



IBM Software Group

## IBM Rational Application Developer V6

### *JavaServer Faces In Application Developer*



@business on demand.

© 2004, 2006 IBM Corporation  
Converted to video July 7, 2015

This presentation will focus on discussing the JavaServer™ Faces support available in IBM Rational® Application Developer V6.

## Goals

- Provide an overview of IBM Rational Application Developer JavaServer Faces (JSF) tools support
- Introduce various JSF components
  - ▶ Standard
  - ▶ IBM Custom Components

The goals for this presentation are to provide an overview of IBM Rational Application Developer JSF support and introduce both standard and IBM custom JSF components.

## Agenda

- Tools Overview
- Page Data
- Faces Component Overview
  - ▶ Input/Output Components
  - ▶ Command Components
  - ▶ Selection Components
  - ▶ Miscellaneous Components
- Summary and References

The agenda begins with providing a JSF tools overview, followed by a description of the page data associated with a particular JSF page. The remaining part of the presentation will give you an overview of the various JSF components that are available.

## Section

# *JavaServer Faces Tools Overview*



This section is an overview of the JSF tools support.

## JSF Support In Application Developer

- JSF tools integrates with the existing Page Designer JSP editor and views associated with web tools
- JSF tools simplify development tasks with
  - ▶ Automatic page code generation and maintenance
  - ▶ Management of the faces-config.xml file
    - Navigation rules
    - Managed Beans
- Includes support for standard and IBM custom JSF components



Support for JSF development in Application Developer integrates with the existing Page Designer JSP editor and the views commonly associated with the web development tools. In addition to this, the JSF tools support simplifies many development tasks. For example, by default, Application Developer automatically provides page code generation and maintenance for each JSF page in the project. This support provides a central location for the Java code and is used to manage the various UI components and data associated with a particular JSF page. Also, the tools provide management of the faces-config.xml file for configuring navigation rules and Managed Beans.

Application Developer provides support for building JSF pages with both the standard JSF and custom IBM components.

## JSF Support In Application Developer (cont.)

- JSF tools integrate with the following tools:
  - ▶ Site Designer
  - ▶ Web Diagram Editor
  - ▶ Portlet/Portal Development
  
- JSF tools integrate with the following technologies:
  - ▶ Page Templates
  - ▶ Enterprise Generation Language (EGL)
  - ▶ Service Data Objects (SDO)
  - ▶ Web Services



JSF tools integrate with the following the other web development technology in Application Developer: Page Templates, Site Designer, Web Diagram Editor, EGL, and Portlet/Portal Development. JSF tools integrate with several other technologies, including SDO and Web services.

## JSF Development Steps

- Create a Dynamic Web Project
  - Create a Faces JSP file
    - ▶ Application Developer creates and manages faces-config.xml file and managed bean classes
  - Add JSF components and data to the page
    - ▶ Page data
    - ▶ Input and Output UI components
    - ▶ Command components
- } Key parts to a JSF page



Before going through some of the specifics of JSF development in Application Developer, this presentation will cover a little bit about the development steps that will you will need to create a JSF application. The first step is to create a dynamic web project. Each JSF application needs one or more JSF page. The next step is to use Application Developer to create a Faces JSP file. When you do this, Application Developer creates and manages the faces-config.xml file and the managed bean classes that represent your JSF page. Once you have created the JSF page you can add you will add UI components and data to the page. A typical JSF page will be include data, both input and output components, and one or more command components. Typically data associated with a particular JSF page will be bound to at least one UI component through a value binding expression, though this is not required.

## JSF Development Steps (cont.)

- Configure attributes for each JSF UI component
- Add event handlers to act on user events
  - ▶ Client and server side events supported
- Specify navigation rules for response
- Test and Debug JSF application



Once UI components have been added to the JSF page you will use the Properties view to configure the attributes for these components. In addition to this event handlers that will be used to act on user events need to be set for various UI components. For example, a command component will need action events specified, and input fields may need value change events added. It is important to note that these events can be handled as either client side (JavaScript) or server side (Java) events. If there are any command components on the page, you will need to use the Properties view to specify navigation rules for the response. Finally, the last step in the JSF development process is to test and debug the JSF application.



## JSF: The Web Perspective

Editor / View	Description
<b>Page Designer</b>	Default editor for building Faces JSP pages
<b>Palette</b>	Provides the <i>Faces Components</i> , <i>Faces Client Components</i> , and <i>Data</i> drawers containing JSF UI and Data components
<b>Page Data View</b>	Used to define and modify data associated with a JSF page. Page data items can be used to bind to JSF components
<b>Client Data View</b>	Used to define and manage client side data associated with a JSF page
<b>Properties View</b>	Used to configure attributes for a JSF UI component that is selected in Page Designer
<b>Quick Edit View</b>	Provides an editor area to add server side (Java) and client side (Javascript) code to handle user events



This slide highlights some of the important views used in JSF application development. The primary perspective for JSF development is the Web Perspective and the views associated with it. As mentioned previously, Page Designer is the default editor for building Faces JSP pages. The Palette has a number of drawers that include Faces components, Faces Client Components, and a Data drawer that are used in conjunction with Faces JSP pages. The Page Data and Client Data view are used to define and modify server and client data associated with a particular JSF page. The Properties view is used to configure attributes associated with a particular JSF UI component that has been selected in the Page Designer editor. Finally, the Quick Edit view allows you to add both server side and client side code to handle user events.

## JSF: Application Organization and Artifacts



10

This slide highlights some of the important application artifacts associated with a JSF project. Highlighted on this slide is (1) the page code classes associated with a specific JSF page, (2) cascading style sheets used to define the appearance of JSF components, (3) the faces-config.xml file, and (4) Faces JSP pages.

## Demonstration: Create a JSF Page

- This demonstration will highlight
  - ▶ How to create a new Faces JSP file
  - ▶ Project organization and artifacts within Application Developer
  - ▶ Common views used to build Faces JSP pages

Create  
a JSF Page

Add  
Page Data

Add JSF  
Components

Test  
Application



Pause this presentation and click the Show Me icon to view a demonstration that will highlight how to create a new Faces JSP file. This demonstration will also point out the project organization and artifacts and the common views used to build Faces JSP pages.

## Section

# *Page Data*

The next section will discuss Page Data associated with JSF development.

## Page Data: Overview

- Page Data includes one or more model objects associated with a JSF page
  - ▶ Added from Page Data view or the Palette (Data drawer)
- Value binding expressions associate page data with JSF UI components
  - ▶ Page data objects do not have to be bound to a JSF component
- Page data is tightly coupled with the page code file
  - ▶ Public get methods show up as available data



In JSF development, page data is used to define one or more model objects associated with a JSF page. Data is added a JSF page from either the Page Data view or the Data drawer on the Palette. Once data has been added to the page data for a JSF page, a value binding expressions is used to associate the page data item with a particular JSF UI component on the page. However, it is important to note that page data objects do not have to be bound to a JSF component. Developers new to JSF support in Application Developer should note that page data that has been added through the Page Data view or the Data drawer is tightly coupled with the page code file associated with the JSF page. This means that data items added will automatically have the appropriate data fields and methods added to the page code file. Likewise, any public getter methods added to the page code file by the developer, will show up in the page data view as available data to bind to JSF components.

## Types of Page Data Objects

- JSP Scripting
  - ▶ Defines variables that are available from various scopes accessible by the JSF page
- JavaBeans
  - ▶ Properties are available for binding like any other page data object
  - ▶ Can be a managed bean
- Web Service
  - ▶ Used to access a Web Service from a JSF page



This slide and the next lists the various types of data that can be added to a JSF page. The first is a set of JSP scripting data items that allow you to define variables that are available from the various scopes accessible by the JSF page. These scopes include application, session, request, and parameter scope variables. Another common page data source is the Java bean data item. The properties on a Java bean are available to binding to JSF UI components just like any other types of page data objects. In addition to Java beans, web services can be added to JSF pages as a data source. In this case the web service is used to access a web service client from a JSF page.

## Types of Page Data Objects (cont.)

- **Relational Record and Record List**
  - ▶ Used to add a record or list of records from a Relational Database to a Faces page
  - ▶ SDO-based data item that uses the JDBC Mediator
- **Stateless Session Bean**
  - ▶ Used to access an EJB session bean from a JSF page



The relational record and record list are page data sources associated with the SDO-based JDBC data mediator. These data items are used to add a record or list of records from a relational data base to a Faces JSP page. Another page data item available is a stateless session bean. This page data source provides access to the methods of an EJB session bean. The input parameters are added to the JSF page as input components, and the result is added as an output component. This data item is intended to be used with SDO-based EJB data mediator. Finally, there are several other SDO-based data mediators available from the page data view. These are the Seibel Record and Record List, and the SAP BAPI and RFI. These technologies are supported for portal JSF projects only.

## Demonstration: Adding Page Data

- This demonstration will highlight
  - ▶ How to create a JavaBean Page Data object
  - ▶ Changes to the page code file when a Java Bean is added
  - ▶ Tight association of page code and page data view



From the slide you can access a demonstration that will highlight how add a JavaBean to a JSF page from the Page Data view. This demonstration will also point out the changes that were made to the page code file after the JavaBean has been added.



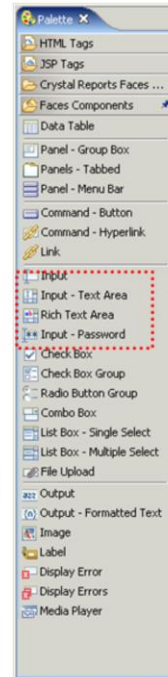
## Section

# *Input / Output / Command Components*

The next section will discuss input, output and command JSF components.

## Input Components

- Several types
  - ▶ Input (String, Date/Time, Number, Mask)
  - ▶ Input – Text Area
  - ▶ Input – Password
  - ▶ Rich Text Area
    - Rich text entry capability like a Word processor
- Can bind input value to page data
- Can use the Quick Edit view to add Value Changed handler code (client or server side)
- Client-side input assist capabilities available

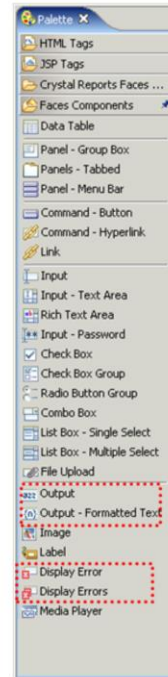


There are several types of input components available from the Faces Components drawer on the palette. The underlined items indicate components that are IBM custom JSF components. The value attribute for an input component is typically bound to a data item in the page data view using a value binding expression. Binding a data item to an input component results in the value entered into this component being used to set the server side data item.

The handling of user events can be added from the Quick Edit view, and from here there is the ability to add both client and server side event handling. Most input components have input assist capabilities that can be set from the properties view. This support provides client side JavaScript to assist user input.

## Output Components

- Several types
  - ▶ Output (String, Date/Time, Number, Mask)
  - ▶ Output- Formatted Text
  - ▶ Display Error
    - Displays error message associated with a single component
  - ▶ Display Errors
    - Displays all error messages associated with the page or only errors that are not associated with a specific component
  
- Can bind output value to:
  - Static value
  - Page data variable
  - Property value from a Resource Bundle



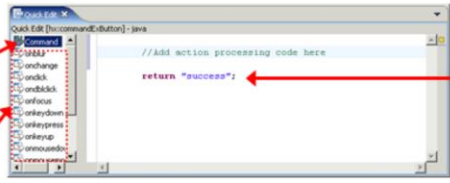
There are several types of output components available from the Faces Components drawer on the palette. The underlined items indicate components that are IBM custom JSF components. The value attribute for an output component is typically bound to a data item in the page data view using a value binding expression. Binding a data item to an output component results in the server side data item being output in the client page. It is possible to an output value to a static variable, page data variable, or a property value from a resource bundle.

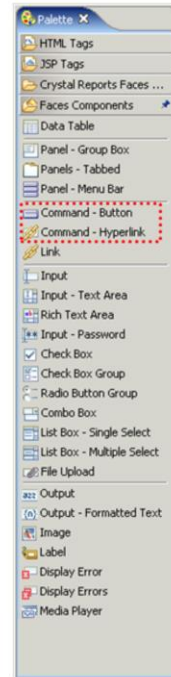
## Command Components

- Two types
  - ▶ Command Buttons – Press button/image
  - ▶ Command Hyperlink – Hyperlink look
- Event handling
 

Server side Java

Client side Java Script


- Navigation rules are defined from the properties view for command components
  - ▶ Navigation rules match String returned from the Command event script



There are two command components available from the Palette, the command button and the command hyperlink. Command components have an action attribute that can be bound to an action method through a method binding expression. In Application Developer, the Quick Edit view is used to provide an action method to the page code file and associated with the command component selected in the Page Designer editor area. The action method is being edited when the Command item is selected from the right hand list in the Quick Edit view. This is server side Java code that is run during the Invoke Application phase of the JSF request processing lifecycle. The other items available in the Quick Edit view are used to provide client side event handling. A final note about command components is that it is from the Properties view for these components that navigation rules can be defined.

## Demonstration: Adding JSF Components

- This demonstration will highlight
  - ▶ Adding input/output components to a page
  - ▶ Binding these components to page data
  - ▶ Configuration
    - Type of input/output
    - Input assist
  - ▶ Defining event handlers



From the slide you can access a demonstration that will highlight how to add input and output components to a JSF page and bind these components to page data sources. This demonstration will also point out the configuration options for these components and how to define event handlers from the Quick Edit view.

## Section

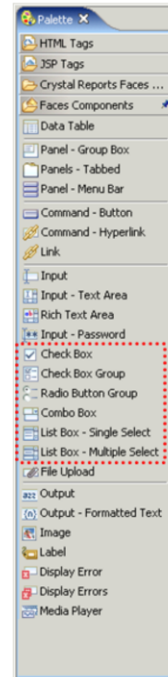
# *Other JSF Components*



The next section will discuss a variety of other JSF components.

## Selection Components

- Several types
  - ▶ Check Box – Boolean value
  - ▶ Radio Button Group – Single choice
  - ▶ Check Boxes Group – Multiple choice
  - ▶ List Box – Single Select and Multiple Select
  - ▶ Combo Box
  
- Two types of list items (choices)
  - ▶ Simple text
  - ▶ Dynamic (Computed Items)
  
- Data variable is bound to user selection

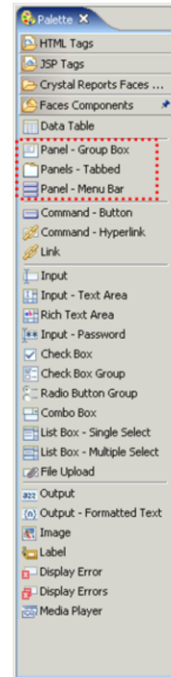


There are several types of selection components available from the Faces Components drawer on the Palette. These selection components range from single selection items to multiple choice selection options and also differ in the way in which they are displayed. All selection components provide two ways to specify the list items or choices available rendered at runtime. The first is a simple text option used when the choices are known prior to runtime. An example of this would be a “yes” or “no” option. A dynamic set of selection choices is one that is not known until runtime. This might include a selection of items that is in a shopping cart. Not only does the selection component need to be bound to the choices in the list, this components also needs to be bound to a data variable that holds the user’s selection.

## Panels

### ■ Several Types

- ▶ Panel – Group Box (Grid, List, Snap to Border, HTML Panel)
  - Support list, positioned and compound HTML/JSF layouts
- ▶ Panel – Menu Bar
  - Used to simplify building horizontal and vertical bars containing commands/links and nested sub-trees
- ▶ Panels – Tabbed

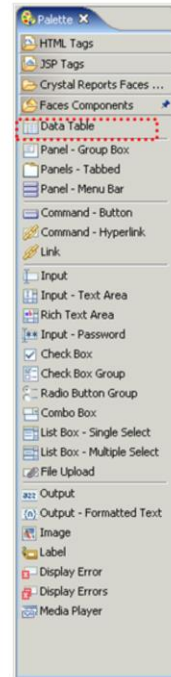


There are several types of Panel components available from the palette. Generally these components are used to group or embed other JSF or HTML components. With the exception of the Tabbed Panel the panel components are not visual by nature.



## Miscellaneous Components

- Data Table
  - ▶ Supports arrays, Vectors, and JDBC ResultSets
  - ▶ Paging
  - ▶ Row Categorization
  - ▶ Row Edit
    - Client-side JavaScript
    - Do not have to go back to the server until submit action



25

JavaServer Faces in Application Developer

© 2004, 2006 IBM Corporation

The data table is used to provide a table component for container-type data items. Specifically, the data table supports array, Vector, and JDBC ResultSets data items. Although the data table is a standard JSF component, there are several types of configuration options available that are IBM custom options that extend the data table functionality. Specifically, the data table provides paging, row categorization, and row edit capabilities. The row edit functionality is provided by client side Java script that allows you to edit data included in the table with the need to go back to the server until a submit action has been completed.

## Miscellaneous Components (cont.)

### ▪ Image

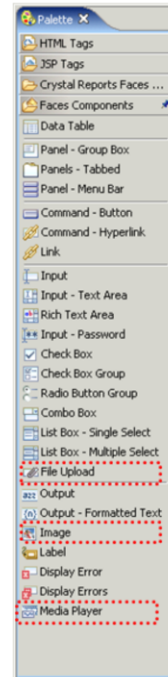
- ▶ Images can be easily retrieved from data sources

### ▪ File Upload

- ▶ Simplifies uploading files from a data store

### ▪ Media Player

- ▶ Easily supports using media managed by data stores



There are several important IBM custom components available from the palette. These include the Image, File Upload, and Media Player components.

## Section

# ***Testing JSF Application***

The next section includes a demonstration of running a JSF Application.

## Demonstration: Testing a JSF Application

- This demonstration will highlight
  - ▶ Deploy application
  - ▶ Test JSF application

Create  
a JSF Page

Add  
Page Data

Add JSF  
Components

Test  
Application



From the slide you can access a demonstration that will highlight how to deploy and test a JSF application.

## Section

# *Summary and Reference*

The next section will provide a summary and references for this presentation.

## Summary

- Java Server Faces technology combined with tools support provides an easier way to create dynamic Web pages with rich UIs
- IBM Rational Application Developer integrates JSF technology with a variety of web development technologies



In summary, this presentation has focused on providing an overview of JSF support available in Application Developer V6. JSF combined with the web tools in Application Developer provides an easier way to create dynamic Web pages with rich user interface components. In addition to this, Application Developer integrates JSF technology with a variety of new and existing Web development technologies.