



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

Developing Web Applications using JavaServer Pages and Servlets

This IBM® Redpaper™ publication illustrates how to develop Java Platform, Enterprise Edition (Java EE) applications using JavaServer Pages (JSP), Java EE servlet technology, and static HTML pages. IBM Rational® Application Developer can help you work with these technologies, and a Redbank example is used to guide you through its features. This paper is intended for web developers interested in Java development.

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

The paper begins by describing the major tools that are available for web developers in Rational Application Developer and then introduces the new features of the latest version. Next, the ITSO RedBank application is built and tested by using Rational Application Developer. The paper concludes with a list of additional information sources about Java EE web components and Rational Application Developer.

The paper is organized into the following sections:

- ▶ Introduction to Java EE web applications
- ▶ Web development tooling
- ▶ Rational Application Developer new features
- ▶ RedBank application design
- ▶ Implementing the RedBank application
- ▶ Web application testing

The sample code for this paper is in the \4880code\webapp folder. For more information about how to download the sample code, see “Locating the web material” on page 64.

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 book is available at the following website:

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

Introduction to Java EE web applications

Java EE is an application development framework that is the most popular standard for building and deploying web applications in Java. Two of the key underlying technologies for building the web components of Java EE applications are servlets and JSP. *Servlets* are Java classes that provide the entry point to logic for handling a web request and return a Java representation of the web response. *JSP* are a mechanism to combine HTML with logic written in Java. After they are compiled and deployed, JSP run as a servlet, where they also take a web request and return a Java object that is representing the response page.

Typically, in a large project, the JSP and servlets are part of the presentation layer of the application and include logic to invoke the higher level business methods. The core business functions are separated into a clearly defined set of interfaces, so that these components can be used and changed independently of the presentation layer (or layers, when using more than one interface).

Enterprise JavaBeans (EJB) are also a key feature included in the Java EE framework and are an option to implement the business logic of an application. The separation of the presentation logic, business logic, and the logic to combine them is referred to as the *model view controller (MVC)* pattern and is described later in this paper.

Technologies, such as Struts, JavaServer Faces (JSF), various JSP tag libraries, and even newer developments, such as Ajax, were developed to extend the JSP and servlets framework to improve various aspects of Java EE web developments. For example, JSF facilitates the construction of reusable user interface (UI) components that can be added to JSP pages. We describe several of these technologies in the original IBM Redbooks publication from which this paper is excerpted (*Rational Application Developer for WebSphere Software V8 Programming Guide*, SG24-7835). However, the underlying technologies of these tools are extensions to Java servlets and JSP.

When planning a new project, the choice of technology depends on several criteria, such as the size of the project, previous implementation patterns, maturity of technology, and skills of the team. Using JSP with servlets and HTML is a comparatively simple option for building Java EE web applications.

Figure 1 shows the relationships among the Java EE, enterprise application, web applications, EJB, servlets, JSP, and additions, such as Struts and JSF.

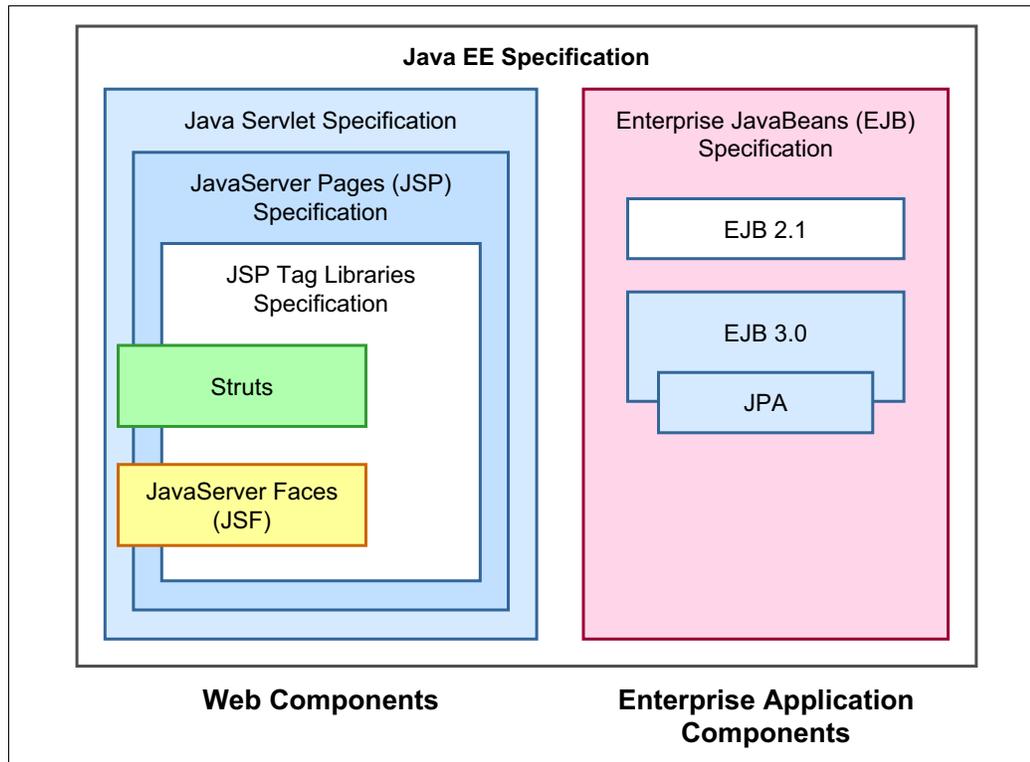


Figure 1 Java EE-related technologies

The focus of this paper is mainly on developing web applications by using JSP, servlets, and static pages that use HTML with the tools included with Rational Application Developer. After you master these concepts, you can more easily understand the other technologies that are available.

Java EE applications

At the highest level, the Java EE specification describes the construction of two application types that can be deployed on any Java EE-compliant application server:

- ▶ Web applications, which are represented by a web archive (WAR) file
- ▶ Enterprise applications, which are represented by an enterprise archive (EAR) file

Both files are constructed in a compressed file format, with a defined directory and file structure. Web applications generally contain the web components that are required to build the information presented to the user and lower-level logic. The enterprise application contains an entire application, including the presentation logic and logic that implements its interactions with an underlying database or other back-end system.

An EAR file can include one or more WAR files where the logic within the web applications usually invokes the application logic in the EAR.

Enterprise applications

An *enterprise application project* contains the collection of resources that are required to deploy an enterprise (Java EE) application to WebSphere Application Server. It can contain a combination of web applications (WAR files), EJB modules, Java libraries, and application

client modules (all stored in JAR format). They also must include a deployment descriptor (an `application.xml` file in the `META-INF` directory), which contains meta information to guide the installation and execution of the application.

On deployment, the EAR file is unwrapped by the application server and the individual components (EJB modules, WAR files, and associated JAR files) are deployed individually. The JAR files within an enterprise application can be used by the other contained modules. This allows code to be shared at the application level by multiple Web or EJB modules.

The use of EJB is not compulsory within an enterprise application. When developing an enterprise application (or even a web application), the developer can write whatever Java logic is the most appropriate for the situation. EJB are the defined standard within Java EE for implementing application logic, but many factors can determine the decision for implementing this part of a solution. In the RedBank sample application, the business logic is implemented by using standard Java classes that use HashMaps to store data.

Web applications

A web application server publishes the contents of a WAR file under a defined URL root (called a *context root*) and then directs web requests to the correct resources and returns the appropriate web response to the requestor. Certain requests can be mapped to a simple static resource, such as HTML files and images. Other requests, which are referred to as *dynamic resources*, are mapped to a specific JSP or servlet class. Through these requests, the Java logic for a web application is initiated and calls to the main business logic are processed.

When a web request is received, the application server looks at the context root of the URL to identify for which WAR the request is intended, and the server reviews the contents after the root to identify to which resource to send the request. This resource might be a static resource (HTML file), the contents of which are returned, or a dynamic resource (servlet or JSP), where the processing of the request is handed over to JSP or servlet code.

In every WAR file, descriptive meta information describes this information and guides the application server in its deployment and execution of the servlets and JSP within the web application.

The structure of these elements within the WAR file is standardized and compatible between various web application servers. The Java EE specification defines the hierarchical structure for the contents of a web application that can be used for deployment and packaging purposes. All Java EE-compliant servlet containers, including the test web environment provided by Rational Application Developer, support this structure.

Figure 2 shows the structure of a WAR file, an EAR file, and a JAR file.

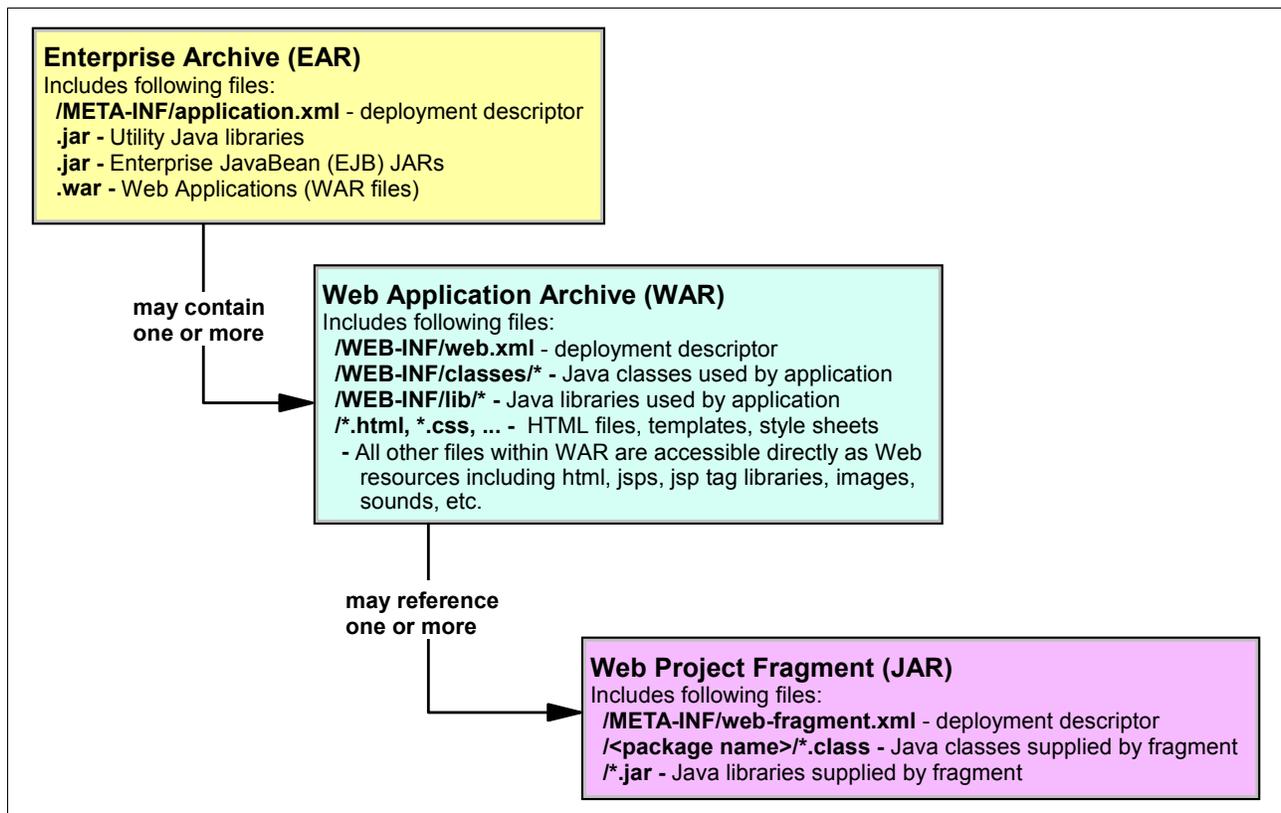


Figure 2 Structures of EAR, WAR, and JAR files

The *Java Specification Requests (JSR) 315: Java Servlet 3.0 Specification* (see <http://jcp.org/en/jsr/detail?id=315>) includes a series of Java annotations for declaring the fundamental classes that make up a JEE web application. These classes are used to define servlets, URL mappings to servlets, security (definition of which user groups can access a particular set of URLs), filters (a mechanism to call certain Java code before a request is processed), listeners (a mechanism to call specific Java code on certain events), and configuration parameters to be passed to servlets or the application as a whole. In previous versions, these classes were defined in the `web.xml` file and the new version removes this dependency. However, it still can be used if required and Rational Application Developer includes tooling support for Version 2.5 and Version 3.0 of the Servlet specification.

A key extension in the latest version of the JEE specification (Version 6) is the addition of Web Fragment projects. Web Fragment projects are a mechanism to partition the web components in a WAR file into separate JAR files (called *Web Fragments*), which enhance or provide utility/framework logic to the main WAR file.

There are no requirements for the directory structure of a web application outside of the `WEB-INF` directory. All these resources are accessible to clients (general web browsers) directly from a URL, given the context root. Naturally, you must structure the web resources in a logical way for easy management. For example, use an `images` folder to store graphics.

Java EE web APIs

Figure 3 shows the main classes that are used within the Java EE framework and the interactions between them. The application servlet class is the only class outside of the Java EE framework and contains the application logic.

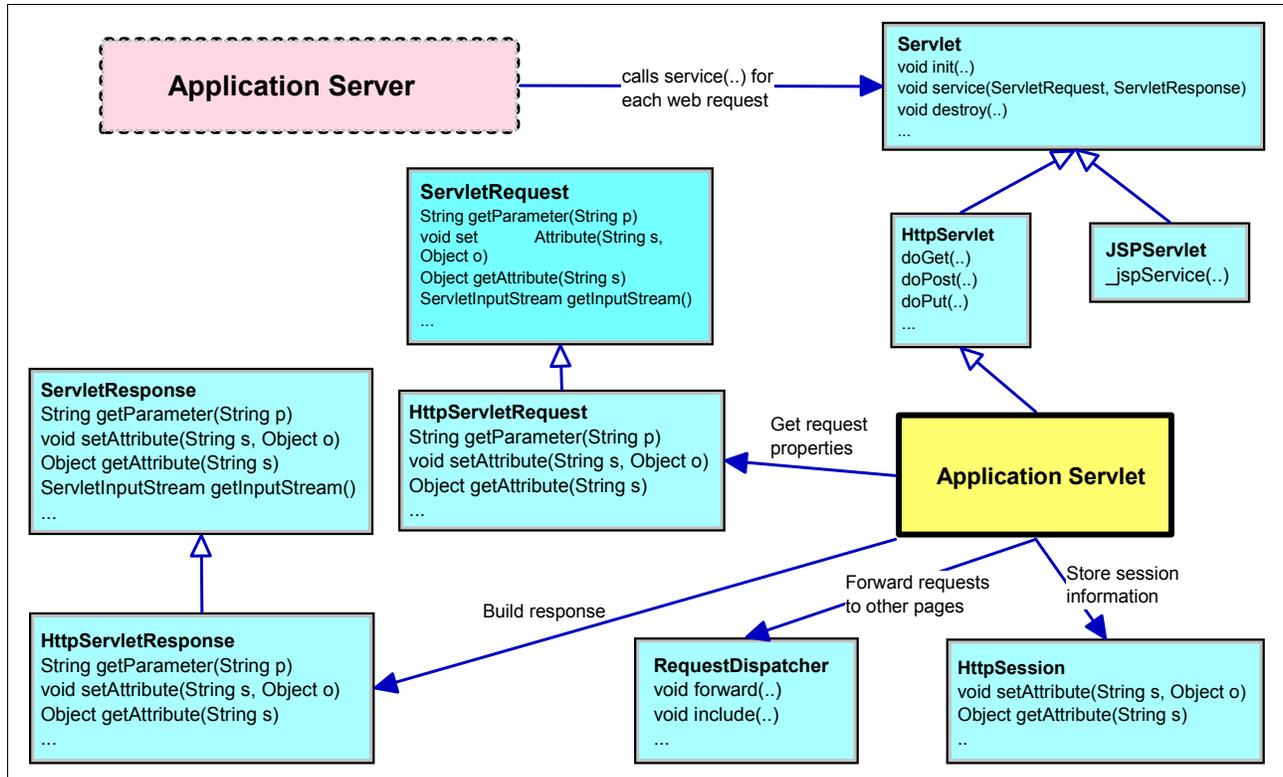


Figure 3 Java EE web component classes

The Java EE framework has the following main classes:

- ▶ `HttpServlet` (extends `Servlet`): The main entry point for handling a web request. The `doGet`, `doPost`, and other methods invoke the logic for building the response given the request and the underlying business data and logic. In the *JEE Servlet 3.0 specification*, these classes are identified by the `@WebServlet` annotation.
- ▶ `HttpJSPServlet` (extends `Servlet`): The WebSphere Application Server automatically compiles a JSP page into a class that extends this type. It runs similarly to a normal servlet, and its only entry point is the `_jspService` method.
- ▶ `HttpRequest` (extends `Request`): This class provides an API to access all pertinent information in a request.
- ▶ `HttpResponse` (extends `Response`): This class provides an API to create a response to a request and the application state.
- ▶ `HttpSession`: This class stores any information required to be stored across a user session with the application (as opposed to a single request).
- ▶ `RequestDispatcher`: Within a web application, redirecting the processing of a request to another servlet is required. This class provides methods to redirect the processing of a request to another servlet.

Other classes are available in the Java EE web components framework. For a full description of the classes that are available, see “More information” on page 63, which provides a link to the Java EE servlet specifications.

JSP

You can include any valid fragment of Java code in a JSP page and mix it with HTML. In the JSP file, the Java code is marked by `<%` and `%>`. On deployment (or sometimes the first page request, depending on the configuration), the JSP is compiled into a servlet class. This process combines the HTML and the scriptlets in the JSP file in the `_jspService` method that populates the `HttpResponse` variable. Combining many complex Java code with HTML can result in a complicated JSP file. Except for simple examples, avoid this practice.

One way around this situation is to use custom *JSP tag libraries*. These libraries are tags defined by developers that initiate calls to a Java class within a JSP page. These classes implement `Tag`, `BodyTag`, or `IterationTag` interfaces from the `javax.servlet.jsp.tagext` package, which is part of the Java EE framework. Each tag library is defined by a `.tld` file that includes the location and content of the `taglib` class file. Although not strictly required, you need to include a reference to this file in the deployment descriptor.

XML-based tag files: JSP 2.0 introduces XML-based tag files. Tag files no longer require a `.tld` file. Tags can now be developed by using JSP or XML syntax.

The most widely available tag library is the JavaServer Pages Standard Tag Library (JSTL), which provides simple tags to handle simple operations required in most JSP programming tasks, including looping, globalization, XML manipulation, and even processing of SQL result sets. The RedBank application uses JSTL tags to display tables and add URLs to a page. The final section of this paper contains references to learn more about JSP and tag libraries.

Model view controller pattern

The model view controller (MVC) concept is a pattern used often when describing and building applications with a user interface component, including Java EE applications.

Following the MVC concept, a software application or module must have its business logic (model) separated from its presentation logic (view). This separation is desirable, because it is likely that the view will change over time, and it might not be necessary to change the business logic each time. Also, many applications might have multiple views of the same business model. If the view is not separated, adding a view causes considerable disruptions and increases the complexity of the component.

You can achieve this separation through the layering of the component into a *model* layer (responsible for implementing the business logic) and a *view* layer (responsible for rendering the user interface to a specific client type). In addition, the *controller* layer sits between those two layers, intercepting requests from the view (or presentation) layer and mapping them to calls to the business model, then returning the response based on a response page selected by the controller layer. The key advantage provided by the controller layer is that the presentation can focus only on the presentation aspects of the application and leave the flow control and mapping to the controller layer.

You can achieve this separation in Java EE applications in several ways. Various technologies, such as JSF and Struts, differ in the ways that they apply the MVC pattern. Our focus in this paper is on JSP and servlets that fit into the view and controller layers of the MVC pattern. If only servlets and JSP are used in an application, the details of how to implement the controller layer are left to whatever mechanism the development team decides is appropriate and that they can create by using Java classes.

In the example later in this paper, a *command* pattern (see Eric Gamma, and others, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN 0-201-63361-2) is applied to encompass the request to the business logic and interactions made with the business logic through a facade class. In the other interactions, the request is sent directly to a servlet that makes method calls through the facade.

Web development tooling

Rational Application Developer includes many web development tools for building static and dynamic web applications. Many of these web development tools focus on technologies, such as Web 2.0 technologies, portals, and JSF. In this section, we highlight the following tools and features, which focus on the more fundamental aspects of web development:

- ▶ Web perspective and views
- ▶ Page Designer
- ▶ Page templates
- ▶ CSS Designer
- ▶ Security Editor
- ▶ File creation wizards

Web perspective and views

The Web perspective helps web developers build and edit web resources, such as servlets, JSP, HTML pages, style sheets images, and deployment descriptor files.

To open the Web perspective, from the workbench, select **Window** → **Open Perspective** → **Web**. Figure 4 on page 9 shows the default layout of the Web perspective with a simple `index.html` that was published to the local test service and is being viewed.

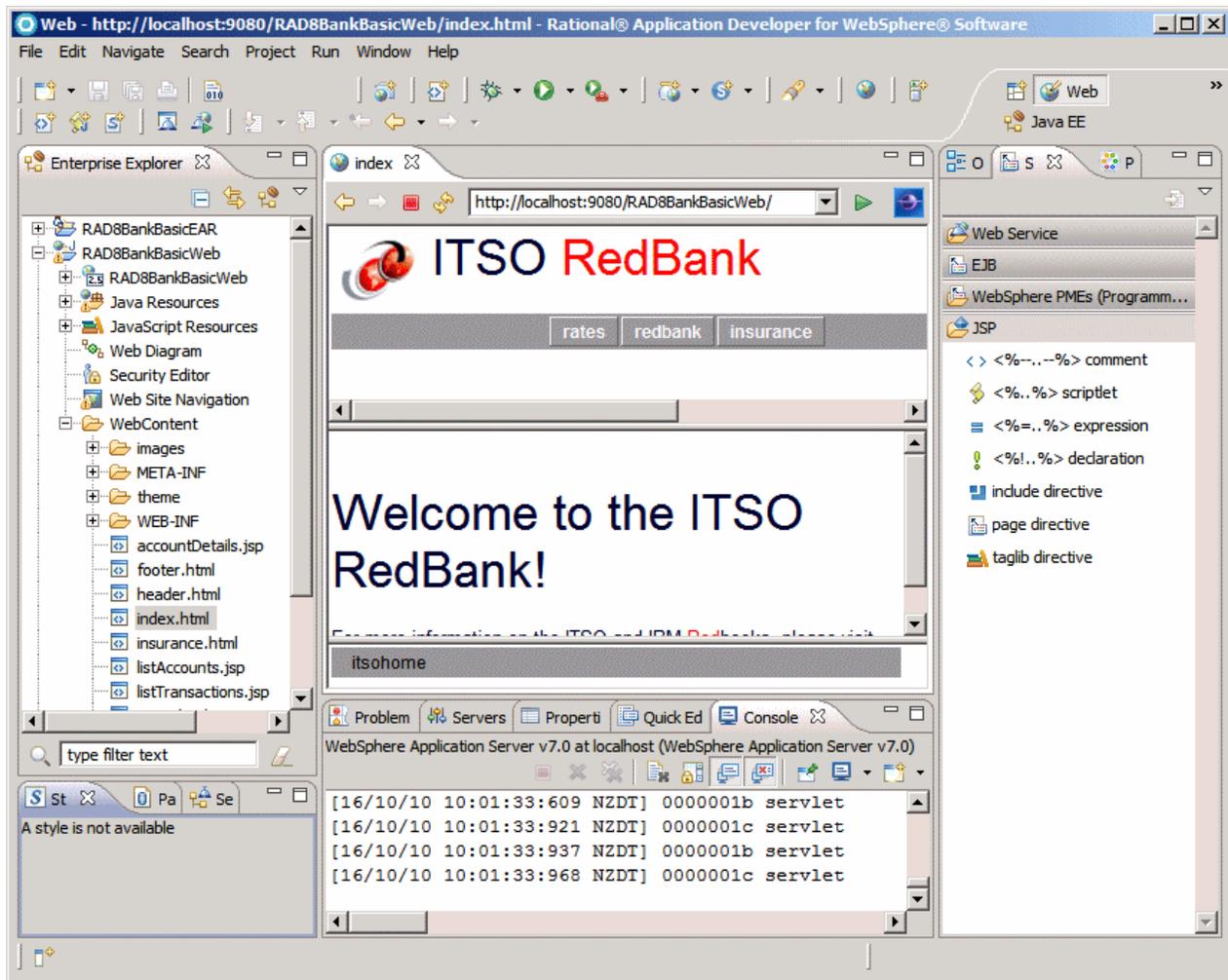


Figure 4 Web perspective in Rational Application Developer

In the Web perspective, many views are accessible (by selecting **Window** → **Show View**), several of which are already open in the Web perspective default setting.

By default, the following views are available in the Web perspective:

- ▶ Console view: This view shows output to SystemOut from any running processes.
- ▶ Outline view: This view shows an outline of the file that is being viewed. For HTML and JSP, this view shows a hierarchy of tags around the current cursor position. Selecting a tag in this view moves the cursor in the main view to the selected element. This view is useful for moving quickly around a large HTML file.
- ▶ Page Data view: When editing JSP files, this view gives a list of any page or scripting variables available.
- ▶ Page Designer view: This view is a “what you see is what you get” (WYSIWYG) editor for JSP and HTML that consists of four tabs:
 - Design: the user can drag components to the page
 - Source: shows the HTML
 - Split: shows the Design in the top half and the Source in the bottom half
 - Preview: shows how the final page looks in either Internet Explorer or Firefox

- ▶ **Palette view:** When editing JSP or HTML files, this view provides a list of HTML items (arranged in drawers) that can be dragged onto pages.
- ▶ **Problems view:** This view shows ding errors, warnings, or informational messages for the current workspace.
- ▶ **Enterprise Explorer view:** This view shows a hierarchy view of all projects, folders, and files in the workspace. In the Web perspective, it structures the information within web projects in a way that makes navigation easier.
- ▶ **Properties view:** This view shows the properties for the item currently selected in the main editor.
- ▶ **Quick Edit view:** When editing HTML or JSP files, this view provides a mechanism to quickly add Java Script to a window component on certain events, for example, `onClick`.
- ▶ **Servers view:** Use this view if you want to start or stop test servers while debugging.
- ▶ **Services view:** This view shows a summary of all the web services present in the projects on the workspace.
- ▶ **Snippets view:** In this view, you can edit small bits of code, including adding and editing actions assigned to tags. You can drag items from the Snippets view to the Quick Edit view.
- ▶ **Styles view:** With this view, you can edit and apply both pre-built and user-defined styles sheets to HTML elements and files.
- ▶ **Thumbnails view:** Given the selection of a particular folder in the Project Explorer view, this view shows the contents of the folder.

Page Designer

The Page Designer is the primary editor within Rational Application Developer for building HTML, XHTML, JSP, and JSF source code. Web designers can drag web page items from the Palette view to the desired position on the web page.

It provides the following representations of a page:

- ▶ The *Design tab* provides a WYSIWYG environment to visually design the contents of the page. A good development technique is to work within the Design tab of the Page Designer and build up the HTML contents by clicking and dragging items from the Palette view onto the page and arranging them with the mouse or editing properties directly from the Properties view. Tags can be positioned as absolute instead of relative.
- ▶ The *Source tab* provides access to the page source code that shows the raw HTML or JSP contents. You can use the Source tab to change details that are not immediately obvious in the Design tab.
- ▶ The *Split tab* (Figure 5 on page 11) combines the Source tab and either the Design tab or the Preview tab in a split-screen view. The Split tab is helpful to see the Design tab and Source tab in one view; the changes are immediately reflected.

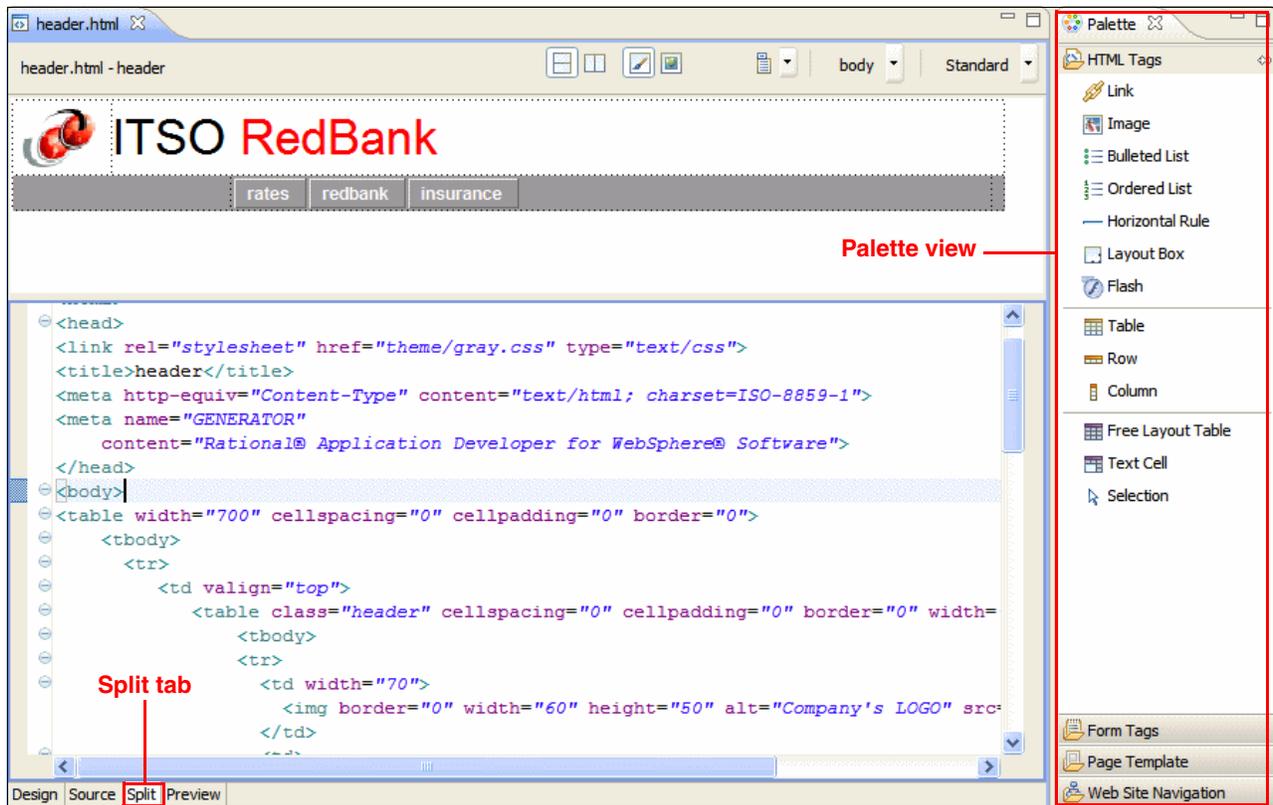


Figure 5 Page Designer Split tab

- ▶ The *Preview tab* shows how the page looks when it is displayed in a web browser. You can use the Preview tab throughout the process to verify the look of the final result in either Firefox or Internet Explorer.

Another important feature of Page Designer is the ability to automatically convert HTML elements to related elements. For example, there are menu options to convert Text entry fields to Password entry fields, which you can achieve from the context menu (**Convert Widget → HTML Form Widgets**) of the element in the design view (see Figure 6 on page 12). Rational Application Developer comments the original element for reference. Rational Application Developer supports conversions between selected HTML and Dojo widgets.

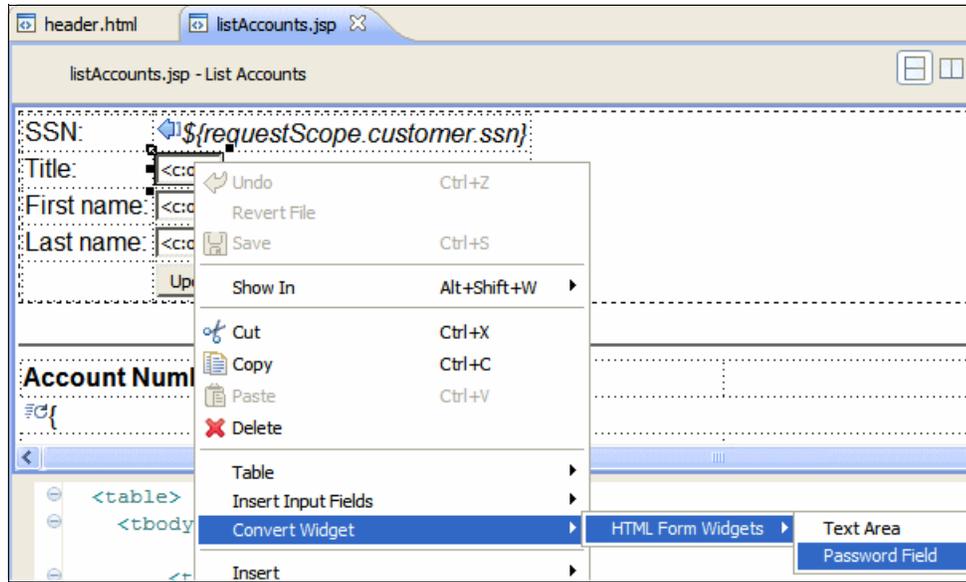


Figure 6 Convert Widget in Page Designer

Finally, HTML content is often provided to a development team and created from tools other than Rational Application Developer. These files can be imported by using the context menu on the target directory, selecting **File** → **Import** → **General** → **File System**, browsing to the new file, and clicking **Import**. When an imported file is opened in the Page Designer, all the standard editing features are available.

More information: For a detailed example of using the Page Designer, see “Developing the static web resources” on page 35, and “Working with JSP” on page 46.

Page templates

A *page template* contains common areas that you want to appear on all pages, and content areas that are intended to be unique on each page. Page templates are used to provide a common design for a web project.

The Page Template File creation wizard is used to create these files. After being created, the file can be modified in the Page Designer. The page templates are stored as *.html files for HTML pages and *.jspx files for JSP pages. Changes to the page template are reflected in pages that use that template. Templates can be applied to individual pages, groups of pages, or applied to an entire web project, and they can be replaced or updated for a website. Areas can be marked as read-only, meaning that the Page Designer does not allow the developer to modify those areas, ensuring that certain designated areas are never changed accidentally.

When creating a page template, you are prompted to choose whether the template is a dynamic page template or a design-time template:

- ▶ *Dynamic page templates* use Struts-Tiles technology to generate pages on the web server.
- ▶ *Design-time templates* allow changes to be made and applied to the template at design or build time. However, after an application is deployed and running, the page template cannot be changed. Design time templates do not rely on any technologies other than the standard Java EE servlet libraries.

CSS Designer

Cascading style sheets (CSS) are used with HTML pages to ensure that an application has consistent colors, fonts, and sizes across all pages. You can create a default style sheet when creating a project. Several samples are included with Rational Application Developer.

At the start of the development effort, decide on the overall theme (color or fonts) for a web application and create the style sheet. Then, as you create the HTML and JSP files, you can select that style sheet to ensure that web pages have a consistent design. Style sheets are commonly kept in the `WebContent/theme` folder.

The CSS Designer is used to modify cascading style sheet (*.css) files. It provides two panels. On the right side, it shows all the text types and their respective fonts, sizes, and colors, which are all editable. On the left side, you see a sample of how the various settings look. Any changes that are made are immediately applied to the design in the Page Designer if the HTML file is linked to the CSS file.

More information: For an example of customizing style sheets used by a page template, see “Customizing a style sheet” on page 33.

Security Editor

Rational Application Developer features the Security Editor, which provides a wizard to specify security groups within a web application and the URLs to which a group has access. With the Java EE specification, you can define, in the deployment descriptor, security groups and levels of access to defined sets of URLs. The Security Editor (Figure 7 on page 14) provides an interface for this information.

Selecting an entry in the Security Roles pane shows the resources members of that role in the Resources pane and the constraint rules that are applicable for the role and resource (if one is selected). Each entry in the Constraints window has a list of resource collections that specify the resources available to it and which HTTP methods can be used to access these resources. Using context menus, it is possible to create new roles and security constraints and to add resource collections to these constraints.

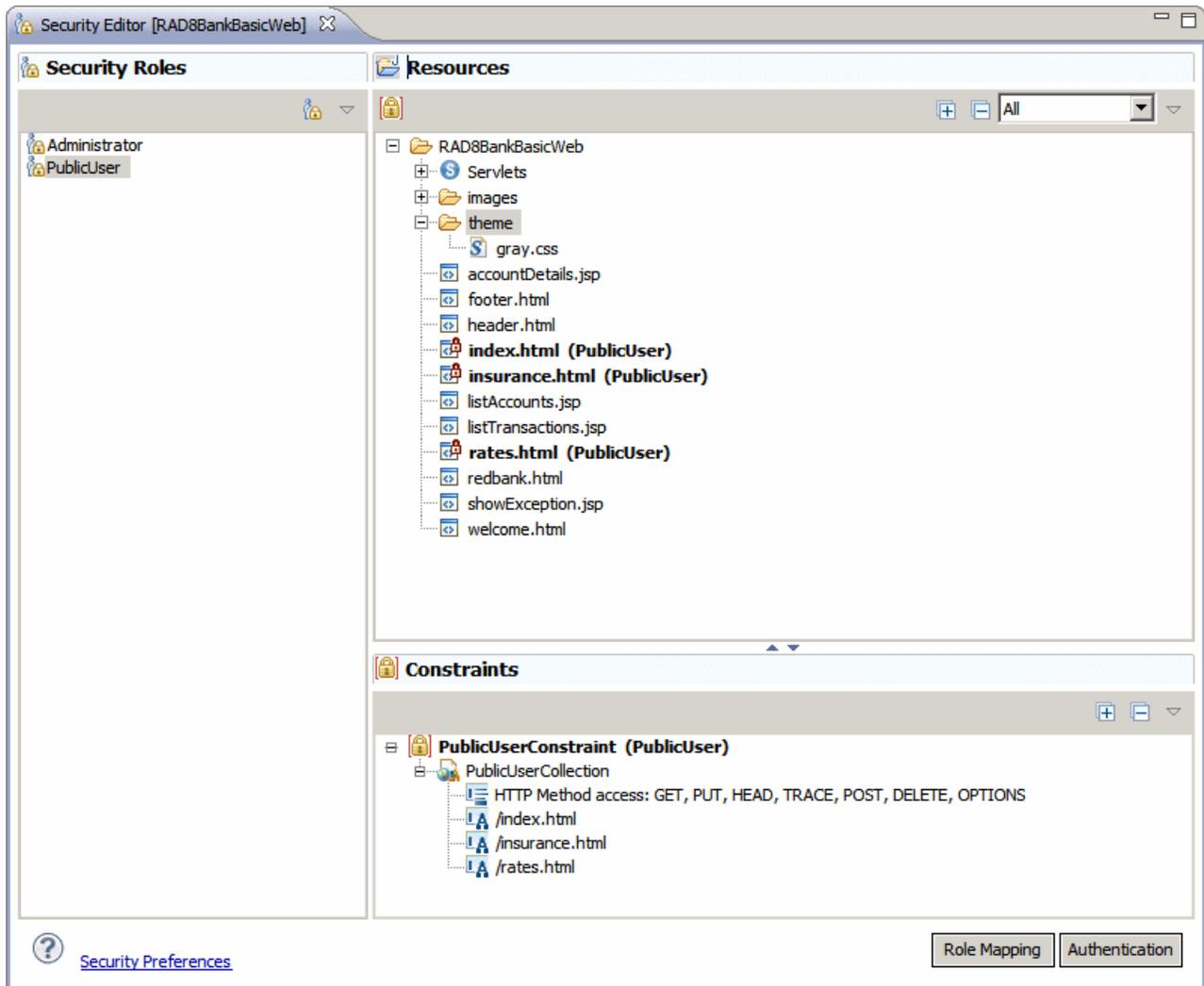


Figure 7 Security Editor

The Java EE security specification defines the mechanism for declaring groups and the URL sets that each group can access, but it is up to the Web container to map this information to an external security system. The WebSphere administrative console provides the mechanism to configure an external Lightweight Directory Access Protocol (LDAP) directory. See the IBM Redbooks publication *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297.

File creation wizards

Rational Application Developer provides many web development file creation wizards. You can access them by selecting **File** → **New** → **Other**. Then from the Select a wizard window, expand the **Web** folder and select the type of file required. These wizards prompt you for the key features of the new artifact and can help you to quickly get a skeleton of the component that you need. You can always manipulate the artifact created by the wizard directly, if necessary.

The following wizards are available in the Web perspective:

- ▶ **CSS:** You can use the CSS file wizard to create a new cascading style sheet (CSS) in a specified folder.
- ▶ **Dynamic Web Project:** This wizard steps you through the creation of a new web project, including which features the project uses and any page templates present.
- ▶ **Web Fragment Project:** This wizard guides you through the steps to make a web fragment project, including the link to the target dynamic web project.
- ▶ **Filter:** This wizard constructs a skeleton Java class for a Java EE filter, which provides a mechanism for processing on a web request before it reaches a servlet. The wizard also updates the `web.xml` file with the filter details.
- ▶ **Filter Mapping:** This wizard steps you through the creation of a set of URLs with which to map a Java EE filter. The result of this wizard is stored in the deployment descriptor.
- ▶ **HTML:** This wizard steps you through the creation an HTML file in a specified folder, with the option to use HTML Templates.
- ▶ **JSP:** This wizard steps you through the creation of a JSP file in a specified folder, with the option to use JSP Templates.
- ▶ **Listener:** You can use a listener to monitor and react to events in the lifecycle of a servlet or application by defining methods that are started when lifecycle events occur. This wizard guides you through the creation of such a listener and to select the application lifecycle events to which to listen.
- ▶ **Security Constraint:** You can use this wizard to populate the `<security-constraint>` in the deployment descriptor that contains a set of URLs and a set of http methods, which members of a particular security role are entitled to access.
- ▶ **Security Role:** This wizard adds a `<security-role>` element to the deployment descriptor.
- ▶ **Servlet:** You can use this wizard to create a skeleton servlet class and add the servlet with appropriate annotations or add to the deployment descriptor.
- ▶ **Servlet Mapping:** This wizard steps you through the creation of a new URL to servlet mapping and adds appropriate annotations or information to the deployment descriptor.
- ▶ **Static Web Project:** This wizard steps you through building a new web project that contains only static pages.
- ▶ **JSP Tag:** This wizard steps you through creating a Tag library file.
- ▶ **Web Page:** By using this wizard, you can create an HTML or JSP file in a specified folder, with the option to create from many page templates.
- ▶ **Web Page Template:** The Page Template File wizard is used to create new page template files in a specified folder. You can optionally create from a page template or create as a JSP fragment and define the markup language (HTML, HTML Frameset, Compact HTML, XHTML, XHTML Frameset, and Wireless Markup Language (WML) 1.3). You can select from one of the following models: Template containing Faces Components, Template containing only HTML, or Template containing JSP.

Dojo, Struts, and JSF: Several wizards are available in the Web perspective specifically for Dojo, widgets, and JSF, which are discussed in the original IBM Redbooks publication from which this paper is excerpted (*Rational Application Developer for WebSphere Software V8 Programming Guide*, SG24-7835).

Rational Application Developer new features

Significant improvements to the previous version of Rational Application Developer were made in the tools that are available for creating artifacts within a web application:

- ▶ Support for JEE Servlet 3.0 specification: The Servlet 3.0 specification revision is a major revision of the specification and includes the following changes:
 - Use of annotations versus deployment descriptor: Under the servlet 3.0 specification, servlets, servlet mappings, filters, and listeners can be declared by using annotations rather than being declared in the deployment descriptor. This feature reduces the amount of configuration required for each web application. The tooling support in Rational Application Developer was changed to cater for declarations made in annotations or the deployment descriptor.
 - Support for Web Fragment projects: A Web Fragment project is a new project type that allows logical partitioning of the web application in such a way that the utility projects or frameworks that are used within the web application can define all the artifacts without requiring you to edit or add information in `web.xml`. There is a reference inside the web project to the Web Fragment project, which contributes to it. You do not need to update the deployment descriptor to declare the web fragment project used.
 - Ability to store an EJB directly in a WAR: New to the Servlet 3.0 specification and Rational Application Developer is the ability to include EJBs directly in the WAR.
 - Ability to programmatically add and remove servlets: The Servlet 3.0 specification includes API changes to allow developers to add and remove servlets as an application runs. However, there are no specific features to achieve this capability in Rational Application Developer.
- ▶ HTML support: Page Designer does not support the new elements and attributes that are defined in the HTML5 specification. HTML5 is only supported in our source editors. Page Designer support for earlier versions of HTML remains. Elements, such as `frameset`, and the `height/width`, `spacing`, and `padding` attributes within tables are not recommended in HTML5. If these elements and attributes are used, Page Designer underlines the attribute or element and raises a warning.
- ▶ Page Designer enhancements:
 - The preview tab now includes options for previewing a web page in Internet Explorer or Mozilla Firefox.
 - There is a feature to convert HTML widgets to related widgets through a context menu.
 - Users can show all pages affected by a CSS style rule. When a CSS file is edited, users can see the affected HTML or JSP files in the search view and decide whether to keep the modified style rule or not.
 - There is support for Scalable Vector Graphics (SVG) images and Flash widgets in Page Designer.
 - *JSP fragment files* (JSPF) are a mechanism to break up JSP pages into reusable blocks (fragments) that can be assembled on a standard JSP page. Page Designer now includes additional tooling to understand how a fragment is used when it is included in separate contexts.

RedBank application design

In this section, we describe the design for the ITS0 RedBank web application. We outline the design of the RedBank application and explain how it fits into the Java EE web framework, particularly about JSP and servlets.

Model

The model for the RedBank project is implemented by using a simple Java project and exposed to other components through a facade interface (called ITS0Bank). The main ITS0Bank object is a Singleton object, accessible by a single static public method called getBank.

The ITS0Bank object is composed of the other business objects that make up the application, including Customer, Account, and Transaction. The facade into the bank object includes methods, such as getCustomer, getAccounts, and withdraw, deposit, and transfer. Figure 8 shows a simplified Unified Modeling Language (UML) class diagram of the model.

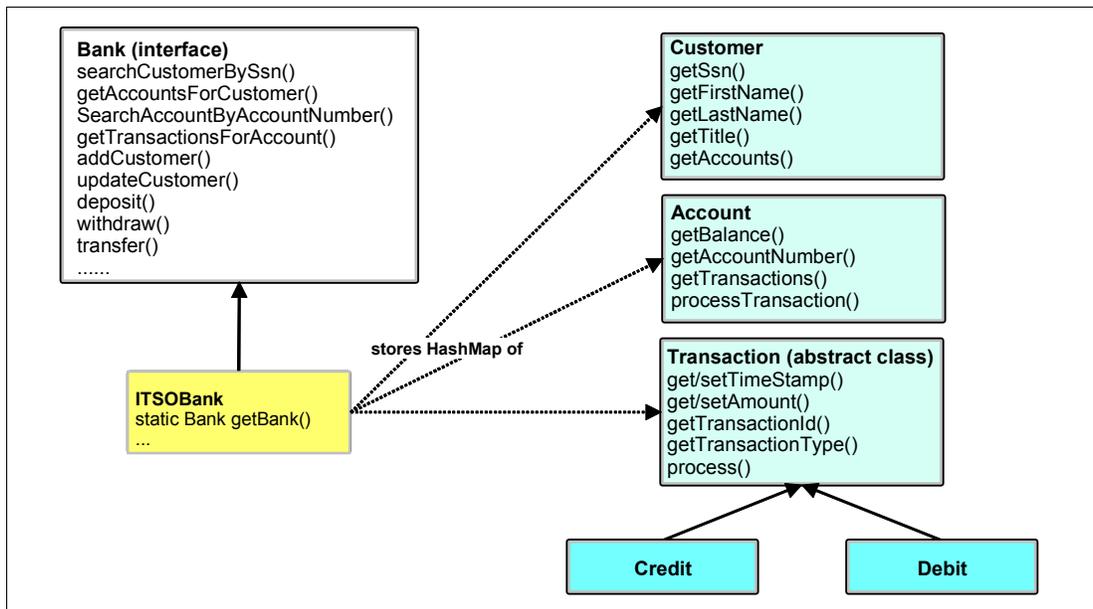


Figure 8 Class diagram for RedBank model

The underlying technology to store data that is used by the ITS0Bank application involves Java HashMaps. These Java HashMaps are populated at startup in the constructor, and the data is lost every time that the application is restarted. In an actual client example, the data might be stored in a database. For the purposes of this example, HashMaps are acceptable.

For information about how to modify the ITS0Bank model to run as EJB and Java Persistence API (JPA) entities and how to store the application data in a database, see the IBM Redpaper publication, *Developing Enterprise JavaBeans Applications*, REDP-4885.

View layer

The view layer of the RedBank application consists of four HTML files and four JSP files. The application home page is the `index.html` file that contains a link to four HTML pages (the `welcome.html`, `rates.html`, `insurance.html`, and `redbank.html` pages). The `welcome.html`, `rates.html`, and `insurance.html` pages are simple static HTML pages that show information without forms or entry fields.

The `redbank.html` page contains a single form in which a user can type the customer ID to access customer services, such as accessing a balance and performing transactions. Although the account number is verified, we do not cover security issues (logon and password) in this example.

From the `redbank.html` page, the user sees the `listAccounts.jsp` page, which shows the customer's details, a list of accounts, and a button to log out.

Selecting an account opens the `accountDetails.jsp` page, which shows the balance for the selected account and a form through which a transaction can be performed. This page also shows the current account number and balance, which are both dynamic values. A simple JavaScript code controls whether the amount and destination account fields are available, depending on the option selected. One of the transaction options on the `accountDetails.jsp` page is List Transactions, which invokes the `listTransactions.jsp` page.

If anything goes wrong in the regular flow of events, the exception page (`showException.jsp`) is displayed to inform the user of the error.

The `listAccounts.jsp`, `accountDetails.jsp`, `listTransactions.jsp`, and `showException.jsp` JSP pages make up the dynamic pages of the RedBank application.

Controller layer

The controller layer was implemented by using two strategies, one straightforward strategy and one complex strategy, which is more applicable to an actual client situation.

The application has the following servlets:

ListAccounts	Gets the list of accounts for one customer.
AccountDetails	Shows the account balance and the selection of operations: list transactions, deposit, withdraw, and transfer.
Logout	Invalidates the session data.
PerformTransaction	Performs the selected operation by calling the appropriate control action: ListTransactions, Deposit, Withdraw, or Transfer.
UpdateCustomer	Updates the customer information.

The first three servlets use a simple function call from the servlet to the model classes to implement their controller logic and then use `RequestDispatcher` to forward control to another JSP or HTML resource. Figure 9 on page 19 shows the pattern that is used in the sequence diagram.

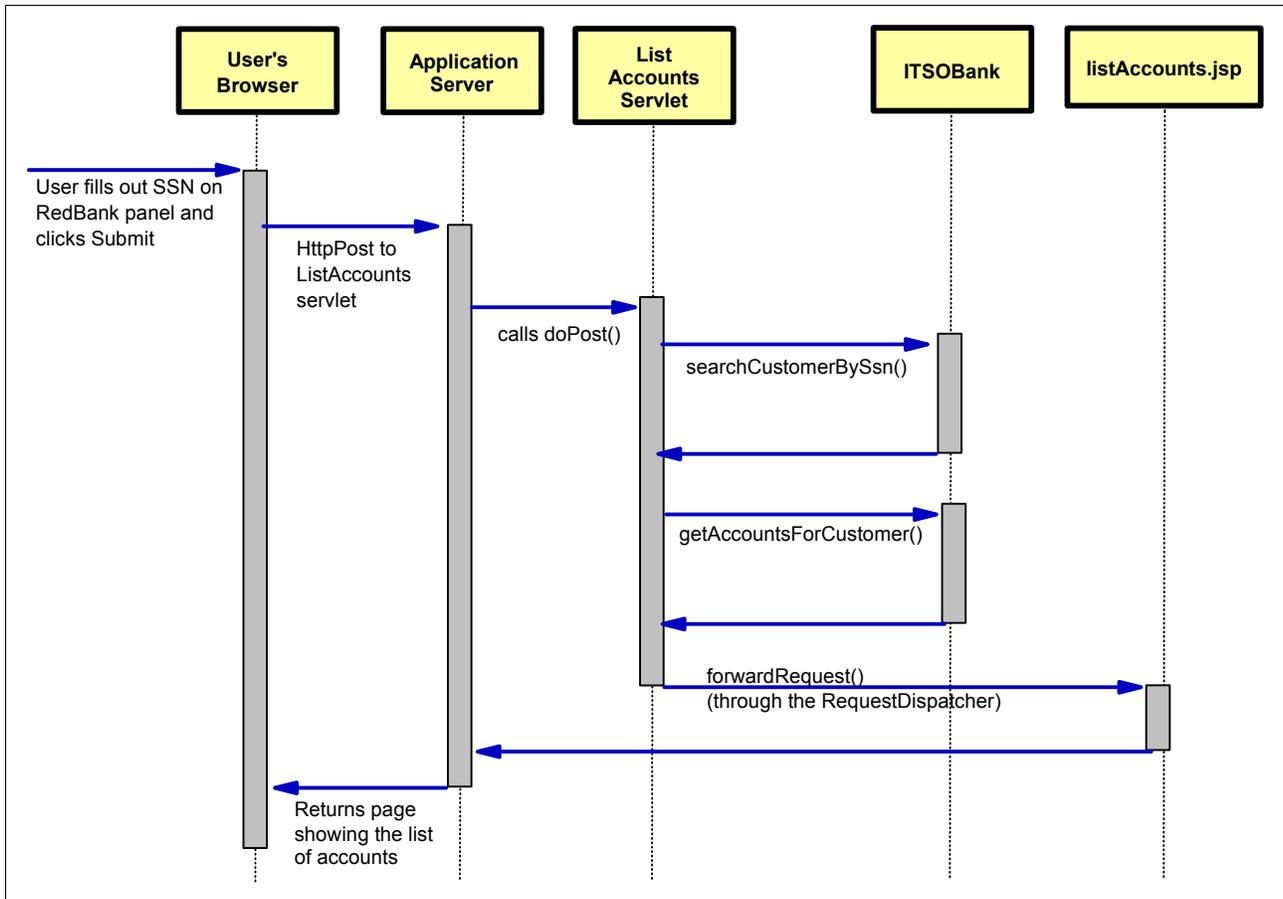


Figure 9 ListAccounts sequence diagram

PerformTransaction uses a separate implementation pattern. It acts as a front controller, receiving the HTTP request and passing it to the appropriate control action object. These objects are responsible for carrying out the control of the application. Figure 10 on page 20 shows a sequence diagram for the list transaction operation from the account details page, including the function calls through PerformTransaction, the ListTransactionsCommand class, onto the model classes, and forwarding to the appropriate JSP.

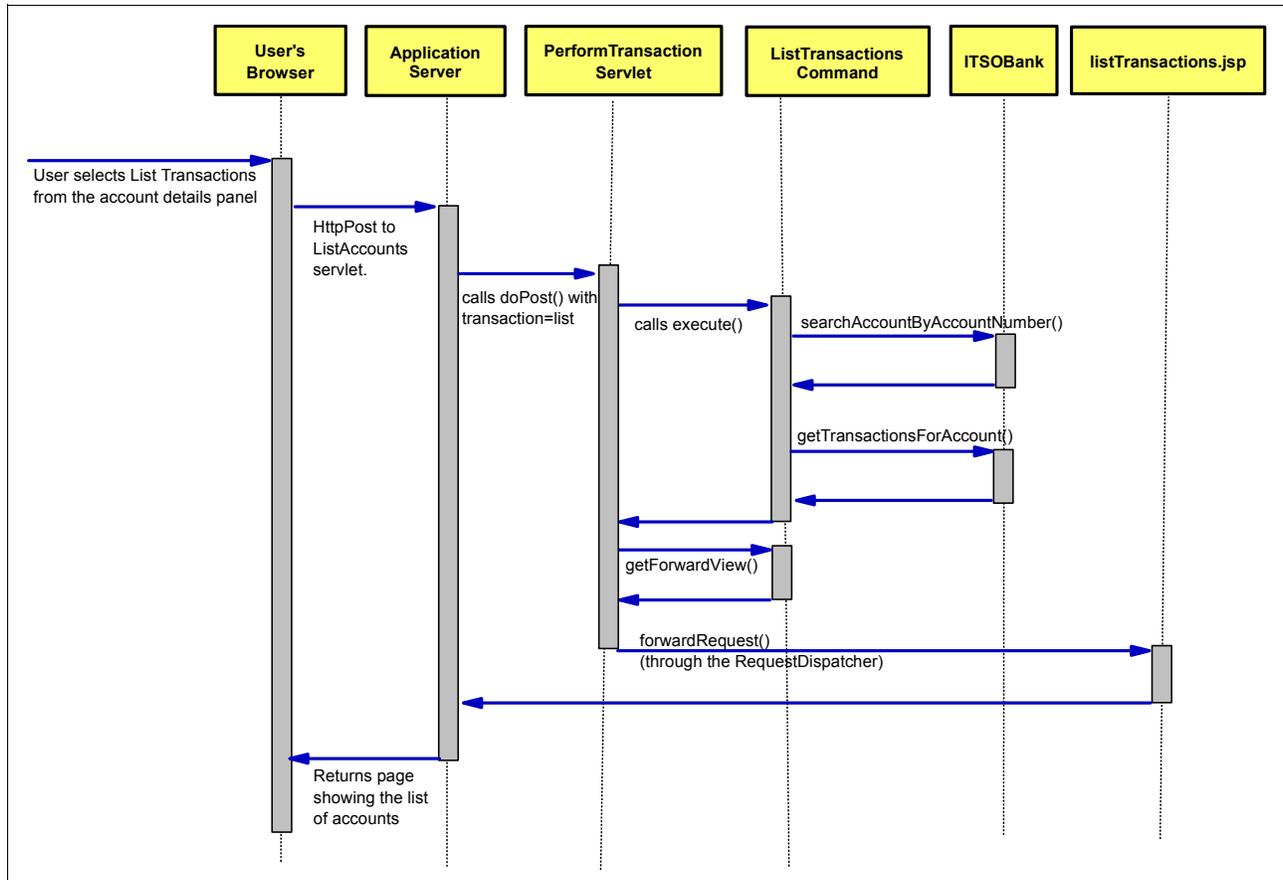


Figure 10 PerformTransaction sequence diagram

The Struts framework provides a much more detailed implementation of this strategy and in a standardized way.

Action objects: Action objects, or commands, are part of the Command design pattern. For more information, see Eric Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN 0-201-63361-2.

Implementing the RedBank application

In this section, we use an example to introduce you to the tools within Rational Application Developer that facilitate the development of web applications. In the example, we create separate web artifacts (including page templates, HTML, JSP, and servlets) and demonstrate how to use the available tools.

The section is organized in the following way:

- ▶ Creating the web project
- ▶ Importing the Java RedBank model
- ▶ Defining the empty web pages
- ▶ Creating frameset pages
- ▶ Customizing frameset web page areas
- ▶ Customizing a style sheet
- ▶ Verifying the site navigation and page templates

- ▶ Developing the static web resources
- ▶ Developing the dynamic web resources
- ▶ Working with JSP

At the end of this section, the RedBank application is ready for testing.

Creating the web project

The first step is to create a web project in the workspace.

Enabling web capabilities: Before you begin, ensure that web capabilities are enabled. Select **Windows** → **Preferences**, expand **General** → **Capabilities**, and ensure that the **Web Developer** options (including basic, typical, and advanced) are selected.

Two types of web projects are available in Rational Application Developer: static and dynamic. Static web projects contain static HTML resources and no Java code and thus are comparatively simple. To demonstrate as many features of Rational Application Developer as possible, and because the RedBank application contains both static and dynamic content, we use a dynamic web project for this example.

In Rational Application Developer, perform the following steps:

1. Select **Window** → **Open Perspective** → **Web** to open the Web perspective.
2. To create a web project, select **File** → **New** → **Dynamic Web Project**.
3. In the New Dynamic Web Project window (Figure 11 on page 22), enter the following items:
 - a. In the Name field, type `RAD8BankBasicWeb`.
 - b. For Project location, select **Use Default** (default). This setting specifies where to place the project files on the file system. It is also acceptable to use the default option of leaving them in the workspace.
 - c. The Target Runtime option shows the supported test environments that are installed. Select **WebSphere Application Server v8.0 Beta**.

Target Server Environment: Although it is possible to re-create the sample by using WebSphere Application Server V7.0 Beta, the sample code provided in this paper does not work if WebSphere Application Server V7 is chosen. It is written in Servlet Version 3.0, which is only supported by WebSphere Application Server V8 Beta and later.

- d. For the Dynamic Web Module version, select **3.0**.
- e. For Configuration, define the configuration as the last item.
- f. For the EAR Membership, select **Add module to an EAR** (default). Dynamic web projects, such as the one we are creating, run exclusively within an enterprise application. For this reason, you must either create an EAR project or select an existing project.
- g. For the EAR Project Name, enter `RAD8BankBasicWebEAR` (the default). Because we select **Add module to an EAR**, the wizard creates an EAR project.

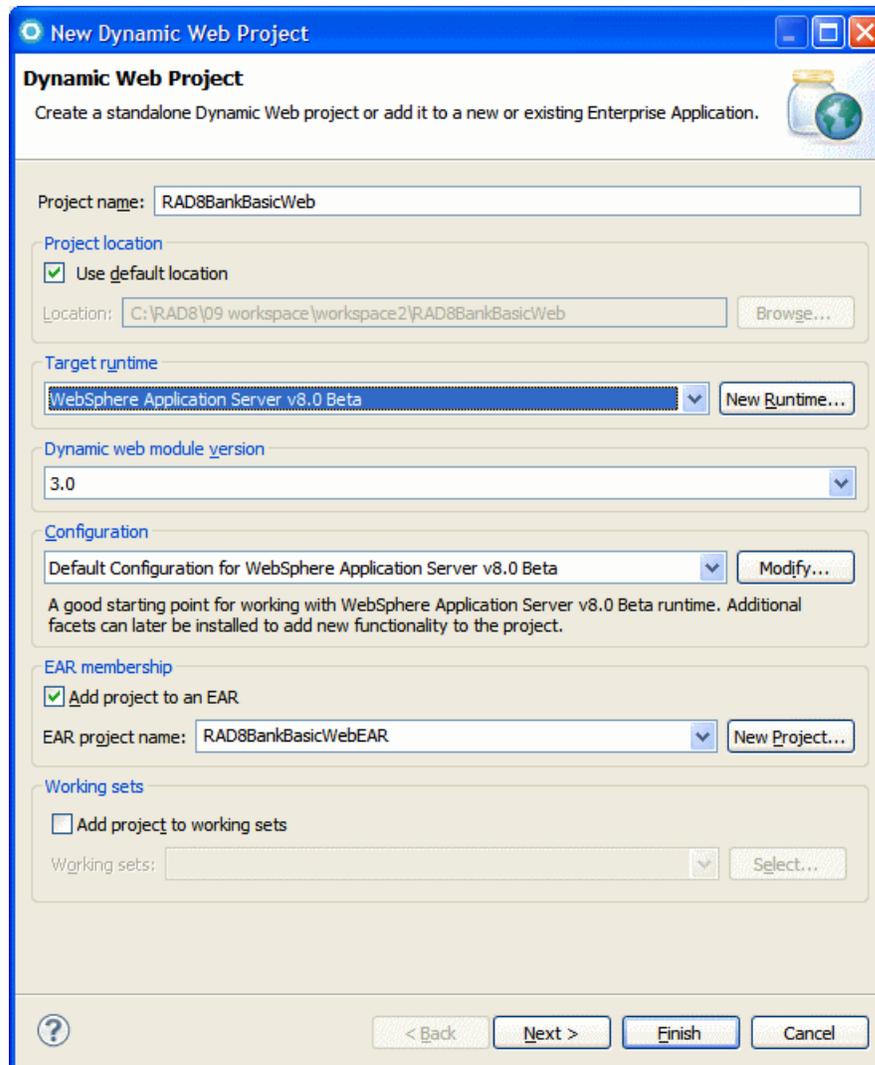


Figure 11 New Dynamic Web Project window

- h. For the Configuration section, click **Modify**.
4. In the Project Facets window (Figure 12 on page 23), select the additional features **Default Style Sheet** and **JSTL**. Click **OK** and the configuration changes to <custom>.

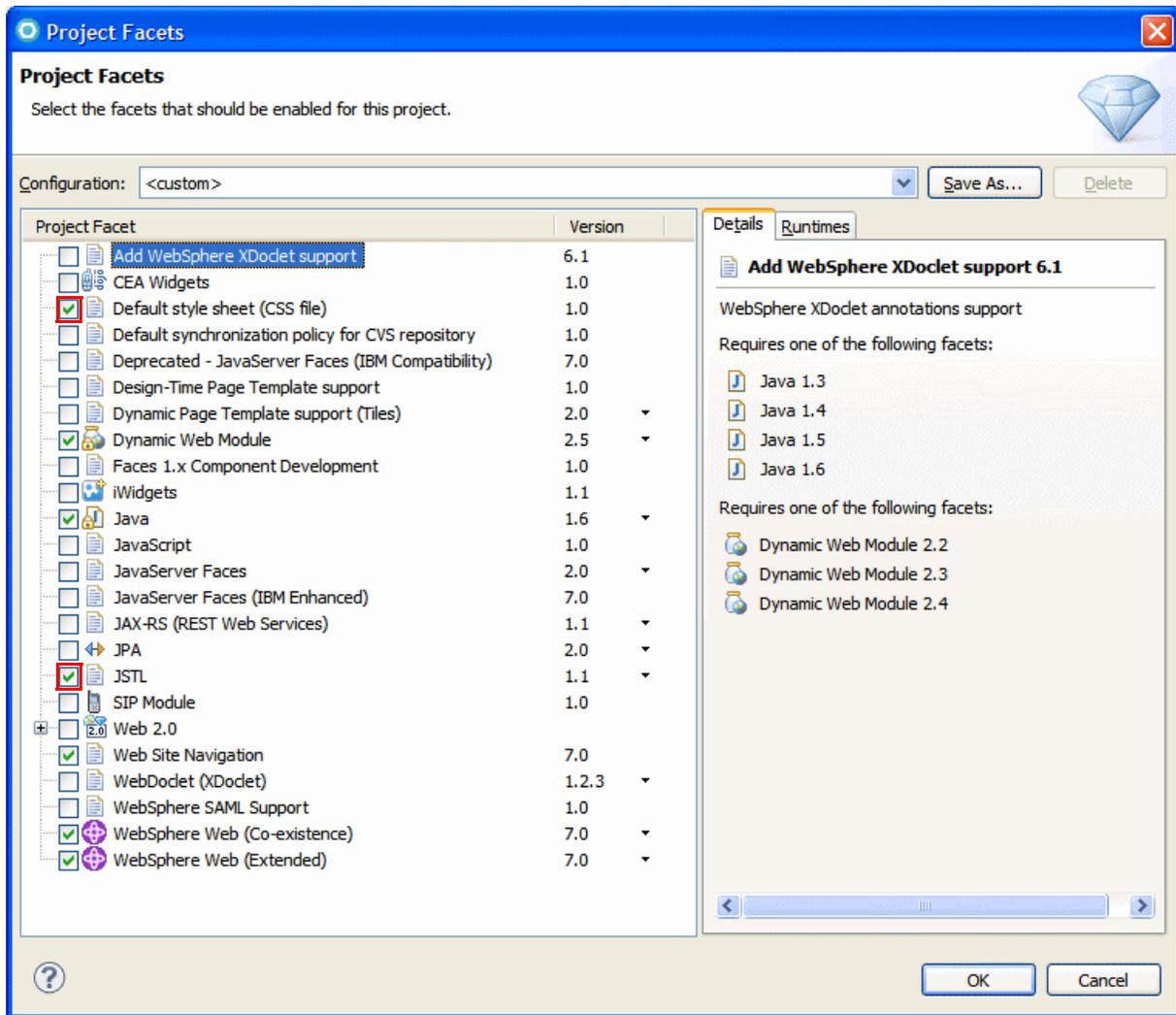


Figure 12 Dynamic Web Project facets

Tips:

- ▶ For the options that have a down arrow, you can alter the underlying version of the feature that is selected. By default, the latest version that is available is selected.
- ▶ From this window, to save the configuration for future projects, you can click **Save** and enter a configuration name and a description.

5. Click **Next**. In the Java window for the New Dynamic Web Project (Figure 13 on page 24), accept the value `src`.

This action specifies the directory where any Java source code used by the web application is stored. The default value of `src` is sufficient for most cases.

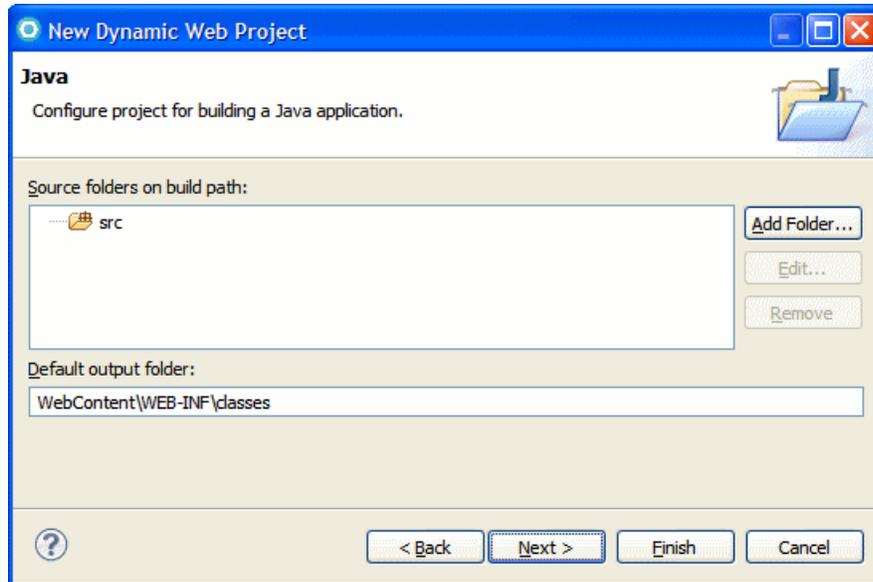


Figure 13 New Dynamic Web Project: Java window

6. In the Web Module window (Figure 14 on page 25), accept the following default options:

- a. For the Context Root, accept the value `RAD8BankBas i cWeb`.

The *context root* defines the base of the URL for the web application. The context root is the root part of the URI under which all the application resources are going to be placed, and by which they are referenced later. It is also the top-level directory for your web application when it is deployed to an application server.

- b. For the Content Directory, accept the value `WebContent`.

This value specifies the directory where the files intended for the WAR file are created. All the contents of the `WEB-INF` directory, HTML, JSP, images, and any other files that are deployed with the application are contained under this directory. Usually, the folder `WebContent` is sufficient.

- c. Select **Generate web.xml deployment descriptor** to create the `web.xml` file and IBM extensions.
- d. Click **Finish**.

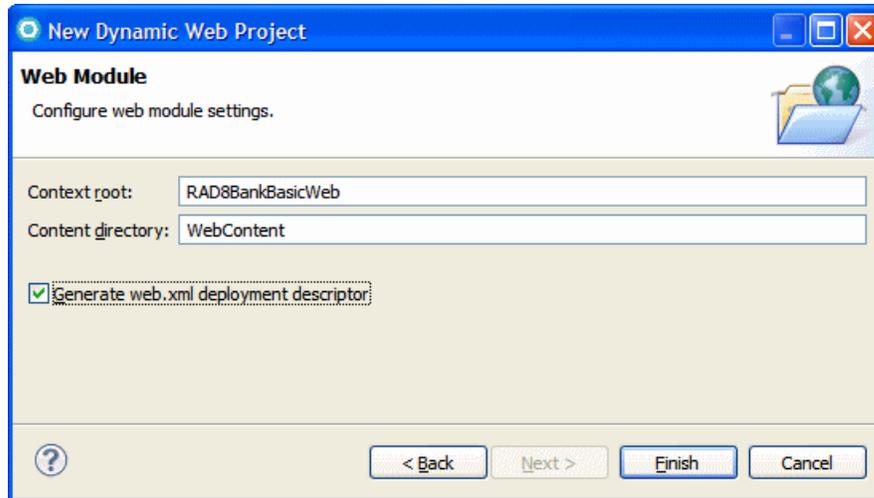


Figure 14 New Dynamic Web Project: Web Module window

The Technology Quickstart window opens. You can browse the Help topics for the features. When you finish, close the Technology Quickstart.

Figure 15 on page 26 shows the web project directory structure for the newly created RAD8BankBasicWeb project and its associated RAD8BankBasicEAR project (enterprise application).

The following folders are shown under the web project:

- ▶ Deployment Descriptor

This folder shows an abstracted view of the contents of the web.xml file of the project. It includes subfolders for the major components that make up a web project configuration, including servlets and servlet mappings, filters and filter mappings, listeners, security roles, and references.

- ▶ Web Site Navigation

Clicking this folder starts the tool for editing the page navigation structure.

- ▶ Java Resources: src

This folder contains the Java source code for regular classes, JavaBeans, and servlets. When resources are added to a web project, they are automatically compiled, and the generated class files are added to the WebContent\WEB-INF\classes folder.

- ▶ WebContent

This folder holds the contents of the WAR file that is deployed to the server. It contains all the web resources, including compiled Java classes and servlets, HTML files, JSP, and graphics needed for the application.

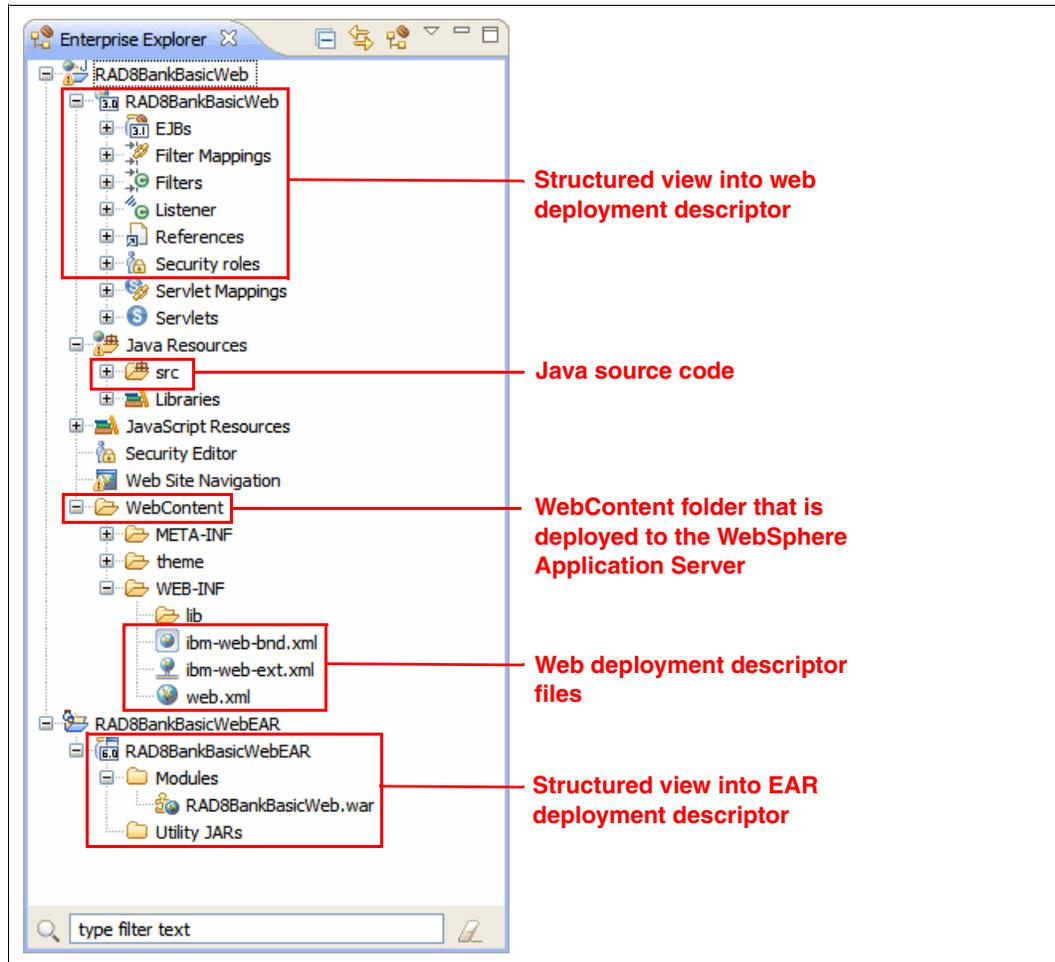


Figure 15 Web project directory structure

Important: Files that are not under WebContent are not deployed when the web project is published. Typically, these files include Java source and SQL files. Make sure that you place everything that is to be published under the WebContent folder.

Importing the Java RedBank model

In this section, we explain how to import the project files. If you already have the final project in the workspace, you can skip this import process.

To import the RedBank model, follow these steps:

1. Locate the file `\jsp\RAD80Java.zip`.
2. From the Import menu option (**File** → **Import**), select **General** → **Existing Projects into Workspace** and click **Next**.
3. Choose the **Select archive file** option that is selected and click **Browse**. Browse to `RAD80Java.zip` and click **OK**.
4. Click **Finish**. The Java project is imported into the workspace.

To verify that the model is running, you can run the main method of the `itso.rad80.bank.client.BankClient` class, which invokes the bank facade classes directly, makes simple transactions directly, and prints the results to the console.

We add a reference to the RAD80Java project in the RAD8BankBasicWeb web application in “Adding the RAD80Java JAR to the web project” on page 39.

Defining the empty web pages

In this section, we show how to define the skeleton pages for both the static and dynamic web pages that make up the RedBank application. We use the New Web Page wizard. When we finish, we have a working application that demonstrates the page navigation of the application.

We create an HTML or JSP page for each of the rows shown in Table 1.

Table 1 Web pages of the RedBank application

HTML or JSP file	HTML version
welcome.html	5
index.html	4.01
rates.html	5
insurance.html	5
redbank.html	5
listAccounts.jsp	5
accountDetails.jsp	5
listTransactions.jsp	5
showException.jsp	5

Importing web resources for the RedBank application

Before we create the web pages, we must import resources to provide the correct design for web pages used in our example, such as images and CSS files. To import the resources, follow these steps:

1. Expand **RAD8BankBasicWeb** → **WebContent**, and from the context menu, select **Import**.
2. Select **General** → **File System** and click **Next**.
3. In the From directory, type `c:\4880code\webapp`, select the **images** and **theme** folders and click **Finish**.

The `itso_logo.gif` and `c.gif` images and the `gray.css` file are imported.

Defining the empty HTML pages

To create the `welcome.html` web page, complete the following steps:

1. From the Menu, select **New** → **Web Page**.
2. In the New Web Page window, enter the following values:
 - a. For the File name, type `welcome.html`.
 - b. For the Folder, type `/RAD8BankBasicWeb/WebContent`.
 - c. For the Template, expand **Basic Templates** and select **HTML/XHTML**.
 - d. Configure the document markup options by clicking **Options**. Ensure that the Markup Language is set to **HTML**. Ensure that the Document Type is set to **HTML 5**.
 - e. Select **Style Sheets**, enter `/theme/gray.css` as the style sheet, and remove the `Master.css`.
3. Click **Close** and then click **Finish** to create the HTML page.
4. If the Split view does not happen automatically, open the new file in the Page Designer view and move to the **Split** view.
5. Locate the title tag and change the value stored inside it to the following value:

```
<title>ITS0 Home</title>
```

The title that is shown in the top half reflects the new title.
6. Locate the body tag and enter text to identify the page:

```
<body>Welcome to Redbank</body>
```
7. Save the web page.

Complete the previous steps to create HTML pages for the other HTML pages (`rates.html`, `insurance.html`, and `redbank.html`).

Defining the empty JSP pages

To create the `listAccounts.jsp` web page, complete the following steps:

1. From the top Menu, select **New** → **Web Page**.
2. In the New Web Page window, complete the following actions:
 - a. For the File name, type `listAccounts.jsp`.
 - b. For the Template, expand **Basic Templates** and select **JSP**.
 - c. Click **Options**. Verify the `gray.css` cascade style sheet and that the document type is **HTML 5**.
 - d. Click **Close**.
3. Click **Finish** to create the page. Close the editor that opens.

Complete the previous steps to create HTML pages for the other JSP pages (`accountDetails.jsp`, `listTransactions.jsp`, and `showException.jsp`).

Important: The spelling and capitalization of the JSP file names must be typed exactly as shown.

Creating frameset pages

The RedBank user interface (view) is made up of a combination of static HTML pages and dynamic JSP. In this section, we explain how to create an HTML frameset page (`index.html`) that defines the layout of the web pages created in “Defining the empty web pages” on page 27.

index.html: By default, a web server looks for the `index.html` (or `index.htm`) page when a web project is run. Although you can change this behavior, use `index.html` as the top-level page name.

Frameset pages provide an efficient method for creating a common layout for the user interface of the web application. A frameset page has the same structure as a table, where the rows are defined in a tag element called `<frameset>` and the columns are individually defined in a tag element called `<frame>`. An alternative approach is to use Page Templates for which Rational Application Developer also has tooling support.

In our example, we have only three areas and no column separations:

- ▶ Header area: With the company logo and the navigation bar to other pages
- ▶ Workspace area: Where the rest of the operational pages are displayed
- ▶ Footer area: With the option to return to the main menu

Creating an HTML frameset page

To create a HTML frameset page (`index.html`) to use for the RedBank layout, complete the following steps:

1. Right-click **WebContent** and select **New** → **Web Page**.
2. In the New Web Page window (Figure 16 on page 30), enter the following values:
 - a. For the File name, type `index.html`.
 - b. For the Template, select **Basic Templates** → **HTML/XHTML**.
 - c. Click **Options**. In the New Web Page Options window, for the Markup Language, select **XHTML Frameset**. For the **Document Type**, select **HTML 4.01 Frameset**. Click **Close**.

Important: In this example, the `index.html` page is HTML 4.01, and all other pages are HTML5. This difference demonstrates the support for both standards and also because the `index.html` page uses frameset tags. Frameset tags are not recommended for HTML5. There is no support for building these frameset tags in Rational Application Developer.

- d. Click **Finish** to create the `index.html` frameset.

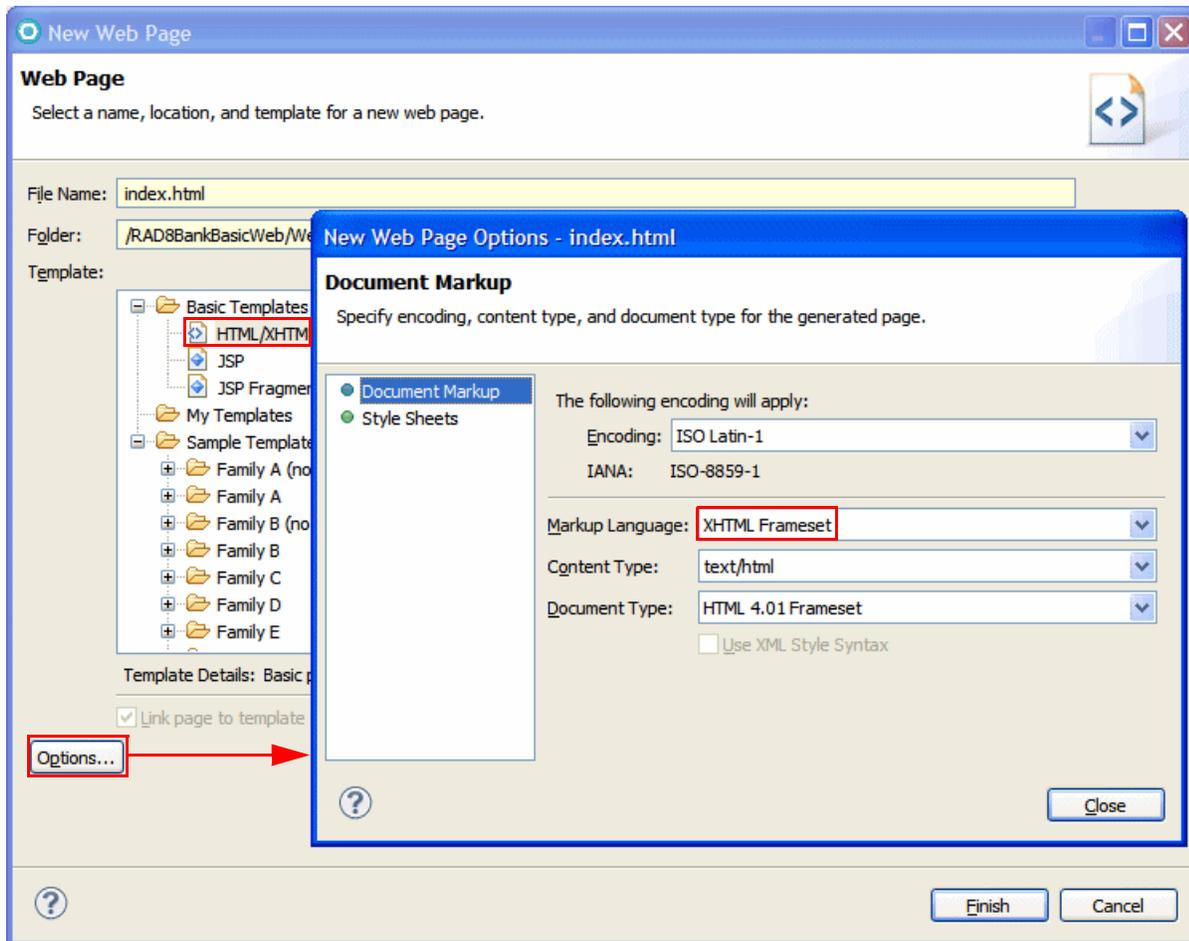


Figure 16 Creating a frameset page

Creating an HTML header for all web pages

Create the static HTML web page for the header area that shows the logo and heading information:

1. Right-click **WebContent** and select **New** → **Web Page**.
2. In the New Web Page window, enter the following values:
 - a. For the File name, type `header.html`.
 - b. For the Template, select **Basic Templates** → **HTML/XHTML**.
 - c. Click **Options**. In the New Web Page Options window, for the Markup Language, select **HTML**. For the Document Type, select **HTML 5** and click **Close**.
 - d. Click **Finish** to create the static `header.html`.
3. Import the code for the logo, title, and action bar into the `header.html` file:
 - a. Locate the `c:\4880code\webapp\html\SnippetForHeaderHTML.txt` file and open it in a simple text editor (for example, Notepad).
 - b. Open the `header.html` file in the Page Designer and select the **Source** tab.
 - c. Paste the code from `SnippetForHeaderHTML.txt` between the `<body>` and `</body>` tags.
 - d. Save the `header.html` file. Select the **Preview** tab to verify that the page has the ITSO RedBank text and logo as desired.

Warnings: The sample provided uses tags that are invalid for HTML5, and Rational Application Developer generates warnings that indicate these attributes and tags are obsolete. To be fully HTML5-compliant, refactor this information. However, these warnings do not affect the operation of the RedBank application, because HTML5-compliant web browsers are compatible with an earlier version.

Creating an HTML footer for all web pages

Create the web page that holds the source of the footer area in the same manner as for the header page:

1. Create a web page named `footer.html` under `WebContent`.
2. Copy and paste the code from `SnippetForFooterHTML.txt` between the `<body>` and `</body>` tags.
3. Save the `footer.html` file. Select the **Preview** tab to see the newly created link.

Customizing frameset web page areas

Next we add frame references to the pages that are part of the user interface frame areas. We explain how to customize the following elements of the HTML page template (`index.html`, `header.html`, and `footer.html`).

Defining the areas in the frameset

To create the mentioned areas/frames in the frameset and link the areas with the previously created `header.html` and `footer.html` web pages, complete these steps:

1. In the Enterprise Explorer, expand **RAD8BankBasicWeb** → **WebContent** and open the **index.html** web page.
2. In the Page Designer, select the **Split** tab to work simultaneously with the source code and interface design.
3. To define the frameset areas, add a `rows` attribute to the frameset tag by following these steps:
 - a. In the Outline view, right-click the **frameset** tag and select **Add Attribute** → **New Attribute**.
 - b. In the New Attribute window, for the Name, type `rows`. For the value, type `20%,70%,10%` and click **OK**.
 - c. Verify in the **Source** tag the value `<frameset rows="20%,70%,10%">`.
4. Link the header area with the `header.html` web page by following these steps on `index.html`:
 - a. In the Outline view, expand **html** → **frameset** and select the **frame** node.
 - b. In the Properties view (Figure 17 on page 32), select the following attributes on the frame tab:
 - i. For the URL, type `header.html`. Or, click the **Browse** icon () and select **File**. Then in the File Selection window, select **header.html** and click **OK**.
 - ii. For the Frame name, type `headerArea`.
 - iii. Leave the rest of the values by default, and save the changes.

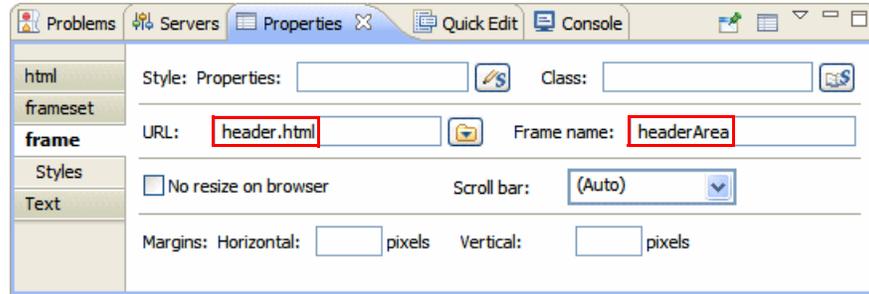


Figure 17 Frameset properties for index.html

- c. On the Split tab of the Page Designer, verify that the `<frame>` code was replaced by `<frame src="header.html" name="headerArea">`.
5. To link the workspace area, create a frame and link it to the `welcome.html` web page by following these steps:
 - a. In the Outline view, right-click the **frame** tag and select **Add After** → **frame**.
 - b. In the Properties view, set the URL to **welcome.html** and the Frame name to **workspaceArea**.
6. To link the footer area, create a frame and link it to the **footer.html** web page and frame name **footerArea**.
7. From the **Design** tab, click the **Show frames** icon () to see the frames in position (Figure 18).

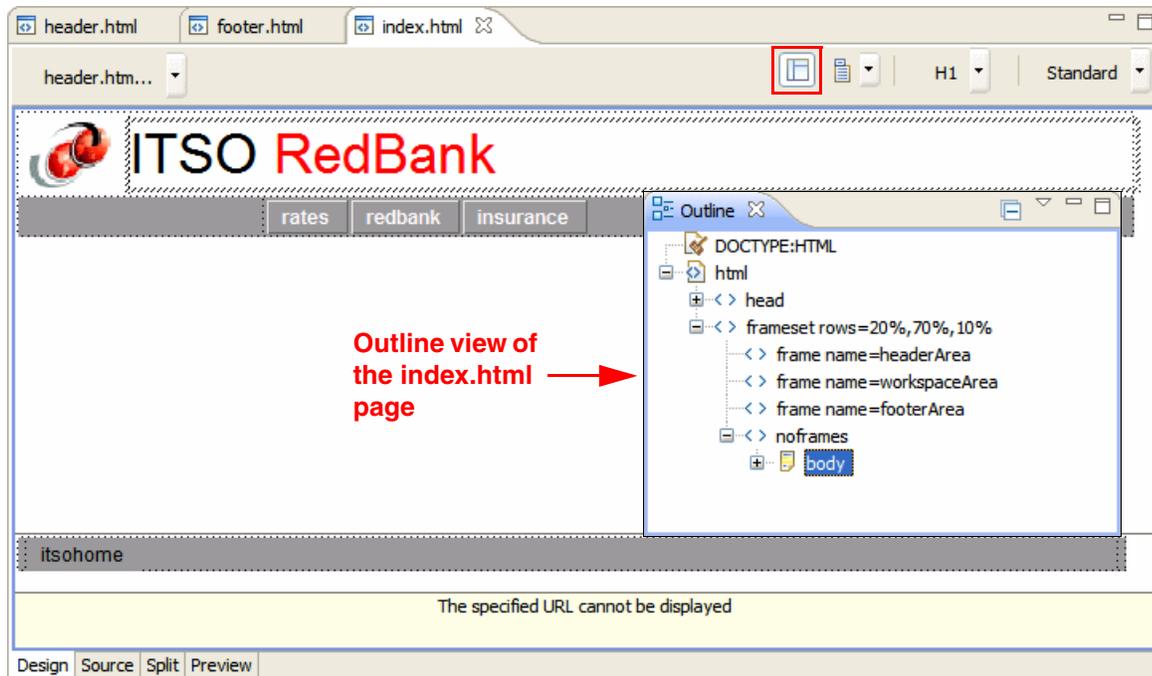


Figure 18 Frame design

Customizing a style sheet

You can create style sheets when a web project is created (by selecting the **Default style sheet (CSS File)** option in the Project Facets window), when a page template is created from a sample, or at any time by starting the CSS File creation wizard.

In the RedBank example, a style sheet named `gray.css` was imported as part of the process of “Importing web resources for the RedBank application” on page 27. Both the HTML and the JSP web pages that we created reference the `gray.css` style sheet to have a common appearance of fonts and colors.

In the following example, we customize the colors that are used on the navigation bar links when you hover over a link. By default, the link text in the navigation bar is orange (`#cc6600`) when hovering. We customize this color to red (`#ff0000`).

To customize the `gray.css` style sheet, complete the following steps:

1. Open the **theme/gray.css** file in the CSS Designer (Figure 19).

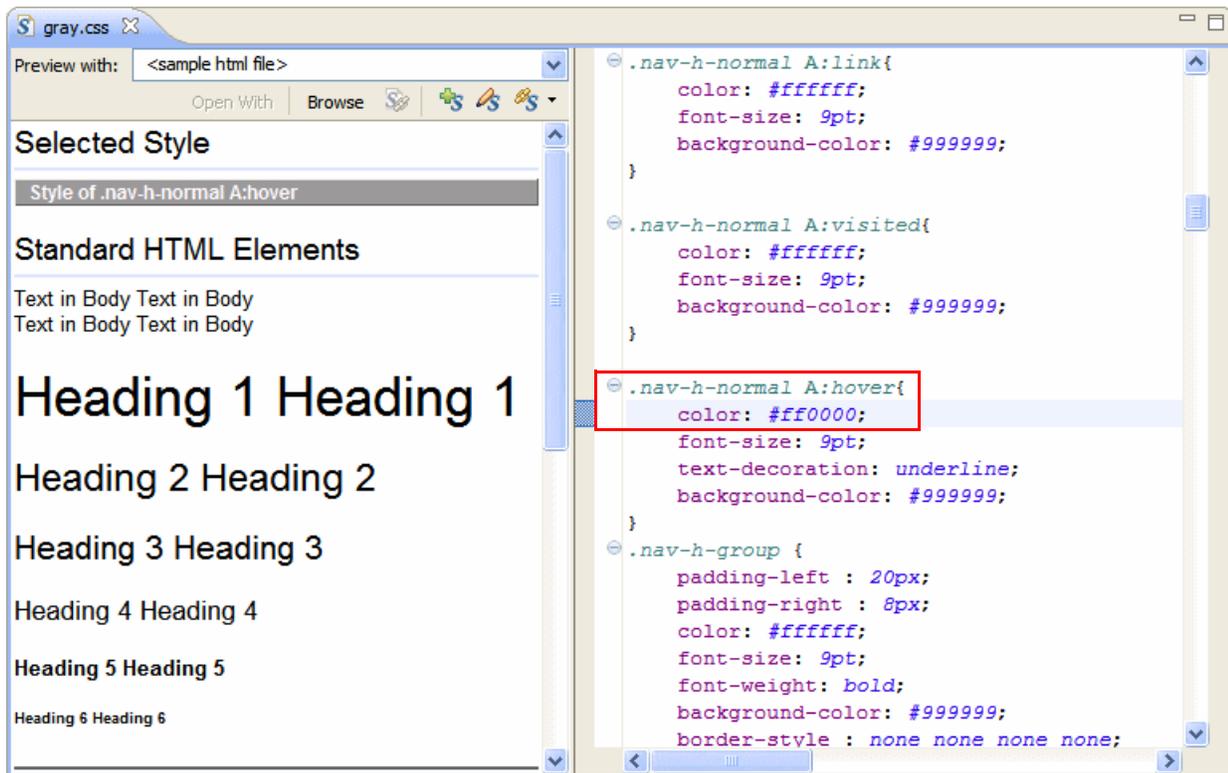


Figure 19 CSS Designer: `gray.css`

By selecting the text style `.nav-h-normal A:hover` in the right pane (scroll down to locate the style, or find the style in the Styles view at the lower left), the text in the left pane is displayed and highlighted. Selecting the text style makes it easy to change the settings and see the change immediately.

2. Change the Hex HTML color code for `.nav-h-normal A:hover` from `color: #cc6600`; (orange) to `color: #ff0000`; (red).
3. Customize the footer highlighted link text. Locate the `.nav-f-normal A:hover` style, and change the color from `#ff6600` (orange) to `#ff0000` (red).
4. Save the file.

Now when you hover over the links in the header and footer, the color changes to red, which can be demonstrated on the Preview tab of `index.html`. Any number of changes can be applied to the style sheets to change the design of the application.

Verifying the site navigation and page templates

At this stage, although the pages have no content, verify that the page templates look as expected and that the navigation links in the header and footer navigation bars work as required:

1. Add text to identify each of the web pages created in the Web Site Navigation:
 - a. Open the **welcome.html** page in the Page Designer and go to the **Source** tab.
 - b. Type `Welcome Page` between the tags `<body></body>`. The final code is displayed:
`<body>Welcome Page</body>`
 - c. Repeat these steps to indicate the names in the body page for `rates.html`, `redbank.html`, and `insurance.html`.
 - d. Start the WebSphere Application Server v8.0 Beta server in the Servers view if it is not running. On the Servers view, from the context menu, select **Start** and wait until the status indicator says Started.
2. In the Enterprise Explorer view, right-click **index.html** in **RAD8BankBasicWeb** and select **Run As** → **Run on Server**.
3. In the Run Server window, select **Choose an existing server** and choose **WebSphere Application Server v8.0 Beta**. Click **Finish**.

After the application is published to the server, the browser pane shows the index page. You can click the tabs for rates, redbank, and insurance to move between these pages (Figure 20).

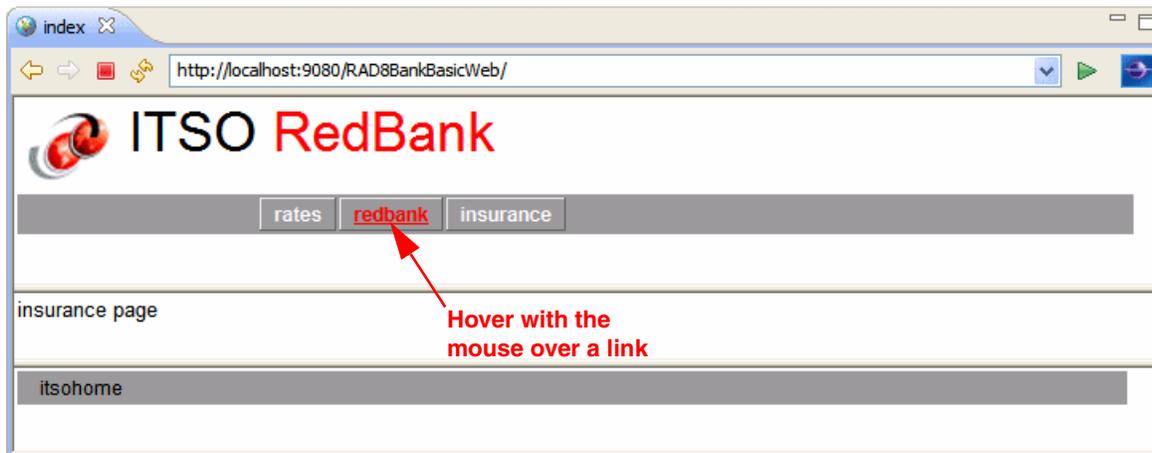


Figure 20 ITSO RedBank website

4. To remove the project from the test server, in the Servers view, right-click **WebSphere Application Server v8.0 Beta**, select **Add Remove Projects**, and remove **RAD8BankBasicEAR**.

Alternatively, expand **WebSphere Application Server v8.0 Beta**, right-click the **RAD8BankBasicEAR** project, and select **Remove**.

Developing the static web resources

In this section, we create the content for the four static pages of our sample with the objective of highlighting several of the features of the Page Designer. The Page Designer facilitates the building of HTML pages by allowing the user to add HTML elements from the Pallet view by using the drag-and-drop method. HTML fragments can also be imported directly into the source tab, as we also demonstrate.

This section covers the following topics:

- ▶ Creating the welcome.html page content (text and links)
- ▶ Creating the rates.html page content (tables)
- ▶ Importing the insurance.html page contents
- ▶ Importing the redbank.html page contents

Creating the welcome.html page content (text and links)

The RedBank home page is `index.html`. The links to the child pages are included as part of the header and footer of our page template. In the following example, we add static text to the page and add a link to the page to the Redbooks website:

1. Open the **welcome.html** file in Page Designer.
2. Select the **Design** tab.
3. Insert the welcome message text:
 - a. Delete the Welcome Page text.
 - b. Insert two line breaks:
 - i. In the Context Area, right-click and select **Insert** → **Line Break**. Leave Type as **Normal** (default).
 - ii. Repeat these steps for the second line break.
 - c. In the Context Area, right-click and select **Insert** → **Paragraph** → **Heading 1**.
 - d. Enter the text Welcome to the ITSO RedBank! at the current cursor position, which places the text between the `<h1>` tags.
4. Insert a link to the Redbooks website:
 - a. Add an empty line after the heading.
 - b. From the menu bar, select **Insert** → **Paragraph** → **Normal**.
 - c. In the new area, enter the text For more information on the ITSO and IBM Redbooks, please visit our Internet site.
 - d. Highlight the text **Internet site**, right-click, and select **Insert** → **Link**.
 - e. In the Insert Link window, select **HTTP**. In the URL field, type `http://www.ibm.com/redbooks` and click **OK**.
5. Customize the text font face, size, and color. In the Properties view, perform these steps:
 - a. Select **Red** in Redbooks from the text created in the previous step (use the keyboard Shift and arrow keys).
 - b. On the Text tab of the Properties view, select the color **Red** to make this partial word stand out. The source changes to `...IBM Redbooks, ...`
6. Save the page.

7. Select the **Preview** tab to view the page (Figure 21). Verify that the page looks correct in both Firefox and Internet Explorer.

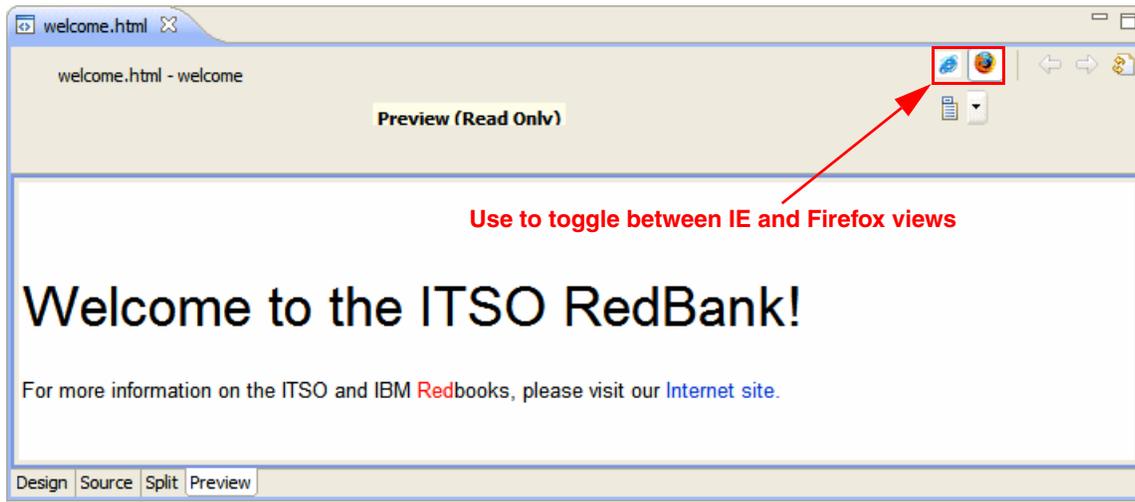


Figure 21 Preview of the welcome.html page

Creating the rates.html page content (tables)

In this example, you add a static table that contains interest rates by using the Page Designer:

1. Open the **rates.html** file in Page Designer and select the **Design** tab.
2. Delete the **Rates Page** text.
3. In the Palette, expand **HTML Tags**.
4. Select and drag a **Table** from the Palette to the content area.
5. In the Insert Table window, for the Rows, Columns, and Padding inside cells fields, enter 5. Then click **OK**.
6. Resize the table as desired.
7. Enter the descriptions and rates (as seen in Figure 22 on page 37) in the table.
8. Select each heading text, and in the Properties view, click the **Bold** icon ().

Table option: You can add additional table rows and columns or delete rows and columns by using the Table menu option.

9. Save the page.

10. Select the **Preview** tab to view the page (Figure 22).



Figure 22 Preview of the rates.html page

Importing the insurance.html page contents

Next, import the body of the insurance.html file:

1. Locate the c:\4880code\webapp\html\SnippetForInsuranceHTML.txt file and open it in a simple text editor (for example, Notepad).
2. Open the **insurance.html** file in Page Designer and select the **Source** tab.
3. Select the text between the tags: <body></body>.
4. Insert the text from the SnippetForInsuranceHTML.txt file.
5. Save the file and switch to the **Preview** tab (Figure 23).



Figure 23 Preview of the insurance.html page

Importing the redbank.html page contents

Repeat the import of the body of the redbank.html file:

1. Locate the c:\4880code\webapp\html\SnippetForBankHTML.txt file.
2. Replace the existing content area text with the text from the snippet.
3. Switch to the **Preview** tab (Figure 24).

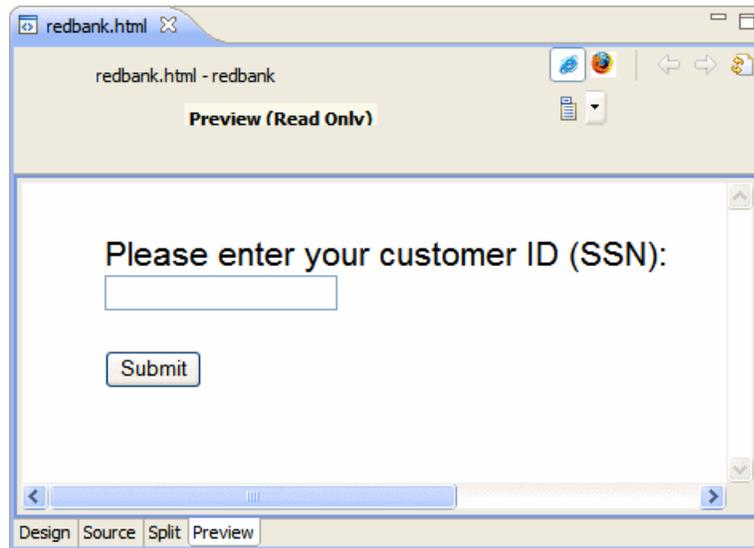


Figure 24 Preview of the redbank.html page

The static HTML pages for the RedBank application are now complete. You can navigate the site by using the header action bar and the footer itsosome link.

Developing the dynamic web resources

In addition to the tools created for building HTML content and designing the flow and navigation in a web application, Rational Application Developer provides several wizards to help you quickly build JavaServer Pages (JSP) and Java servlets. You can use the products of these wizards as is or modify them to fit specific needs.

The wizards support the creation of servlets and JSP. They also compile the Java code and store the class files in the correct folders for publishing to your application servers. As the wizards generate project resources, the appropriate annotations are added to the generated classes (Servlet 3.0 and up) or the deployment descriptor file (web.xml) is updated automatically with the appropriate configuration information for the servlets that are created (versions earlier than Servlet 3.0).

In the previous section, we explained how to create each of the static web pages. In this section, we demonstrate the process of creating and working with servlets. The example servlets are first built by using the wizards. Then the code contents are imported from the sample solution. In “Working with JSP” on page 46, the JSP pages, which invoke the logic in these servlets, are created.

Working with servlets

As described in “Introduction to Java EE web applications” on page 2, servlets are flexible and scalable server-side Java components based on the Java Servlet API, as defined in the JEE Servlet Specification. Servlets generate dynamic content by responding to web client

requests. When an HTTP request is received by the application server, the web server determines, based on the request URI, which servlet is responsible for answering that request and forwards the request to that servlet. The servlet then performs its logic and builds the response HTML that is returned to the web client, or forwards the control to a JSP page.

Rational Application Developer supports Version 3.0 of the JEE servlet specification which means that servlets are defined by the `WebServlet` annotation in the servlet class rather than by an entry in the `web.xml` file.

Rational Application Developer provides the features to make servlets easy to develop and integrate into your web application. From the workbench, you can develop, debug, and deploy servlets. You can also set breakpoints within servlets and step through the code in a debugger. Any changes made are dynamically folded into the running web application, without restarting the server each time.

In the sections that follow, we implement the `ListAccounts`, `UpdateCustomer`, `AccountDetails`, and `Logout` servlets. Then the command or action pattern is applied in Java to implement the `PerformTransaction` servlet.

Adding the RAD80Java JAR to the web project

Before the implementation of the servlet classes can proceed, we must add a reference from the `RAD8BankBasicWeb` project to the `RAD80Java` project, because the servlets call the methods from classes in this project. Adding this reference is achieved in Rational Application Developer by using a feature called *Deployment Assembly*, which is available in the Properties window for each web project.

Creating the reference: In previous versions of Rational Application Developer, this reference was created in the J2EE Module Dependencies dialog window. The new Deployment Assembly dialog window can be used to achieve the same result.

To add `RAD80Java` to the Deployment Assembly for `RAD8BankBasicWeb`, complete the following steps.

To add the JAR file to the class path of the `RAD8BankBasicWeb` project, follow these steps:

1. Highlight the **RAD8BankBasicWeb** project, right-click and select **Properties**.
2. On the Properties for `RAD8BankBasicWeb` dialog window, select the **Deployment Assembly** link and click **Add**.
3. From the New Assembly Directive: Select Directive Type window, select **Project** and click **Next**.
4. From the New Assembly Directive: Projects window, highlight the **RAD80Java** project and click **Finish**.

`RAD80Java.jar` now appears in the Java Build path of the project under Web App Libraries (see Figure 25 on page 40), which indicates that the Java classes are available at compile time. `RAD80Java.jar` also appears in the Web Deployment Assembly list (see Figure 26 on page 40), which indicates that the JAR file is added to a WAR file generated from `RAD8BankBasicWeb`. Click **OK** to close the Properties dialog window.

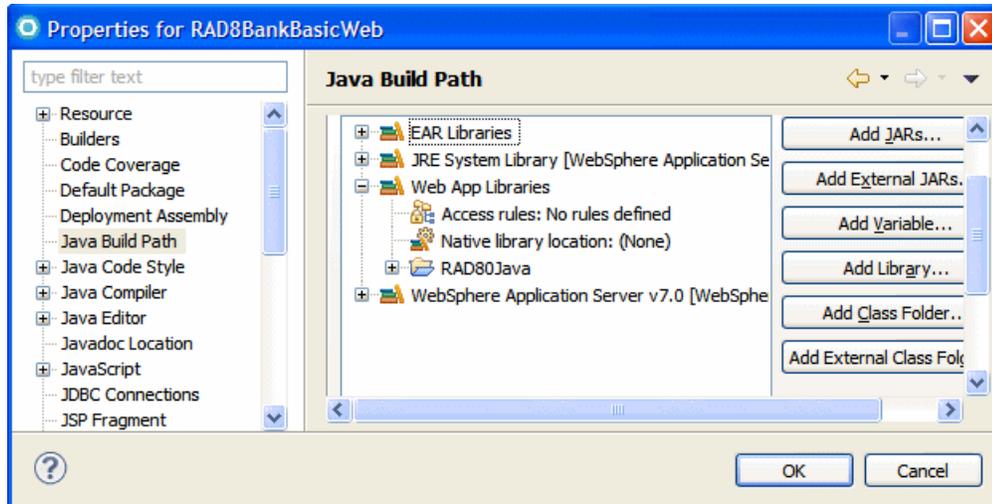


Figure 25 Updated Java Build Path for RAD8BankBasicWeb

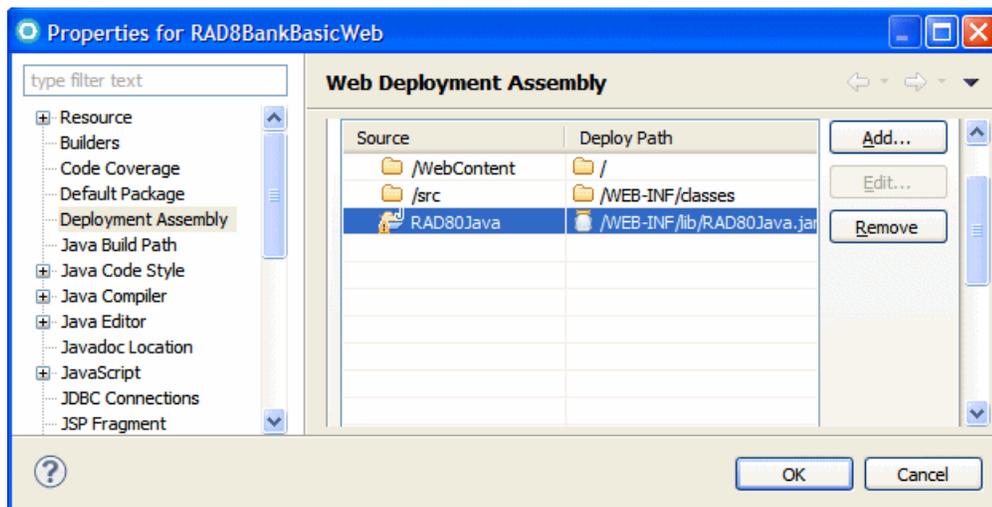


Figure 26 Updated Web Deployment Assembly for RAD8BankBasicWeb

Adding the ListAccounts servlet to the web project

Rational Application Developer provides a servlet wizard to assist you in adding servlets to your web application:

1. Select **File** → **New** → **Other** and then select **Web** → **Servlet**. Click **Next**.

Tip: You can also access the Create Servlet wizard by right-clicking the project and selecting **New** → **Servlet**.

2. In the first window of the Create Servlet wizard (Figure 27 on page 41), for the Java package, type `itso.rad8.webapps.servlet`. For the Class name, type `ListAccounts`. Click **Next**.

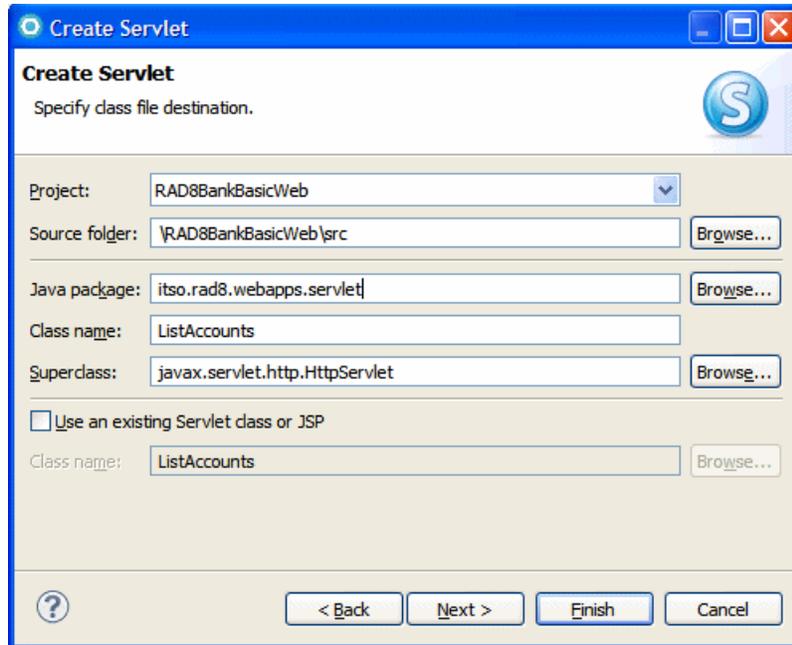


Figure 27 New Servlet wizard: Create Servlet window (part 1 of 3)

The second window (Figure 28) provides space for the name and description of the new servlet.

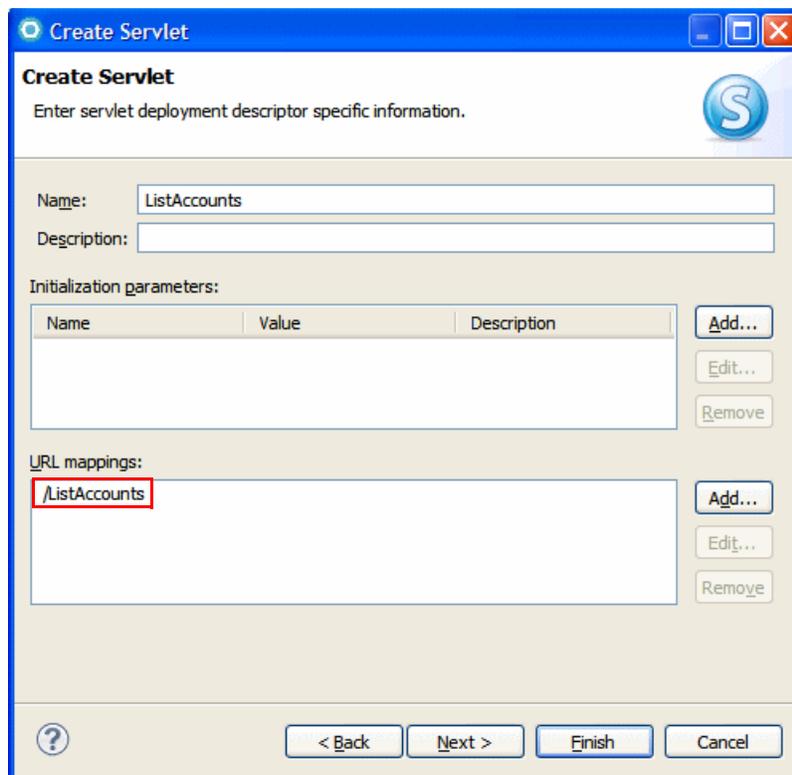


Figure 28 New Servlet wizard: Create Servlet window (part 2 of 3)

On this window, you can also add servlet initialization parameters, which are used to parameterize a servlet. You can change servlet initialization parameters at run time from within the WebSphere Application Server administrative console.

The wizard automatically generates the URL mapping `/ListAccounts` for the new servlet. If other, or additional, URL mappings are required, you can add them here. In our sample, we do not require additional URL mappings or initialization parameters. Click **Next**.

3. In the third window (Figure 29), click **Finish**. In this window, you can have the wizard create method stubs for methods that are available from the `HttpServlet` interface. The `init` method is called at start, and `destroy` is called at shutdown.

The `doPost`, `doGet`, `doPut`, and `doDelete` methods are called when an HTTP request is received for this servlet. All of the `do` methods have two parameters: an `HttpServletRequest` and an `HttpServletResponse`. These methods are responsible for extracting the pertinent details from the request and for populating the response object.

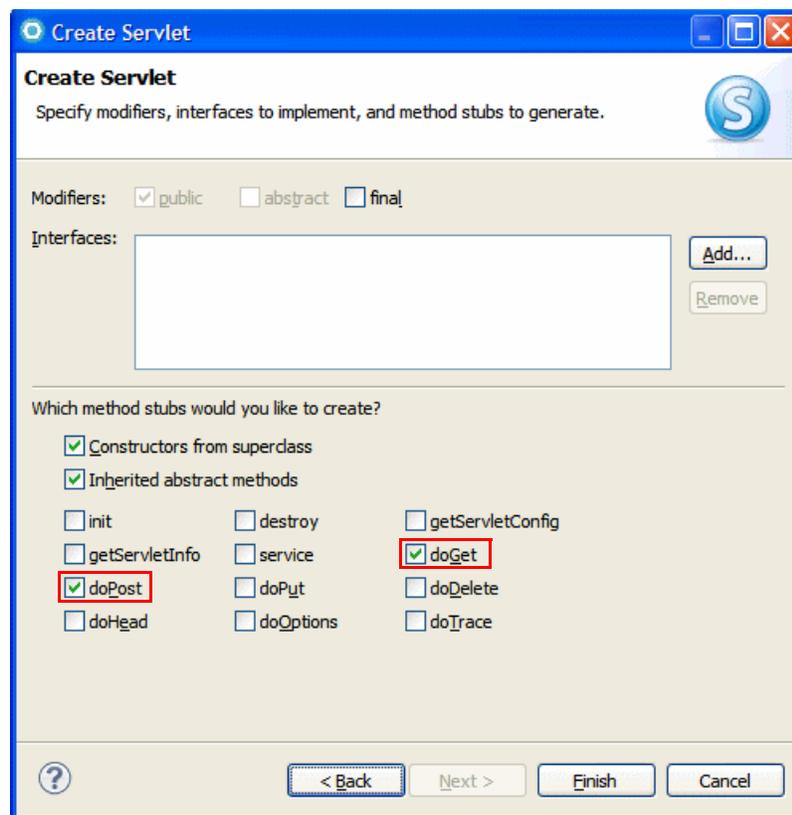


Figure 29 New Servlet wizard: Create Servlet window (part 3 of 3)

For the `ListAccounts` servlet, ensure that only `doGet` and `doPost` are selected. Usually, HTTP gets are used with direct links, when no information has to be sent to the server. HTTP posts are typically used when information in a form has to be sent to the server.

No initialization is required for the new servlet. Therefore, the `init` method is not selected.

The servlet is generated and added to the project. You can find the source code in the Java Resources folder of the project.

4. Expand the deployment descriptor list for the **RAD8BankBasicWeb** (immediately under the project in the Enterprise Explorer), and you see that the `ListAccounts` servlet is shown.

Implementing the ListAccounts servlet

A skeleton servlet now exists, but it does not perform any actions when it is started. Now add code to the servlet to implement the required behavior. The `ListAccounts.java` code of the servlet is already opened.

Follow these steps:

1. Locate the `c:\4880code\webapp\servlet>ListAccounts.java` file.
2. Replace the contents of `ListAccounts.java` with the sample file. The file compiles successfully with no errors.
3. Examine the source code for the `ListAccounts.java` servlet. Note the annotation before the class definition:

```
@WebServlet("/ListAccounts")
```

This annotation declares the class as a servlet for the web project and describes the URL mapping.

This class implements the `doPost` and `doGet` methods, both of which call the `performTask` method.

The `performTask` method does the following tasks:

- a. The method deals with the HTTP request parameters supplied in the request. This servlet expects to either receive a parameter called `customerNumber` or none at all. If the parameter is passed, we store it in the HTTP session for future use. If it is not passed, we look for it in the HTTP session, because it was stored there earlier.
- b. The method implements the control logic. Access to the Bank facade is obtained through the `ITS0Bank.getBank` method, and it is used to retrieve the customer object and the array of accounts for that customer.
- c. The third section adds the customer and account variables to the `HttpRequest` object so that the presentation renderer (`ListAccounts.jsp`) receives the parameters that it requires to perform its job. The control of processing the request is then passed through to `ListAccounts.jsp` by using the `RequestDispatcher.forward` method, which builds the response to be shown on the browser.
- d. The final part of the method is the error handler. If an exception is thrown in the previous code, the catch block ensures that control is passed to the `showException.jsp` page.

Figure 9 on page 19 shows a sequence diagram of the design of this class.

The `ListAccounts` servlet is now complete.

4. Save the changes and close the source editor.

Implementing the UpdateCustomer servlet

The `UpdateCustomer` servlet is used for updating the customer information and is started from the `ListAccounts` JSP through a push button.

The servlet requires that the Social Security number (SSN) of the customer that is to be updated is already placed on the session (as must be done in the `ListAccounts` servlet). It extracts the `title`, `firstName`, and `lastName` parameters from the `HttpRequest` object, calls the `bank.getCustomer(String customerNumber)` method, and uses the simple setters on the `Customer` class to update the details.

Follow the procedures in “Adding the `ListAccounts` servlet to the web project” on page 40 and “Implementing the `ListAccounts` servlet” on page 43 to build the servlet, including the `doGet`

and `doPost` methods. The code to use for this class is in the `c:\4880code\webapp\servlet\UpdateCustomer.java` file.

Implementing the AccountDetails servlet

The `AccountDetails` servlet retrieves the account details and forwards control to the `accountDetails.jsp` page to show these details. The servlet expects the parameter `accountId`, which specifies the account for which data must be shown, in the request. The servlet calls the `bank.getAccount(..)` method, which returns an `Account` object and adds it as a variable to the request. It then uses the `RequestDispatcher` to forward the request onto the `accountDetails.jsp`.

Security and authorization: An actual client implementation performs security and authorization where the current user has the required access rights to the requested account. You can implement security and authorization by using the Security Editor tool, as described in, “Security Editor” on page 13.

Follow the procedures in “Adding the ListAccounts servlet to the web project” on page 40 and “Implementing the ListAccounts servlet” on page 43 to build the servlet. The code to use for this class is in the `c:\4880code\webapp\servlet\AccountDetails.java` file.

Implementing the Logout servlet

The Logout servlet is used for logging the customer off from the RedBank application. The servlet requires no parameters. The only logic performed in the servlet is to remove the SSN from the session, simulating a logoff action. You remove the SSN by calling the `session.removeAttribute` and `session.invalidate` methods. Finally, the servlet uses the `RequestDispatcher` class to forward the browser to the `index.html` page.

Follow the procedures in “Adding the ListAccounts servlet to the web project” on page 40 and “Implementing the ListAccounts servlet” on page 43 to build the servlet. The code to use for this class is in the `c:\4880code\webapp\servlet\Logout.java` file.

Implementing the PerformTransaction command classes

In the `PerformTransaction` servlet, a Command design pattern is used to implement it as a front controller class that forwards control to one of the four command objects: `Deposit`, `Withdraw`, `Transfer`, and `ListTransactions`.

You can find this design in, “Controller layer” on page 18. This implementation is based on the sequence diagram (Figure 10 on page 20).

Next we import the code for the commands package. The source is in the `C:\4880code\webapp\command` folder. Follow these steps:

1. Create the `itso.rad8.webapps.command` package. In the Enterprise Explorer, right-click the **Java Resources: src** folder and select **New** → **Package**.
2. For the package name, type `itso.rad8.webapps.command` and click **Finish**.
3. From the context menu of the new package, click **Import** and select **General** → **File system**. Click **Next**.
4. Click **Browse** and navigate to the `C:\4880code\webapp\command` folder. Click **OK**.
5. Select the following Java files and click **Finish**:
 - `Command.java`
 - `DepositCommand.java`
 - `ListTransactionsCommand.java`

- TransferCommand.java
- WithdrawCommand.java

The command classes perform the operations on the RedBank model classes (from the RAD80Java project) through the Bank facade. They also return the file name for the next page to be displayed after the command runs.

Implementing the PerformTransaction servlet

Now that all the commands for the PerformTransaction framework are realized, you can create the PerformTransaction servlet. The servlet uses the value of the transaction request parameter to determine which command to execute.

You can use the Create Servlet wizard to create a servlet named PerformTransaction. The servlet class must be placed in the `itso.rad8.webapps.servlet` package.

Follow the procedures in “Implementing the ListAccounts servlet” on page 43 to prepare the servlet, including the `doGet` and `doPost` methods. The code to use for this class is in the `c:\4880code\webapp\servlet\PerformTransaction.java` file.

PerformTransaction stores a `HashMap` of the action strings (deposit, withdraw, transfer, and list) to instances of Command classes. Both the `doGet` and `doPost` methods call `performTask`. In the `performTask` method, the `execute` method is called on the appropriate Command class that performs the transaction on the Bank classes. After the `execute` method completes, the `getForwardView` method is called on the Command class, which returns the next page to display, and PerformTransaction uses the `RequestDispatcher` to forward the request to the next page.

After this step, all the servlets required for the sample application are built. The Enterprise Explorer view shows a list of all the servlets in the web project and the URL mappings to these servlets (Figure 30 on page 46).

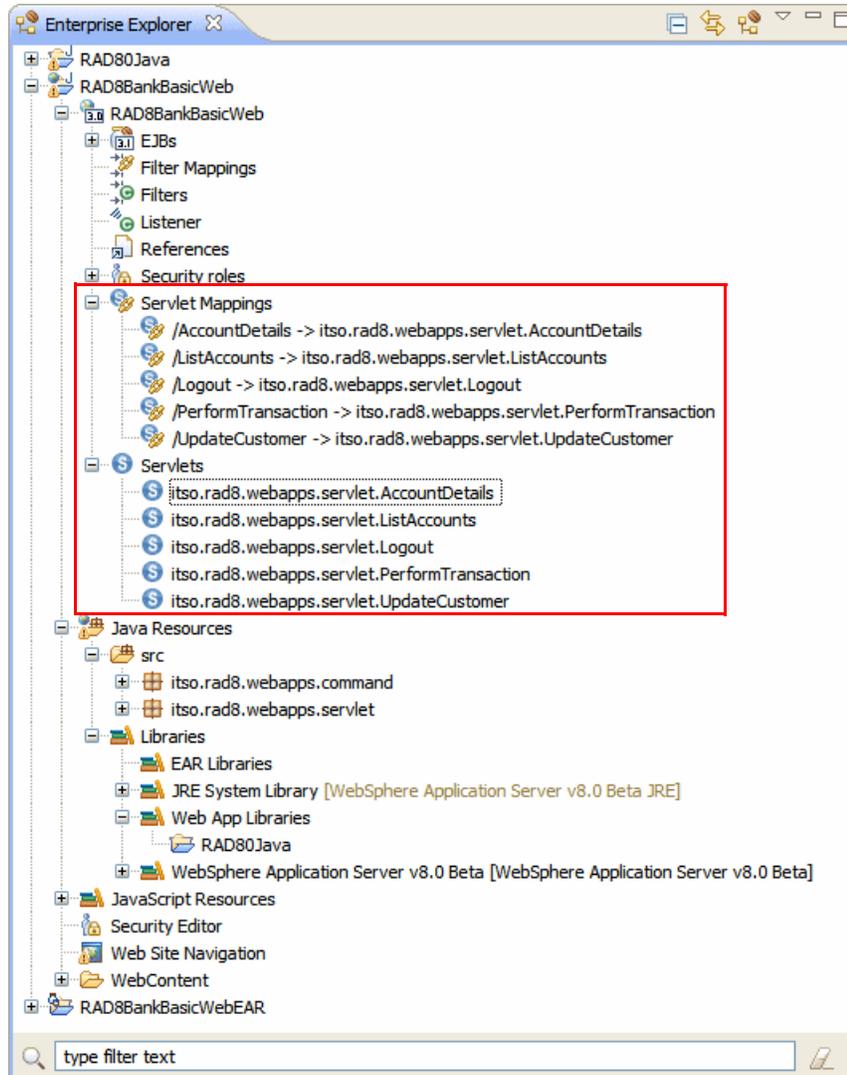


Figure 30 Servlets and Servlets Mappings in Enterprise Explorer

Working with JSP

JSP files are edited in the Page Designer, which is the same editor that is used to edit the HTML page. When working with a JSP page in the Page Designer, the Palette view has a separate drawer for JSP Tags, which includes elements, such as JavaBean references, JavaServer Pages Standard Tag Library (JSTL) tags, and scriptlets that contain Java code.

In this section, we describe the implementation of `listAccounts.jsp` in detail, and the other JSP (`accountDetails.jsp`, `listTransactions.jsp`, and `showException.jsp`) are imported from the solution.

Implementing the List Accounts JSP

Customizing a JSP file by adding static content is done in the Page Designer tool in the same way that an HTML file can be edited. You can also add the standard JSP declarations, scriptlets, expressions, tags, or any other custom tag that is developed or retrieved from the Internet.

In this example, the `listAccounts.jsp` file is built by using page data variables for *customer* and *accounts* (and an array of `Account` classes for that customer). These variables are added to the page by the `ListAccounts` servlet and are accessible to the Java code and tags used in the JSP.

To complete the body of the `listAccounts.jsp` file, perform the following steps:

1. Open the **listAccounts.jsp** in Page Designer and select the **Design** tab.
2. Add the *customer* and *accounts* variables to the page data meta information in the Page Data view (by default, one of the views in the upper-left corner). These variables are added to the request object in the `ListAccounts` servlet, as discussed in “Implementing the `ListAccounts` servlet” on page 43. Page Designer must be aware of these variables. Complete these steps:

- a. In the Page Data view, expand **Scripting Variables**, right-click **requestScope**, and select **New** → **Request Scope Variable**.
- b. In the Add Request Scope Variable window, complete the following tasks:
 - i. For the Variable name, select **customer**.
 - ii. Type `itso.rad80.bank.model.Customer`.
 - iii. Click **OK**.

Tip: You can use the browse button (marked with an ellipsis (...)) to find the class that is using the class browser.

- c. Repeat this procedure to add the following request scope variable:
 - i. For the Variable name, select **accounts**.
 - ii. Type `itso.rad80.bank.model.Account []`.

Important: The square brackets indicate that the variable `accounts` is an array of `accounts`.

3. In the Palette view, select **Form Tags** → **Form** and click anywhere on the JSP page in the content table. A dashed box appears on the JSP page, representing the new form.
4. In the Properties view for the new Form element, enter the following items:
 - a. For the Action, type `UpdateCustomer`.
 - b. For the Method, select **Post**.

Tip: You can use the Outline view to navigate to the form tag quickly.

5. Add a table with the customer information:
 - a. In the Page Data view, expand and select **Scripting Variables** → **requestScope** → **customer** (`itso.rad80.java.model.Customer`).
 - b. Select and drag the customer object to the form that was previously created.
 - c. In the Insert JavaBean window (Figure 31 on page 48), follow these steps:
 - i. Select **Displaying data (read-only)**.
 - ii. Use the arrow up and down buttons to arrange the fields in the order shown and type over the labels.
 - iii. Clear the **accounts** field (we do not display the accounts).

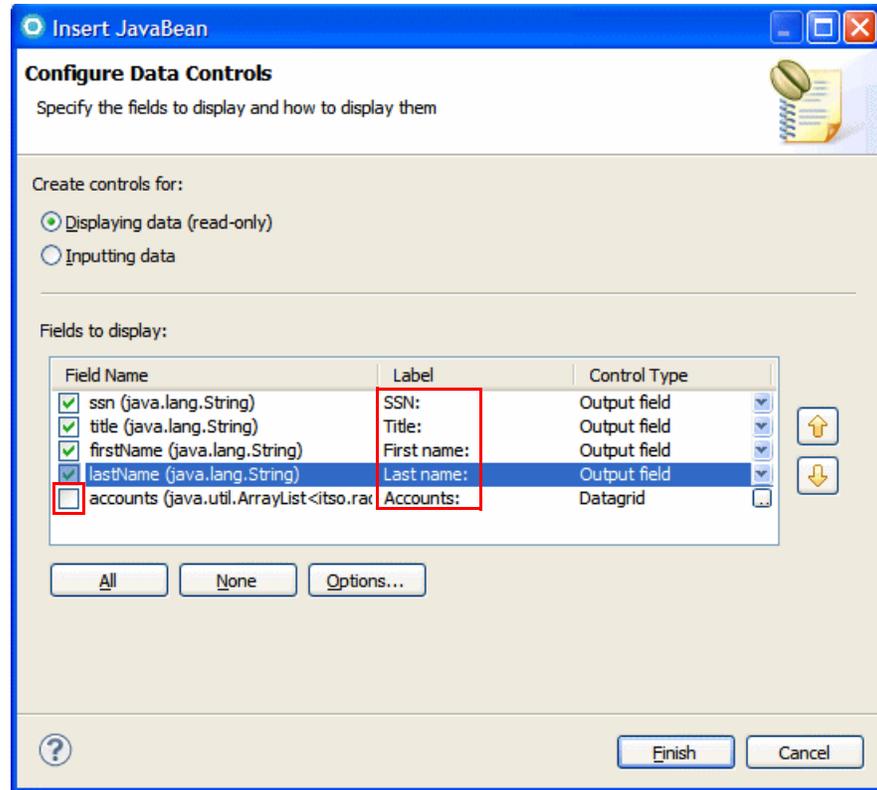


Figure 31 Inserting the customer JavaBean

- d. Click **Finish** to add the Data control for the Customer.

Customer data in table: The newly created table with customer data is changed in a later stage to use input fields for the title, first name, and last name fields. Creating an editable version of this information is not possible with the available wizards.

- e. Right-click the last row of the newly created table (select the **LastName** cell) and select **Table** → **Add Row Below**.
6. In the Palette view, select **Form Tags** → **Submit Button** and click in the right cell of the new row. In the Label field, enter Update and click **OK**. You can leave the Name field empty.
7. In the Palette view, select **HTML Tags** → **Horizontal Rule** and click in the area immediately beneath the form.
8. In the Page Data view, expand **Scripting Variables** → **requestScope** → **accounts** (itso.rad8.java.model.Account []).
9. Drag the accounts object beneath the Horizontal Rule that was created.
10. In the Insert JavaBean: Configure Data Controls wizard (Figure 32 on page 49), follow these steps:
 - a. Clear **transactions**. We do not display the transactions.
 - b. For both fields, in the Control Type column, select **Output link**.
 - c. In the Label field for the accountNumber field, enter AccountNumber.
 - d. Ensure that the order of the fields is accountNumber and balance.

- e. Click **Finish**. The accounts bean is added to the page and is displayed as a list.

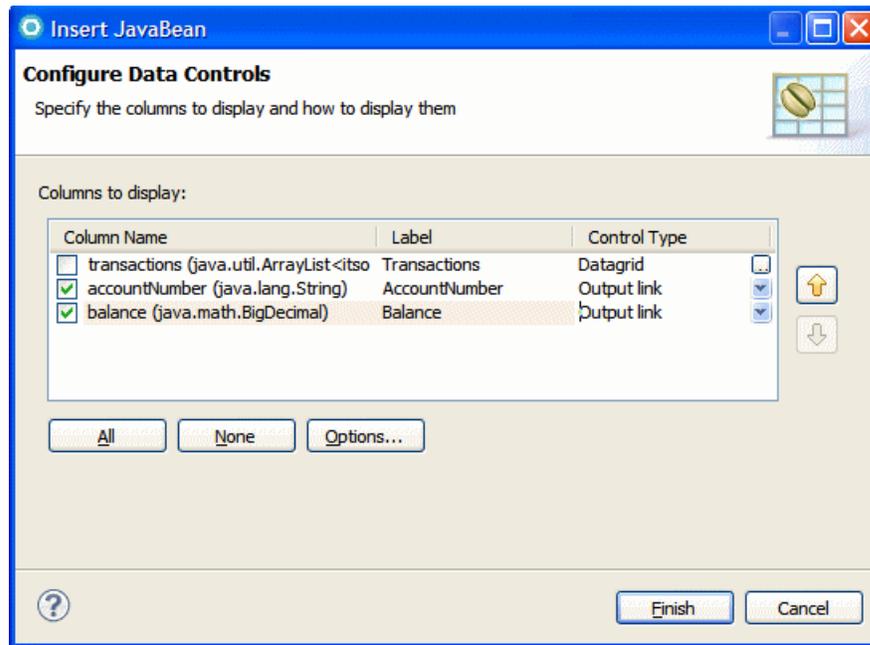


Figure 32 Inserting the accounts JavaBean

- f. The wizard inserts a JSTL `c:forEach` tag and an HTML table with headings, as entered in the Insert JavaBean window. Because we selected **Output link** as the Control Type for each column, corresponding `c:url` tags are inserted. We now edit the URL for these links to make sure that they are identical and to pass the `accountId` variable as a URI parameter:
- g. From the Design view, select the first `<c:url>` tag under the heading AccountNumber, which has the text `#{varAccounts.accountNumber}`. In the Properties view (Figure 33 on page 50), in the Value field, enter `AccountDetails`.
The tag changes to `AccountDetails{}`.

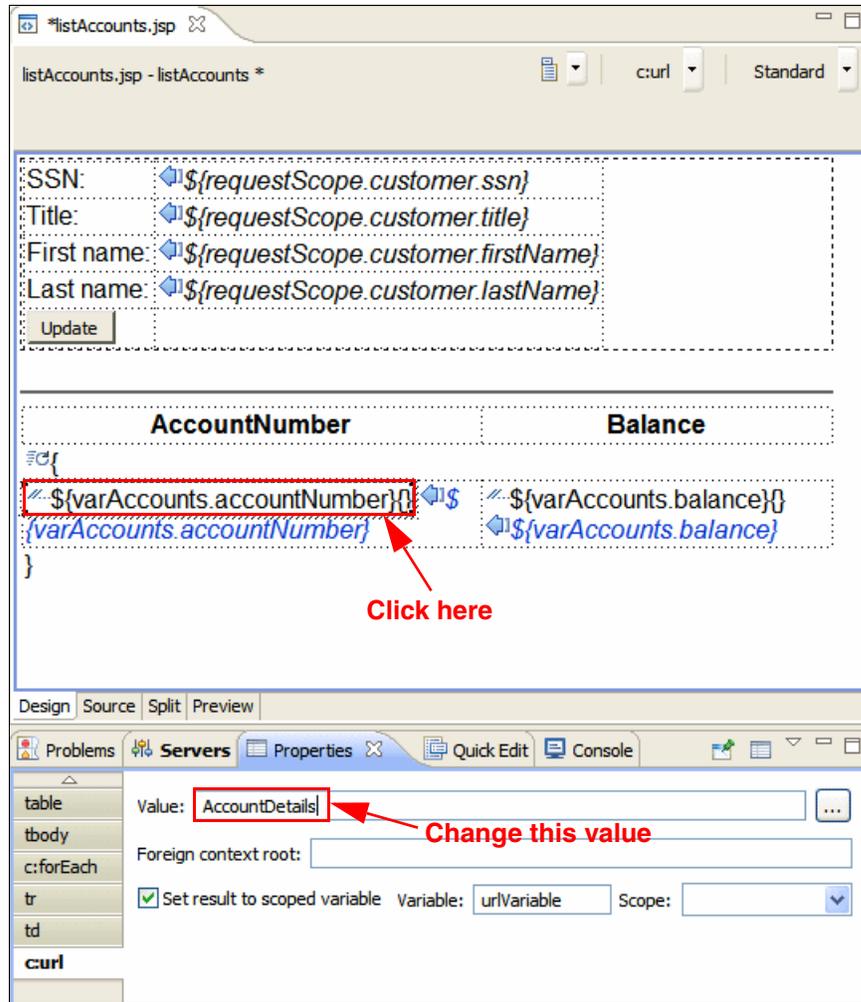


Figure 33 Configuring the AccountDetails URL

- h. Under the heading Balance, which has the text `${varAccounts.balance}`, select the second `<c:url>` tag. In the Properties view, in the Value field, enter `AccountDetails`. This value specifies the target URL for the link, which, in this case, maps to the `AccountDetails` servlet.
- i. Add a parameter to this URL to ensure that the link maps to the correct account. In the Palette view, select **JSP Tags** → **Parameter** (📄) and click the first `<c:url>` in the AccountNumber column. The column has the text `AccountDetails{}` as shown in Figure 34 on page 51.
- j. In the Properties view, on the **c:param** tab, in the Name field, enter `accountId`, and in the Value field, enter `${varAccounts.accountNumber}`. This action adds a parameter to the account URL with a name of `accountId` and the value of the `accountNumber` request variable.

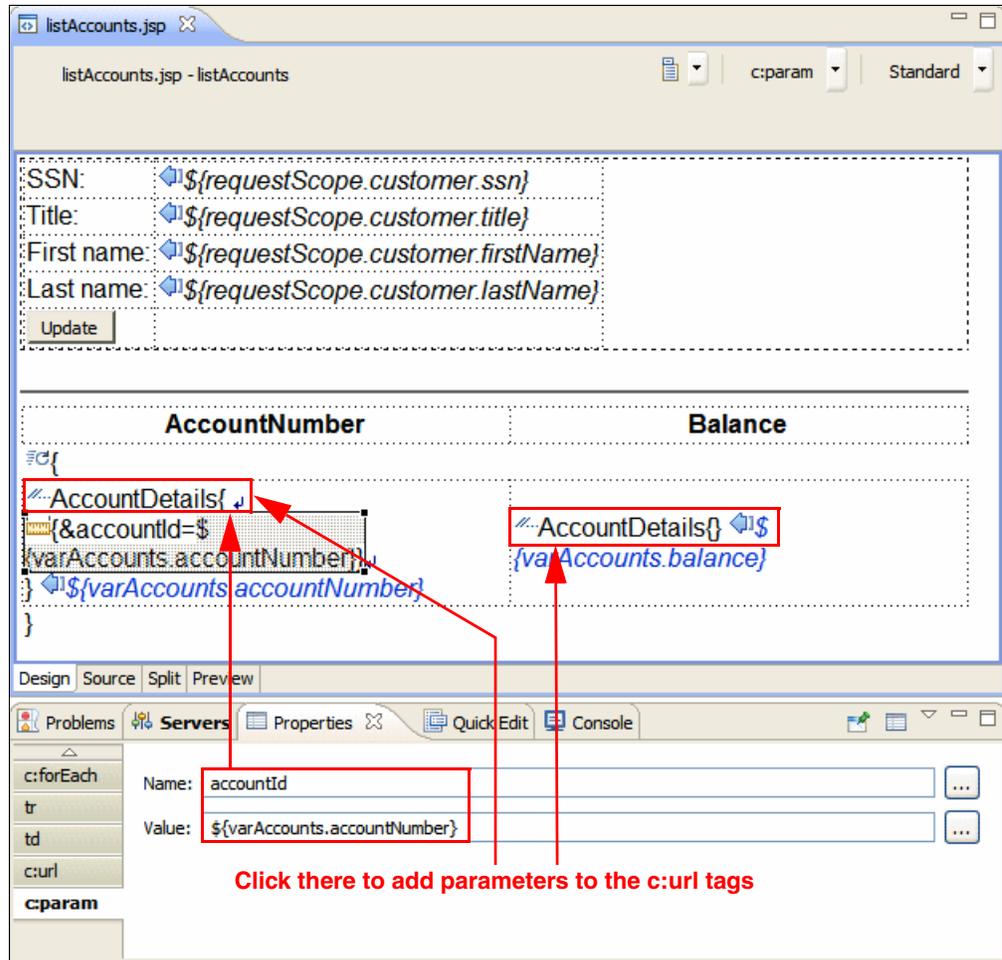


Figure 34 Adding parameters to `c:url` tags

- k. Repeat the two previous steps to add a parameter to the second `<c:url>` tag in the Balance column, showing the text Account Number{}. In the Name field, type `accountId`, and in the Value field, type `${varAccounts.accountNumber}`.
11. Click anywhere in the Balance column and select the **td** tag in the Properties view. In the Horizontal alignment list box, select **Right** to right-align the contents of the Balance cells.
12. Select the **Source** tab and compare the code to display the accounts to the code that is shown in Example 1 on page 52. This JSP code shows the accounts as a list, uses the `<c:forEach>` tag to loop through each account, and uses the `<c:out>` tag to reference the current loop variable. The `<c:url>` tag builds a URL to `AccountDetails` and the `<c:param>` tag adds the `accountId` parameter (with the account number value) to that URL.

Example 1 JSP code with JSTL tags to display accounts (formatted)

```
<c:forEach var="varAccounts" items="${requestScope.accounts}">
  <tr>
    <td>
      <c:url value="AccountDetails" var="urlVariable">
        <c:param name="accountId"
          value="${varAccounts.accountNumber}"></c:param>
      </c:url>
      <a href="<c:out value='${urlVariable}' />">
        <c:out value="${varAccounts.accountNumber}"></c:out>
      </a>
    </td>
    <td align="right">
      <c:url value="AccountDetails" var="urlVariable">
        <c:param name="accountId"
          value="${varAccounts.accountNumber}"></c:param>
      </c:url>
      <a href="<c:out value='${urlVariable}' />">
        <c:out value="${varAccounts.balance}" />
      </a>
    </td>
  </tr>
</c:forEach>
```

13. Select the **Split** tab. From the Palette view, select **HTMLTags** → **Horizontal Rule** and click in the area beneath the account details table.

14. Add a logout form:

- a. In the Palette view, select **Form Tags** → **Form** and click beneath the new horizontal rule. A dashed box is displayed on the JSP page, representing the new form.
- b. In the Properties view for the new form tag, enter the following items:
 - i. For the Action, type Logout.
 - ii. For the Method, select **Post**.
- c. In the Palette view, select **Form Tags** → **Submit Button** and click in the new form. When you click the Logout button, the doPost method is called on the Logout servlet.
- d. In the Insert Submit Button window, in the Label field, enter Logout and click **OK**.

15. Change the title, first name, and last name to entry fields, so that the user can update the customer's details. To convert the Title, First name, and Last name text fields to allow text entry, follow these steps:

- a. Select the `${requestScope.customer.title}` field.
- b. Select the **Source** tab and you can see the following code:

```
<td><c:out value="${requestScope.customer.title}" /></td>
```
- c. Change the code to this code:

```
<td><input type="text" name="title"
  value="<c:out value='${requestScope.customer.title}' />" /></td>
```

d. Repeat these steps for the first name and last name fields:

```
<td><input type="text" name="firstName"
    value="<c:out value='${requestScope.customer.firstName}' />" /></td>
.....
<td><input type="text" name="lastName"
    value="<c:out value='${requestScope.customer.lastName}' />" /></td>
```

The customer fields change from display-only fields to editable fields, so that the details can be changed.

You can change the length of the three input fields in the Properties view. For example, you can specify 6 columns and 3 as the maximum length for the title, and 32 columns for the names.

You can also change the width of the content areas in the source code.

16. Format the account balance, which is a `BigDecimal`. Otherwise, it is displayed with many digits. Complete these steps:

- In the Balance column, click the balance field `${varAccounts.balance}`.
- From the context menu, select **JSP** → **Insert Custom**.
- In the Insert Custom Tag window (Figure 35), click **Add** to add another tag library.
- Locate and select the `http://java.sun.com/jsp/jstl/fmt` URI and click **OK**.
- Select the new tag library and select `formatNumber` as the custom tag. Click **Insert** and click **Close**.

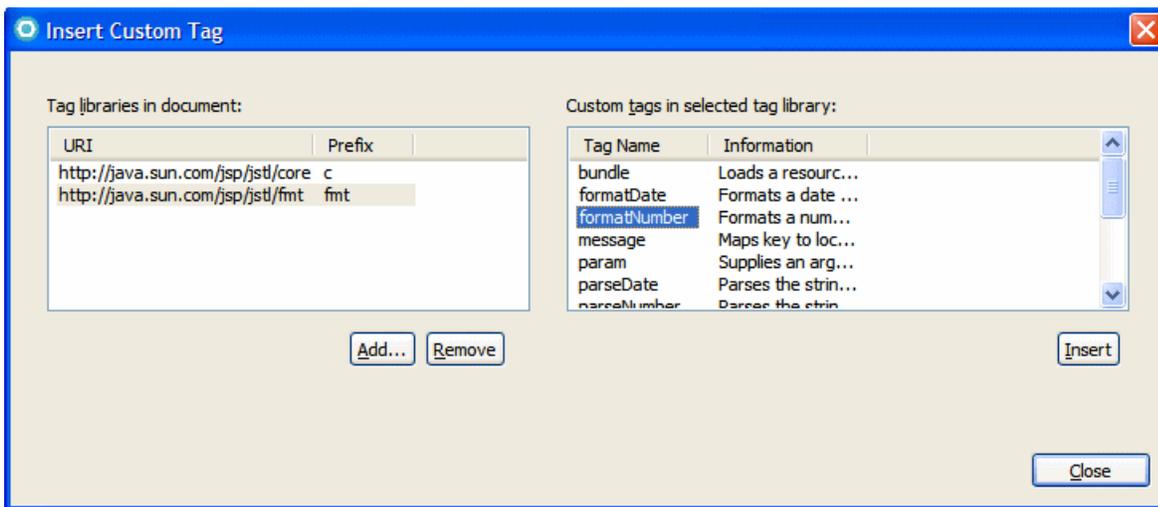


Figure 35 Inserting a custom tag

f. On the Source tab, select the `<fmt:formatNumber>` tag. In the Properties view, set `maxFractionDigits` and `minFractionDigits` to **2**. For the value, type `${varAccounts.balance}`.

g. Remove the `<c:out value=... >` and `</c:out>` tags:

```
<a href="<c:out value='${urlVariable}' />">
  <del>c:out value='${varAccounts.balance}' />
  <fmt:formatNumber maxFractionDigits="2" minFractionDigits="2"
    value="${varAccounts.balance}"></fmt:formatNumber>
  </del>c:out
</a>
```

17. Select any field in the accounts table. In the Properties view, select the **table** tab. Set the width to **100** and select **%**.
18. Select the **Account Number** heading. In the Properties view, set the horizontal alignment to **Left**. Select the **Balance** heading and set the horizontal alignment to **Right**.
19. Save the file.

Figure 36 shows the JSP in the **Design** tab.

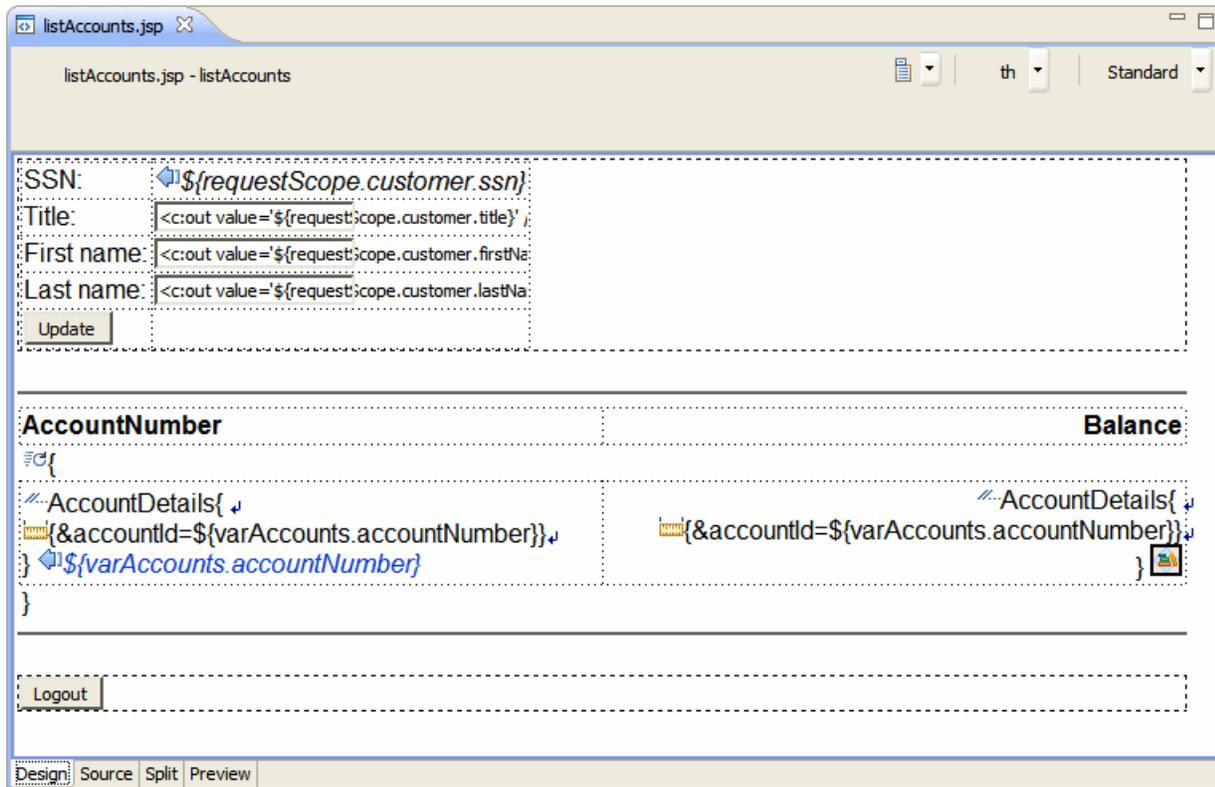


Figure 36 List AccountsJSP finished

JSP source code for the listAccounts.jsp file: The JSP source code for the `listAccounts.jsp` is in the `c:\4880code\webapp\jsp\` directory. You can import the code into the WebContent folder, or copy and paste it directly into Rational Application Developer.

Implementing the other JSP

The other JSPs are already created as part of the model solution. We built them by using a similar process of adding request beans to the JSP and building HTML and JSP elements around them.

To import the other JSP files and to view them in Page Designer, perform the following steps:

1. Select the **WebContent** folder, click **Import**, and select **General** → **File System**. Click **Next**.
2. Click **Browse** and navigate to the `c:\4880code\webapps\jsp` file.
3. Select all the JSP files, except `listAccounts.jsp` (which is completed). Click **Finish**.
4. When prompted whether to override the existing files, click **Yes to All**.

Associated request beans of the imported pages: The imported pages do not have the associated request beans that show in the Page Data view, because they are maintained in the `.jspPersistence` file immediately under the web project directory. You have to specifically add them to the Page Data view.

Add the required variables to the appropriate Page Data view by following the next steps for each JSP file in Table 2:

1. Open each JSP file in the Page Designer by double-clicking the file from the Project Explorer view.
2. In the Page Data view, select **Scripting Variables** → **New** → **Request Scope Variable** and add the variables for each JSP, as specified in Table 2.

Table 2 Request scope variables for each JSP

JSP file	Variable	Type
accountDetails.jsp	account	itso.rad80.bank.model.Account
listTransactions.jsp	account	itso.rad80.bank.model.Account
listTransactions.jsp	transactions	itso.rad80.bank.model.Transaction[]
showException.jsp	message	java.lang.String
showException.jsp	forward	java.lang.String

The following sections describe briefly the logic that is contained within each JSP.

Account Details JSP

The `accountDetails.jsp` page shows the details for a particular customer account and gives options to execute a transaction:

- ▶ The JSP uses a single request variable called `account` to populate the top portion of the body of the page, which shows the account number and balance.
- ▶ The middle section is a simple static form, which provides fields for the details of a transaction (transaction type, amount, and destination account) and posts the request to the `PerformTransaction` servlet for processing.
- ▶ The Customer Details button brings the user to the `listAccounts.jsp` page.

Figure 37 on page 56 shows the Page Designer view of this page.

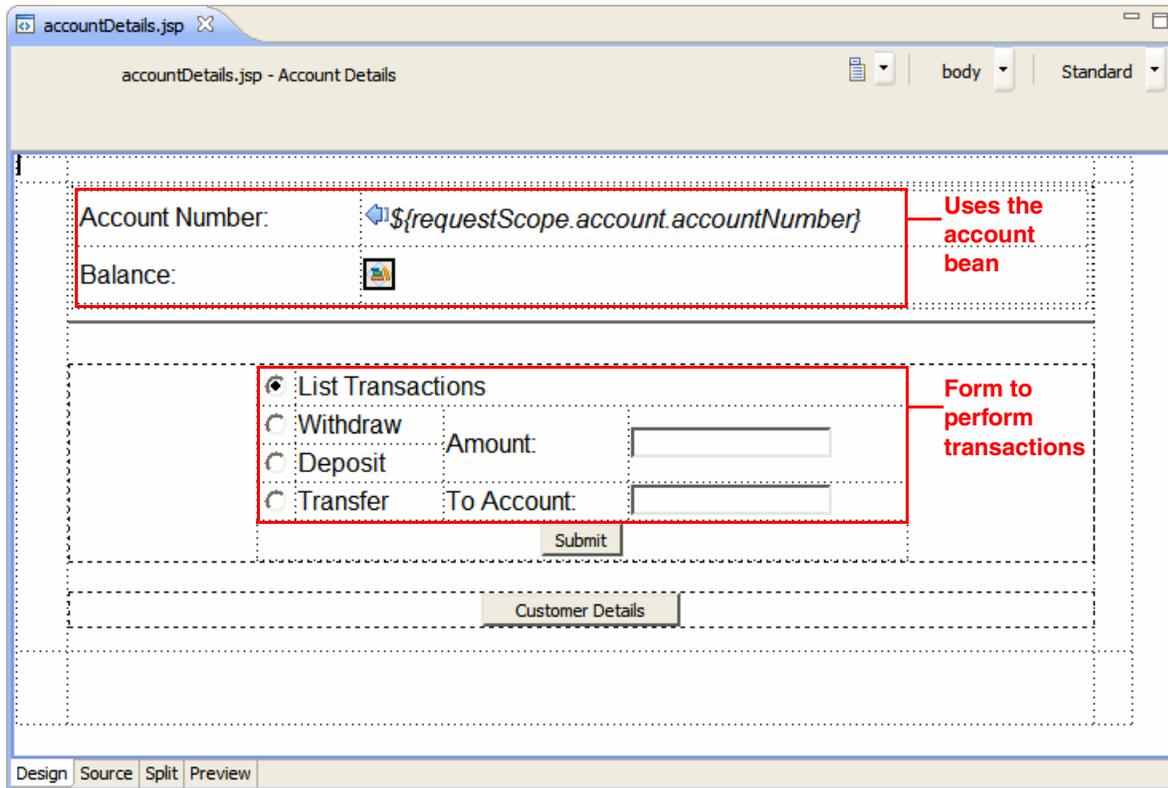


Figure 37 Completed `accountDetails.jsp` in a preview

List Transactions JSP

The `listTransactions.jsp` page shows a read-only view of the account, including the account number and balance, plus a list of all transactions:

- ▶ The JSP uses two request variables called `account` and `transactions`. The first section of the page uses the `account` request bean to populate a table that shows the account number and balance.
- ▶ The middle section uses the `transactions[]` request bean to show a list of transactions. This transaction array bean uses the JSTL tag library to iterate through the transaction list and build up an HTML representation of the transaction history.
- ▶ The Account Details button returns the browser to the `accountDetails.jsp` page.

Figure 38 on page 57 shows the Page Designer view of this page.

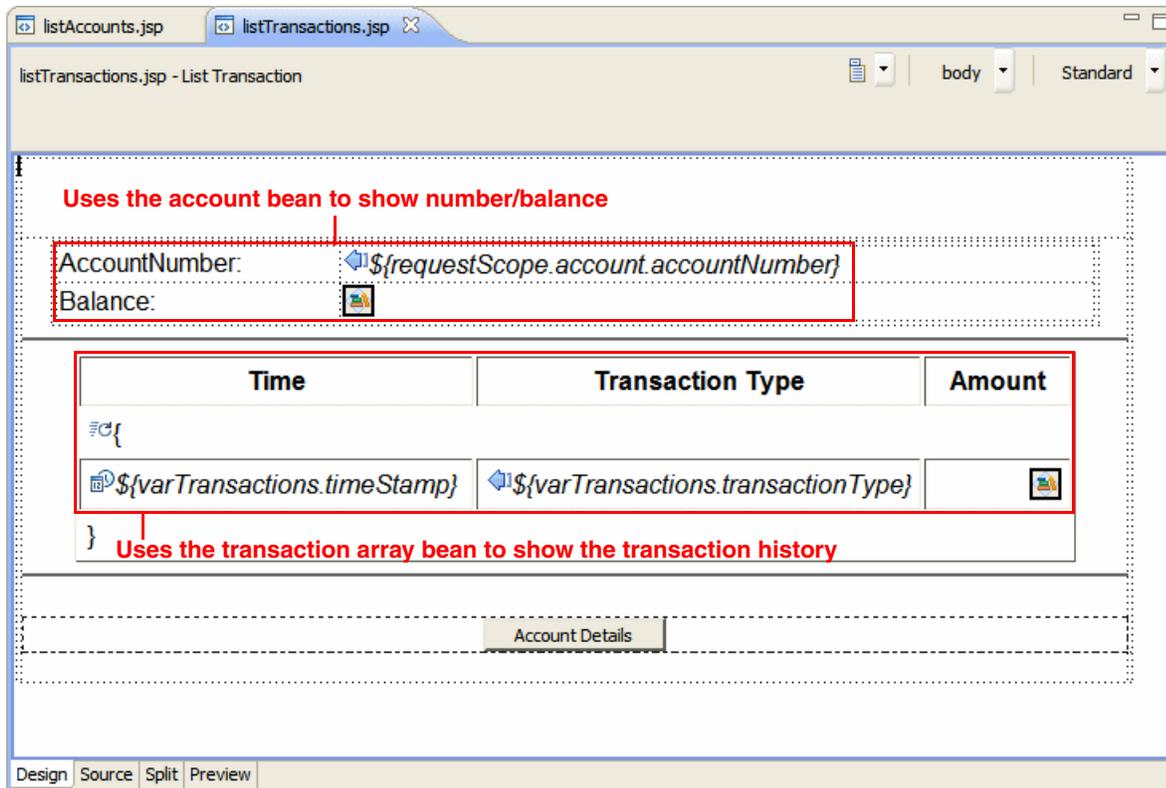


Figure 38 Completed listTransactions.jsp in a preview

Show Exception JSP

The showException.jsp page is displayed when an exception occurs in the processing of a request:

- ▶ The JSP shows a simple error message and gives a link to another page within the RedBank application to allow the user to continue.
- ▶ Two request beans are used on this page. The message bean stores the text to display to the user, and the forward bean stores a URL for the next page to continue. The URL is hidden behind the text `Click here to continue`.

Figure 39 shows this page in the Design view of the Page Designer.

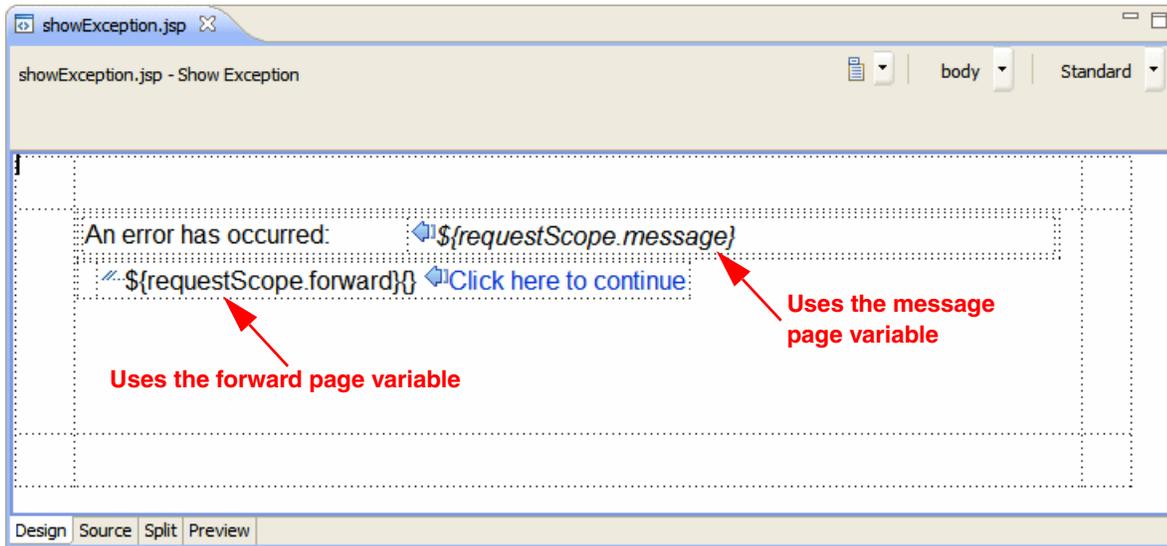


Figure 39 Completed showException.jsp in a preview

The RedBank application is finished and ready to be tested.

Web application testing

In this section, we demonstrate how to run the sample RedBank application that we built in the previous sections.

Prerequisites to run the sample web application

To run the RedBank application, you must choose one of the following actions:

- ▶ Complete the sample that follows the procedures that are described in “Implementing the RedBank application” on page 20.
- ▶ Import the completed projects from `\4880codesolution\jsp\RAD8Web-JSP.zip`.

Running the sample web application

To run the RedBank web application in the test environment, follow these steps:

1. Right-click **RAD8BankBasicWeb** in the Enterprise Explorer view and select **Run As** → **Run on Server**.
2. In the Server Selection window, select **Choose an existing server** and select **WebSphere Application Server v8.0 Beta**. Select the **Always use this server when running my project** option so that this step can be skipped next time. Click **Finish**.

The main page of the web application is displayed in a web browser inside Rational Application Developer.

Verifying the RedBank web application

After you start the application by running it on the test server, verify that the web application works properly:

1. From the main page, select the **redbank** menu option.
2. On the RedBank page (Figure 40), type a customer's Social Security number (SSN), for example, 222-22-2222.

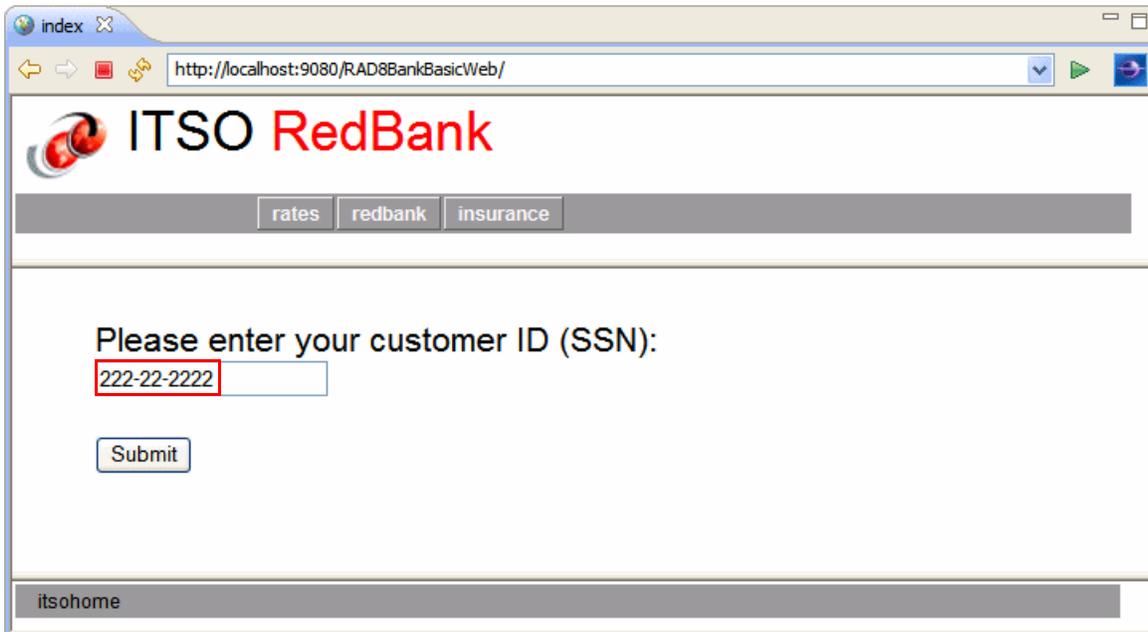


Figure 40 ITSO RedBank login page

3. Click **Submit**. The page now lists the customer and accounts (Figure 41).

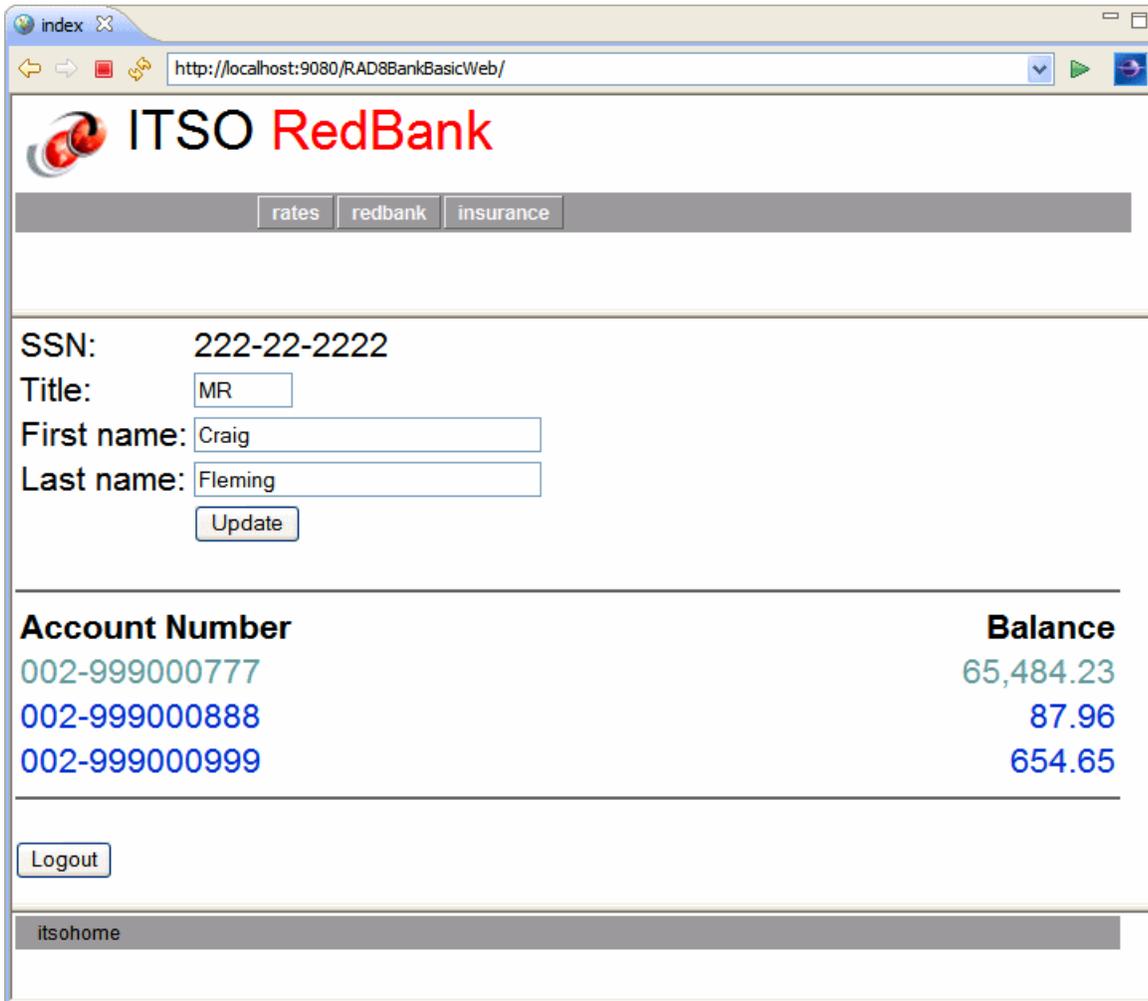


Figure 41 Listing of customer accounts

4. From the list of accounts, choose one of the following actions. In this example, we click an account to view the account information.
 - Change the customer title or name fields. For example, change the title to **Sir**. Then click **Update** to perform the doPost method of the UpdateCustomer servlet.
 - Click **Logout** to return to the Login page.
 - Re-enter the Social Security number (SSN), click **Submit**, and verify that the Customer name changed.
5. Click the link for one of the accounts, and from the account view (Figure 42 on page 61), choose one of the following actions:
 - Select **List Transactions** and click **Submit**. There are no transactions yet.
 - Select **Deposit** or **Withdraw**, enter an amount, and click **Submit** to execute a banking transaction. The page is redisplayed with the updated balance.
 - Select **Transfer** and enter an amount and a target account. Then click **Submit**. The page is redisplayed with the updated balance.
 - Click **Customer Details** to return to the account listing.

In this example, we run a few transactions (deposit, withdraw, and transfer). Then we select **List Transactions** and click **Submit**.

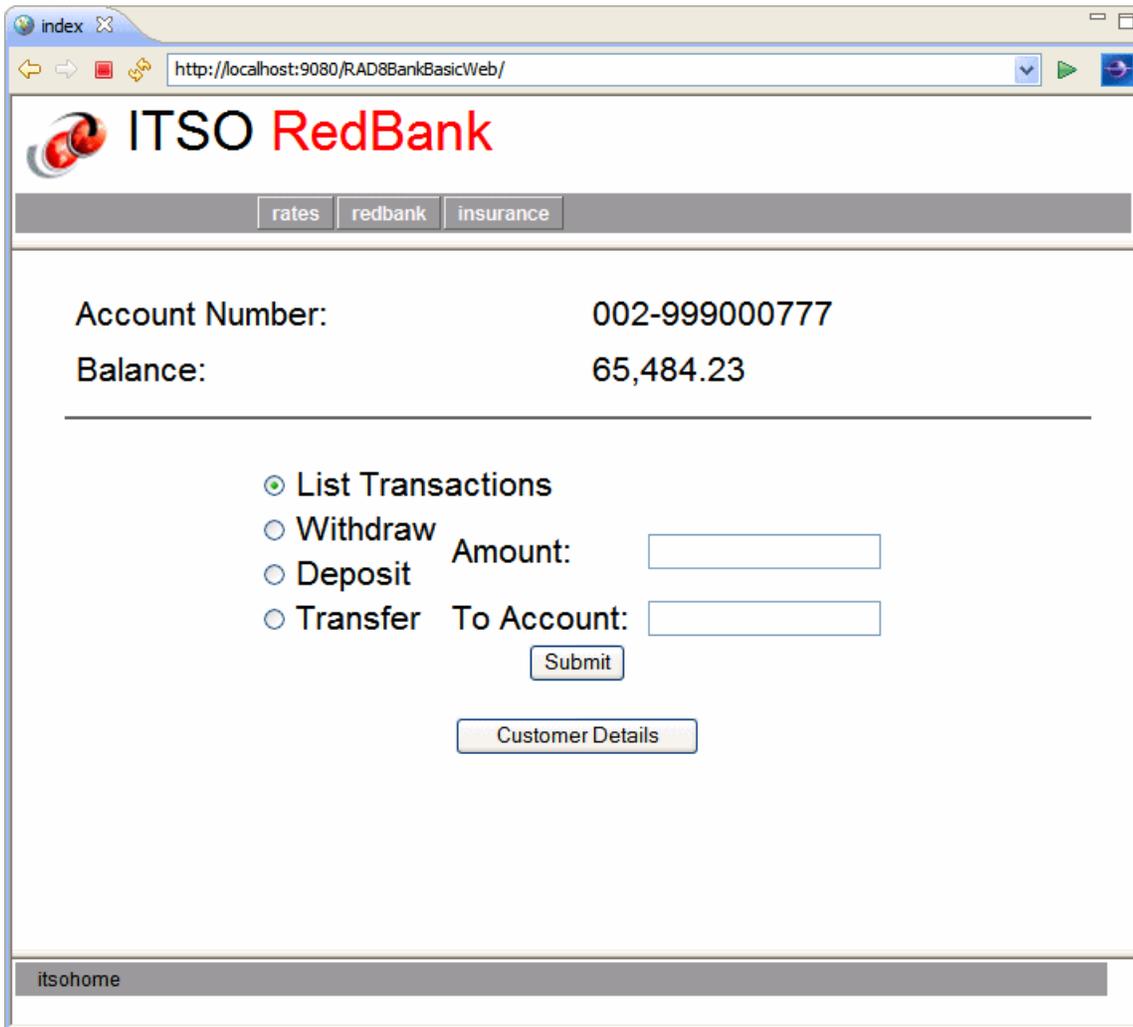
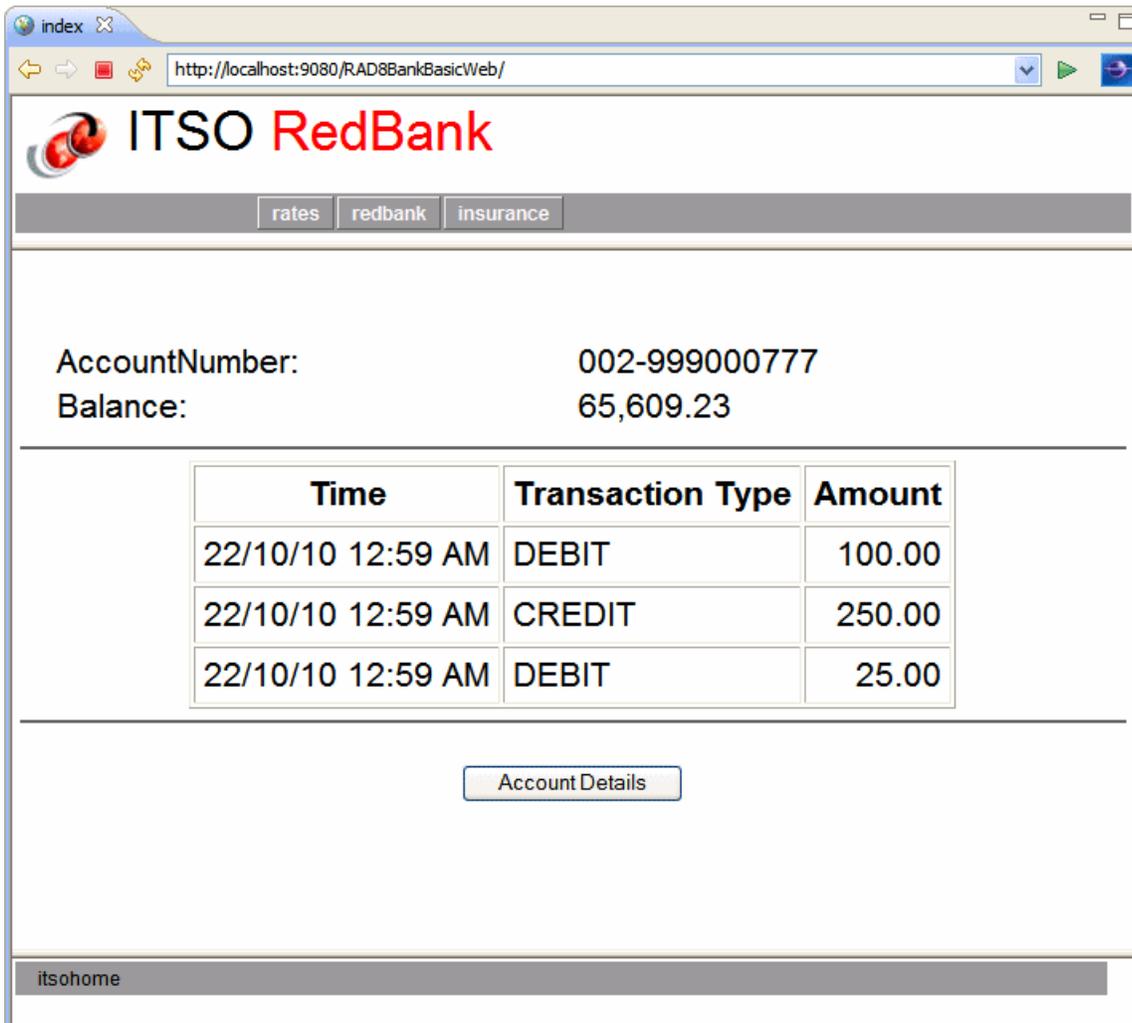


Figure 42 Details for a selected account

Figure 43 shows the transaction listing.



The screenshot shows a web browser window with the URL `http://localhost:9080/RAD8BankBasicWeb/`. The page header features the ITSO RedBank logo and navigation tabs for `rates`, `redbank`, and `insurance`. The account information is displayed as follows:

AccountNumber: 002-999000777
Balance: 65,609.23

Time	Transaction Type	Amount
22/10/10 12:59 AM	DEBIT	100.00
22/10/10 12:59 AM	CREDIT	250.00
22/10/10 12:59 AM	DEBIT	25.00

Below the table is a button labeled `Account Details`. The footer of the page contains the text `itsohome`.

Figure 43 List of transactions for an account

6. Try a withdrawal of an amount greater than the balance. The Show Exception JSP shows an error message (Figure 44 on page 63).

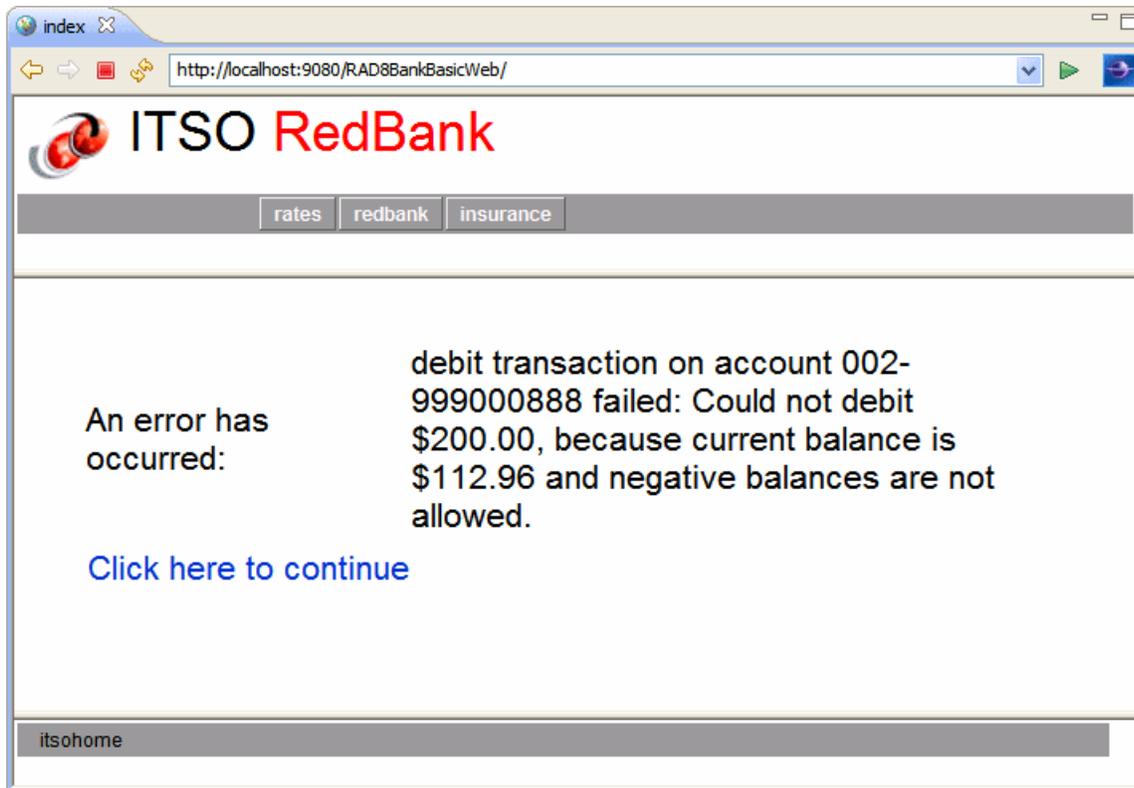


Figure 44 Withdrawal over the limit error

More information

The RedBank application can be improved in many ways; for example, by adding features or by using other technologies. These methods are explained in the following chapters of the original IBM Redbooks publication from which this paper is excerpted (*Rational Application Developer for WebSphere Software V8 Programming Guide*, SG24-7835):

- ▶ To use a database rather than HashMaps, see Chapter 9.
- ▶ To use EJB to store the model, see Chapter 12.
- ▶ To use JSF components rather than JSTL, see Chapter 19.
- ▶ To debug the application, see Chapter 28.

The book is available at the following website:

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

The Help feature provided with Rational Application Developer includes many topics about developing websites and applications. It contains reference information for all the features presented in this paper and further information about topics only covered briefly here, including JSP tag libraries and security.

For more information about the topics in this paper, see the following websites:

- ▶ Oracle Java Servlet Technology home page, which contains links to the specification, API Javadoc, and articles about servlets:

<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>

- ▶ Oracle JavaServer Pages Technology home page for technical information about JSP:
<http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
- ▶ JavaServer Pages Standard Tag Library (JSTL) home page for technical information about JSTL:
<http://www.oracle.com/technetwork/java/index-jsp-135995.html>
- ▶ “JSP and Servlets best practices,” article that articulates clearly the various ways of applying an MVC pattern to JSP and servlets:
<http://www.oracle.com/technetwork/articles/javase/servlets-jsp-140445.html>
- ▶ *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316
<http://www.redbooks.ibm.com/abstracts/sg246316.html?Open>

Locating the web material

The web material that is associated with this paper 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/SG247835>

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-4880.

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:

<i>File name</i>	<i>Description</i>
4880code.zip	Compressed file that contains sample code
4880codesolution.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, Raleigh 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 IBM 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:

<http://www.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-4880-00 was created or updated on July 24, 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.



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:

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

The following terms are trademarks of other companies:

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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.