Oracle® Fusion Middleware

Mobile Browser Developer's Guide for Oracle Application Development Framework

11*g* Release 1 (11.1.1)

E10140-04

April 2010

ORACLE®

Oracle Fusion Middleware Mobile Browser Developer's Guide for Oracle Application Development Framework 11*g* Release 1 (11.1.1)

E10140-04

Primary Author: John Bassett

Contributing Author: Tadashi Enomori

Contents

4 Skinning

5 Supporting Basic HTML Mobile Browsers

6 Design Guidelines for BlackBerry 4.2 to 4.5

7 Narrow Screen Support and User-Agent Details Support

8 Extending ADF Mobile Browser Applications

Index

Preface

Welcome to *Mobile Browser Developer's Guide for Oracle Application Development Framework*.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Audience

This document is intended for developers of browser applications.

Related Documents

You can also find information about ADF (Application Development Framework) in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Overview of Oracle ADF Mobile Browser

This chapter provides an overview of Oracle Application Development Framework Mobile (ADF Mobile) browser.

This chapter includes the following sections:

1.1 About ADF Mobile Browser

Oracle Application Development Framework Mobile (ADF Mobile) browser is a standards-based framework that enables the rapid development of enterprise mobile applications. Oracle Fusion Middleware 11*g* release 1 of ADF Mobile browser extends Oracle ADF to browsers running on mobile devices. Because ADF Mobile browser is built upon the component model of Java Server Faces (JSF), you can quickly develop applications for mobile browsers. ADF Mobile browser's mobile-specific extensions to JSF enable you to develop mobile applications using the same methodologies for developing JSF applications for the desktop.

When developing an ADF Mobile browser application, you need not focus on the limitations or capabilities of different browsers, as ADF Mobile browser enables you to develop applications that function properly on different browser types. The ADF Mobile browser renderer ensures that contents can be consumed correctly by the target browser. It handles the variations in both browser implementations of HTML, JavaScript, CSS, DOM, `XMLHttpRequest`, and in system performance. For example, if a browser does not support `XMLHttpRequest` and is incapable of posting a partial page request to a server, ADF Mobile browser's support for AJAX (Asynchronous JavaScript and XML) enables the application to revert automatically to a full page submit so that the same page functions whether the browser supports `XMLHttpRequest` or not.

> **Note:** For Oracle Fusion Middleware 11*g* release 1, ADF Mobile browser requires HTML and JavaScript support.

1.1.1 About Java Server Faces and the Application Development Framework

Java Server Faces (JSF) is a standard specified by JSR-127 that enables developers to create applications using pre-built components that define functionality and behavior. JSF provides a clean Model-View-Controller (MVC) mechanism that simplifies the development of Web applications through its renderkit, which converts components both to and from a specific markup language. The renderkit's renderers abstract the

production of markup and responses to browser requests by generating the markup representations of components and the way in which these components should interpret browser requests.

JSF development focuses on components, not markup. Using JSF, you create a JSP page containing JSF component tags. When a user visits this page (through the FacesServlet) JSF uses the renderkit specified by the user's device to encode the markup for the appropriate output. For example, if the user's device specifies HTML for a desktop browser, then the renderkit's markup encoding results in an HTML page. In addition to rendering appropriate content, JSF supports user interaction.

Application Development Framework (ADF) is built on the standard JSF technology and provides the following:

- A large component set (since JSF provides only basic components)
- Renderers that support these components in HTML browsers, including a rich renderkit for applications using AJAX technologies
- Converters, validators, and events

1.1.2 Developing Mobile Applications Using ADF Mobile Browser

You can use the same programming model and component set for developing desktop browser applications to develop mobile browser applications for mobile devices. ADF Mobile browser application development is almost identical to ADF Web application development, except that ADF Mobile browser application development uses only mobile JSF pages that consist of Apache MyFaces Trinidad components. For more information on developing ADF Web applications, see *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

> **Note:** You cannot use ADF Faces components to develop an ADF Mobile browser application. You must use Apache MyFaces Trinidad components.

Developing mobile browser applications for mobile devices with ADF Mobile browser leverages the same methodologies used in developing JSF applications for the desktop but with a few specific mobile extensions. With support for over 60 Apache MyFaces Trinidad components, you can build applications with the rich component set, each of which renders appropriately for small-screen mobile devices. In this way, you can reuse the desktop browser application's model and controller layers to assemble a new view layer for mobile devices by using similar Apache MyFaces Trinidad components.

> **Note:** Oracle JDeveloper only supports the JSF page flows for ADF Mobile browser application development. The ADF task flow is not supported.

How ADF Mobile Browser Improves Performance

The PDA component renderers have been optimized to minimize the payload of the Web page sent to the mobile device for improved performance over wireless networks. In mobile environments with high-latency and low-bandwidth wireless networks, Partial Page Rendering (PPR) is essential in providing end-users with an efficient application. For mobile browsers supporting AJAX, ADF Mobile browser supports PPR for certain components to minimize the amount of data requested from

the server and improve the responsiveness of the applications. See also Section 3.1.2, "Partial Page Rendering."

1.2 Supported Mobile Browsers

ADF Mobile browser supports Apache MyFaces Trinidad components on the browsers listed in Table 1–1. Later versions of Trinidad can be integrated into Oracle JDeveloper and used with Oracle Fusion Middleware 11*g* release 1 of ADF Mobile browser.

Table 1–1 Supported Browsers and Supported Mobile Features

Browser	JavaScript Support	CSS Support	PPR Support
BlackBerry version 4.6 and later	Yes	Yes	Yes
Blackberry versions 4.2 through 4.5	No	Yes	No
Microsoft Windows Mobile 5	Yes	Yes	Yes (with nuances)
Microsoft Windows Mobile 6	Yes	Yes	Yes
Apple iPhone Safari	Yes	Yes	Yes
Nokia s60 series	Yes	Yes	No
Plain HTML (such as Opera Mini, Opera Mobile and Skyfire)	No	Yes	No

2

Configuring the ADF Mobile Browser Environment

This chapter describes how to configure the environment for ADF Mobile browser applications and how to build and test mobile browser applications.

This chapter includes the following sections:

- Section 2.1, "About the ADF Mobile Browser Development Environment"
- Section 2.2, "Configuring the ADF Mobile Browser Development Environment"
- Section 2.3, "Developing an ADF Mobile Browser Application"
- Section 2.4, "Testing an ADF Mobile Browser Application"

2.1 About the ADF Mobile Browser Development Environment

ADF Mobile browser application development is almost identical to ADF Web application development, except that ADF Mobile browser application development uses only mobile JSF pages that consist of Apache MyFaces Trinidad components.

> **Note:** Oracle JDeveloper supports only the JSF page flows for ADF Mobile browser application development. The ADF task flow is not supported.

To create an ADF Mobile browser application:

- Configure the environment by creating an application and project.
- Add a Web project.
- Add the JSF pages using Apache MyFaces Trinidad components.

2.2 Configuring the ADF Mobile Browser Development Environment

ADF Mobile browser application development differs only from ADF Web application development for desktop browsers in the creation of the Web project. For more information, see Section 2.3, "Developing an ADF Mobile Browser Application."

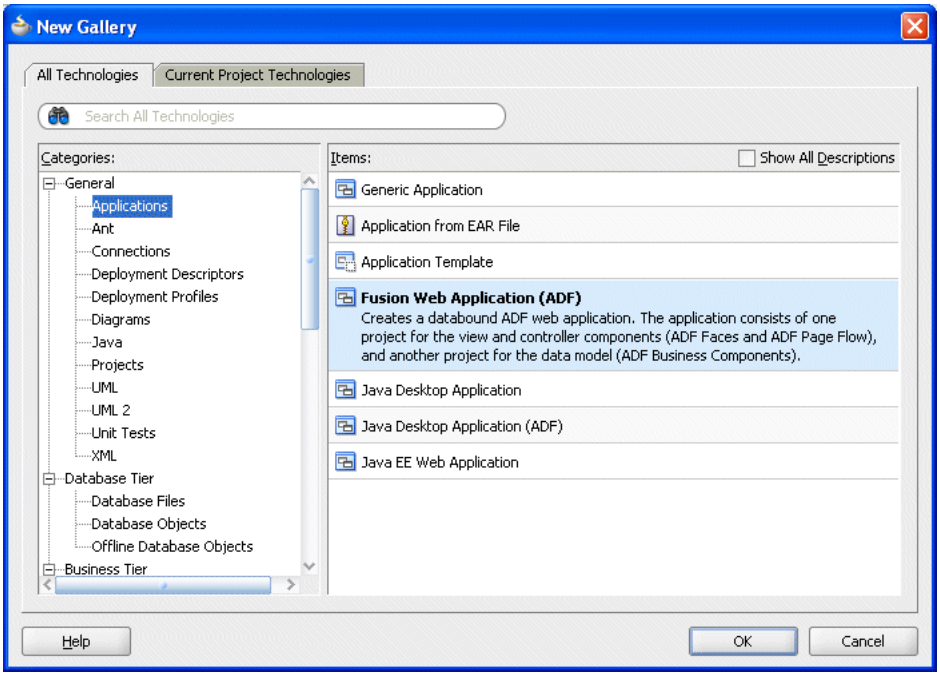
2.2.1 How to Configure the Environment by Creating a Mobile Application and Project

To configure the environment, first create an ADF Mobile browser application that includes a project with the ADF Mobile browser technology.

To create the ADF Mobile Browser Application and the ADF Mobile Browser Project:

1. Choose **File** and then **New**.

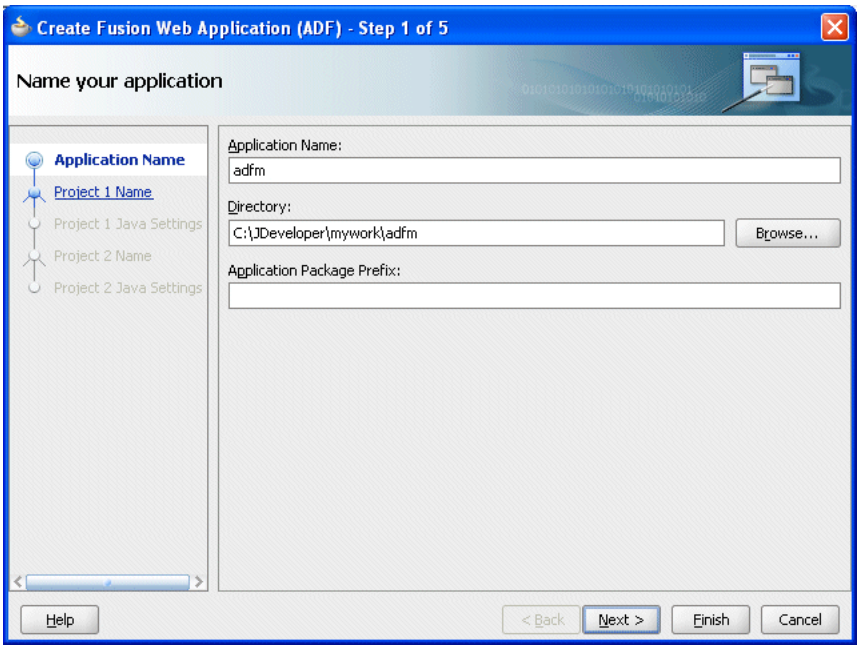
Figure 2–1 The New Gallery



2. In the New Gallery, expand **General**, select **Applications** and then **Fusion Web Application (ADF)** and then click **OK**.

3. In the Name your application page of the Create Fusion Web Application (ADF) wizard, enter a name and, if needed, a location for the application in the Directory field, as shown in Figure 2–2.

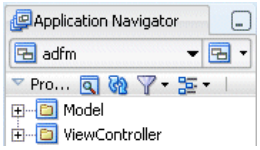
Figure 2–2 The Name your application Page



4. Click **Finish**.

   Selecting **Fusion Web Application (ADF)** creates the model project used by the mobile view project. Figure 2–3 shows the application's Model and its generated Model-View-Controller projects that appear in the Application Navigator.

Figure 2–3 The Fusion Web Application and its Projects in the Application Navigator



> **Note:** Define the business logic for the Model project, but do not use the generated Model-View-Controller project. Instead, create a mobile Model-View-Controller project as described in the following steps.

5. Choose **File** and then **New**. The New Gallery appears.
6. In the New Gallery, expand **Categories**, select **Projects** and then select **Generic Project** and click **OK**.
7. In the Create Generic Project wizard, complete the wizard by first entering a name for the project. For example, enter *mvc* (a short name for Model-View-Controller mobile).

   > **Tip:** To make entering a URL on a mobile device easier, enter short, lower-case names for both the application and the project.

8. Select the ADF Mobile browser technology for the project by moving **Mobile Browser** from the **Available** list to the **Selected** list.

Figure 2–4 Selecting the Mobile Browser Technology for a Project

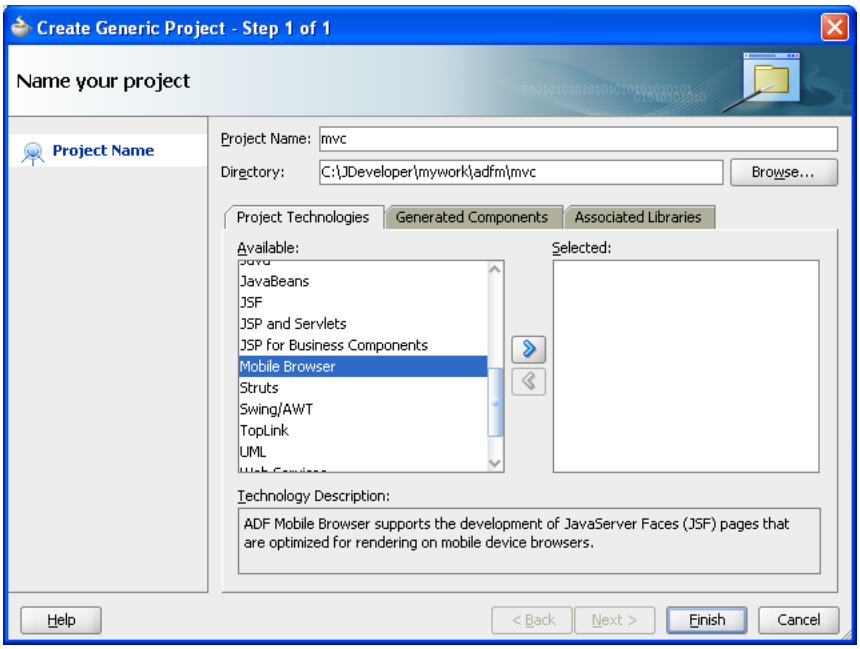
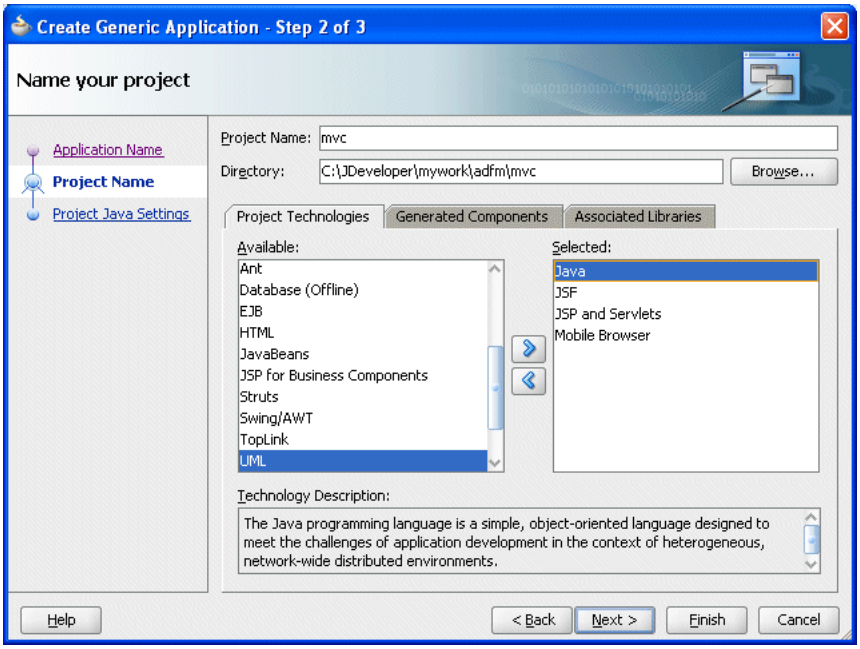


Figure 2–4 shows the Mobile Browser technology in the Available list. After you move the Mobile Browser technolgy to the Selected list, the following technologies are made available to the project and also appear in the Selected list, as shown in in Figure 2–5:

- Java
- JSF (JavaServer Faces)
- JSP and Servlets

Figure 2–5 Mobile Browser and Supporting Technologies Selected for a Project

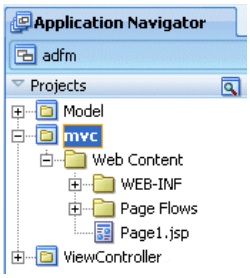


9. Click **Next** to navigate through the Configure Java settings page and then click **Finish**.

2.2.2 What Happens When You Create a Mobile Application and Project

As shown in Figure 2–6, the mobile Model-View-Controller project (*mvc*) appears in the Application Navigator within the Fusion Web application (*adfm*).

Figure 2–6 The Mobile ADF Model-View-Controller Project in the Application Navigator



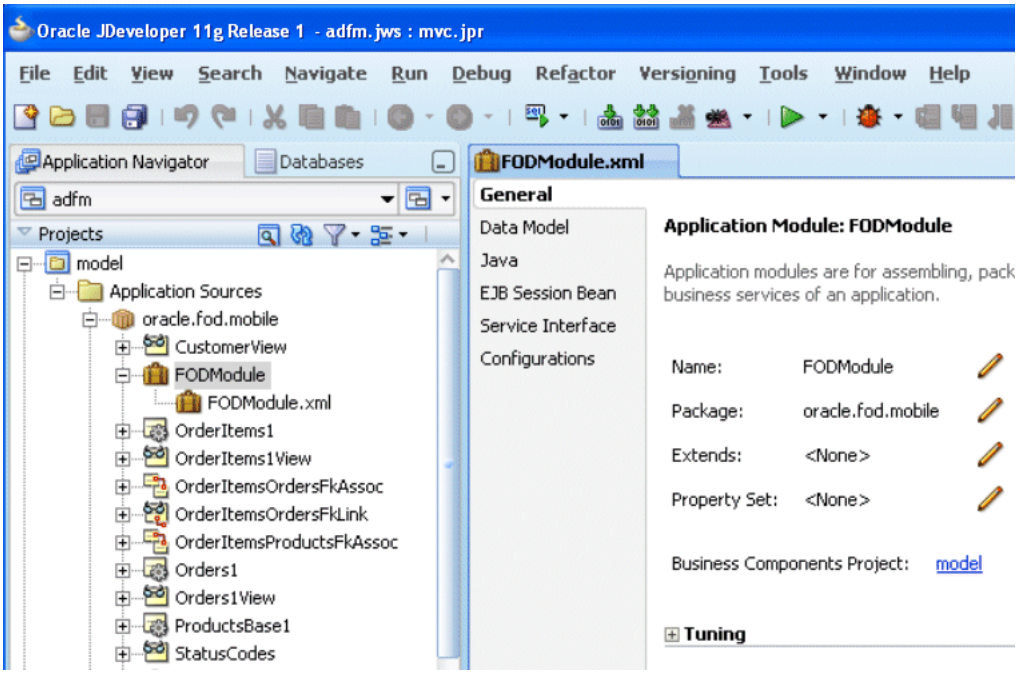
Because you added the mobile browser technology scope, the Apache MyFaces Trinidad library is automatically loaded to the workspace and the Trinidad component palette is loaded when you create mobile JSF pages, shown in Figure 2–10.

2.3 Developing an ADF Mobile Browser Application

ADF Mobile browser application development is nearly identical to ADF Web application development for desktop browsers; the two only differ in how you create the Web project. For mobile browser applications, you develop an application by creating Web pages within the Web project. Otherwise, you develop a mobile browser application the same way that you develop an ADF Web application for a desktop browser. Typically, you create a Web project within the application to implement a user interface and ADF Business Components or an Oracle TopLink project to implement a business layer.

Figure 2–7 shows a mobile application (*adfm*) that contains *model*, a business components project.

Figure 2–7 Creating a Business Components Project within an ADF Mobile Browser Application



2.3.1 How to Develop an ADF Mobile Browser Application by Creating a Mobile JSF Page

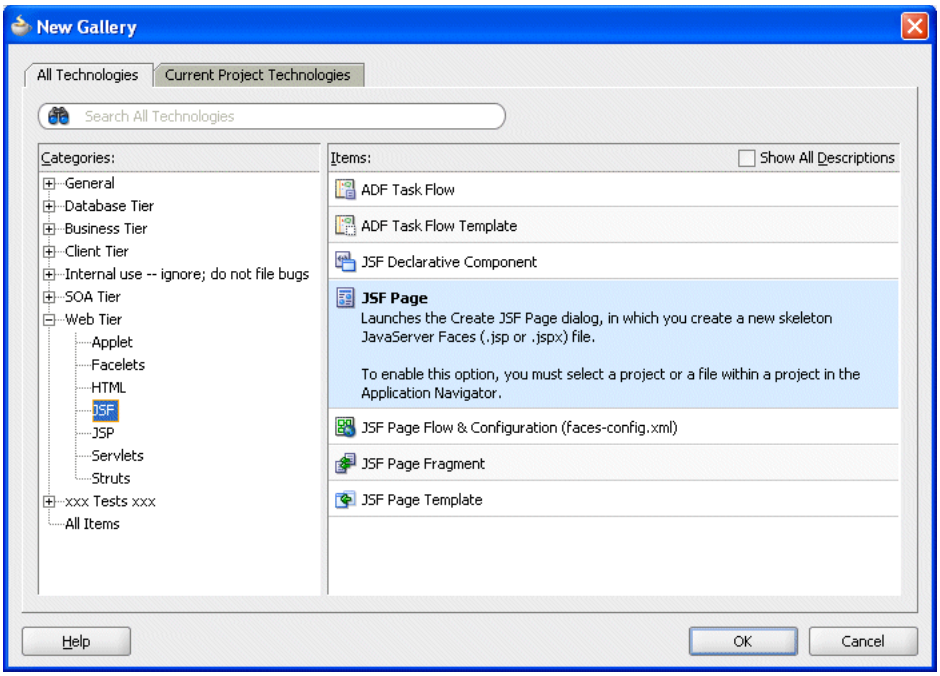
You develop an ADF Mobile browser application by first creating a JSP page and then populating it with the Apache My Faces Trinidad components.

To create a mobile JSF page:

1. Choose **File** and then **New**. The New Gallery appears.
2. In the New Gallery, expand **Categories**, select **Web Tier**, and then select **JSF** and then select **JSF Page** and click **OK**.

Note: **Project Technologies** (the default) must be selected from the Filter By list.

Figure 2–8 The New Gallery for JSF Pages



3. Enter a name for the JSF page, and if needed a directory location for it in the JSF Page dialog, shown in Figure 2–9.

> **Note:** Because you added the Mobile Browser technology scope for the application, the **Render in Mobile Device** option is selected by default, as shown in Figure 2–9.

Figure 2–9 The Create JSF Page Dialog Box

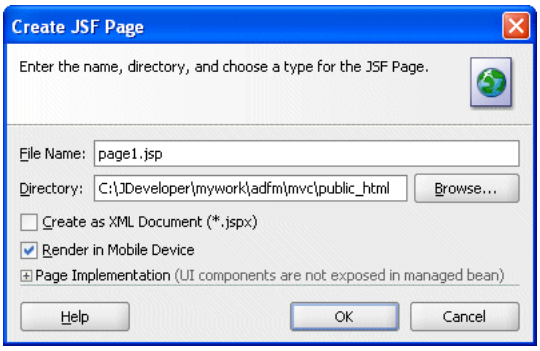
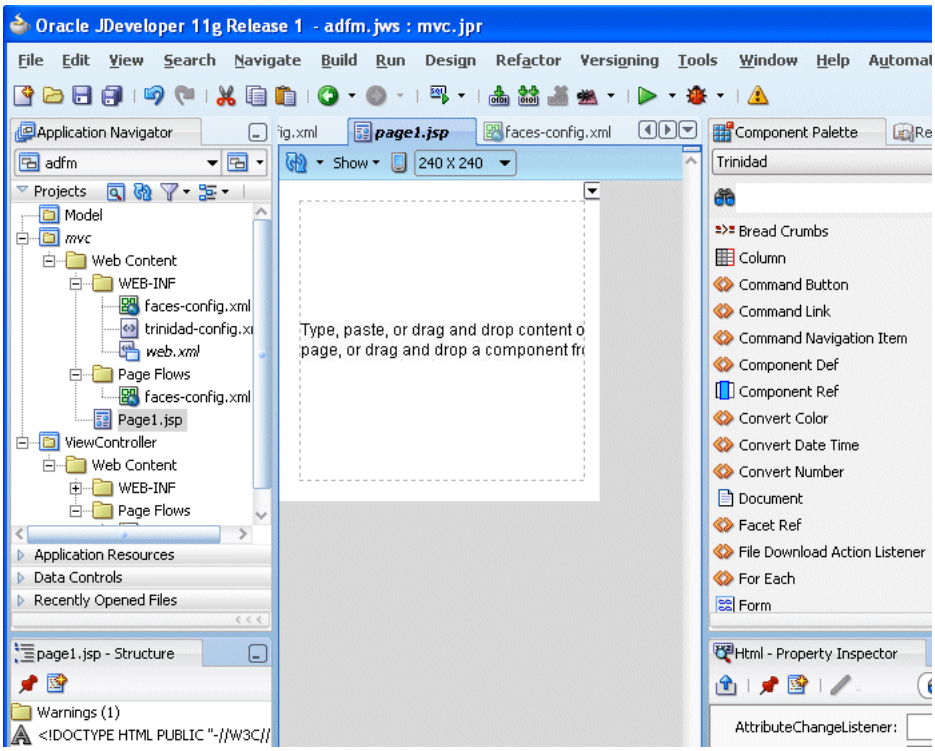


Figure 2–10 shows the designer for a mobile JSP page called *page1.jsp*.

4. From the Components Palette, select the Trinidad components and then create the page using the Apache MyFaces Trinidad components. You can create the page in the same manner as a desktop ADF Web page.

Figure 2–10 Using the Trinidad Component Palette



2.3.2 What Happens When You Create a Mobile JSF Page

Because the **Render in Mobile Device** option is selected by default, the page designer in the visual editor reflects the size of a mobile device, as illustrated in Figure 2–10.

> **Tip:** You change the size of the page in the visual editor or by clicking **Tools**, then **Preferences**, then **Mobile**.

2.4 Testing an ADF Mobile Browser Application

You can test an ADF Mobile browser application on a mobile device, a mobile device emulator, or a desktop browser. Testing on an actual mobile device or mobile device emulator provides more accurate results than does testing on a desktop browser.

Using a desktop browser to test an ADF Mobile browser application produces only approximate results. Desktop browsers provide a fairly uniform testing environment: Web pages appear and behave similarly and business logic executes identically in any type of browser.

Testing an application on an actual mobile device, however, produces more accurate results, because the capabilities of mobile browsers may cause controls to behave differently than they do on a desktop browser. In addition, mobile browsers are usually smaller than desktop browsers. They also render pages differently than desktop browsers because Web servers optimize the look and feel by generating pages that are specific to the mobile browser.

Testing ADF Mobile browser applications directly on mobile devices has limitations as well, in that you may not have access to all of the devices that you must test. Furthermore, firewalls can complicate testing. Many mobile devices can only access the Internet and therefore cannot reach development environments behind a firewall. In such cases, mobile device emulators provide an alternative testing method. For

example, to test applications on BlackBerry or Windows Mobile emulators (Figure 2–11 and Figure 2–13, respectively), first download device emulators from the RIM developer site (`http://na.blackberry.com`) and the Microsoft developer site (`http://www.microsoft.com`). Before you test applications on the emulator, you must first configure the emulator and connect it to the Web server. The Oracle Technology Network (`http://www.oracle.com/technology/tech/wireless`) provides information on downloading and configuring simulators for ADF Mobile browser.

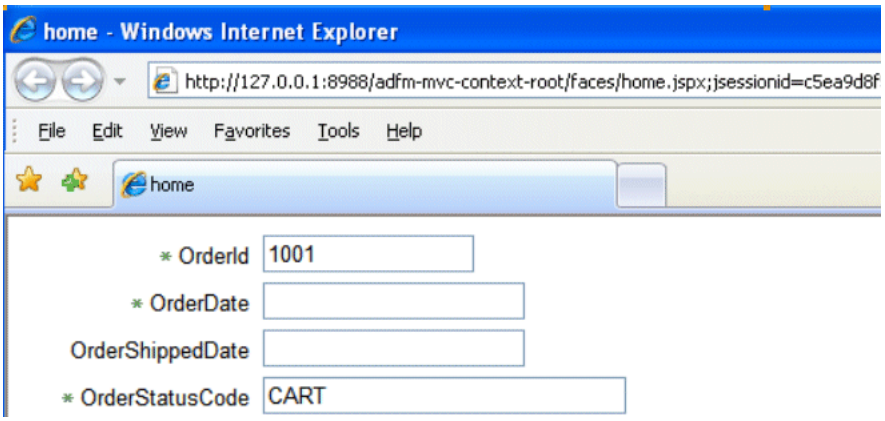
Figure 2–11 Testing an ADF Mobile Browser Application on a BlackBerry Emulator

2.4.1 How to Test ADF Mobile Browser Applications on Emulators

After you test an application on a desktop browser, you can then test it on an emulator. You can use the URL displayed in the desktop browser, but if it uses the localhost IP address (127.0.0.1), you must change it to the network IP address of your computer.

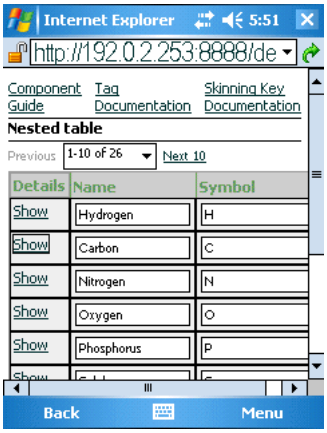
> **Tip:** To obtain the network IP address, use the `ipconfig` command interface on Windows systems and the `ifconfig` command on Linux/UNIX systems.

Figure 2–12 Testing an ADF Mobile Browser Application on a Desktop Browser



For example, to test an application using a Windows Mobile 6 emulator, change the address from the desktop's localhost IP address (*127.0.0.1*, shown in Figure 2–12) to that of the computer's network IP address (*192.0.2.253*, shown in Figure 2–13).

Figure 2–13 Testing an ADF Mobile Browser Application on a Windows Mobile Emulator



In addition, you must remove the session specification that follows the page name. The page name is typically appended with either `.jspx` or `.jsp`. In Figure 2–12, the page name, *home*, is appended with `.jspx`.

In general, you debug an application by repeating cycles of code and then by testing the application. When you test an application that has been modified, you must do one or both of the following:

- Refresh the page.
- Clear the browser's cache.

> **Tip:** Because the URL does not change if you develop the same application, you are not required to enter it again.

2.4.2 What You May Need to Know About Browser Settings

To view ADF Mobile browser applications properly requires adjustments to the browser settings for Windows Mobile and BlackBerry browsers.

Microsoft Windows Mobile 5 and 6, Microsoft Pocket Internet Explorer

For optimal viewing, select the *Fit to Screen* view (accessed by selecting **Menu**, **View** and then **Fit to Screen**).

> **Note:** Do not select the *One Column* view because it causes layout problems.

BlackBerry Browser 4.x

ADF Mobile browser only works if JavaScript support is enabled. To ensure that JavaScript support is enabled:

1. Select **Options** and then **Browser Configuration**.
2. Ensure that the following tables are selected:
   - Support JavaScript
   - Allow JavaScript Popup
   - Support HTML Tables

3 Component Support

This chapter describes the ADF Components that are supported by ADF Mobile browser.

This chapter includes the following sections:

3.1 About Apache My Faces Trinidad Components

ADF Mobile browser supports more than 60 of Apache MyFaces Trinidad components, enabling you to build applications with a rich component set that renders appropriately to the screens of mobile devices. For more information, refer to the Apache MyFaces Trinidad site (`http://myfaces.apache.org/trinidad/`).

3.1.1 Supported Features

ADF Mobile browser supports the following renderer-specific features for the supported browsers:

- Partial Page Rendering
- Dialogs

3.1.2 Partial Page Rendering

The high latency and low bandwidth of networks in mobile environments decrease application responsiveness for mobile users. Screens refresh slowly, diminishing the mobile user experience. ADF Mobile browser's support of Partial Page Rendering (PPR) compensates for the negative impact that slow connections have on screen updates by minimizing the amount of data requested from the server; using PPR, mobile device screen updates do not require a full refresh. Browsers that do not support AJAX (Asynchronous JavaScript and XML) use full page rendering instead of

PPR. For example, a page submission on basic HTML browsers (which do not support JavaScript) results in the refresh of a full page.

Note: Browsers for BlackBerry 4.5 and earlier versions do not support PPR. Specifying the `autosubmit` attribute on certain form components results in the submission of the page after the user exits the field. A full, not partial, refresh of the page then follows.

3.1.3 Dialogs

ADF Mobile browser supports dialogs, pages used by applications to obtain user input. Because mobile browsers cannot open a new window that contains a dialog (a pop-up window), dialogs appear as new pages within the main browser window after automatically preserving the state of the current page.

3.1.4 Rendering Specific to the BlackBerry Browser 4.5 and Earlier Versions

On browsers for BlackBerry 4.5 and earlier versions, the bullets in a list sublevel (such as those in a `tr:panelList` component) appear large and are not indented. The BlackBerry browser's table handling may affect complex layouts; the BlackBerry browser does not allow horizontal scrolling. Instead, it wraps a table row onto multiple display lines which may disturb the layout. For more information, see Chapter 6, "Design Guidelines for BlackBerry 4.2 to 4.5."

3.2 Input Components

ADF Mobile browser supports input text fields and lists, core components that support user input.

3.2.1 Creating Input Text Fields

You can create input fields using the following components:

- `tr:inputColor`

  Note: Mobile browsers do not support an inline `chooseColor` or a `color` dialog for the `tr:inputColor` component.

- `tr:inputDate`
- `tr:inputHidden`
- `tr:inputText`

  Note: Basic HTML browsers do not support the `autosubmit` attribute of the `tr:inputText` component.

  Note: Trinidad optimizes the `tr:inputText` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7, "Narrow Screen Support and User-Agent Details Support."

3.2.2 Creating Lists

You can create lists using the following components:

- `tr:panelChoice`
- `tr:panelList`
- `tr:selectBooleanCheckBox`
- `tr:selectBooleanRadio`
- `tr:selectItem`

> **Note:** Mobile browsers do not support the `disabled` attribute for the `tr:selectItem` component.

- `tr:selectManyCheckBox`
- `tr:selectManyListBox`
- `tr:selectOneChoice`
- `tr:selectOneListBox`
- `tr:selectOneRadio`
- `tr:resetButton`

> **Note:** Basic HTML browsers do not support the `autosubmit` attribute for the `tr:resetButton` component.

3.3 Output Components

ADF Mobile browser uses the Apache MyFacesTrinidad core components that support output on mobile device applications. These components include those for displaying text and images and also components for displaying or hiding text.

3.3.1 Displaying Text

The following components enable you to display text:

- `tr:iterator`
- `tr:message`
- `tr:messages`

> **Note:** When using the `tr:message` and `tr:messages` components, the component-specific messages do not display as they do in a desktop browser. Instead, they display in the region where the message component is placed on the Web page.

- `tr:outputDocument`
- `tr:outputForwarded`
- `tr:outputLabel`
- `tr:outputText`

3.3.2 Displaying Images

The following components enable you to display images:

- `tr:icon`
- `tr:image`
- `tr:panelTip`

3.3.3 Showing (or Hiding) Components

The following components enable showing or hiding items:

- `tr:panelAccordion`

  Note: Mobile browsers only support a full-page update; they do not support the `partialTriggers` attribute of the `tr:panelAccordion` component.

- `tr:panelTabbed`

  Note: To save space on mobile devices, the renderer intentionally prevents the display of tab bars on both the top and bottom of the `tr:panelTabbed` component. Valid values for the attribute positions are `top` and `bottom`. If `both` is specified, then the renderer displays the tabs on top.

- `tr:showDetail`

  Note: For the `tr:showDetail` component, the disclosure arrow does not display; instead [+] and [-] display.

- `tr:showDetailHeader`

  Note: For the `tr:showDetailHeader` component, the disclosure arrow does not appear on mobile browsers.

- `tr:showDetailItem`

  Note: For the `tr:showDetailItem` component, the disclosure arrow does not appear on mobile browsers and `flex` is not supported.

3.4 Layout Components

The layout components supported by ADF Mobile browser include those for managing the page itself (such as `tr:document` and `tr:form`) as well as such components for laying out the sections of a page as `tr:group`, `tr:panelFormLayout`, and `tr:panelGroupLayout`.

3.4.1 Managing the Page

The following components enable you to manage the page:

- `tr:document`
- `tr:form`

  Note: Mobile browsers do not support the `defaultCommand` attribute of the `tr:form` component.

- `tr:page`

  Note: Mobile browsers do not support the `tr:page` facet of the `tr:page` component.

3.4.2 Laying Out Sections of the Page

The following ADF Faces core tags support page layout for mobile device applications:

- `tr:group`
- `tr:panelBorderLayout`

  Note: Only the `top` and `bottom` facets are supported for the `tr:panelBorderLayout` component. Mobile browsers do not support the following facets:

  - `left`
  - `right`
  - `start`
  - `end`
  - `innerLeft`
  - `innerRight`
  - `innerStart`
  - `innerEnd`

  The `tr:panelBorderLayout` component does not render if you use any of these unsupported facets.

- `tr:panelBox`
- `tr:panelFormLayout`
- `tr:panelGroupLayout`
- `tr:panelHeader`
- `tr:panelHorizontalLayout`

  Note: Mobile devices do not support the `halign=end` in the `tr:panelHorizontalLayout` component.

- `tr:panelLabelAndMessage`

  Note: Trinidad optimizes the `tr:panelLabelAndMessage` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Section 7.1, "Determining Narrow Screen Support."

- `tr:panelPage`
- `tr:panelPageHeader`

  Note: Mobile devices only support the following facets of the `tr:panelPageHeader` component:

  - `branding`
  - `brandingApp`
  - `navigation1`
  - `navigation2`

- `tr:panelRadio`

  Note: Trinidad optimizes the `tr:panelRadio` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Section 7.1, "Determining Narrow Screen Support."

- `tr:panelCaptionGroup`

3.4.3 Inserting Spaces

The following components control the space allocation on pages:

- `tr:separator`
- `tr:spacer`
- `tr:subform`

3.5 Navigation Components

ADF Mobile browser supports components as buttons, links, and breadcrumbs that enable users to navigate to other pages of the application or to external locations.

3.5.1 Creating Buttons

ADF Mobile browser supports the following button types:

- `tr:commandButton`

Note: Because the `text` attribute cannot display if the `icon` attribute has been set, buttons on mobile devices can have either text or an image, but not both. If you set the `disabled` attribute to `true`, then the `tr:commandButton` component with an `icon` attribute renders as a static image with no links.

- `tr:goButton`

See Chapter 8, "Extending ADF Mobile Browser Applications" for information on how to use the `tr:goButton` component to integrate e-mail, telephony, and Google maps into an application.

3.5.2 Creating Links

ADF Mobile browser supports the following components for creating hyper-links:

- `tr:commandLink`

Note: Because the `tr:commandLink` component renders as an input element in basic mobile HTML browsers, its child components cannot render. For more information on input elements in basic mobile HTML browsers, see Section 5.2, "Developing Applications for Basic HTML Mobile Browsers."

- `tr:goLink`

See Chapter 8, "Extending ADF Mobile Browser Applications" for information on how to use the `tr:goLink` component to integrate e-mail, telephony, and Google maps into an application.

3.5.3 Navigation Components

ADF Mobile browser supports the following navigation components:

- `tr:breadcrumbs`

Note: Trinidad optimizes the `tr:breadcrumbs` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7.1, "Determining Narrow Screen Support."

- `tr:commandNavigationItem`

Note: `tr:commandNavigationItem` does not render when you set the `disabled` attribute to `true` for the following:

- `tr:selectOneListBox`
- `tr:selectOneChoice`
- `tr:processChoiceBar`
- `tr:navigationPane` with hint, `"choice"`
- `tr:selectRangeChoiceBar`

- `tr:navigationPane`

  Note: `tr:navigationPane hint = "choice"` with a destination value is not supported for basic HTML browsers.

  Note: Trinidad optimizes the `tr:navigationPane` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7.1, "Determining Narrow Screen Support."

- `tr:train`

  Note: The `tr:train` component appears as x of y instead of listing each item. This is a display-only component in ADF Mobile browser; users cannot navigate through the application by clicking the x of y component. To enable navigation, you must add a separate link or button.

- `tr:processChoiceBar`

  Note: Trinidad optimizes the `tr:processChoiceBar` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Section 7.1, "Determining Narrow Screen Support."

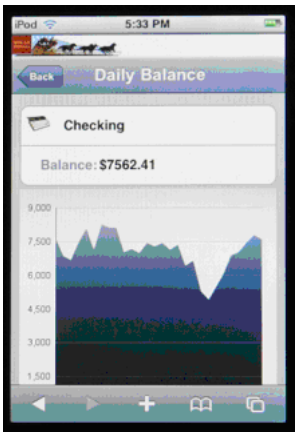
- `tr:selectRangeChoiceBar`

  Note: Trinidad optimizes the `tr:selectRangeChoiceBar` component for narrow-screen devices (that is, devices with screen width measuring less than 240 pixels). For more information see Chapter 7.1, "Determining Narrow Screen Support."

3.6 Data Visualization (Graphs and Gauges)

ADF Mobile browser supports data visualization components (DVTs) used to create a wide variety of graphs and gauges in mobile application pages, such as the area graph representing a user's bank balance in Figure 3–1.

Figure 3–1 Graph Displays in Mobile Applications



ADF Mobile browser supports the following types of graphs:

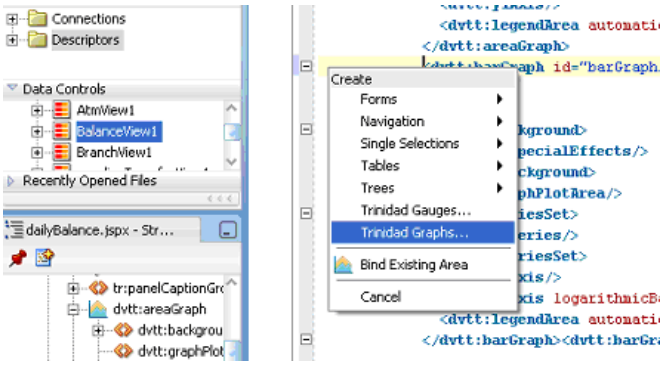
- area
- bar
- bar (horizontal)
- bubble
- combination (horizontal bar and line)
- funnel
- line
- pareto
- pie
- radar
- scattar/polar
- stock

ADF Mobile browser supports the following types of gauges:

- dial
- status meter
- status meter (vertical)
- LED

Because Oracle JDeveloper incorporates DVTs, you can quickly add graphs and gauges. To add these components to an application, first move a data control into the editor window using a drag-and-drop operation and then select **Trinidad Gauges** or **Trinidad Graphs** from the context menu. For example, Figure 3–2 illustrates the context menu that appears when a data control called *BalanceView1* is dragged and dropped into the editor window.

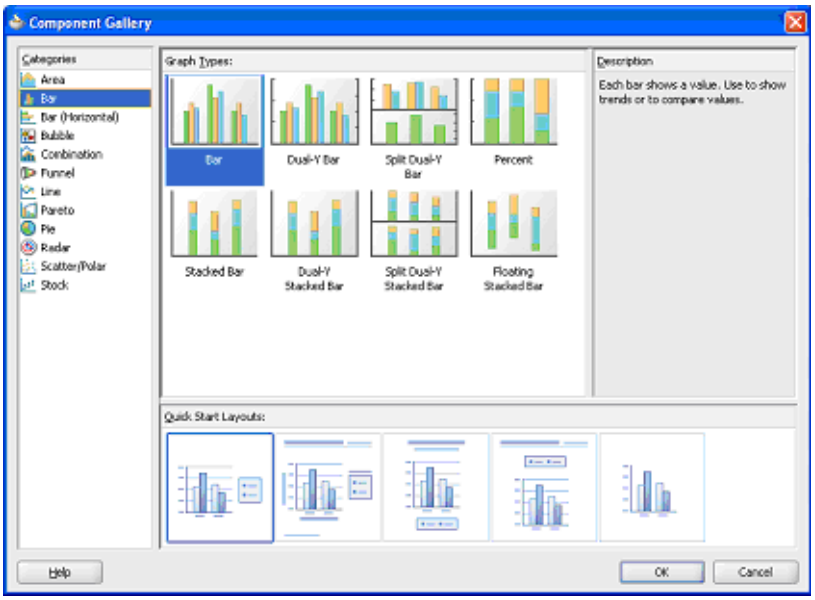
Figure 3–2 Selecting a Data Control



After you select either the **Trinidad Graphs** or **Trinidad Gauges** options, the DVT wizard appears and opens to the Component Gallery page, shown in Figure 3–3. You select the DVT type from this page.

Note: For Oracle Fusion Middleware 11*g* release 1, ADF Mobile browser supports only static graphs and gauges which are rendered as PNG images. Any mobile device that supports this image format can display graphs and gauges.

Figure 3–3 The Component Gallery for the DVT Wizard



3.7 Tables and Trees

ADF Mobile browser applications can display structured data in the rows and columns of a table or hierarchically in and trees.

3.7.1 Creating Tables

ADF Mobile browser supports tables comprised of the following components:

- `tr:table`

Note: ADF Mobile browser does not support the `allDetailsEnabled` attribute for the `tr:table` component; this attribute is always set to `false`.

- `tr:column`

Note: When you nest `tr:column` tags to create column groups, the header facets do not render for the column groups.

3.7.2 Creating Trees

ADF Mobile browser supports the `tr:tree` component.

Note: `tr:tree` may not render on basic HTML browsers.

3.8 Unsupported Components and Attributes

Release 11*g* of ADF Mobile browser does not support some components or attributes.

3.8.1 Unsupported Components

Release 11*g* of ADF Mobile browser does not support the following components:

- `tr:chart`
- `tr:chooseColor`
- `tr:chooseDate`
- `tr:inputFile`
- `tr:inputListOFVariables`
- `tr:inputNumberSpinbox`
- `tr:legend`
- `tr:media`
- `tr:navigationTree`
- `tr:panelButtonBar`
- `tr:panelPopup`
- `tr:panelSideBar`
- `tr:poll`
- `tr:progressIndicator`
- `tr:selectManyShuttle`
- `tr:selectOrderShuttle`
- `tr:singleStepButtonBar`
- `tr:statusIndicator`
- `tr:switcher`
- `tr:treeTable`

3.8.2 Unsupported Attributes

Release 11*g* of ADF Mobile browser does not support the following component attributes on any component.

- `accessKey`
- `shortDesc (tooltip)`

4 Skinning

This chapter describes skinning for ADF Mobile browser applications.

This chapter includes the following sections:

- Section 4.1, "About ADF Mobile Browser Skinning"
- Section 4.2, "Implementing ADF Mobile Browser Skinning"
- Section 4.3, "Example iPhone Components"

4.1 About ADF Mobile Browser Skinning

Skinning enables a page to display consistently on a variety of devices through the automatic delivery of device-dependent style sheets. These style sheets enable optimal display of pages that share the same page definitions on various mobile browsers. Within these style sheets, which enable you to set the look and feel of an application, you not only tailor a component to a specific browser by setting its size, location, and appearance, but you also specify the types of browsers on which components can be displayed or hidden. For more information, see Section 4.2, "Implementing ADF Mobile Browser Skinning." For examples of how to use skinning, see Section 4.3, "Example iPhone Components," which includes an example of an iPhone skin. You can apply a similar style sheet to other mobile browsers, such as BlackBerry, Windows Mobile 6, and Nokia S60. Sample implementations are available from Oracle Technology Network (`www.oracle.com/technology`).

> **Note:** Browsers must support the Cascading Style Sheet (CSS) syntax.

Features supported on specific browsers require means other than customizing style sheets.

4.2 Implementing ADF Mobile Browser Skinning

To create a skin, refer to *Apache Trinidad Skinning* in the *Development Guidelines for Apache MyFaces Trinidad* (`http://myfaces.apache.org/trinidad/devguide/skinning.html`) which includes descriptions on how to:

1. Create a skin (`trinidad-skins.xml`, located in the either the WEB-INF or META-INF directories).
2. Create a style sheet.

3. Set the skin family in `trinidad-config.xml` (located in the WEB-INF directory).

4.2.1 How to Implement Skinning in an ADF Mobile Browser Application

For ADF Mobile browser, you implement skinning by performing the following tasks:

- Within the `trinidad-config.xml` component, define the `<skin-family>` tag with the EL (Expression Language) expression, `#{requestContext.agent.skinFamilyType}`, that returns the skin family type of the browser. See Section 7.2.1.1, "Determining the Skin Type."
- Specify the renderkit and style sheet in `trinidad-skins.xml`
- Include the CSS files within the ADF Mobile browser project

4.2.1.1 How to Define the <skin-family> in trinidad-config.xml

As illustrated in Example 4–1, add the `<skin-family>` tag within the `<trinidad-config>` element and specify an EL expression that evaluates to the string that returns the skin family type of the browser.

Example 4–1 Defining the Skin Family

```
<?xml version="1.0" encoding="windows-1252"?>
<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">
   <skin-family>#{requestContext.agent.skinFamilyType}</skin-family>
         ...
</trinidad-config>
```

4.2.1.2 How to Define <skin-family> in trinidad-config.xml to Enable Switching Between Skins

After you create the skin, you can switch between the default skin and another skin, such as an iPhone skin as illustrated in Example 4–2, using the `<skin-family>` element in `Trinidad-config.xml`. As shown in Figure 4–1, this component, which is located within WEB-INF enables you to set the default skins for an application. To switch between the default skin and an alternate skin, use Expression Language (EL).

To enable switching between skins:

1. Open the `Trinidad-config.xml` file.
2. Define the EL expression in the `<skin-family>` element as illustrated in Example 4–2, which shows switching between the default (`minimal`) and iPhone skins.

Example 4–2 Setting an Alternative Skin

```
<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">
  <skin- family>
      #{requestContext.agent.skinFamilyType == 'iPhonewebkit' ? 'iPhonewebkit':
'minimal'}
  </skin- family>
</trinidad-config>
```

3. Save the file. See also Section 7.2.1.1, "Determining the Skin Type."

4.2.2 How to Specify the Renderkit and Style Sheet Name in trinidad-skins.xml

Under `<skins>`, define the `<skin>` tags that specifies the render-kit-id and style-sheet-name (`org.apache.myfaces.trinidad.desktop` and `iPhone/iPhone.css`, respectively in Example 4–3) for browser types identified in `<family>`. The value of `<family>` is the result string from the EL expression in `<skin-family>` tag in `trinidad-config.xml` (illustrated in Example 4–1). See also Section 7.2.1.1, "Determining the Skin Type."

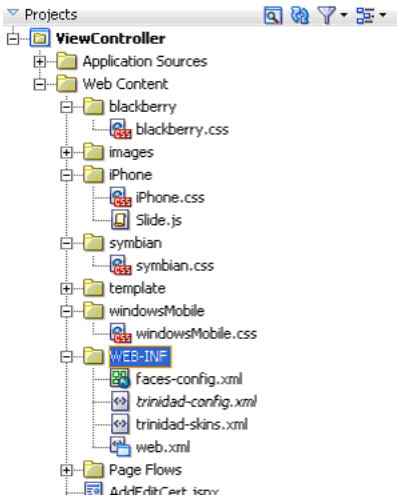
Example 4–3 Defining the Skins

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
<skin>
    <id>iphone</id>
    <family>iPhonewebkit</family>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
     <style-sheet-name> iPhone/iPhone.css </style-sheet-name>
</skin>
<skin>
    <id>symbian</id>
    <family>nokiawebkit</family>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
     <style-sheet-name> symbian/symbian.css </style-sheet-name>
</skin>
<skin>
    <id>windowsMobile</id>
    <family>windowsmobile</family>
    <render-kit-id>org.apache.myfaces.trinidad.pda</render-kit-id>
    <style-sheet-name> windowsMobile/windowsMobile.css </style-sheet-name>
 </skin>
<skin>
```

4.2.3 How to Add the CSS Files to the ADF Mobile Browser Application Project

Include all of the CSS files (such as `blackberry.css` and `iphone.css` in Figure 4–1) in the View-Controller project as specified in `trinidad-skins.xml`.

Figure 4–1 CSS Files in the ADF Mobile Browser Project



4.2.4 What Happens at Runtime

The EL expressions defined within `<skin-family>` returns the skin family type of the browser.

4.3 Example iPhone Components

CSS 3.0 features enables a Web application to have the same look and feel as a native iPhone application. By creating a new skin in Trinidad for iPhone, you can include iPhone-specific components. Examples of these components include:

- Header
- Navigation Panel
- Field Set

These components illustrate how to apply style classes and how to define style classes using the `styleClass` attribute.

4.3.1 How to Create Headers in iPhone Applications

The `backButton`, `toolBar`, `toolBar > h1`, and `button` style classes used with the `<tr:panelHeader>` and `<tr:commandLink>` components set the appearance of the Header (Figure 4–2).

Figure 4–2 The Header Component

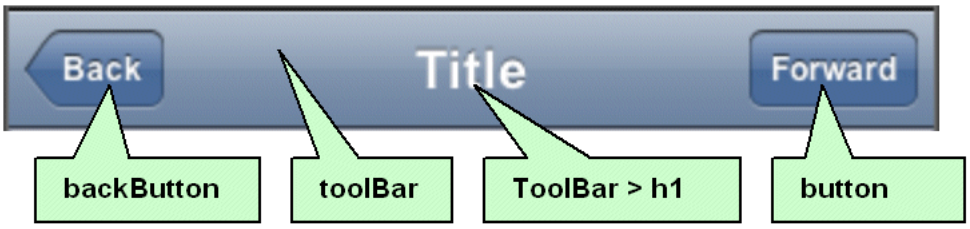


Table 4–1 lists the tags used to build headers, the style classes that you define within them, and the layout effects of these classes.

Table 4–1 Header Component Classes

Tag	Style Class	Layout Effects
`<tr:panelHeader>`	toolbar, toolbar > h1	Sets the height, width, border, and background of the header
`<tr:commandLink>`	backButton	Sets the width, height, color, and position of the back button in the header
`<tr:commandLink>`	button	Sets the width, height, color, and position of the button in the header

toolbar

Example 4–4 illustrates the `toolbar` style class, which sets the height, width, border, and background for the header.

Example 4–4 The toolbar Style Class

```
.toolbar {
    box-sizing: border-box !important;
```

```
   -webkit-box-sizing: border-box !important;
   -moz-box-sizing: border-box !important;
    border-bottom: 1px solid #2d3642 !important;
    border-top: 1px solid #000000 !important;
    padding: 10px !important;
    height: 45px !important;
    background: url(/images/toolbar.png) #6d84a2 repeat-x !important;
    display: block !important;
}
```

toolbar > h1

Example 4–5 illustrates the `toolbar > h1` style class, which sets the height, width, font size, and style of the toolbar title.

Example 4–5 The toolbar > h1 Style Class

```
.toolbar > h1 {
    position: absolute !important;
    overflow: hidden !important;
    left: 50% !important;
    margin: 1px 0 0 -75px !important;
    height: 45px !important;
    font-size: 20px !important;
    width: 150px !important;
    font-weight: bold !important;
    text-shadow: rgba(0, 0, 0, 0.4) 0px -1px 0 !important;
    text-align: center !important;
    text-overflow: ellipsis !important;
    white-space: nowrap !important;
    color: #FFFFFF !important;
    border-bottom: none !important;
}
```

button

Example 4–6 illustrates the `button` style class, which sets the width, height, color, and position of a button in the header.

Example 4–6 The button Style Class

```
.button {
    position: absolute !important;
    overflow: hidden !important;
    top: 8px !important;
    right: 6px !important;
    margin: 0 !important;
    border-width: 0 5px !important;
    padding: 0 3px !important;
    width: auto !important;
    height: 30px !important;
    line-height: 30px !important;
    font-family: inherit !important;
    font-size: 12px !important;
    font-weight: bold !important;
    color: #FFFFFF !important;
    text-shadow: rgba(0, 0, 0, 0.6) 0px -1px 0 !important;
    text-overflow: ellipsis !important;
    text-decoration: none !important;
    white-space: nowrap !important;
```

```
    background: none !important;
   -webkit-border-image: url(/images/toolButton.png) 0 5 0 5 !important;
}
```

backButton

Example 4–7 illustrates the `backbutton` style class, which sets the width, height, color, and position of the back button in the header.

Example 4–7 The backbutton style class

```
.backButton {
    position: absolute !important;
    overflow: hidden !important;
    top: 8px !important;
    left: 6px !important;
    margin: 0 !important;
    height: 30px !important;
    max-width: 45px !important;
    line-height: 30px !important;
    font-family: inherit !important;
    font-size: 12px !important;
    font-weight: bold !important;
    color: #FFFFFF !important !important;
    text-shadow: rgba(0, 0, 0, 0.6) 0px -1px 0 !important;
    text-overflow: ellipsis !important;
    text-decoration: none !important;
    white-space: nowrap !important;
    background: none !important;
   -webkit-border-image: url(/images/toolButton.png) 0 5 0 5 !important;
    padding: 0  !important;
    border-width: 0 8px 0 14px !important;
   -webkit-border-image: url(/images/backButton.png) 0 8 0 14 !important;
}
```

4.3.1.1 Using the styleClass Attribute to Create Header Components

Example 4–8 illustrates how to define the `styleClass` attribute to create the header components.

Example 4–8 Defining the Header Component

```
<tr:panelHeader id = "panelHeader" styleClass="toolbar" text="Title">
    <tr:commandLink styleClass="button"  text="Forward"/>
    <tr:commandLink styleClass="backButton"  text="Back"/>
</tr:panelHeader>
```

4.3.2 How to Create Navigation Panels in iPhone Applications

There are two style classes that define the navigation panel:

- For static lists, use the `Panel List` style class. This style class displays a simple list of navigation items. It sets the width, position, and height of this list.
- For dynamic lists, use the `Table List` style class.

4.3.2.1 Using the Panel List Style Class to Create a Static List of Navigation Panels

You define the `Panel List` style class within a `<tr:panelList>` component, using `<tr:commandLink>` tags for each navigation item as illustrated in Example 4–9.

Example 4–9 Defining a Static List of Navigation Items

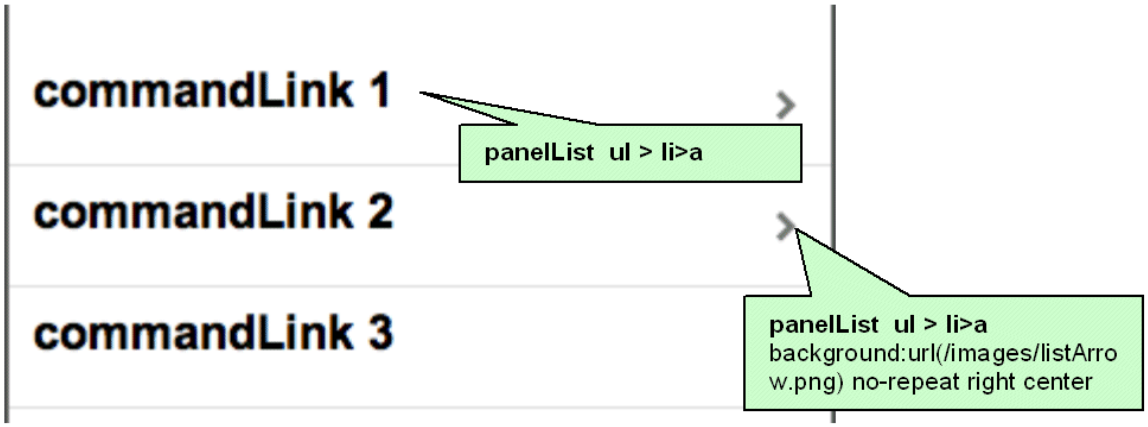
```
<tr:panelList styleClass="panelList">
    <tr:commandLink text="commandLink 1"/>
    <tr:commandLink text="commandLink 2"/>
    <tr:commandLink text="commandLink 3"/>
</tr:panelList>
```

Many CSS features are applied by default on this component when using expressions similar to the ones listed in Table 4–2 on an iPhone skin, as shown in Figure 4–3.

Table 4–2 CSS Expressions

CSS Expression	Layout Effect
panelList ul	Sets the width, position, and height of the list
panelList ul > li	Sets the position and border at the bottom for each item in the list
panelList ul > li > a	Sets the margin, font size, height, and background for each navigation item defined within the `<tr:commandLink>` elements

Figure 4–3 A Static List of Navigation Items



panelList ul

Example 4–10 illustrates the `panelList ul` style class, which sets the width, position, and height of the list.

Example 4–10 The panelList ul Style Class

```
.panelList ul {
    position: absolute !important;
    margin: 0  !important;
    padding: 0 !important;
    left: 0 !important;
    top : 45px !important;
    width: 100% !important;
    min-height: 372px !important;
}
```

panelList ul > li

Example 4–11 illustrates the `panelList ul > li` style class, which sets the position and border at the bottom for each item in the list.

Example 4–11 The panelList ul > li Style Class

```
.panelList ul > li {
    position:relative  !important;
    margin:0  !important;
    border-bottom:1px solid #E0E0E0 !important;
    padding:8px 0 8px 10px  !important;
    list-style:none  !important
}
```

panelList ul > li > a

Example 4–12 illustrates the `panelList ul > li > a` style class, which sets the margin, font size, height, and background for each navigation item.

Example 4–12 The panelList ul > li > a Style Class

```
.panelList ul > li > a {
    display:block  !important;
    margin:-8px 0 -8px -10px  !important;
    padding:8px 32px 8px 10px !important;
    text-decoration:none  !important;
    color:inherit  !important;
    background:url(/images/listArrow.png) no-repeat right center  !important;
    min-height:34px !important;
    font-size:20px;
    font-weight:bold;
 }
```

4.3.2.2 Using the Table List Style Component to Create a Dynamic List of Navigation Items

The `Table List` component enables you to build dynamic tables, such as a table that includes a list of dynamic links as illustrated by Example 4–13. Because the `Table List` component is a table, it includes built-in navigation. Unlike `Panel List`, the `Table List` includes style classes for including images and detailed descriptions below the navigation items, shown in Figure 4–4.

Example 4–13 Building a List of Dynamic Links

```
<tr:table value="#{bindings.EmployeesView15.collectionModel}"
                            var="row"
                            rows="7"
                            width="100%"
                           styleClass = "iphoneTable"
           emptyText="#{bindings.EmployeesView15.viewable ? 'No rows yet.' :
                     id="mainTable" horizontalGridVisible="false"  >
   <tr:column >
      <tr:panelGroupLayout layout="vertical" styleClass="listing">
   <tr:outputText value="#{row.bindings.PhoneNumber.inputValue}"
                                      styleClass="listingDetails"/>
         <tr:commandLink text="#{row.bindings.LastName.inputValue} ,
                                       #{row.bindings.FirstName.inputValue} "
```

```
                                        styleClass="listingLink"
                                        partialSubmit="true"
                                        actionListener = "#{agentUtil.gotoPage2}"
                                        id="myLink1"
                                        disabled="#{!bindings.Execute.enabled}"
                                        onclick='iPhone.slideFragments("page2",
"page1")'>
         </tr:commandLink>
         <tr:image styleClass="listingImage"
                                    source="/images/326425649.png"/>
      </tr:panelGroupLayout>
   </tr:column>
</tr:table>
```

To create a table of dynamic links:

1. Create a Trinidad read-only table using data control.
2. Set the `styleClass` attribute for the table as `iphoneTable`.

   The expressions listed in Table 4–3 apply the needed iPhone-related CSS properties when you set the `styleClass` as `iPhoneTable`.

Table 4–3 CSS Expression

Expression	Layout Effects
`.iphoneTable .af_table_content`	Sets the background color for the table content. It overrides the table's default outer-border style to none.
`.iphoneTable .af_table_control-bar-top`	Sets the background color for the table controller (pagination)
`.iphoneTable .af_column_cell-text`	Sets the background color of the column

3. Set the width of the table to 100.
4. Set the `horizontalGridVisible` attribute to false.

> **Note:** There must be only one column within the `<tr:table>` tag. Within this column, all tags must be wrapped by a `<tr:panelGroupLayout>` component with a `styleClass` set as `listing`.

Table 4–4 lists the style classes used within the subelements of the `<column>` tag.

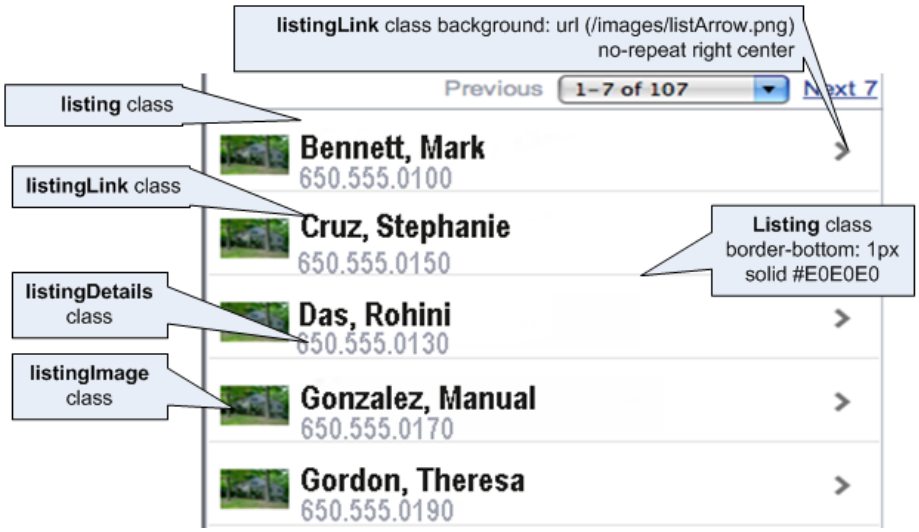
Table 4–4 Table Listing Style Classes

Element	Style Class	Layout Effects
`<tr:panelGroupLayout>` with layout attribute as vertical	listing	Sets the position and the border for each row
`<tr:panelList>`	listingImage	Sets the width, position, and height of the image
`<tr:commandLink>` : (navigation items)	listingLink	Sets the position, height, font size, text alignment, background image, and color of the navigation item

Table 4–4 (Cont.) Table Listing Style Classes

Element	Style Class	Layout Effects
`<tr:outputText>` : (description of the navigation)	listingDetails	Sets the position, height, font size, text alignment, background image, and color of the navigation description

Figure 4–4 A Listing of Dynamic Links



listing

Example 4–14 illustrates the `listing` style class, which sets the position and the border for each row.

Example 4–14 The listing StyleClass

```
.listing {
    position: relative !important;
    margin: 0 !important;
    border-bottom: 1px solid #E0E0E0 !important;
    padding: 8px 0 8px 10px !important;
    font-size: 20px !important;
    font-weight: bold !important;
    list-style: none !important;
}
```

listingLink

Example 4–15 illustrates the `listingLink` style class, which sets the width, position, and height of the image.

Example 4–15 The listingLink StyleClass

```
.listingLink  {
    display: block !important;
    margin: -8px 0 -8px -10px !important;
    padding: 8px 32px 8px 10px !important;
    text-decoration: none !important;
```

```
    color: inherit !important;
    background: url(/images/listArrow.png) no-repeat right center !important ;
    padding-left: 54px !important;
    padding-right: 40px !important;
    min-height: 34px !important;
    font-size: 20px !important;
    font-weight: bold !important;
}
```

listingDetails

Example 4–16 illustrates the `listingDetails` style class, which sets the position, height, font size, text alignment, background image, and color of the navigation item.

Example 4–16 The listingDetails StyleClass

```
.listingDetails {
    display: block !important;
    position: absolute !important;
    margin: 0 !important;
    left: 54px !important;
    top: 27px !important;
    text-align: left !important;
    font-size: 12px !important;
    font-weight: normal !important;
    color: #666666 !important;
    text-decoration: none !important;
    height: 13px !important;
    padding: 3px 0 0 0 !important;

}
```

listingImage

Example 4–17 illustrates the `listingImage` style class, which sets the position, height, font size, text alignment, background image, and color of the navigation description.

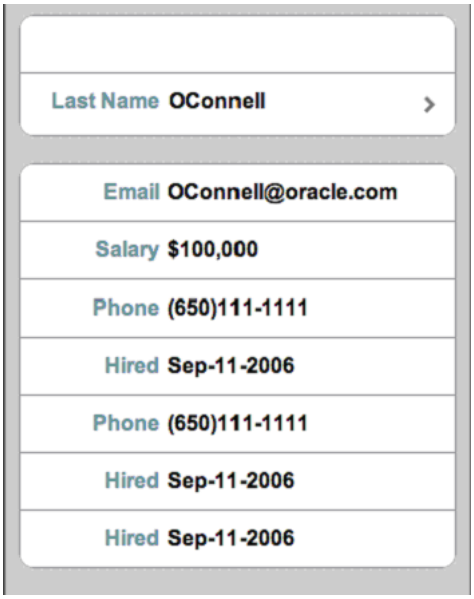
Example 4–17 The listingImage Style Class

```
.listingImage  {
    display: block !important;
    position: absolute !important;
    margin: 0 !important;
    left: 6px !important;
    top: 7px !important;
    width: 35px !important;
    height: 27px !important;
    padding: 7px 0 10px 0 !important;
}
```

4.3.3 How to Create Detail Items in iPhone Applications

On the destination page, this component displays the detail of an item selected through panel navigation. As illustrated in Figure 4–5, these details include salary, phone numbers, and a hire date for a selected employee.

Figure 4–5 Field Set



The `Destination Page - Field Set` component contains one or more rows where each row contains a label or a message (which can be simple text or another navigation item). As illustrated in Example 4–18, you use the `<div>` tags to create these rows. The `<div>` tags are subelements of a `<tr:panelCaptionGroup>` component.

Example 4–18 Creating a Field Set

```
<div  class="panelBase">
       <tr:panelCaptionGroup>
            <div class="row">
                 <tr:outputText styleClass="labeltext" value="#{agentUtil.name}"
                 truncateAt="0"/>
                 <tr:outputText styleClass="messageText"
                   value="#{sessionScope.FirstName}" />
             </div>
              <div class="row">
                 <tr:outputText styleClass="labeltext" value="Last Name"/>

                 <tr:commandLink text="#{sessionScope.LastName}"
                                 styleClass="messageLink"
                                 partialSubmit="true"
                                 id="myLink2"
                                 actionListener="#{agentUtil.gotoPage3}"
                                 onclick='iPhone.slideFragments("page3",
"page2");'
                                 />
                 </div>
          </tr:panelCaptionGroup>

           <tr:panelCaptionGroup>
                 <div class="row">
                     <tr:outputText styleClass="labeltext" value="Email"/>
                       <tr:outputText styleClass="messageText"
                                value="#{bindings.LastName}@oracle.com"/>
                   </div>
```

```
          <div class="row">
                <tr:outputText styleClass="labeltext" value="Salary"/>
                <tr:outputText styleClass="messageText" }"
                                    value="#{sessionScope.Salary}"/>
              </div>
            <div class="row">
                <tr:outputText styleClass="labeltext" value="Phone"
                  truncateAt="5"/>
                <tr:outputText styleClass="messageText"
                value="#{sessionScope.PhoneId}"/>
            </div>
            <div class="row">
                <tr:outputText styleClass="labeltext" value="Hired"
                truncateAt="7"/>
                <tr:outputText styleClass="messageText"
                value="#{sessionScope.HireDate}"/>
            </div>
            <div class="row">
               <tr:outputText styleClass="labeltext" value="Phone"
                truncateAt="5"/>
                <tr:outputText styleClass="messageText"
                  value="#{sessionScope.PhoneId}"/>
             </div>
            <div class="row">
                <tr:outputText styleClass="labeltext" value="Hired"
                truncateAt="7"/>
                <tr:outputText styleClass="messageText"
                value="#{sessionScope.HireDate}"/>
            </div>
             <div class="row">
                <tr:outputText styleClass="labeltext" value="Hired"
                truncateAt="7"/>
                <tr:outputText styleClass="messageText"
                value="#{sessionScope.HireDate}"/>
            </div>
        </tr:panelCaptionGroup>

    </div>
```

To create field set components:

1. Insert as many `<div>` tags as needed within a `<tr:panelCaptionGroup>` component (illustrated in Example 4–18).
2. To create rows, define each `<div>` tag with the row class attribute. For example:

   ```
   <div class="row">
   ```

   The `row` attribute sets the position, height, and border for each row.
3. Within each `<div>` tag, create a label element as follows:
   a. Create a `<tr:outputText>` tag.
   b. Set the position, width, font, and color of the label element by defining the `StyleClass` as labeltext.

   For example:

   ```
   <tr:outputText styleClass="labeltext" value="Phone"
                       truncateAt="5"/>
   ```

4. Create a message element using either the `<tr:outputText>` tag or the `<tr:commandLink>` component as follows:

   - The `<tr:outputText>` component with `styleClass` set as `messageText`. For example:

     ```
     <tr:outputText styleClass="messageText"
                    value="#{sessionScope.PhoneId}"/>
     ```

     The messageText style class sets the position, width, font, and color for the label element.

   - Example 4–19 illustrates the `<tr:commandLink>` component with `styleClass` set as `messageLink`.

Example 4–19 Setting the styleClass Attribute as messageLink

```
<tr:commandLink text="#{sessionScope.LastName}"
   styleClass="messageLink"
   partialSubmit="true"
   id="myLink2"
   actionListener="#{agentUtil.gotoPage3}"
   onclick='iPhone.slideFragments("page3", "page2");'
   />
```

The messageLink element sets the position, width, font, height, and color for the message element.

5. For a panel base background, wrap the `<div>` tags with the panelBase class attribute (illustrated in Example 4–18).

Note: The panelBase fieldset sets rounded edges. The `fieldset` element is added by the renderer for the `<tr:panelCaptionGroup>` component.

4.3.3.1 Field Set Style Classes

This section lists the style classes for field set components and their layout properties.

labeltext

Example 4–19 illustrates the `labeltext` style class, which sets the position, width, font, and color of the label element

Example 4–20 The labeltext Style Class

```
.labeltext {
    position: absolute !important;
    margin: 0 0 0 14px !important;
    line-height: 42px !important;
    font-weight: bold !important;
    color: #7388a5 !important;
    text-align: right !important;
    width: 90px !important;
    white-space: nowrap !important;
}
```

messageText

Example 4–21 illustrates the `messageText` style class, which sets the position, width, font, and color for the message element.

Example 4–21 The messageText Style Class

```
.messageText {
    display: block !important;
    margin: 0 !important;
    border: none !important;
    padding: 12px 10px 0 110px !important;
    text-align: left !important;
    font-weight: bold !important;
    text-decoration: inherit !important;
    height: 42px !important;
    color: inherit !important;
    box-sizing: border-box !important;
   -webkit-box-sizing: border-box !important;
}
```

messageLink

```
.messageLink  {
    display: block !important;
    text-align: left !important;
    text-decoration: none !important;
    color: inherit !important;
    background: url(/images/listArrow.png) no-repeat right center !important ;
    padding-top: 12px !important;
    padding-left: 111px !important;
    padding-right: 40px !important;
    min-height: 34px !important;
    font-size: 16px !important;
    font-weight: bold !important;
}
```

panelBase

Example 4–22 illustrates the `panelBase` style class, which sets the background of the panel base.

Example 4–22 The panelBase Style Class

```
.panelBase {
    box-sizing: border-box !important;
   -webkit-box-sizing: border-box !important;
    padding: 10px !important;
    background: #c8c8c8 url(/images/pinstripes.png) !important;
}
```

panelBase fieldset

Example 4–23 illustrates the `panelBase fieldset` style class, which sets rounded edges. The `<fieldSet>` element is rendered by the renderer for the `<tr:panelCaptionGroup>` component.

Example 4–23 The panelBase fieldset Style Class

```
.panelBase fieldset {
    position: relative;
    margin: 0 0 20px 0;
    padding: 0;
    background: #FFFFFF;
   -webkit-border-radius: 10px;
    border: 1px solid #999999;
    text-align: right;
    font-size: 16px;
}
```

row

Example 4–24 illustrates the `row` style class, which sets the position, height, and border for each row.

Example 4–24 The row Style Class

```
.row {
    position: relative !important;
    min-height: 42px !important;
    border-top: 1px solid #999999 !important;
   -webkit-border-radius: 0 !important;
    text-align: right !important;
}
```

row:first-child

Example 4–25 illustrates the `row:first-child` style class.

Example 4–25 The row:first-child style class

```
.row:first-child {
    border-top: none !important;

}
```

4.3.4 What You May Need to Know About CSS Classes in iPhone Applications

Although you manually apply most of the CSS classes to specific components using the `styleClass` attribute (as in Example 4–8), some CSS features are applied by default when you use the iPhone skin.

5 Supporting Basic HTML Mobile Browsers

This chapter describes ADF Mobile browser's support for basic HTML mobile browsers.

This chapter includes the following sections:

- Section 5.1, "About Basic HTML Mobile Browser Support"
- Section 5.2, "Developing Applications for Basic HTML Mobile Browsers"
- Section 5.3, "Styling Basic HTML Mobile Browsers"

5.1 About Basic HTML Mobile Browser Support

For Oracle Fusion Middleware 11*g* release 1, ADF Mobile browser supports mobile browsers that do not provide support for JavaScript. This lack of JavaScript support makes basic HTML mobile browsers less robust than such supported browsers as the BlackBerry or Apple iPhone. Aside from the browsers listed in Section 1.2, "Supported Mobile Browsers," ADF Mobile browser considers most common browsers as basic HTML mobile browsers. ADF Mobile browser may not recognize certain mobile browsers, however.

5.1.1 Requirements for Basic HTML Mobile Browser Support

The minimum requirement for ADF Mobile brower's support is XHTML Basic or the XHTML Mobile Profile that includes WAP2.x browsers.

> **Note:** ADF Mobile browser does not support WAP1.x browsers that do not support XHTML Basic or the XHTML Mobile Profile.

5.2 Developing Applications for Basic HTML Mobile Browsers

Because the ADF Mobile browser framework serves pages to mobile browsers that are appropriate to a browser's capabilities or limitations, you do not have to create user interfaces that are specific to basic HTML mobile browsers. However, the absence of JavaScript support by these browsers limits the functionality of certain HTML elements.

- Basic HTML mobile browsers do not support the `autosubmit` attribute. Add a submit button to the form only if the form submission responds to a component's autosubmit feature. For composite components with built-in autosubmit features, ADF Mobile browser adds a submit button to enable users to submit the form.

- Basic HTML mobile browsers do not support form-submitting links. All submitting elements are rendered as buttons. Basic HTML mobile browsers do not support the child components of such input elements. As a consequence, the child components of the `tr:commandLink` component cannot render in a basic HTML mobile browser. For more information on `tr:commandLink`, see Section 3.5.2, "Creating Links."

5.3 Styling Basic HTML Mobile Browsers

ADF Mobile browser provides basic CSS support for basic HTML mobile browsers. While most of these browsers support CSS, ADF Mobile browser applications can still run on the browsers that do not support CSS. However, without design time considerations, the user interface may be difficult to use. Test the ADF Mobile browser application on as many browsers as possible.

6

Design Guidelines for BlackBerry 4.2 to 4.5

This chapter describes how to accommodate the behavior of BlackBerry browsers 4.2 to 4.5.

This chapter includes the following sections:

6.1 About BlackBerry Browser Display Behavior

The BlackBerry browser behaves differently than many other browsers in that it does not display pages using horizontal scrolling. Instead, it fits a page to the width of the screen. This chapter presents guidelines to help you format pages to display properly on BlackBerry smartphones.

6.2 Formatting Tables to Prevent Wrapping

When formatting tables, avoid long words on lines that contain multiple fields because browsers wrap long words between fields.

> **Note:** Within this chapter, a word refers to a series of characters. In this context, a word does not include white space.

Because the default mode of the BlackBerry browser limits the browser width to that of the physical screen, any field that does not fit in a line is displayed on the next line. If the intent of an application is to display multiple elements in one line, then you must ensure that the total width of the fields are within the width of the browser. Like other browsers, the BlackBerry browser wraps multiple lines when needed. The column width cannot be reduced beyond the size of the longest word in the field.

6.2.1 How to Prevent Fields from Wrapping in Tables

To prevent fields from wrapping, ensure that the total of the size attribute values in a table's row satisfies the following formula when all of the fields in a row are input fields.

```
3*Number of columns + the Sum of the size attributes in all columns <=X, when X=48
```

In general, field sizes in table columns should satisfy the following formula:

```
3 * Number of Columns +
Sum of size attributes in all input field columns +
Sum of number of characters in longest words in all output field columns <=  X,
when X=48
```

If the fields still wraps, decrease the value of X until it fits.

6.3 Formatting Label and Message Panels

To preserve the intended programming flexibility, `nowrap` attributes are supported and inserted when they are explicitly programmed for the Trinidad component. You may encounter problems if you add `nowrap` to a component definition when you program pages.

6.4 Formatting Column Width

When formatting columns, set the percentage width specification for both the label and the field in the `tr:panelFormLayout` component so that the total width is at 100%.

6.5 What You May Need to Know About Display Variations on BlackBerry Smartphones

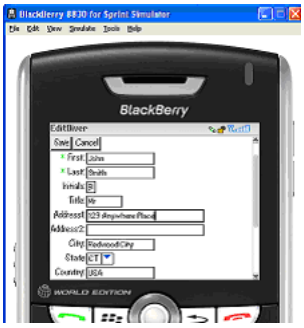
This section describes how the same application can display differently on different smartphones. This section includes the following topics:

- Changing the Minimum Font Size
- Form Factor Variations

6.5.1 Changing the Minimum Font Size

Changing the minimum font size through user preferences affects the formatting ability of the ADF Mobile browser renderer. For example, input fields and their corresponding labels align properly when the font is set to its default size of 6 pt, as shown in Figure 6–1.

Figure 6–1 Application Display Using the Default Font Size of 6 pt.



However, increasing the font size to 10 pt. disrupts the display by shifting the input fields beneath their corresponding labels. As a result, the page is difficult to read.

Figure 6–2 shows a page that is too large for the display screen.

Figure 6–2 Increasing the Font Size

6.5.2 Form Factor Variations

Differing screen sizes can affect display. Even if the font size is at the default of 6 pt (illustrated in Figure 6–1), the same application appears differently on different devices. In Figure 6–3, the input fields barely fit the device's screen, even though they are easily accommodated on other devices running the same application as shown in Figure 6–1.

Figure 6–3 Difficulty Displaying Input Fields and Labels with Font Size at 6 pt.

In addition, input fields may display properly on the screen of one device, but may appear crowded on the screen of another type of device.

Figure 6–4 shows an application whose table cells are not wide enough to accomodate the text, causing it to wrap.

Figure 6–4 Wrapping Text

7

Narrow Screen Support and User-Agent Details Support

This chapter describes how the Trinidad infrastructure determines narrow screen support and how it uses EL expressions to expose user-agent details.

This chapter includes the following sections:

7.1 Determining Narrow Screen Support

Mobile devices come with a wide range of screen widths. As a result, the UI components of a web application may render properly on a device with a screen width measuring 240 pixels, but not align correctly when the application runs on a device that has a screen width of only 100 pixels. In such a situation, Trinidad optimizes its rendering for narrow-screen devices. Trinidad considers any device with a screen width of less than 240 pixels as a narrow screen and optimizes the rendering for the following components accordingly:

- `tr:breadcrumbs`
- `tr:inputText`
- `tr:navigationPane`
- `tr:panelFormLayout`
- `tr:panelLabelAndMessage`
- `tr:panelRadio`
- `tr:processChoiceBar`
- `tr:selectRangeChoiceBar`

7.1.1 How Trinidad Determines Narrow-Screen Optimization

Because Trinidad only considers a device with a screen width that measures less than 240 pixels as a narrow screen, it does not consider iPods (Safari browsers) or BlackBerry smartphones (BlackBerry browsers), which usually have screens that are greater than 240 pixels, as such. For a Windows Mobile browser, Trinidad determines the screen width from the UA-pixels request header and only applies narrow screen optimization if the screen-width is less than 240 pixels. For all other user agents, however, Trinidad optimizes its rendering for a narrow screen device.

7.2 Determining User-Agent Capabilities Using EL Expressions

Trinidad exposes a requesting user-agent's details to developers using the EL expression, `#{requestContext.agent}`, which returns an `agent` object that describes the requesting user agent. By adding the detail name or capability name properties to this expression, you enable Trinidad to return details that include the user-agent's name, version, platform, the version of the platform, the model (which is applicable only to BlackBerry), and the browser's support for JavaScript and PPR (Partial Page Rendering). For information on exposing user-agent details, see Section 7.2.1, "How To Determine User-Agent Details." For information on determing browser capabilities, see Section 7.2.2, "How to Determine Browser Capabilities."

7.2.1 How To Determine User-Agent Details

When Trinidad receives a request, it parses user-agent strings for a variety of user-agent details (listed in Table 7–1) that include type, the name and version of the agent, and the agent's platform name and platform version. Trinidad uses the EL expression `#{requestContext.agent.<detail-name>}` to expose these details to developers. For example, to enable developers to retrieve the category appropriate to the user-agent type (that is, desktop for a desktop browser or PDA for mobile browsers), Trinidad uses the `type` detail in the EL expression as follows:

```
#{requestContext.agent.type}
```

Note: Trinidad may return a null value for such details as `PlatformName`, `PlatformVersion` if it cannot parse them from the user-agent string.

Table 7–1 Browser Details Exposed through EL Expressions

Detail Name	Description
`type`	Identifies a user-agent type. For desktop and mobile browsers, the values are *desktop* and *PDA*, respectively. Because Safari provides all desktop browser features when it runs in a mobile device, the agent object exposes this detail as a desktop type.
`agentName`	The name of the agent
`agentVersion`	The version of the agent
`platformName`	The platform on which the agent runs
`platformVersion`	The version of the platform on which the agent runs.
`hardwareMakeModel`	The model of the mobile device
`skinFamilyType`	Trinidad categorizes the mobile browsers into different skin types based on their CSS capabilties. For more information, see Section 7.2.1.1, "Determining the Skin Type."

7.2.1.1 Determining the Skin Type

Trinidad categorizes incoming user-agents into different skin family types based on the CSS support and exposes the skin family type to developers using the `#{requestContext.agent.skinFamilyType}` EL expression. For example, for a Safari user-agent running in a Windows platform, Trinidad uses this EL expression to provide developers with the value of `windowswebkit`. For Safari browsers running on Symbian devices, this expression returns the Nokia Webkit (`nokiawebkit`). Table 7–2 lists the skin family types returned by

`#{requestContext.agent.skinFamilyType}` according to user-agent, platform, and platform version.

Table 7–2 Skin Family Types Returned by the SkinFamilyType Attribute

User-Agent	Platform	Skin Family Type
	Windows mobile	`windowsmobile`
Safari	iPhone/iPod	`iPhonewebkit`
Safari	linux	`linuxwebkit`
Safari	Macintosh	`macwebkit`
Safari	Symbian	`nokiawebkit`
Safari	Windows	`windowswebkit`
Safari	Unknown platforms	`defaultwebkit`
Blackberry		`blackberryminimal`
Blackberry (versions 4.5 and higher)		`blackberry`
All other mobile browsers		`genericpda`

7.2.2 How to Determine Browser Capabilities

Trinidad sends its response to a user-agent's request based on capabilities it assigns to a user agent. These capabilities include a user-agent's support for JavaScript, PPR, and so on. Some of these capabilities (listed in Table 7–3) are exposed to developers through the EL expression `#{requestContext.agent.capabilities}`.

Use the EL expression `#{requestContext.agent.capabilities.<capability-name>}` to determine the specific capability assigned to a user-agent by Trinidad. For example, to determine whether Trinidad assigns JavaScript capability to a user agent, use the following EL expression:

```
# {requestContext.agent.capabilities.scriptingSpeed!='none'}.
```

Table 7–3 Browser Capabilities Exposed through EL Expressions

Capability Name	Detail
`narrowScreen`	Indicates whether Trinidad optimizes is rendering for a narrow-screen device. It returns `true` (a boolean type) if Trinidad optimizes its rendering for a narrow-screen device.
`scriptingSpeed`	Indicates JavaScript support for a user-agent. Returns `"none"` (a String type) if the user-agent does not support JavaScript.
`partialRendering`	Indicates PPR support for a user-agent. Returns `true` (a boolean type) if the browser supports PPR.

8

Extending ADF Mobile Browser Applications

This chapter describes how to add functionality to ADF Mobile browser applications.

This chapter includes the following sections:

8.1 Introduction to Extending Applications for E-Mail, Telephony, and Google Maps

In addition to using style sheets described in Chapter 4, "Skinning", you can further tailor an ADF Mobile browser application to include support for e-mail, telephony, and Google Maps. This chapter, through the following sections, describes how to use the `tr:goButton` and `tr:goLink` components to integrate links to phone numbers, e-mail addresses, and Google Maps into ADF Mobile browser applications.

- "Integrating an E-Mail Client"
- "Integrating Telephony"
- "Integrating Google Maps"

8.2 Integrating an E-Mail Client

To invoke an e-mail application from a Web application:

1. Use either the `tr:goButton` or the `tr:goLink` components.
2. Prepend the `mailto:` protocol in an HTML link.
3. Set the `destination` property to the HTML link (represented as the Expression Language statement `#{sessionScope.empDetails.Email}` in Example 8–1).

Example 8–1 Integrating the iPhone E-Mail Client using the mailto: Protocol

```
<tr:goLink styleClass="messageText"
                                text="#{sessionScope.empDetails.Email}"

destination="mailto:#{sessionScope.empDetails.Email}"/>
```

8.2.1 Adding Mail Properties

The `mailto:` protocol enables you to add the mail properties that are listed in Table 8–1.

Table 8–1 Mail Properties

Property	Syntax
Multiple Recipients	A comma (,) separating each e-mail address
Message Subject	`subject =<subject text>`
cc Recipients	`cc=<name@address.com>`
bcc Recipients	`bcc=<name@address.com>`
Message Text	`body=<Message Text>`

To specify these properties, append the e-mail address with question mark (`?`) as illustrated by `#{sessionScope.empDetails.Email}?` in Example 8–2 and then add the properties, separating each with an ampersand (`&`).

Example 8–2 Adding E-Mail Properties

```
<tr:goLink styleClass="messageText"
                 text="#{sessionScope.empDetails.Email}"
destination="mailto:#{sessionScope.empDetails.Email}?subject=howdy&cc=myboss@oracl
e.com&bcc=me@oracle.com&body=howdy partner!"/>
```

8.3 Integrating Telephony

To invoke a call dialog box for a phone number displayed in the application, prepend the phone number with the `tel:` protocol in an HTML link.

> **Note:** The phone number must support the portion of the RFC 2806 protocol (`http://www.ietf.org/rfc/rfc2806.txt`) which enables you to add pauses or dial extensions after a user dials the primary phone number. Because Apple does not specify which portions of RFC 2086 that it supports, you must test each portion.

In Example 8–3, the EL expression, `#{sessionScope.empDetails.PhoneNumber}` represents the phone number.

Example 8–3 Enabling the Call Dialog Box

```
<tr:goLink styleClass="messageText"
                                  text="#{sessionScope.empDetails.PhoneNumber}"
destination="tel:#{ sessionScope.empDetails.PhoneNumber}"/>
```

8.4 Integrating Google Maps

To create a link that displays a map that shows the data available in the application, specifying the `destination` property of the `tr:goLink` component as follows:

1. Define `destination=` as the URL of Google Maps. (`destination=http://maps.google.com/maps`, as illustrated in Example 8–4.)
2. To search for a location, append the Google Maps URL with `?q=`.
3. Define `q=` using the address string of the target location. This value can be a full street address, a city, landmark, or any item that Google Maps can search and locate. If multiple items are found, Google Maps drops multiple pins automatically.

> **Note:** The address string must be well formatted, including commas between words. Also, replace spaces with plus sign (+) characters.

Example 8–4 illustrates how to define the `tr:goLink` component to invoke a Google Maps application and then drop a pin on 200 Oracle Parkway.

Example 8–4 Specifying Locations in Google Maps

```
<tr:goLink styleClass="messageAddrText"
                                    text="200 Oracle Parkway, Redwood City, CA, USA"
destination="http://maps.google.com/maps?q=200+Oracle+Parkway,+Redwood+City,+CA,+U
SA"/>
```

Example 8–5 illustrates specifying a location using an address represented by EL expressions.

Example 8–5 Specifying Locations in Google Maps Using EL Expressions

```
<tr:goLink styleClass="messageAddrText"
                                    text=" #{sessionScope.empDetails.StreetAddress},
#{sessionScope.empDetails.City}, #{sessionScope.empDetails.StateProvince},
#{sessionScope.empDetails.CountryName}"
destination="
http://maps.google.com/maps?q=#{sessionScope.empDetails.StreetAddress},+#{sessionS
cope.empDetails.City},+#{sessionScope.empDetails.StateProvince},+#{sessionScope.em
pDetails.CountryName}"/>
```

The address string, such as the one in Example 8–4, must be have plus sign (+) characters rather than spaces.

8.4.1 Programming Driving Directions

Google Maps also supports driving directions. Modify the string following the question mark (?) in the Google Maps URL with the starting and destination addresses (`saddr=<starting address>&daddr=<destination address>`). Using this format, the directions from Oracle headquarters at 200 Oracle Parkway in Redwood City to Oracle's San Francisco office at 1 Front Street in San Francisco are as follows:

```
http://maps.google.com/maps?saddr=200+Oracle+Parkway,+Redwood+City,+CA,+USA&
daddr=1+Front+Street,+San+Francisco,+CA,+USA
```

Note: Apple and Google have not yet published other APIs.

8.4.2 Supporting Google Maps on iPhone

iPhone Safari supports both Google Maps and YouTube applications in that it automatically intercepts certain URL calls and invokes a native application rather than opening the URL using the target Web site. For example, when a user clicks an HTML link to Google Maps (`http://maps.google.com`), Safari invokes a native Google Maps application rather than navigating to the Google Maps Web site. Because the native Google maps application accepts some URL parameters supported by `maps.google.com`, users can specify a location and drop a pin.

8.5 What You May Need to Know About Page Display Dimensions

To control the correct zoom ratio, add a `viewport` meta tag in the header of a page. The `viewport` is a device-specific meta tag used to ensure that a page displays in the correct scale. Example 8–6, illustrates setting the viewports for both iPhones and BlackBerry smartphones. For more information on using the `viewport` specification, see `http://developer.apple.com/`.

Example 8–6 Setting Viewports

```
<trh:head title="Online Banking Demo">
         <meta http-equiv="Content-Type"
               content="text/html; charset=windows-1252"/>
         <f:verbatim rendered="#{requestContext.agent.skinFamilyType eq
 'blackberry'}">
           <meta name="viewport"
                 content="width=device-width; height=device-height;
 initial-scale=1.0; maximum-scale=1.0; user-scalable=0;"/>
         </f:verbatim>
         <f:verbatim rendered="#{requestContext.agent.skinFamilyType eq
 'iPhonewebkit'}">
           <meta name="viewport"
                 content="width=device-width; initial-scale=1.0;
 maximum-scale=1.0; user-scalable=0;"/>
         </f:verbatim>
       </trh:head>
```

Note: Versions 4.6 and later of BlackBerry support the `HandheldFriendly` meta tag which is similar to `viewport`. Include the following line in the header to enable the page to scale appropriately:

```
<meta name="HandheldFriendly" content="True">
```

8.5.1 Setting the Viewports for iPhone

While some mobile browser applications may display correctly on desktop Safari browsers, they may not scale not correctly for the smaller screen of the iPhone and appear too large. As a result, the iPhone shrinks pages until they are too small to read. As illustrated by the following line from Example 8–6, set the iPhone `viewport` specifications in the `<head>` element to ensure that applications display properly on iPhones.

```
<f:verbatim rendered="#{requestContext.agent.skinFamilyType eq 'iPhonewebkit'}">
    <meta name="viewport" content="width=device-width; initial-scale=1.0;
maximum-scale=1.0; user-scalable=0;"/>
 </f:verbatim>
```

Index

A

B

D

E

G

J

M

N

P

R

S

W