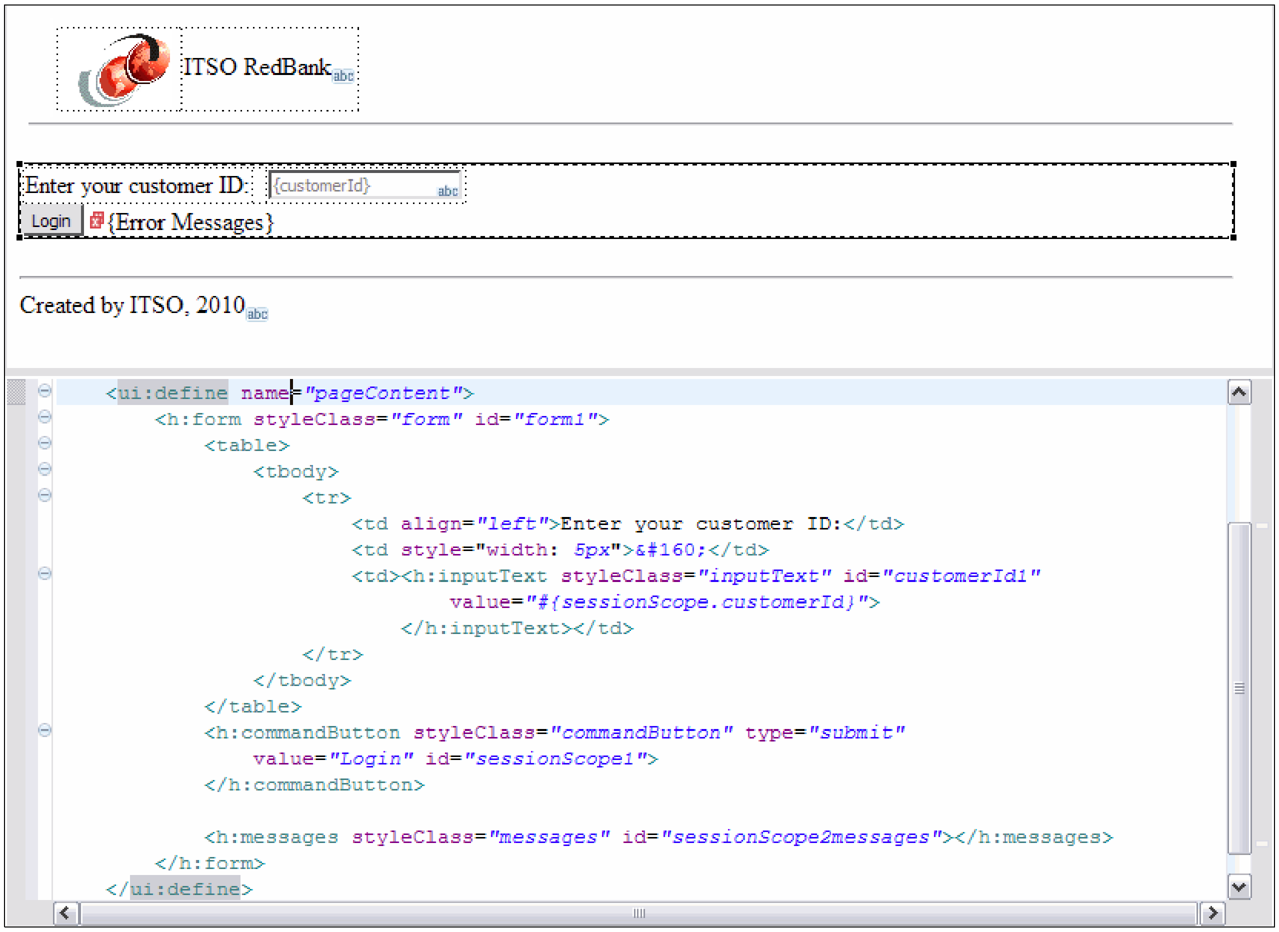
Displaying data (read-only) [✓ Inputting data]

Fields to display:

Field Name	Label	Control Type
✓ customerId (java.lang.String)	Enter your customer ID:	Input Text

Figure 21 Create a login input field

7. Click **Options**. Select the **Buttons** tab of the Options dialog window, and change the label to `Login`.
8. Click **OK** and then click **Finish**. Figure 22 on page 27 shows the login page.



```
<ui:define name="pageContent">
    <h:form styleClass="form" id="form1">
        <table>
            <tbody>
                <tr>
                    <td align="left">Enter your customer ID:</td>
                    <td style="width: 5px">&#160;</td>
                    <td><h:inputText styleClass="inputText" id="customerId1"
                            value="#{sessionScope.customerId}">
                        </h:inputText></td>
                </tr>
            </tbody>
        </table>
        <h:commandButton styleClass="commandButton" type="submit"
            value="Login" id="sessionScope1">
        </h:commandButton>

        <h:messages styleClass="messages" id="sessionScope2messages"></h:messages>
    </h:form>
</ui:define>
```

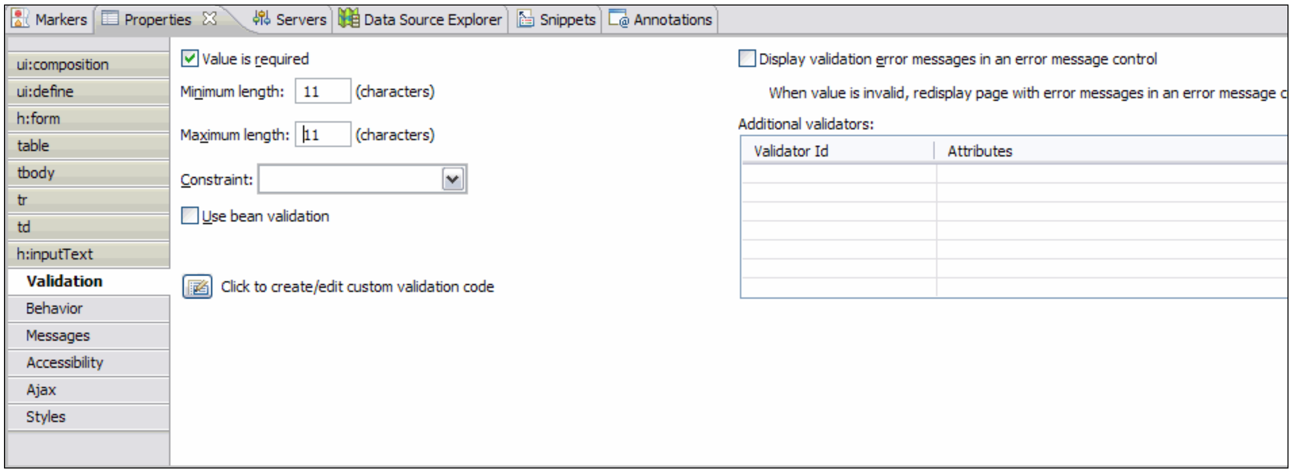
Figure 22 Login page with login input text

9. Save the page.

Adding validation

You can add simple validation to your web page easily by using JSF. Here, we ensure that the user types an 11-digit number for the `customerId`:

1. Click the **customerId** Input Text.
2. Go to the **Properties** view and switch to the **Validation** tab (under **h:inputText**).
3. Check **Value is required**.
4. Enter `11` for both the Minimum length and the Maximum length (Figure 23 on page 28).



☑ Value is required
Minimum length: 11 (characters)
Maximum length: 11 (characters)
Constraint:
☐ Use bean validation
☐ Display validation error messages in an error message control
When value is invalid, redisplay page with error messages in an error message c
Additional validators:

Figure 23 Validation tab of the Properties view

5. Save the page.

Verifying the customer ID

We want to ensure that the customerId is valid and that our `RedBank` application has a customer with that ID:

1. In the Enterprise Explorer, select **RAD8JSFWeb** → **Java Resources** → **src** → **pagecode** → **Login.java**. This bean is a Faces-managed bean that was created automatically by Rational Application Developer. We add a method that runs when the button is clicked.
2. Paste in the code that is shown in Example 1. This code is available in the `4883code/jsf/LoginButton.txt` file.

Example 1 LoginButton

```
public String doLoginAction() {
   try {
      String id = (String) getSessionScope().get("customerId");
      System.out.println("Logon id: " + id);
      CustomerManager customerManager =
(CustomerManager)getManagedBean("CustomerManager");
      Customer customer = customerManager.findCustomerBySsn(id);
      if (customer == null) {
         throw new Exception("Customer " + id + " was not found.");
      }
      return "login";
   } catch (Exception e) {
      getFacesContext().addMessage("id", new FacesMessage(e.getMessage()));
      return null;
   }
}
```

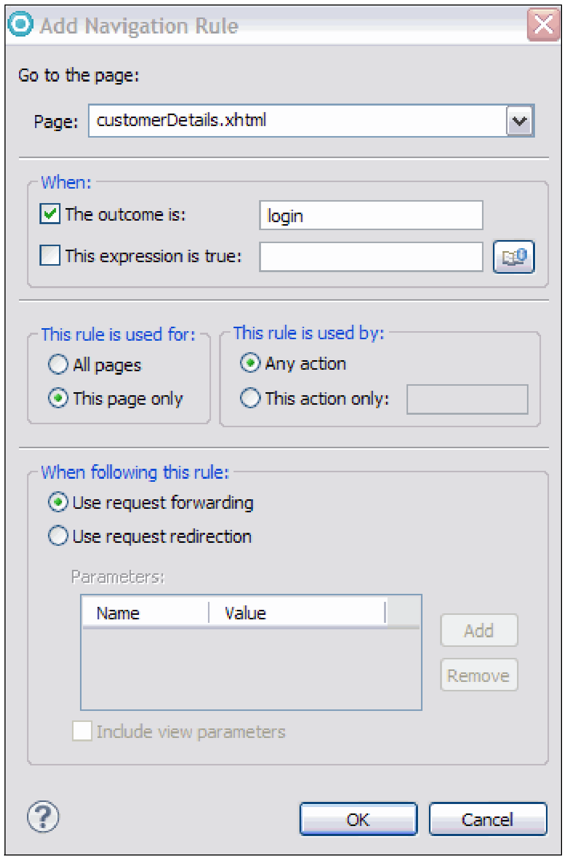
3. If you get errors for unknown imports, right-click the new text and select **Source** → **Organize Imports**. The following import statements are automatically added for you:
   - `import itso.bank.entities.Customer;`
   - `import itso.bank.entities.controller.CustomerManager;`
   - `import javax.faces.application.FacesMessage;`
4. Save the `login.xhtml` and `Login.java` files.

5. Open **login.xhtml**. Click the button.
6. Go to the **Properties** view. Click **Select or code an action button** (next to Action or outcome) and then choose **Select an action**.
7. In the Faces Action selection dialog window, click **Page Code** → **doLoginAction** and click **OK.**
8. Save `login.xhtml`.

Adding navigation

We add navigation so that the customer's details can be displayed if the customer ID is valid:

1. Open **login.xhtml**. Click the button.
2. Go to the **Properties** view. Click **Add Rule**.
3. In the **Add Navigation Rule** dialog window, select **customerDetails.xhtml** for the page.
4. Select **The outcome is** and then type `login` (Figure 24).


Add Navigation Rule

Go to the page:

Page: customerDetails.xhtml

When:

[✓] The outcome is: login

[ ] This expression is true:

This rule is used for: All pages [✓ This page only]

This rule is used by: [✓ Any action] This action only:

When following this rule: [✓ Use request forwarding] Use request redirection

Parameters:

Name	Value

[ ] Include view parameters

Figure 24   Login page

5. Click **OK.**

The `login.xhtml` page is now complete.

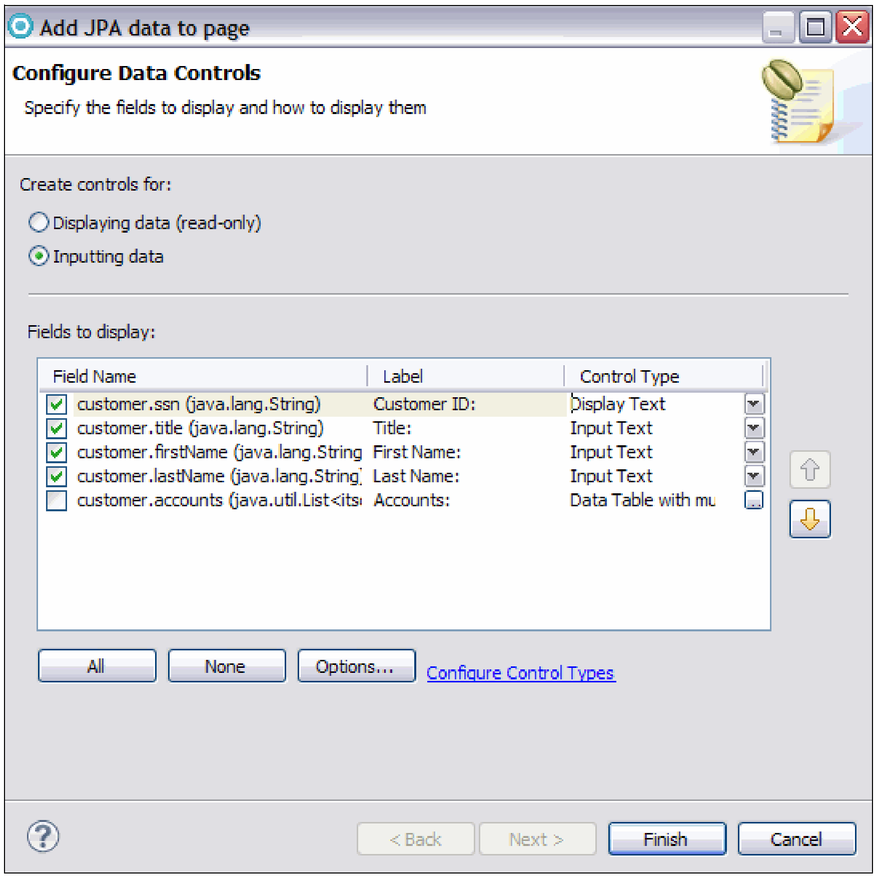
Editing the customer details page

For our `customerDetails` page, we display the information about a particular customer and show the associated bank account balances.

Displaying customer information

Follow these steps:

1. Open `customerDetails.xhtml`. Go to the **Page Data** view.
2. Expand **JPA** → **JPA Page Data**. Click **customer**. Drag **customer** to the page, and drop it on top of the Drop Controls here text.
3. In the Add JPA data to page wizard, make the following changes (Figure 25):
   a. Clear **customer.accounts**.
   b. Move **customer.title** up so that it is over `customer.firstName`.
   c. Change the label of customer.ssn to "`Customer ID:`".
   d. Change the label of customer.firstname to "`First Name:`".
   e. Change the label of customer.lastname to "`Last Name:`".
   f. Change the control type of customer.ssn to **Display Text**.



Create controls for:

( ) Displaying data (read-only)
(●) Inputting data

Fields to display:

	Field Name	Label	Control Type
✓	customer.ssn (java.lang.String)	Customer ID:	Display Text
✓	customer.title (java.lang.String)	Title:	Input Text
✓	customer.firstName (java.lang.String	First Name:	Input Text
✓	customer.lastName (java.lang.String)	Last Name:	Input Text
	customer.accounts (java.util.List<its	Accounts:	Data Table with mu

Figure 25 Add JPA data to page wizard for customer

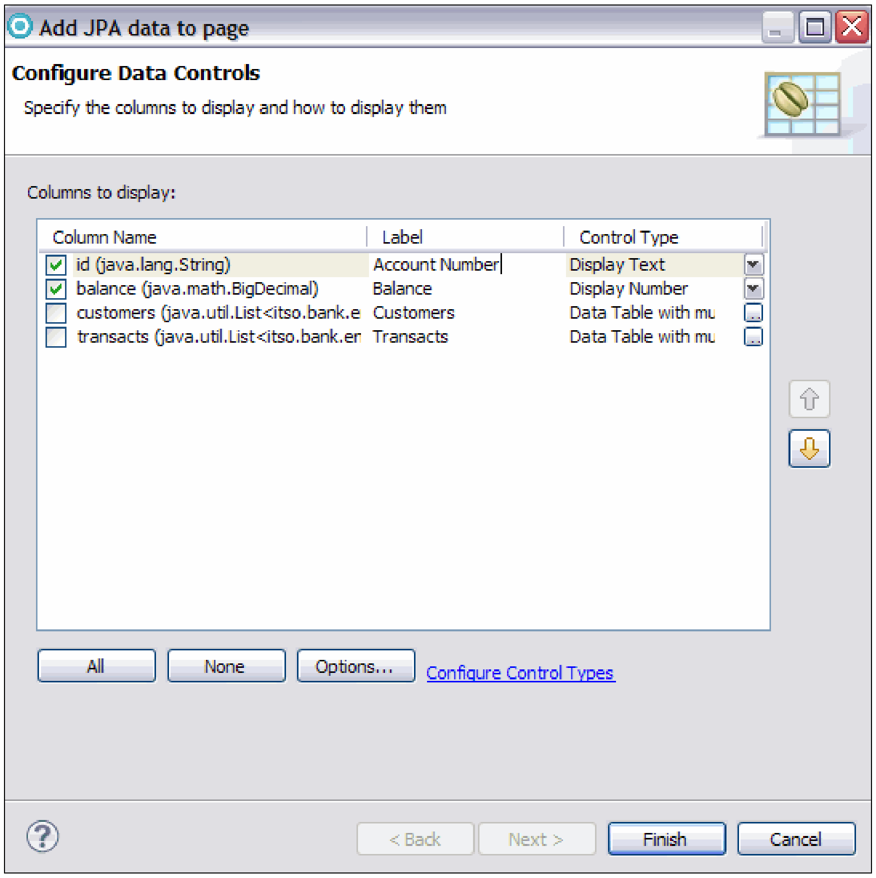
4. Click **Options**.

5. On the Buttons tab, change the label to `Update`.
6. Click **OK** and click **Finish**.

Displaying account information

Follow these steps:

1. In the Page Data view, expand **JPA** → **JPA Page Data** → **accountList** → **accountList**. Click the inner **accountList** and drag it to the page. Drop it after Error Messages.
2. In the Add JPA data to page wizard, make the following changes (Figure 26):
   a. Clear **customers** and **transacts**.
   b. Change the id label to `Account Number`.



Column Name	Label	Control Type
✓ id (java.lang.String)	Account Number	Display Text
✓ balance (java.math.BigDecimal)	Balance	Display Number
☐ customers (java.util.List<itso.bank.e	Customers	Data Table with mu
☐ transacts (java.util.List<itso.bank.er	Transacts	Data Table with mu

Figure 26 Add JPA data to page wizard for accounts

3. Click **Finish**.

   The `customerDetails` page contains customer and account information (Figure 27 on page 32).

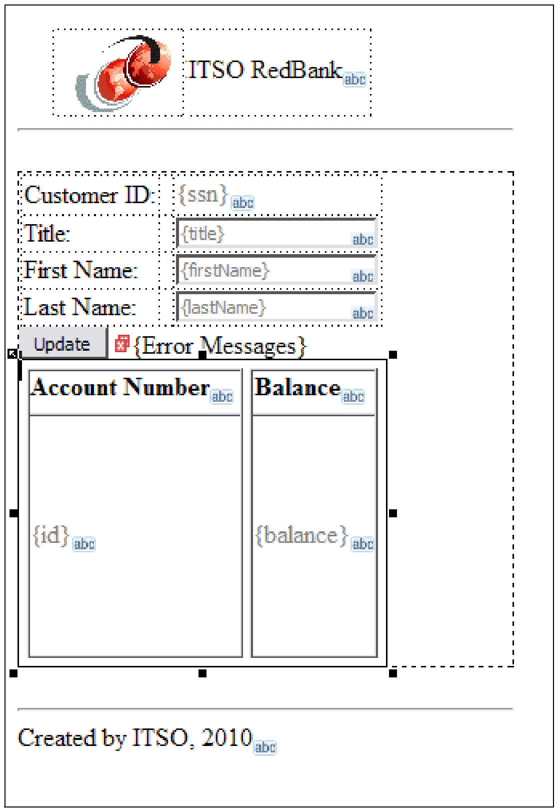
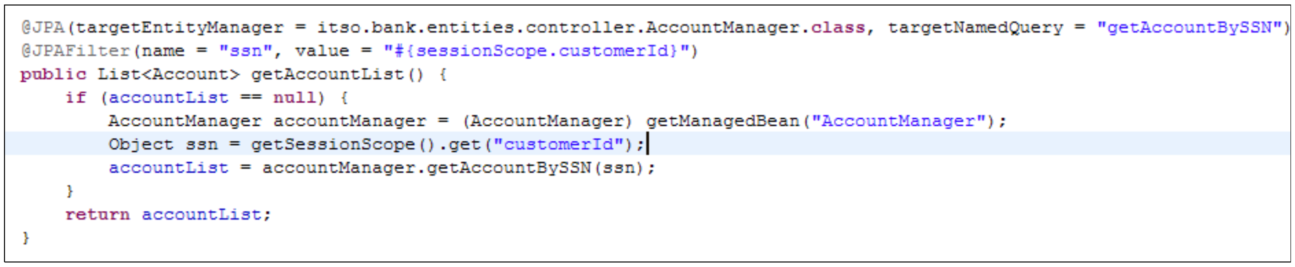


Figure 27 customerDetails page with customer and account information

4. Save `customerDetails.xhtml`.
5. In the Enterprise Explorer view, open **RAD8JSFWeb** → **Java Resources** → **src** → **pagecode** → **CustomerDetails.java**.
6. Find the method **getAccountList()**.
7. Replace the line `Object ssn =` with `Object ssn = getSessionScope().get("customerId");`

This code is available in `4883code/jsf/getAccountList.txt`. See Figure 28.



```
@JPA(targetEntityManager = itso.bank.entities.controller.AccountManager.class, targetNamedQuery = "getAccountBySSN")
@JPAFilter(name = "ssn", value = "#{sessionScope.customerId}")
public List<Account> getAccountList() {
    if (accountList == null) {
        AccountManager accountManager = (AccountManager) getManagedBean("AccountManager");
        Object ssn = getSessionScope().get("customerId");
        accountList = accountManager.getAccountBySSN(ssn);
    }
    return accountList;
}
```

Figure 28 New getAccountList() method

8. Save `CustomerDetails.java`.

Updating customer information

Our page has an Update button. We ensure that it can update simple customer information:

1. In the Enterprise Explorer view, open **RAD8JSFWeb** → **Java Resources** → **src** → **pagecode** → **CustomerDetails.java**.
2. Paste in the code that is shown in Example 2, which is available in the `4883code/jsf/UpdateButton.txt` file.

Example 2 updateButton

```
public String doUpdateAction() {
   CustomerManager customerManager =
(CustomerManager)getManagedBean("CustomerManager");
   try {
      customerManager.updateCustomer(customer);
   } catch (Exception e) {
      logException(e);
   }
   return "update";
   }
```

3. Save `CustomerDetails.java`.
4. Open **customerDetails.xhtml**. Click **Update** and go to the **Properties** view.
5. Click **Select or Code an action** and then choose **Select an action**.
6. In the Faces Action selection dialog window, click **Page Code** → **doUpdateAction** and click **OK**.
7. In the Properties view, click **Add Rule**.
8. In the Add Navigation Rule dialog window, select **CustomerDetails.xhtml** for the Page. Select **The outcome is** and type `update`.
9. Click **OK.**
10. Save the page.

Using Ajax

We already have an Update button on our `customerDetails.xhtml` page to update customer information. We decide to add a second update button that uses Ajax.

Follow these steps:

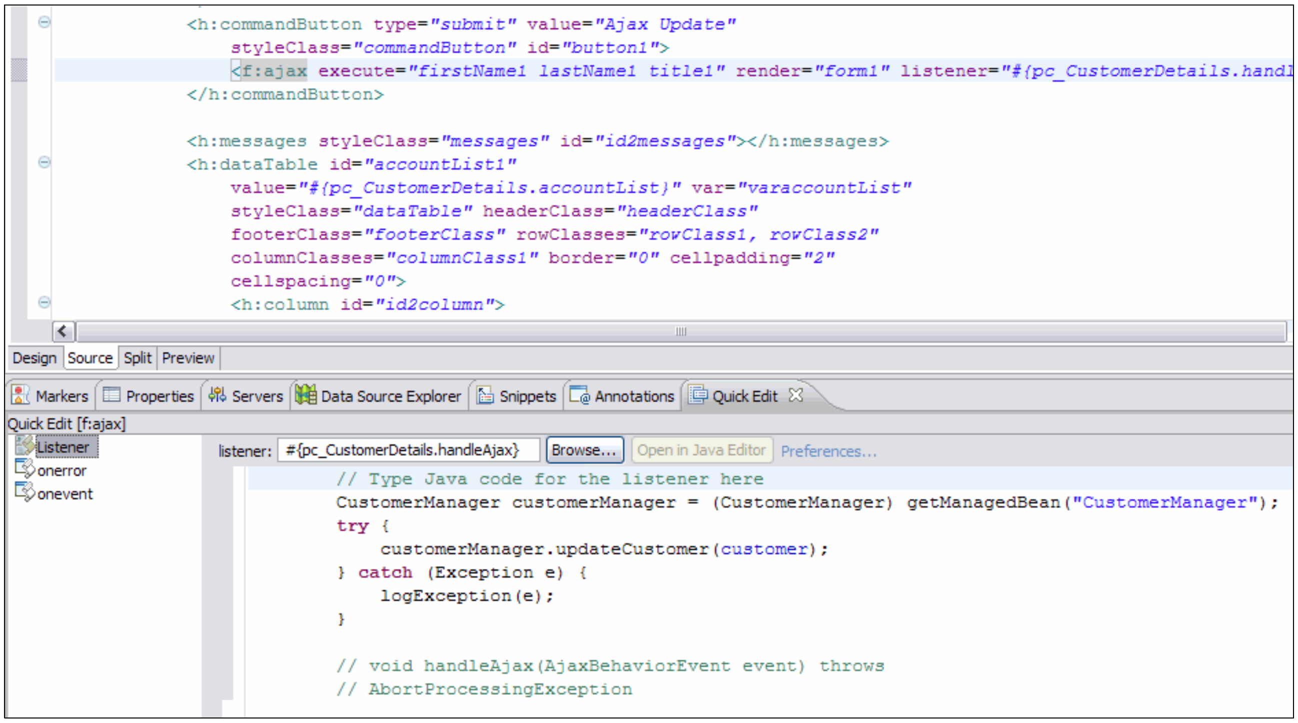
1. Open **customerDetails.xhtml**. Drag a **Button – Command** from the Standard Faces Components drawer of the palette and drop it next to the existing button.
2. In the source, change the value of the new button to `Ajax Update`, which changes the label of the button.
3. Go to the **Properties** view. The **h:commandButton** tab is selected.
4. Go to the **Ajax** tab.
5. Click **Support Ajax**.
6. We want to allow the user to update the customer's first name, last name, and title. Type `firstName1 lastName1 title1` into the Execute combination box.
7. Select **form1** for Render.
8. Select **Click to create/edit custom code**.

9. The Quick Edit view automatically opens. Ensure that **listener** is selected.
10. Paste in the code that is shown in Example 3, which is also available in the `4883code/jsf/ajaxButton.txt` file.

Example 3   ajaxButton

```
CustomerManager customerManager =
(CustomerManager)getManagedBean("CustomerManager");
try {
   customerManager.updateCustomer(customer);
} catch (Exception e) {
   logException(e);
}
```

The code is shown in Figure 29.



```
<h:commandButton type="submit" value="Ajax Update"
    styleClass="commandButton" id="button1">
    <f:ajax execute="firstName1 lastName1 title1" render="form1" listener="#{pc_CustomerDetails.handl
</h:commandButton>

<h:messages styleClass="messages" id="id2messages"></h:messages>
<h:dataTable id="accountList1"
    value="#{pc_CustomerDetails.accountList}" var="varaccountList"
    styleClass="dataTable" headerClass="headerClass"
    footerClass="footerClass" rowClasses="rowClass1, rowClass2"
    columnClasses="columnClass1" border="0" cellpadding="2"
    cellspacing="0">
    <h:column id="id2column">
```

listener: #{pc_CustomerDetails.handleAjax}

```
// Type Java code for the listener here
CustomerManager customerManager = (CustomerManager) getManagedBean("CustomerManager");
try {
    customerManager.updateCustomer(customer);
} catch (Exception e) {
    logException(e);
}

// void handleAjax(AjaxBehaviorEvent event) throws
// AbortProcessingException
```

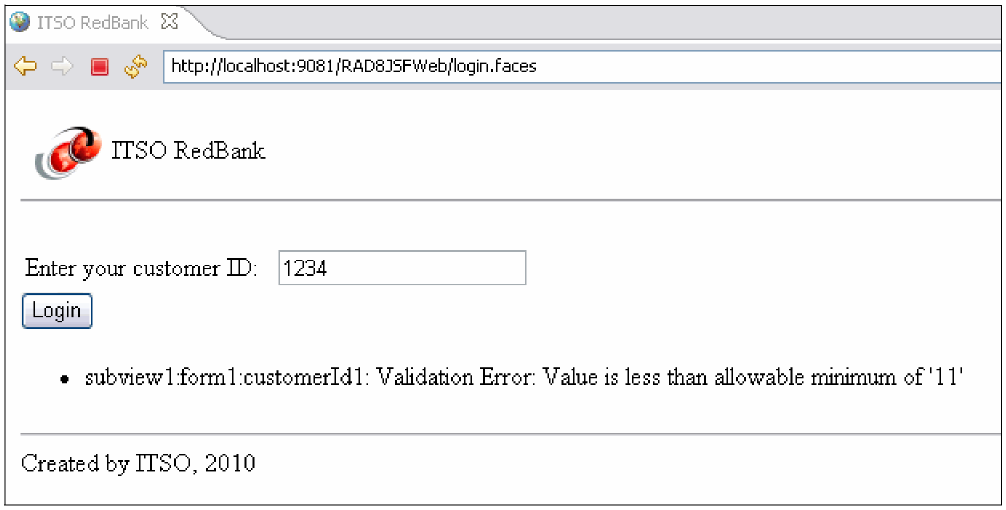
Figure 29   Button with Ajax

11. Save the page.

Running the JSF application

Now we run our web application and see the results:

1. In the Enterprise Explorer view, right-click **login.xhtml** and select **Run As** → **Run on server**.
2. Select a server and make any other necessary selections.
3. When the login page opens, you can interact with your web application.
4. If you enter a user ID that is too short, such as `1234`, or that does not have a customer associated with it, an error message displays. See Figure 30 on page 35.



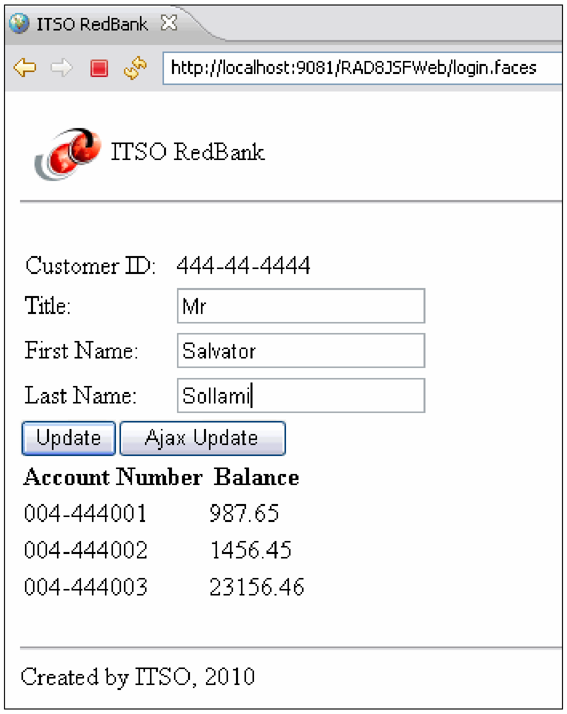
Enter your customer ID: 1234

- subview1:form1:customerId1: Validation Error: Value is less than allowable minimum of '11'

Created by ITSO, 2010

Figure 30   Login page with error

5. If you enter a valid user ID, such as `444-44-4444`, the details for that customer are shown See Figure 31.



Customer ID:	444-44-4444
Title:	Mr
First Name:	Salvator
Last Name:	Sollami

Account Number	Balance
004-444001	987.65
004-444002	1456.45
004-444003	23156.46

Created by ITSO, 2010

Figure 31   Customer details page

6. On the customer details page, you can change the first name, last name, and title of the customer. Click either **Update** or **Ajax Update** to save those changes.

Final code

To run the web application, you must complete the previous steps or import the sample from `4883codesolution/jsf/RAD8JSFWeb.zip`.

You also must set up the `ITSOBANK` database, as described “Creating the JSF Project” on page 11.

More information

For more information about JSF 2.0, see the following resources:

- *JSR 314: JavaServer Faces 2.0*:

  http://www.jcp.org/en/jsr/detail?id=314

- Rational Application Developer Information Center:
  - http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tjsfover.html
  - http://publib.boulder.ibm.com/infocenter/radhelp/v8/topic/com.ibm.etools.jsf.doc/topics/tcrtfaceletcomposite.html
- IBM developerWorks:
  - http://www.ibm.com/developerworks/web/library/wa-aj-gmaps/
  - http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html
  - http://www.ibm.com/developerworks/java/library/j-jsf2fu3/
  - http://www.ibm.com/developerworks/java/library/j-facelets2.html
  - http://www.ibm.com/developerworks/wikis/download/attachments/140051369/radjsffacelet_template.swf?version=1
  - http://www.ibm.com/developerworks/java/library/j-jsf2fu-0410/index.html

Locating the web material

The web material that is associated with this is available in softcopy on the Internet from the IBM Redbooks web server. Enter the following URL in a web browser and then download the two ZIP files:

ftp://www.redbooks.ibm.com/redbooks/REDP4883

Alternatively, you can go to the IBM Redbooks website:

http://www.ibm.com/redbooks

Accessing the web material

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks publication form number, REDP-4883.

Additional information: For more information about the additional material, see *Rational Application Developer for WebSphere Software V8 Programming Guide*, SG24-7835.

The additional web material that accompanies this paper includes the following files:

File name	*Description*
`4883code.zip`	Compressed file that contains sample code
`4883codesolution.zip`	Compressed file that contains solution interchange files

System requirements for downloading the web material

We recommend the following system configuration:

Hard disk space:	20 GB minimum
Operating system:	Microsoft Windows or Linux
Processor:	2 GHz
Memory:	2 GB

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Rochester Center.

Martin Keen is a Consulting IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products and service-oriented architecture (SOA). He also teaches IBM classes worldwide about WebSphere, SOA, and enterprise service bus (ESB). Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, U.K. Martin holds a Bachelors degree in Computer Studies from Southampton Institute of Higher Education.

Rafael Coutinho is an IBM Advisory Software Engineer working for Software Group in the Brazil Software Development Lab. His professional expertise covers many technology areas ranging from embedded to platform-based solutions. He is currently working on IBM Maximo® Spatial, which is the geographic information system (GIS) add-on of IBM Maximo Enterprise Asset Management (EAM). He is a certified Java enterprise architect and Accredited IT Specialist, specialized in high-performance distributed applications on corporate and financial projects.

Rafael is a computer engineer graduate from the State University of Campinas (Unicamp), Brazil, and has a degree in Information Technologies from the Centrale Lyon (ECL), France.

Sylvi Lippmann is a Software IT Specialist in the GBS Financial Solutions team in Germany. She has over seven years of experience as a Software Engineer, Technical Team Leader, Architect, and Customer Support representative. She is experienced in the draft, design, and realization of object-oriented software systems, in particular, the development of Java EE-based web applications, with a priority in the surrounding field of the WebSphere product family. She holds a degree in Business Informatic Engineering.

Salvatore Sollami is a Software IT Specialist in the Rational brand team in Italy. He has been working at IBM with particular interest in the change and configuration area and web application security. He also has experience in the Agile Development Process and Software Engineering. Before joining IBM, Salvatore worked as a researcher for Process Optimization Algorithmic, Mobile Agent Communication, and IT Economics impact. He developed the return on investment (ROI) SOA investment calculation tool. He holds the "Laurea" (M.S.) degree in Computer Engineering from the University of Palermo. In cooperation with IBM, he received an M.B.A. from the MIP - School of Management - polytechnic of Milan.

Sundaragopal Venkatraman is a Technical Consultant at the IBM India Software Lab. He has over 11 years of experience as an Architect and Lead working on web technologies, client server, distributed applications, and IBM System z®. He works on the WebSphere stack on process integration, messaging, and the SOA space. In addition to handling training on WebSphere, he also gives back to the technical community by lecturing at WebSphere technical conferences and other technical forums.

Steve Baber has been working in the Computer Industry since the late 1980s. He has over 15 years of experience within IBM, first as a consultant to IBM and then as an employee. Steve has supported several industries during his time at IBM, including health care, telephony, and banking and currently supports the IBM Global Finance account as a Team Lead for the Global Contract Management project.

Henry Cui works as an independent consultant through his own company, Kaka Software Solution. He provides consulting services to large financial institutions in Canada. Before this work, Henry worked with the IBM Rational services and support team for eight years, where he helped many clients resolve design, development, and migration issues with Java EE development. His areas of expertise include developing Java EE applications with Rational Application Developer tools and administering WebSphere Application Server servers, security, SOA, and web services. Henry is a frequent contributor of developerWorks articles. He also co-authored five IBM Redbooks publications. Henry holds a degree in Computer Science from York University.

Craig Fleming is a Solution Architect who works for IBM Global Business Services® in Auckland, New Zealand. He has worked for the last 15 years leading and delivering software projects for large enterprises as a solution developer and architect. His area of expertise is in designing and developing middleware solutions, mainly with WebSphere technologies. He has worked in several industries, including Airlines, Insurance, Retail, and Local Government. Craig holds a Bachelor of Science (Honors) in Computer Science from Otago University in New Zealand.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks publications

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright International Business Machines Corporation 2012. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.

This document REDP-4883-00 was created or updated on July 25, 2012.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Redpaper™

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®	Maximo®	Redbooks (logo) ®
developerWorks®	Rational®	System z®
Global Business Services®	Redbooks®	WebSphere®
IBM®	Redpaper™	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.