

## **Oracle® Fusion Middleware**

Developing Oracle ADF Mobile Browser Applications

12c (12.1.2)

**E24161-01**

June 2013

Documentation for Oracle Application Development Framework (Oracle ADF) developers that describes how to use Oracle JDeveloper to create mobile browser-based applications comprised of Apache MyFaces Trinidad web-client components.

Oracle Fusion Middleware Developing Oracle ADF Mobile Browser Applications 12c (12.1.2)

E24161-01

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Primary Author: John Bassett

Contributing Author: Mamallan Uthaman

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

---

---

# Contents

<b>Preface</b> .....	vii
Documentation Accessibility .....	vii
Audience .....	vii
Related Documents .....	vii
Conventions .....	vii
<b>1 Overview of Oracle ADF Mobile Browser</b>	
1.1 About ADF Mobile Browser.....	1-1
1.1.1 How ADF Mobile Browser Improves Application Performance .....	1-1
1.1.2 About Java Server Faces and the Application Development Framework .....	1-2
1.1.3 Developing Mobile Applications Using ADF Mobile Browser .....	1-3
1.2 Supported Mobile Browser Features.....	1-3
<b>2 Configuring the ADF Mobile Browser Environment</b>	
2.1 About ADF Mobile Browser Application Development.....	2-1
2.2 Configuring the ADF Mobile Browser Development Environment .....	2-2
2.2.1 How to Create a Mobile Application and Project .....	2-2
2.2.2 What Happens When You Create a Mobile Application and Project.....	2-5
2.3 Developing an ADF Mobile Browser Application.....	2-6
2.3.1 How to Develop a Mobile JSF Page .....	2-6
2.4 Testing an ADF Mobile Browser Application .....	2-8
2.4.1 How to Test ADF Mobile Browser Applications on Simulators.....	2-9
2.4.2 What You May Need to Know About Browser Settings.....	2-10
<b>3 Component Support</b>	
3.1 About Apache My Faces Trinidad Components.....	3-1
3.1.1 Supported Features .....	3-1
3.1.2 Partial Page Rendering.....	3-1
3.1.3 Dialogs.....	3-2
3.1.4 Rendering Specific to the BlackBerry Browser 4.5 and Earlier Versions.....	3-2
3.2 Input Components.....	3-2
3.2.1 Creating Input Text Fields.....	3-2
3.2.2 Creating Lists.....	3-3
3.3 Output Components.....	3-3
3.3.1 Displaying Text.....	3-3

3.3.2	Displaying Images .....	3-4
3.3.3	Showing (or Hiding) Components.....	3-4
3.4	Layout Components .....	3-4
3.4.1	Managing the Page .....	3-5
3.4.2	Laying Out Sections of the Page.....	3-5
3.4.3	Inserting Spaces .....	3-6
3.5	Navigation Components.....	3-6
3.5.1	Creating Buttons .....	3-6
3.5.2	Creating Links .....	3-7
3.5.3	Navigation Components.....	3-7
3.6	Data Visualization (Graphs and Gauges).....	3-8
3.7	Tables and Trees.....	3-10
3.7.1	Creating Tables .....	3-10
3.7.2	Creating Trees .....	3-10
3.8	Generating HTML <meta> Tags.....	3-11
3.8.1	Using <trh:meta> to Generate HTML <meta> Tags .....	3-11
3.8.1.1	About Default Viewport Size on Mobile Devices .....	3-12
3.9	Unsupported Components and Attributes .....	3-12
3.9.1	Unsupported Components.....	3-12
3.9.2	Unsupported Attributes .....	3-13

## 4 Skinning ADF Mobile Browser Applications

4.1	About ADF Mobile Browser Skinning.....	4-1
4.2	Implementing ADF Mobile Browser Skinning.....	4-1
4.2.1	Extending the ADF Mobile Skins .....	4-3
4.3	Applying ADF Mobile Browser Skinning.....	4-3
4.3.1	Headers .....	4-3
4.3.1.1	Creating a Title-Only Header .....	4-4
4.3.1.2	Creating Headers with Titles and Links .....	4-4
4.3.2	Table Components.....	4-5
4.3.2.1	Multi-Column Tables .....	4-5
4.3.2.2	Adding Images and Primary Details with Links .....	4-7
4.3.2.3	Creating Primary Details with Links.....	4-9
4.3.2.4	Creating Primary Details Without Links .....	4-10
4.3.3	Panel List Components .....	4-12
4.3.4	PanelFormLayout .....	4-13
4.3.5	Panel Accordion.....	4-15

## 5 Supporting Basic HTML Mobile Browsers

5.1	About Basic HTML Mobile Browser Support.....	5-1
5.1.1	Requirements for Basic HTML Mobile Browser Support.....	5-1
5.2	Developing Applications for Basic HTML Mobile Browsers .....	5-1
5.3	Styling Basic HTML Mobile Browsers .....	5-2

## 6 Design Guidelines for BlackBerry 4.2 to 4.5

6.1	About BlackBerry Browser Display Behavior.....	6-1
-----	--	-----

6.2	Formatting Tables to Prevent Wrapping.....	6-1
6.2.1	How to Prevent Fields from Wrapping in Tables.....	6-1
6.3	Formatting Label and Message Panels .....	6-2
6.4	Formatting Column Width.....	6-2
6.5	What You May Need to Know About Display Variations on BlackBerry Smartphones .	6-2
6.5.1	Changing the Minimum Font Size .....	6-2
6.5.2	Form Factor Variations .....	6-3

## **7 Narrow Screen Support and User Agent Details Support**

7.1	Determining Narrow Screen Support.....	7-1
7.1.1	How Trinidad Determines Narrow-Screen Optimization.....	7-1
7.2	Determining User Agent Capabilities Using EL Expressions .....	7-2
7.2.1	How To Determine User Agent Details .....	7-2
7.2.2	How to Determine Browser Capabilities .....	7-2

## **8 Extending ADF Mobile Browser Applications**

8.1	Introduction to Extending Applications for E-Mail, Telephony, and Google Maps .....	8-1
8.2	Integrating an E-Mail Client.....	8-1
8.2.1	Adding Mail Properties .....	8-2
8.3	Integrating Telephony.....	8-2
8.4	Integrating Google Maps .....	8-3
8.4.1	Programming Driving Directions .....	8-3
8.4.2	Supporting Google Maps on iPhone.....	8-4
8.5	What You May Need to Know About Page Display Dimensions .....	8-4
8.5.1	Setting the Viewports for iPhone .....	8-5



---

---

# Preface

Welcome to *Developing Oracle ADF Mobile Browser Applications*.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Audience

This document is intended for developers of browser applications for mobile devices.

## Related Documents

For more information, see the following related documents:

- *Understanding Oracle Application Development Framework*
- *Developing Applications with Oracle JDeveloper*
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Developing Web User Interfaces with Oracle ADF Faces*
- *Administering Oracle ADF Applications*
- *Oracle JDeveloper 12c Release Notes*, included with your JDeveloper 12c installation, and on Oracle Technology Network
- Oracle JDeveloper online help

## Conventions

The following text conventions are used in this document:

---

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

---



---

---

# 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:

- [Section 1.1, "About ADF Mobile Browser"](#)
- [Section 1.2, "Supported Mobile Browser Features"](#)

## 1.1 About ADF Mobile Browser

Oracle Application Development Framework Mobile (ADF Mobile) browser enables you to create browser-based, enterprise mobile applications that adapt to the requirements of the mobile browsers used on different types of smartphones and feature phones. Using ADF Mobile's standards-based framework, you can, for example, create web applications that enable users to approve expense reports, or search corporate contact information using a smartphone. Because ADF Mobile browser provides automatic detection and delivery of content based on both the form factor of the mobile phone itself and the capabilities of its mobile browser, you need only build the application once before you deploy it. In other words, because ADF Mobile browser applications are compatible with many mobile browsers, you do not have to focus on the limitations of any particular mobile browser.

---

---

**Note:** To learn more about ADF Mobile browser and Oracle's mobile strategy, see the Oracle ADF Mobile browser page on Oracle Technology Network (<http://www.oracle.com/technetwork/index.html>).

---

---

The ADF Mobile browser renderer ensures that contents can be consumed correctly by the target browser. It not only handles variations in browser-specific implementations of HTML, JavaScript, CSS (Cascading Style Sheets), DOM (Document Object Model), and the XMLHttpRequest (XHR) API, but variations in system performance as well. If, for example, a browser does not support the XMLHttpRequest interface and is incapable of posting a partial-page request to a server, ADF Mobile browser enables the application to revert automatically to a full-page submit so that the page functions properly.

### 1.1.1 How ADF Mobile Browser Improves Application Performance

The ADF Mobile browser renderer is optimized to improve performance by minimizing the payload of the web page that is sent over the wireless network to the mobile phone. In mobile environments with high-latency and low-bandwidth wireless

networks, Partial Page Rendering (PPR) is essential in providing an optimal end user experience. For mobile browsers that support AJAX (Asynchronous JavaScript and XML), ADF Mobile browser supports PPR for certain components, which minimizes the amount of data requested from the server and improves application responsiveness. See also [Section 3.1.2, "Partial Page Rendering."](#)

---

**Note:** Although ADF Mobile leverages AJAX features in smartphone browsers, it also degrades gracefully to support plain HTML browsers.

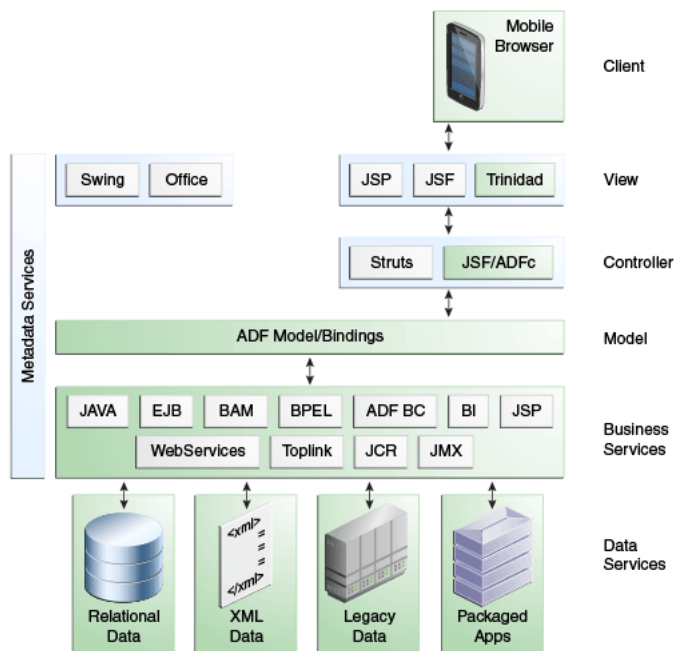
---

### 1.1.2 About Java Server Faces and the Application Development Framework

Java Server Faces (JSF) is a standard specified by JSR-127 that enables you to create applications using pre-built components that define functionality and behavior. As illustrated in [Figure 1-1](#), JSF provides a Model-View-Controller (MVC) mechanism that simplifies the development of web applications. Its renderkit converts components both to, and from, a specific markup language. The kit's renderers generate markup that represents components and how they 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 to 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.

**Figure 1-1 ADF Mobile Architecture**



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

- A large component set (because 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.

### 1.1.3 Developing Mobile Applications Using ADF Mobile Browser

Because ADF Mobile browser is built upon the component model of Java Server Faces (JSF), you can quickly develop applications for mobile browsers. Its mobile-specific extensions to JSF enable you to develop mobile applications using the same methodologies for developing JSF applications for the desktop (or tablet), as described in *Developing Web User Interfaces with Oracle ADF Faces*.

---

---

**Note:** ADF Mobile browser applications are intended to run on the mobile browsers used by smartphones or feature phones. They are not intended to run on desktop browsers.

---

---

ADF Mobile browser application development is almost identical to ADF web application development, except that you construct the application's JSF pages from Apache MyFaces Trinidad components<sup>1</sup>. ADF Mobile provides a rich component set by supporting more than 60 Apache MyFaces Trinidad components.

---

---

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

---

---

The mobile-optimized style sheets that are automatically included in the ADF Mobile browser project ensure that each of the components render appropriately to the device's browser. You can extend a desktop browser application to run on a smartphone or feature phone by reusing its model and controller layers and assembling a new view layer using the Apache MyFaces Trinidad counterparts to the ADF Faces components. For more information, see [Section 2.2, "Configuring the ADF Mobile Browser Development Environment"](#) and [Section 4.2, "Implementing ADF Mobile Browser Skinning."](#)

## 1.2 Supported Mobile Browser Features

ADF Mobile supports different types of touch-screen user interfaces, such as those used on Android- and iOS-powered smartphones. For the latter, ADF Mobile's support of Apple standards provides a native iOS user experience. [Table 1-1](#) lists the mobile browsers and features that ADF Mobile supports.

---

---

**Note:** ADF Mobile browser requires HTML and JavaScript support. See [Chapter 5, "Supporting Basic HTML Mobile Browsers."](#)

---

---

---

<sup>1</sup> The Apache MyFaces Trinidad component set is the first-generation set of ADF Faces components that were donated to the Apache Software Foundation.

**Table 1–1 Supported Browsers and Supported Mobile Features**

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

---

---

## Configuring the ADF Mobile Browser Environment

This chapter describes how to configure the ADF Mobile browser development environment, how to create ADF Mobile browser views, and how to build and test applications.

This chapter includes the following sections:

- [Section 2.1, "About ADF Mobile Browser Application Development"](#)
- [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 ADF Mobile Browser Application Development

An ADF Mobile browser application is an ADF Faces application that is optimized for mobile devices. As such, a mobile browser application can use the model project created for a standard ADF Faces application. Alternatively, you can add the ADF Mobile browser view controller project to an existing ADF faces application. After you create the ADF Mobile browser view controller project, you can construct the mobile browser application from JSF pages built from Apache MyFaces Trinidad components.

To create an ADF Mobile browser application:

- Configure the environment by creating an application and model project.

**Tip:** You can use the business logic components from an existing ADF application.

- Add an ADF Mobile browser view controller project.
- Create an ADF task flow or ADF page flow.

---

---

**Note:** ADF task flows used in ADF Mobile browser applications only support whole pages. They do not support page fragments, pop-ups in dialogs, or regions. ADF Mobile browser applications that use the ADF page flow only support the `trinidad-simple` skin family.

---

---

- Add the JSF pages populated with Apache MyFaces Trinidad components.
- Optimize the page layout with ADF Mobile browser skinning.

- Test the application on a smartphone or smartphone simulator.

## 2.2 Configuring the ADF Mobile Browser Development Environment

ADF Mobile browser application development requires a mobile-optimized view controller project.

### 2.2.1 How to Create 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.

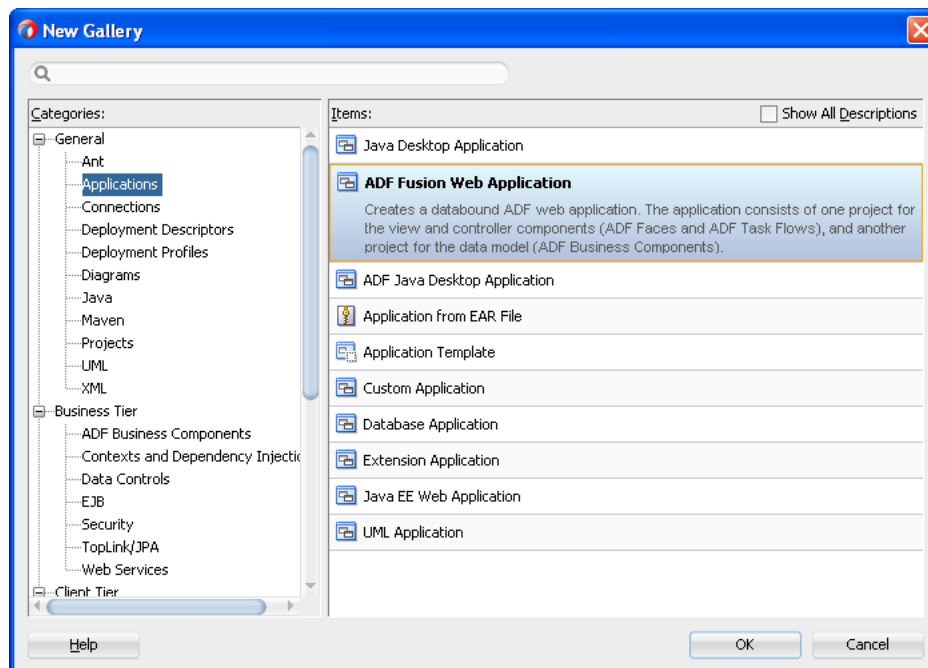
#### Before you begin:

If needed, create an ADF model project. Otherwise, the ADF Mobile browser application can share a model project with an existing ADF application.

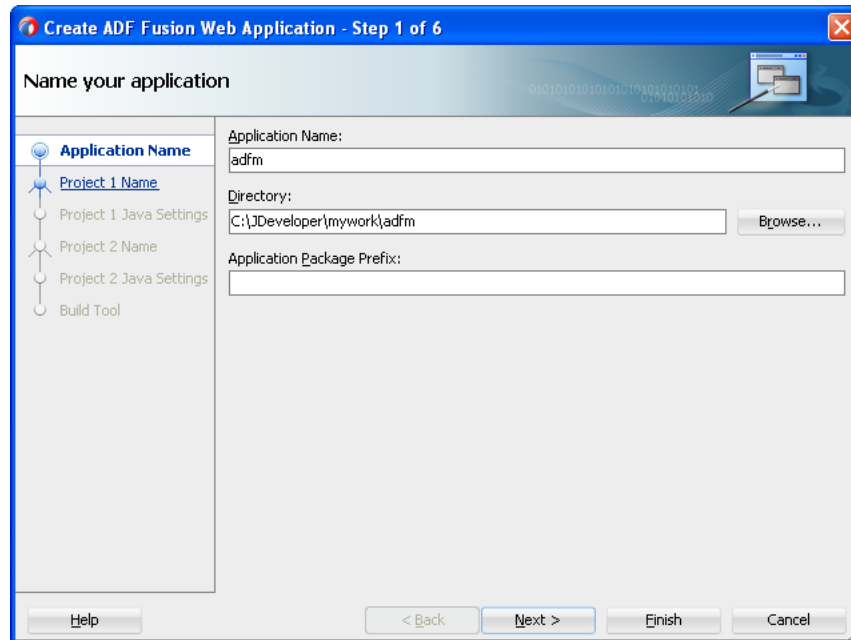
#### To create the ADF Mobile browser application and the ADF Mobile browser project:

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

Figure 2–1 The New Gallery

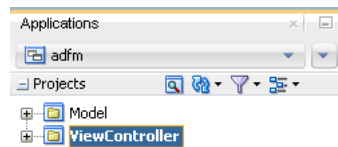


2. In the New Gallery, select **Applications** and then choose **ADF Fusion Web Application**. Click **OK**.
3. In the Name your application page of the Create ADF Fusion Web Application wizard, enter a name and, if needed, a location for the application in the Directory field, as shown in [Figure 2–2](#).

**Figure 2–2 Naming the ADF Mobile Browser Application**

4. Click **Finish**.

Selecting **ADF Fusion Web Application** creates the model project used by the mobile view project. [Figure 2–3](#) shows the application's model and its generated view controller projects that appear in the Applications window.

**Figure 2–3 The Fusion Web Application and its Projects in the Applications Window**


---

**Note:** Do not use the generated view controller project to design the business logic in the model project. Instead, create a mobile view controller project as described in the following steps.

---

5. Create the mobile view controller project as follows:

- a. Choose **File**, then **New**, and then **Project**.
- b. Select **Custom Project** and then click **OK**.
- c. In the Create Custom Project wizard, complete the wizard by first entering a name for the project, as shown in [Figure 2–4](#). For example, enter *mvc* (a short name for mobile view controller).

**Tip:** To create a short URL string that is more easily accommodated by the browsers on mobile devices, use lower-case characters and short names for ADF Mobile browser projects and their `.jspx` pages. See also [Section 2.4, "Testing an ADF Mobile Browser Application."](#)

- d. Select the ADF Mobile browser feature for the project by moving **ADF Mobile Browser** from the **Available** list to the **Selected** list.

**Figure 2–4** *Selecting the ADF Mobile Browser Technology for an ADF Mobile Browser Project*

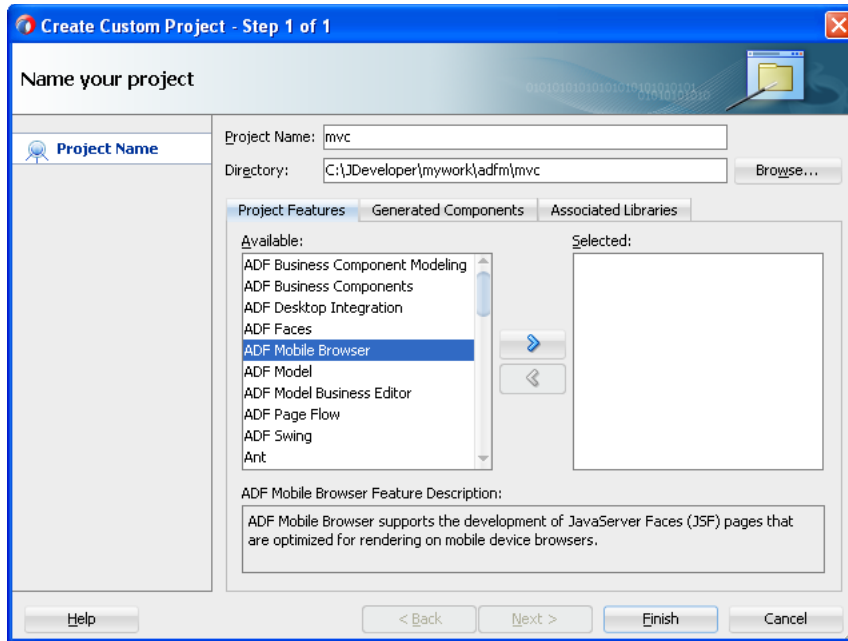
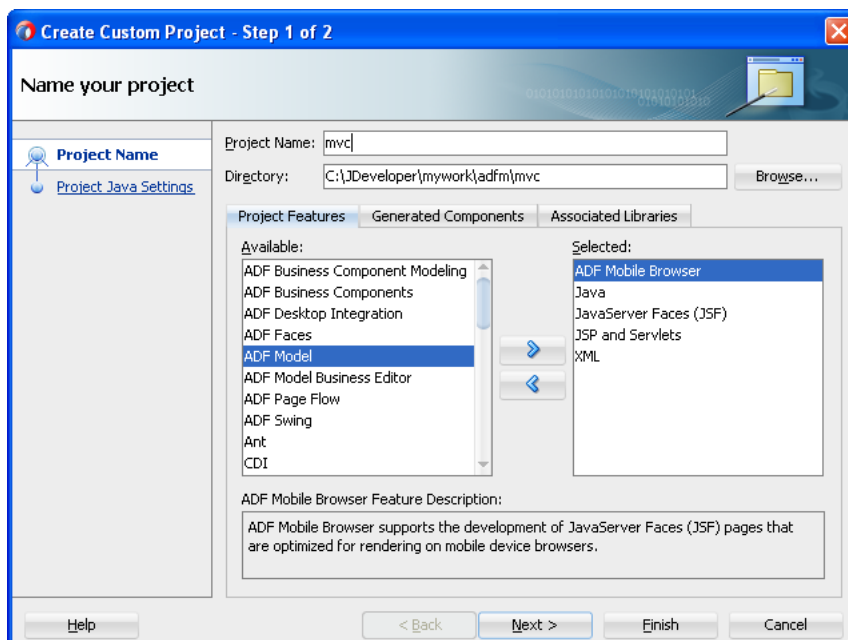


Figure 2–4 shows the ADF Mobile Browser feature in the **Available** window. After you move the **ADF Mobile Browser** feature to the **Selected** window, the Java, JSF (JavaServer Faces), JSP and Servlets technologies are made available to the project and also appear in the **Selected** window, as shown in Figure 2–5.

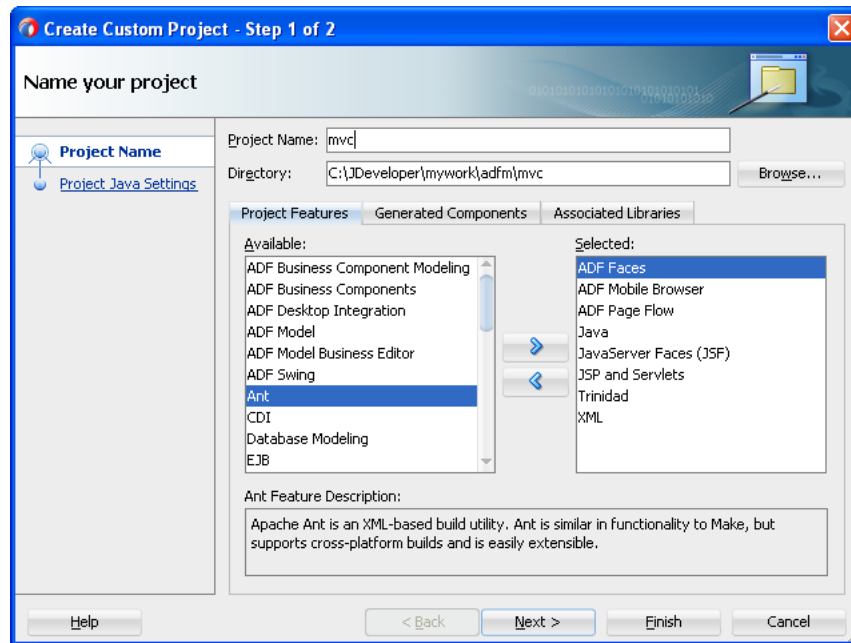
**Figure 2–5** *ADF Mobile Browser and Supporting Features Selected for an ADF Mobile Browser Project*





- e. If the application uses an ADF task flow, add the **ADF Page Flow** feature to the **Selected** window. As shown in [Figure 2–6](#), the ADF Page Flow feature adds the ADF Faces, ADF Page Flow, Trinidad, and XML technologies.

**Figure 2–6 Adding ADF Page Flow Feature to the ADF Mobile Browser Project**

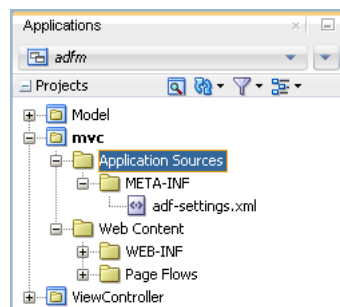


6. Click **Next** to navigate through the Configure Java settings page and then click **Finish**.

## 2.2.2 What Happens When You Create a Mobile Application and Project

As shown in [Figure 2–7](#), the mobile view controller project (*mvc*) appears in the Applications window within the Fusion web application (*adfm*).

**Figure 2–7 The Mobile ADF Mobile Browser View Controller Project in the Applications Window**



The creation of the view controller project also results in the creation of the mobile application's page-flow definition file, *faces-config.xml*. For information on creating a page flow, see "Defining Page Flows" in *Developing Web User Interfaces with Oracle ADF Faces*.

Adding the ADF Mobile Browser and ADF Page Flow technologies to the ADF Mobile custom view project results in the creation of the following ADF skin-related

configuration files and style sheets, which ensure that the components render appropriately to the target mobile browser. [Table 2–1](#) describes these files, which are located in the ADF Mobile browser view controller project's Web Content folder.

**Table 2–1 Skinning-Related Artifacts**

Skinning File	Location	Description
<code>mobile.css</code>	Styles node of the mobile view controller project	Used for basic HTML browsers and browsers used in older versions of smartphones. See also <a href="#">Section 4.2, "Implementing ADF Mobile Browser Skinning."</a>
<code>richmobile.css</code>	Styles node of the mobile view controller project	Used for smartphones, such as those powered by iOS or BlackBerry. See also <a href="#">Section 4.2, "Implementing ADF Mobile Browser Skinning."</a>

## 2.3 Developing an ADF Mobile Browser Application

For mobile browser applications, you develop an application by creating web pages within the web project. Typically, you create a web project within the application to implement a user interface and the ADF Business Components, or an Oracle EclipseLink project, to implement a business layer.

### 2.3.1 How to Develop a Mobile JSF Page

You develop ADF Mobile browser application pages by first creating a JSF page and then by populating it with the Apache My Faces Trinidad components. Typically, you first create the navigation rules using the `faces-config.xml` file before creating the JSF page for the mobile application.

#### To create a mobile JSF page:

1. Select the mobile view controller project in the Applications window, then select **New** and then **Page**.
2. Enter a name for the JSF page in the Create JSF Page dialog, as shown in [Figure 2–8](#). If needed, enter a directory location for the page. You can select either **Facelets** or **JSP XML**.

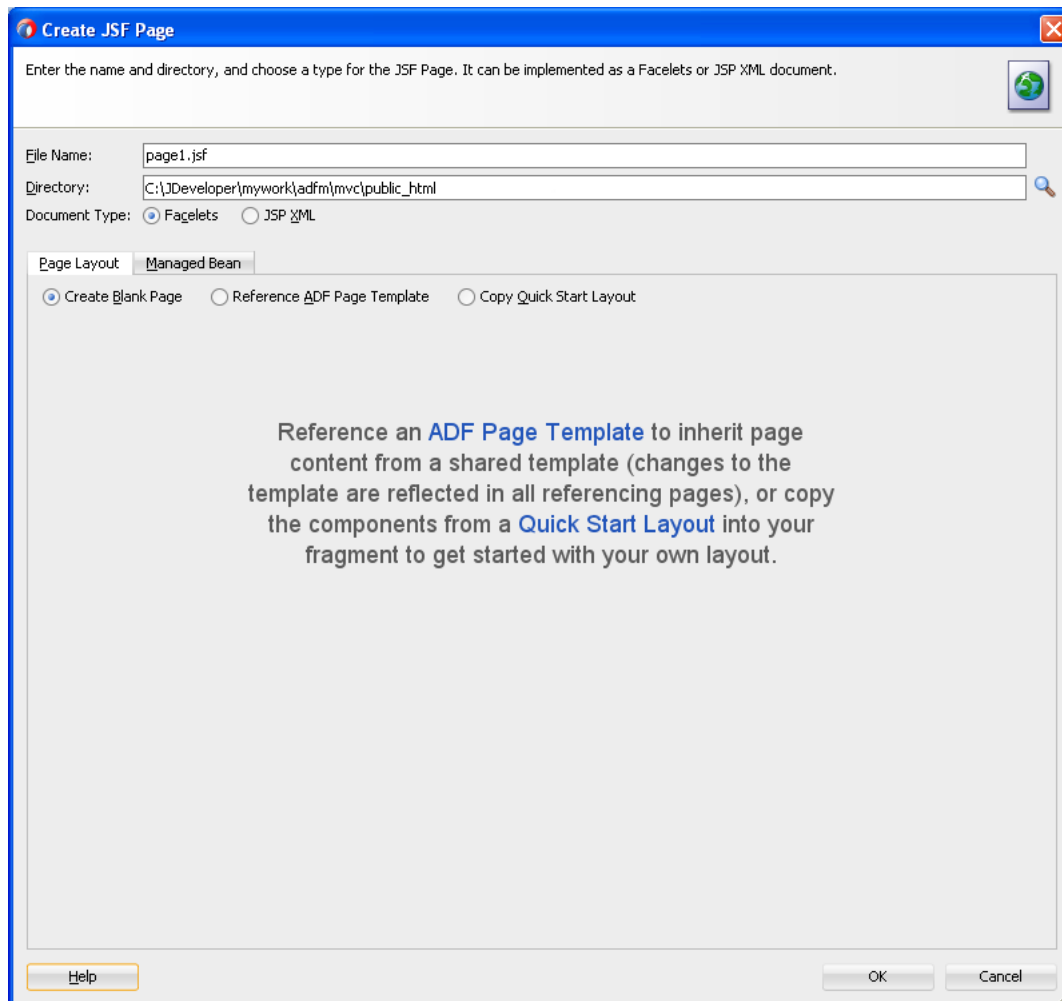
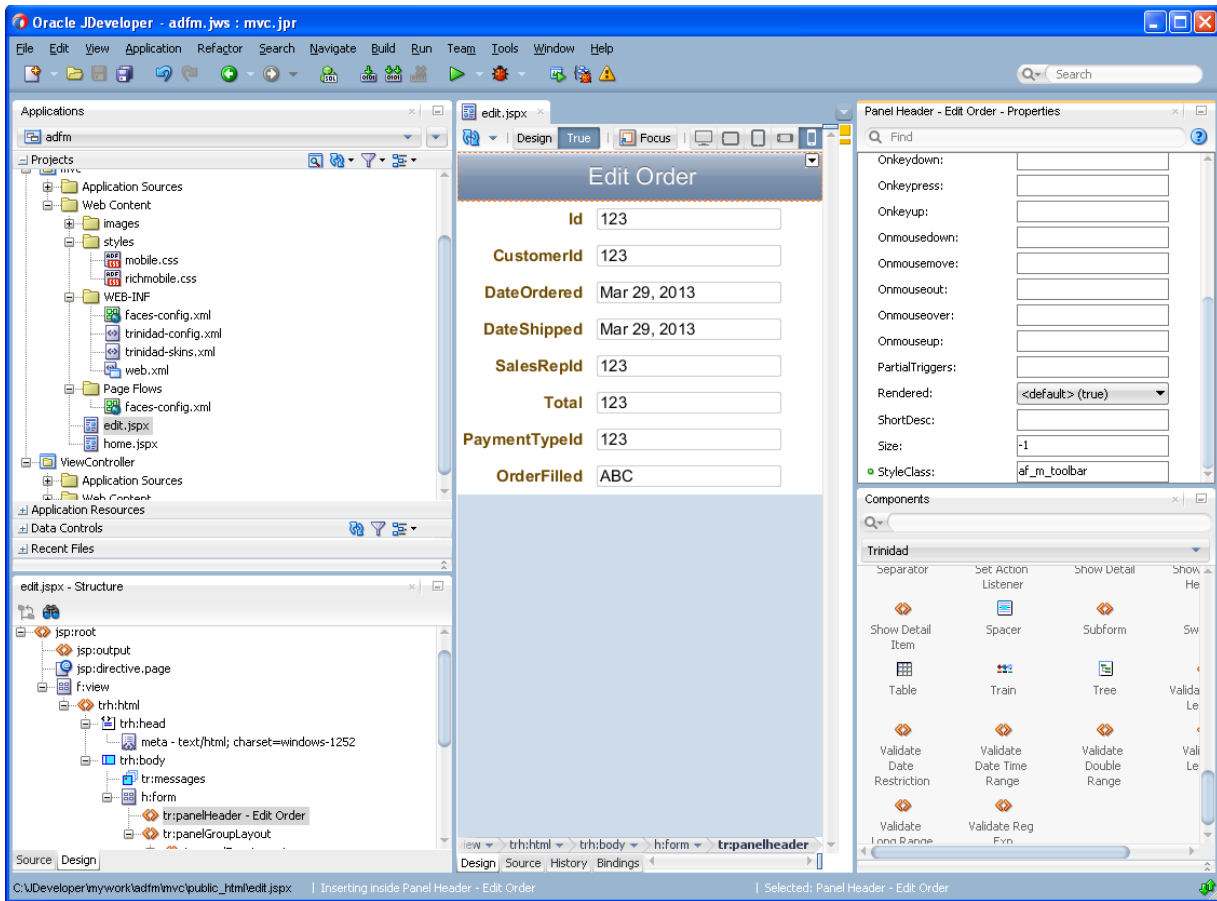
**Figure 2–8 The Create JSF Page Dialog**

Figure 2–9 shows the designer for a mobile JSF page called *page1.jsf*.

- From the Components window, select **Trinidad**, as shown in Figure 2–9, and then create the page using the Apache MyFaces Trinidad components. To optimize the display of these components, refer to the styles and patterns described in Chapter 4, "Skinning ADF Mobile Browser Applications." Figure 2–9 illustrates applying the `af_m_toolbar` style class to the `tr:panelHeader` component, which renders the page header as a toolbar-like element on smartphones.

Figure 2–9 Using Trinidad Components



## 2.4 Testing an ADF Mobile Browser Application

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

Testing an ADF Mobile browser application with a desktop browser produces only approximate results because it provides a fairly uniform testing environment; in desktop browsers, web pages appear and behave similarly and business logic executes identically. Testing an application on an actual smartphone, however, produces more accurate results, because the capabilities of mobile browsers may cause controls to behave differently than they do on a desktop browser. Mobile browsers, which are usually smaller than desktop browsers, render pages differently because web servers optimize the look and feel by generating mobile browser-specific pages.

Testing ADF Mobile browser applications directly on smartphones 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, as many mobile devices can access only the internet and cannot reach development environments behind firewalls. In such cases, smartphone simulators provide an alternative testing method. For example, to test applications on BlackBerry smartphone simulators or Windows Mobile device emulators (shown in Figure 2–10 and Figure 2–12, respectively), download smartphone simulators from the RIM developer site (<http://us.blackberry.com>) and device emulators from the Microsoft developer site (<http://www.microsoft.com>). You must then configure them and connect them to

the web server. For information on downloading and configuring simulators and emulators for ADF Mobile browser, refer to *Running Mobile Device Simulators with ADF Mobile and JDeveloper*, available through the *Mobile Application Development with Oracle ADF Mobile* page of the Oracle Technology Network. You can access this page by selecting **Oracle ADF Mobile** from the Oracle Technology Network's Oracle Application Development Framework overview page (<http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html>).

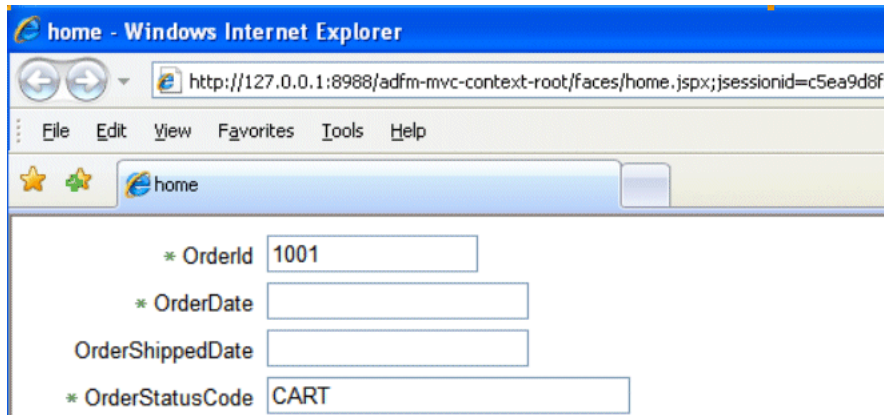
**Figure 2–10** Testing an ADF Mobile Browser Application on a BlackBerry Smartphone Simulator



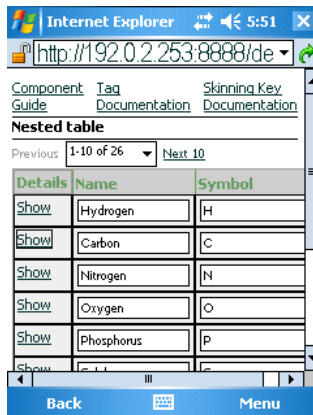
## 2.4.1 How to Test ADF Mobile Browser Applications on Simulators

After you test an application on a desktop browser, you can then test it on a simulator. 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 2–11 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 2–11) to that of the computer's network IP address (192.0.2.253, shown in Figure 2–12).

**Figure 2–12 Testing an ADF Mobile Browser Application on a Windows Mobile Device 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 2–11, 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

Viewing 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:** Selecting the *One Column* view option results in layout problems. Do not select this option.

---

---

**BlackBerry Browser 4.n**

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





---

---

## Component Support

This chapter describes the Apache MyFaces Trinidad components that are supported by ADF Mobile browser.

This chapter includes the following sections:

- [Section 3.1, "About Apache My Faces Trinidad Components"](#)
- [Section 3.2, "Input Components"](#)
- [Section 3.3, "Output Components"](#)
- [Section 3.4, "Layout Components"](#)
- [Section 3.5, "Navigation Components"](#)
- [Section 3.6, "Data Visualization \(Graphs and Gauges\)"](#)
- [Section 3.7, "Tables and Trees"](#)
- [Section 3.8, "Generating HTML <meta> Tags"](#)
- [Section 3.9, "Unsupported Components and Attributes"](#)

### 3.1 About Apache My Faces Trinidad Components

ADF Mobile browser supports more than 60 of the 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:

- [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-related 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), they present dialogs as new pages within the main browser window after they automatically preserve 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. BlackBerry's table handling may affect complex layouts, as its 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:** Component-specific messages do not display on a mobile browser in the same manner as they do in a desktop browser. Instead, they display in the region where the message component (`tr:message` or `tr:messages`) 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 conserve space within mobile browsers, 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; the `[+]` and `[-]` symbols display instead.

---

---

- `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 its `flex` attribute 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 components for laying out the sections of a page, such 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:** ADF Mobile browser does not support the `defaultCommand` attribute of the `tr:form` component.

---

---

- `tr:page`

---

---

**Note:** ADF Mobile browser does 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. ADF Mobile browser does 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:** ADF Mobile browser does not support the `halign=end` attribute definition for the `tr:panelHorizontalLayout` component.

---

---

- `tr:panelLabelAndMessage`

---

---

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

---

---

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

---

---

**Note:** ADF Mobile browser only supports 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 a 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 such 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, you can define buttons as text or as an image, but not both. If you set the `disabled` attribute to `true`, then a `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 a 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 a 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 for 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 a 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 a 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 tools (DVTs), described in the "Creating Databound ADF Data Visualization Components" section in *Developing Fusion Web Applications with Oracle Application Development Framework*.

ADF Mobile browser supports the following types of graphs:

- area
- bar
- bar (horizontal)
- bubble
- combination (horizontal bar and line)
- funnel
- line



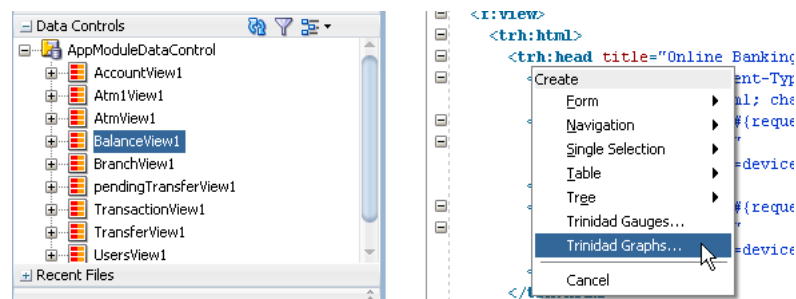
- pareto
- pie
- radar
- scattar/polar
- spark
- stock

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 3–1](#) illustrates the context menu that appears when you drag a collection into the editor.

**Figure 3–1** *Creating a DVT by Dragging and Dropping a Data Control*



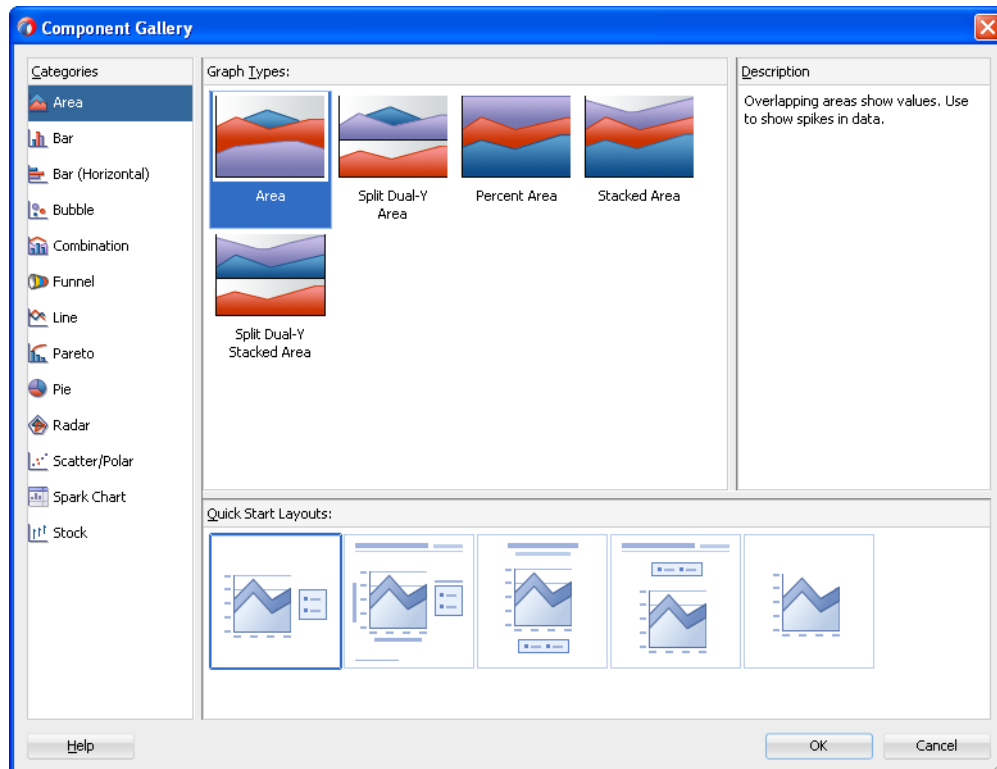
After you select either the **Trinidad Graphs** or **Trinidad Gauges** options, the DVT wizard appears and opens to the Component Gallery page, shown in [Figure 3–2](#). You select the DVT type from this page.

---

**Note:** 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 3–2 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.

---



---

Creating a single-column table optimizes how tables display on mobile devices.

### 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 Generating HTML <meta> Tags

The <meta> tag designates how pages display within a browser. [Example 3-1](#) illustrates examples of <meta> tag usage that include setting an application to display in full-screen mode and setting the viewport width for such devices as the Apple iPhone and iPad. This example also includes a <meta> tag used to designate how a page reloads after a given number of sections.

### **Example 3-1 Using the <meta> Tag to Set Page Behavior**

```
<meta name="viewport"
      content="width=device-width, user-scalable=no">
<meta name="apple-mobile-web-app-capable"
      content="yes">
<meta http-equiv="refresh"
      content="2;url=./test/index.jspx">
```

For more information on the use of the <meta> tag in applications running on Apple devices, see the descriptions of the Apple-specific meta tag keys in the "Supported Meta Tags" section of *Safari HTML Reference*, which is available through the iOS Developer Library (<http://developer.apple.com/library/ios/navigation/>).

### 3.8.1 Using <trh:meta> to Generate HTML <meta> Tags

The <trh:meta> component tag generates an HTML <meta> tag. [Example 3-2](#) shows using this component to:

- Configure the viewport dimensions on mobile devices.
- Disable user resizing of the viewport on mobile devices.
- Enable the home screen bookmark of the web page to launch it in its own process rather than in Safari on iOS devices.
- Configure the page to navigate to another page after two seconds.

### **Example 3-2 Using <trh:meta> to Generate Several HTML <meta> Tags**

```
<af:document ...>
  <f:facet name="metaContainer">
    <af:group id="metaContainer">
      <trh:meta name="viewport"
        content="width=device-width, user-scalable=no"/>
      <trh:meta name="apple-mobile-web-app-capable"
        content="yes"/>
      <trh:meta type="httpEquiv"
        name="refresh"
        content="2;url=./test/index.jspx"/>
    </af:group>
  </f:facet>
</af:document>
```

### 3.8.1.1 About Default Viewport Size on Mobile Devices

Web pages commonly have a hard-coded width that leaves white space on both sides of the page. While this does not interfere with pages displaying on desktop browsers, it can make the same pages appear too small on mobile devices. In addition, the display of the page controls can be too small for a user's finger tips as well, forcing the users to zoom in to interact with the page. Operating systems for Apple (iOS), Google (Android), and Microsoft (Windows Phone 7) assume a standard viewport width and render the page to fit the smaller screens, thereby causing this scaling to occur when web pages display on mobile devices .

To enable a page to display properly at the native resolution of a mobile device, you can specify the viewport dimension using the `viewport` meta key within the `<trh:meta>` tag, as illustrated in [Example 3-3](#).

#### **Example 3-3** Setting the Viewport Dimensions

```
<af:document ...>
  <f:facet name="metaContainer">
    <trh:meta name="viewport"
              content="width=device-width" />
  </f:facet>
  ...
</af:document>
```

You can set a specific numeric value for pixels for the `width` property of the `viewport` meta key or you can use the `device-width` constant as shown in [Example 3-3](#). On Apple (iOS) systems, the value set for the page's width in both the portrait and landscape orientations is the same (that is, for the landscape width, the Safari browser uses the value set for the portrait width).

---

---

**Note:** The pixels on high-resolution displays, such as Retina on iPhone 4, do not have a one-to-one match to pixels on a lower-resolution devices such as iPhone 3GS. Despite this, the number of pixels defined for the `width` property for an iPhone 3GS equates to the same physical length on the iPhone 4, even though the iPhone 4 has more display pixels.

---

---

## 3.9 Unsupported Components and Attributes

ADF Mobile browser does not support some Apache MyFaces Trinidad components or certain component attributes.

### 3.9.1 Unsupported Components

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`

### 3.9.2 Unsupported Attributes

ADF Mobile browser does not support the following attributes on any Apache MyFaces Trinidad component.

- `accessKey`
- `shortDesc` (tooltip)



---

---

# Skinning ADF Mobile Browser Applications

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, "Applying ADF Mobile Browser Skinning"](#)

## 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 the 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 in which components can be displayed or hidden.

---

---

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

---

---

## 4.2 Implementing ADF Mobile Browser Skinning

As noted in [Section 2.2.2, "What Happens When You Create a Mobile Application and Project,"](#) JDeveloper creates two mobile-specific stylesheets, `mobile.css` and `richmobile.css` within the ADF Mobile browser-specific view controller project, as shown in [Figure 4-1](#).

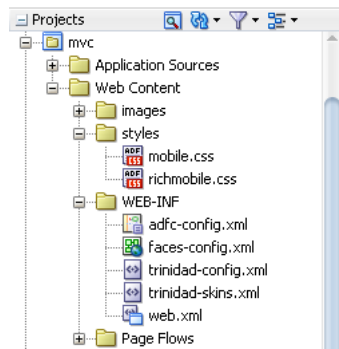
**Figure 4–1 CSS Files in the ADF Mobile Browser Project**

Table 4–1 lists the skinning files provided to ADF Mobile pages.

**Table 4–1 The ADF Mobile Browser Skins**

CSS File	Skin Family	Use
mobile.css	mobile	Used for basic HTML browsers. This family is used for rendering on Windows Mobile and BlackBerry, Version 4.6 and higher. See <a href="#">Chapter 5, "Supporting Basic HTML Mobile Browsers."</a>
richmobile.css	richmobile	Used for smartphone browsers that use the Webkit rendering engine. The Nokia S60 and also iOS- and Android-powered devices use these browsers.

Creating an ADF Mobile browser view controller project populates the `<skin-family>` element of the `trinidad-config.xml` file with the following EL expression that selects the renderkit appropriate to the browser's user agent.

```
#{requestContext.agent.type == 'desktop'? 'richmobile': 'mobile'}
```

The ADF Mobile browser framework then applies the style defined for the renderkit. This expression, which is shown in [Example 4–1](#), must be added to the `trinidad-config.xml` file to enable ADF Mobile browser applications to use the mobile and richmobile skin families.

---

**Note:** Ensure that the `trinidad-config.xml` file includes this statement.

---

#### **Example 4–1 The Skin Family Selection Logic within Trinidad-config.xml**

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

The `trinidad-skins.xml` file, shown in [Example 4–2](#), defines the default skin families that are applied using the EL expression.

#### **Example 4–2 trinidad-skins.xml Populated with ADF Mobile browser <skin> Definitions**

```
<?xml version="1.0" encoding="windows-1252"?>
```



```

<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  <skin>
    <id>richmobile</id>
    <family>richmobile</family>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
    <style-sheet-name>styles/richmobile.css</style-sheet-name>
    <extends>trinidad-simple.desktop</extends>
  </skin>
  <skin>
    <id>mobile</id>
    <family>mobile</family>
    <render-kit-id>org.apache.myfaces.trinidad.pda</render-kit-id>
    <style-sheet-name>styles/mobile.css</style-sheet-name>
    <extends>trinidad-simple.pda</extends>
  </skin>
</skins>

```

## 4.2.1 Extending the ADF Mobile Skins

You can configure the `trinidad-config.xml` file and the `trinidad-skins.xml` file to call CSS files other than `richmobile.css` and `mobile.css`. You can also modify the `richmobile.css` and `mobile.css` files to render to a specific mobile device or platform using at-rules.

To add mobile features to a non-mobile project:

1. Create a skin. This includes creating the `trinidad-skins.xml` file within the **WEB-INF** node of the ADF Mobile browser view project.
2. Create a style sheet.
3. Set the skin family in the `trinidad-config.xml` file (located in the **WEB-INF** node).

For more information, see "Create a Skin—An Overview" in *Development Guidelines for Apache MyFaces Trinidad* (<http://myfaces.apache.org/trinidad/devguide/skinning.html>).

Skin families in Apache MyFaces Trinidad are associated with a renderkit and a unique CSS file. Because Trinidad uses the desktop renderkit for Webkit-based mobile browsers and the PDA renderkit for all other mobile browsers, you can support all mobile browsers by creating two skin families, both of which reference one of these renderkits and a CSS file. Use the `@agent` and `@platform` selector rules to enable rendering based on the browser's name, version, or platform. For more information, see "Skinning CSS Features" in *Development Guidelines for Apache MyFaces Trinidad* (<http://myfaces.apache.org/trinidad/devguide/skinning.html>).

## 4.3 Applying ADF Mobile Browser Skinning

Although CSS styles are applied automatically for many components, you can optimize the rendering of a component by defining its `styleClass` attribute.

### 4.3.1 Headers

Augmenting the `<tr:panelHeader>` component with the `styleClass` attribute enables the application to display title-only headers, as well as headers with both a title and links.

### 4.3.1.1 Creating a Title-Only Header

To create a title-only header, add `styleClass="af_m_toolbar"` to the `<tr:panelHeader>` component, as illustrated in [Example 4-3](#).

**Example 4-3 Adding Attributes to Create a Title-Only Header**

```
<tr:panelHeader styleClass="af_m_toolbar" text="Welcome"/>
```

[Figure 4-2](#) shows how this ADF Mobile browser attribute creates a title-only header on an Apple iPhone.

**Figure 4-2 A Title-Only Header on the Apple iPhone**



[Table 4-2](#) lists examples of how title-only headers display on Windows Mobile devices, BlackBerry smartphones, and the Nokia Webkit.

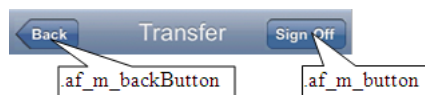
**Table 4-2 Title-Only Header Displays on Various Platforms**

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

### 4.3.1.2 Creating Headers with Titles and Links

As illustrated in [Figure 4-3](#), you can add links and a title within a header. [Figure 4-3](#) shows such a header as it displays in on the Apple iPhone.

**Figure 4-3 Title and Links Within a Header on Apple iPhone**



As described in [Section 4.3.1.1, "Creating a Title-Only Header,"](#) you define the title for the header (in [Figure 4-3](#), a title called *Transfer*) by adding `styleClass="af_m_toolbar"` within the `<tr:panelHeader>` element. The links are defined as buttons (`styleClass="af_m_backButton"` and `styleClass="af_m_button"`, respectively) within the child `<tr:commandLink>` element as illustrated in [Example 4-4](#). In [Example 4-4](#), the `<tr:panelHeader>` element includes these attributes (in bold).

**Example 4-4 Adding Titles and Links to Headers**

```
<tr:panelHeader styleClass="af_m_toolbar"
```





```

        text="Transfer">
<tr:commandLink styleClass="af_m_backButton"
        text="Back"
        action="back"/>
<tr:spacer rendered="{requestContext.agent.skinFamilyType eq
'blackberryminimal'}"
        height="5" width="105"/>
<tr:spacer rendered="{requestContext.agent.skinFamilyType eq
'windowsmobile'}"
        height=""
        width="28"/>
<tr:commandLink styleClass="af_m_button"
        text="Sign Off"
        action="signoff"/>
</tr:panelHeader>

```

Table 4–3 lists examples of how the `<tr:panelHeader>` that includes a title and links display on Windows Mobile devices, BlackBerry smartphones, and the Nokia Webkit.

**Table 4–3** Title and Link Headers on Various Platforms

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

## 4.3.2 Table Components

Using the `styleClass` attribute enables table components within ADF Mobile browser application to render appropriately on various browsers.

### 4.3.2.1 Multi-Column Tables

Unlike panel headers, which require that you include the `styleClass` attribute to apply the style appropriately on the target platform, the table column headers do not require any attributes. Instead, you use the `<tr:columns>` component described in Section 3.7.1, "Creating Tables." Figure 4–4 illustrates how column headers render on the Apple iPhone.

**Figure 4–4 Column Headers and Cells on Apple iPhone**

FirstName	LastName	Phone
Nathan	Chan	650-555-0134
Frank	Davis	650-555-0186
Alok	Gupta	650-555-0188
Louis	Robert	650-555-0145

Example 4–5 illustrates how to define the `<tr:column>` element (in bold).

**Example 4–5 Creating Column Headers**

```

<tr:table var="row" .../>
  <tr:column headerText="LastName">
    <tr:outputText value="{row.bindings.LastName.inputValue}"/>
  </tr:column>
  <tr:column headerText="FirstName">
    <tr:outputText value="{row.bindings.FirstName.inputValue}"/>
  </tr:column>
  <tr:column headerText="Phone">
    <tr:outputText value="{row.bindings.Phone.inputValue}"/>
  </tr:column>
</tr:table>
    
```

Table 4–4 shows examples of how column headers display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4–4 Column Headers on Various Platforms**

Platform	Example															
BlackBerry 4.6	<table border="1"> <thead> <tr> <th>FirstName</th> <th>LastName</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nathan</td> <td>Chan</td> <td>650-555-0134</td> </tr> <tr> <td>Frank</td> <td>Davis</td> <td>650-555-0186</td> </tr> <tr> <td>Alok</td> <td>Gupta</td> <td>650-555-0188</td> </tr> <tr> <td>Louis</td> <td>Robert</td> <td>650-555-0145</td> </tr> </tbody> </table>	FirstName	LastName	Phone	Nathan	Chan	650-555-0134	Frank	Davis	650-555-0186	Alok	Gupta	650-555-0188	Louis	Robert	650-555-0145
FirstName	LastName	Phone														
Nathan	Chan	650-555-0134														
Frank	Davis	650-555-0186														
Alok	Gupta	650-555-0188														
Louis	Robert	650-555-0145														
Windows Mobile	<table border="1"> <thead> <tr> <th>FirstName</th> <th>LastName</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nathan</td> <td>Chan</td> <td>650-555-0134</td> </tr> <tr> <td>Frank</td> <td>Davis</td> <td>650-555-0186</td> </tr> <tr> <td>Alok</td> <td>Gupta</td> <td>650-555-0188</td> </tr> <tr> <td>Robert</td> <td>Louis</td> <td>650-555-0145</td> </tr> </tbody> </table>	FirstName	LastName	Phone	Nathan	Chan	650-555-0134	Frank	Davis	650-555-0186	Alok	Gupta	650-555-0188	Robert	Louis	650-555-0145
FirstName	LastName	Phone														
Nathan	Chan	650-555-0134														
Frank	Davis	650-555-0186														
Alok	Gupta	650-555-0188														
Robert	Louis	650-555-0145														

**Table 4–4 (Cont.) Column Headers on Various Platforms**

Platform	Example															
Nokia Webkit	<table border="1"> <thead> <tr> <th>FirstName</th> <th>LastName</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nathan</td> <td>Chan</td> <td>650-555-0134</td> </tr> <tr> <td>Frank</td> <td>Davis</td> <td>650-555-0186</td> </tr> <tr> <td>Alok</td> <td>Gupta</td> <td>650-555-0188</td> </tr> <tr> <td>Louis</td> <td>Robert</td> <td>650-555-0145</td> </tr> </tbody> </table>	FirstName	LastName	Phone	Nathan	Chan	650-555-0134	Frank	Davis	650-555-0186	Alok	Gupta	650-555-0188	Louis	Robert	650-555-0145
FirstName	LastName	Phone														
Nathan	Chan	650-555-0134														
Frank	Davis	650-555-0186														
Alok	Gupta	650-555-0188														
Louis	Robert	650-555-0145														
BlackBerry 4.2	<table border="1"> <thead> <tr> <th>FirstName</th> <th>LastName</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nathan</td> <td>Chan</td> <td>650-555-0134</td> </tr> <tr> <td>Frank</td> <td>Davis</td> <td>650-555-0186</td> </tr> <tr> <td>Alok</td> <td>Gupta</td> <td>650-555-0188</td> </tr> <tr> <td>Louis</td> <td>Robert</td> <td>650-555-0145</td> </tr> </tbody> </table>	FirstName	LastName	Phone	Nathan	Chan	650-555-0134	Frank	Davis	650-555-0186	Alok	Gupta	650-555-0188	Louis	Robert	650-555-0145
FirstName	LastName	Phone														
Nathan	Chan	650-555-0134														
Frank	Davis	650-555-0186														
Alok	Gupta	650-555-0188														
Louis	Robert	650-555-0145														

### 4.3.2.2 Adding Images and Primary Details with Links

Figure 4–5 demonstrates creating the links and details within a table using the styleClass values `af_m_listingLink` and `af_m_listingDetails`.

**Figure 4–5 Images, Links and Details as Rendered on the Apple iPhone**

As illustrated in Example 4–6, you create these features by adding a `<tr:panelGroupLayout>` component as a child of a `<tr:column>` component. You then add the `styleClass="af_m_listingLink"` and `styleClass="af_m_listingDetails"` attributes to the `panelGroupLayout`'s `<tr:commandLink>` and `<tr:outputText>` subcomponents. See Chapter 3, "Component Support" for information on the `tr:panelGroupLayout`, `tr:commandLink`, and `tr:outputText`.

#### Example 4–6 Adding Links with Details

```
<tr:table horizontalGridVisible="false"
```

```

var="row"
width="100%">
<tr:column>
  <tr:image source="#{row.bindings.TypeIconUrl.inputValue}"/>1
</tr:column>
<tr:column inlineStyle="width:100%;">
  <tr:panelGroupLayout layout="vertical">
    <tr:commandLink text="#{row.bindings.DescShort.inputValue}"
      action="detail"
      styleClass="af_m_listingLink">
    </tr:commandLink>
    <tr:outputText value="#{row.bindings.Balance.inputValue}"
      styleClass="af_m_listingDetails">
    </tr:outputText>
  </tr:panelGroupLayout>
</tr:column>
</tr:table>

```

Table 4-5 shows examples of how images, links, and details display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4-5 Images, Links, and Details on Various Platforms**

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	

**Table 4–5 (Cont.) Images, Links, and Details on Various Platforms**

Platform	Example
BlackBerry 4.2	

### 4.3.2.3 Creating Primary Details with Links

Figure 4–6 illustrates how to create primary details and links within a table.

**Figure 4–6 Primary Details with Links as Rendered on Apple iPhone**

Similar to adding the primary links and details with images described in Section 4.3.2.2, "Adding Images and Primary Details with Links," you create these features by adding a `<tr:panelGroupLayout>` component as a child of a `<tr:column>` component. As illustrated in Example 4–7, you then add the `styleClass="af_m_listingLink"` and `styleClass="af_m_listingDetails"` attributes to the `panelGroupLayout`'s `<tr:commandLink>` and `<tr:outputText>` subcomponents. See Chapter 3, "Component Support" for information on the `tr:panelGroupLayout`, `tr:commandLink`, and `tr:outputText`.

#### Example 4–7 Primary Details and Links

```
<tr:table horizontalGridVisible="false"
  var="row"
  width="100%">
  <tr:column>
    <tr:panelGroupLayout layout="vertical">
      <tr:commandLink text="{row.bindings.Email.inputValue}"
        styleClass="af_m_listingLink">
      </tr:commandLink>
      <tr:outputText value="{row.bindings.FirstName.inputValue}"
        styleClass="af_m_listingDetails"/>
    </tr:panelGroupLayout>
  </tr:column>
</tr:table>
```

Table 4–6 shows examples of how links and details display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4–6 Images and Links on Various Platforms**

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

#### 4.3.2.4 Creating Primary Details Without Links

As illustrated in Figure 4–7, `af_m_listingPrimaryDetails` and `af_m_listingDetails` style classes enable you to create details that do not function as links; their behavior is different from the primary details described in Section 4.3.2.2, "Adding Images and Primary Details with Links."



**Figure 4–7 Primary Details without Links on Apple iPhone**

<b>\$836.94 on 2008-12-09</b> From Mortgage To Credit
<b>\$663.39 on 2008-12-09</b> From Saving To Checking
<b>\$588.92 on 2008-12-09</b> From Credit To Saving

As illustrated in [Example 4–8](#), you create non-linking primary details by adding `styleClass="af_m_listingPrimaryDetails"` and `styleClass="af_m_listingDetails"` to the `<tr:outputText>` element. This element is a child of the `<tr:panelGroupLayout>` element (which is itself a child of the `<tr:column>` element).

**Example 4–8 Adding Non-Linking Primary Details**


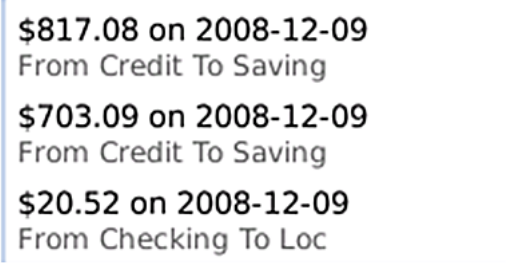
```
tr:table horizontalGridVisible="false"
  var="row"
  width="100%">
  <tr:column>
    <tr:panelGroupLayout layout="vertical">
      <tr:outputText value="{row.bindings.Amount.inputValue}"
        styleClass="af_m_listingPrimaryDetails">
      </tr:outputText>
      <tr:outputText value="{row.bindings.FromAccountName.inputValue}"
        styleClass="af_m_listingDetails"/>
    </tr:panelGroupLayout>
  </tr:column>
</tr:table>
```

[Table 4–7](#) shows examples of how non-linking details display on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4–7 Non-Linking Details on Various Platforms**

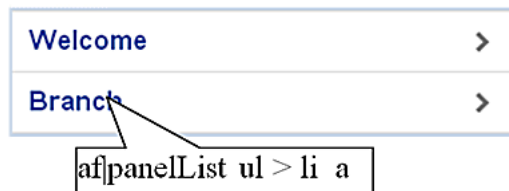
Platform	Example			
BlackBerry 4.6	<table border="1"> <tr> <td><b>\$586.6 on 2008-12-09</b> From Credit To Saving</td> </tr> <tr> <td><b>\$53.9 on 2008-12-09</b> From Saving To Checking</td> </tr> <tr> <td><b>\$263.31 on 2008-12-09</b> From Checking To Loc</td> </tr> </table>	<b>\$586.6 on 2008-12-09</b> From Credit To Saving	<b>\$53.9 on 2008-12-09</b> From Saving To Checking	<b>\$263.31 on 2008-12-09</b> From Checking To Loc
<b>\$586.6 on 2008-12-09</b> From Credit To Saving				
<b>\$53.9 on 2008-12-09</b> From Saving To Checking				
<b>\$263.31 on 2008-12-09</b> From Checking To Loc				
Windows Mobile	<table border="1"> <tr> <td><b>\$836.94 on 2008-12-09</b> From Mortgage To Credit</td> </tr> <tr> <td><b>\$663.39 on 2008-12-09</b> From Saving To Checking</td> </tr> <tr> <td><b>\$588.92 on 2008-12-09</b> From Credit To Saving</td> </tr> </table>	<b>\$836.94 on 2008-12-09</b> From Mortgage To Credit	<b>\$663.39 on 2008-12-09</b> From Saving To Checking	<b>\$588.92 on 2008-12-09</b> From Credit To Saving
<b>\$836.94 on 2008-12-09</b> From Mortgage To Credit				
<b>\$663.39 on 2008-12-09</b> From Saving To Checking				
<b>\$588.92 on 2008-12-09</b> From Credit To Saving				

**Table 4–7 (Cont.) Non-Linking Details on Various Platforms**

Platform	Example
Nokia Webkit	
BlackBerry 4.2	

### 4.3.3 Panel List Components

Defining the value of the `styleClass` as `af_m_panelBase` within the `<tr:panelGroupLayout>` component applies padding to the `<tr:panelList>` components, as shown in [Figure 4–8](#).

**Figure 4–8 Rendering Padding on an Apple iPhone**

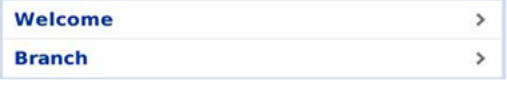

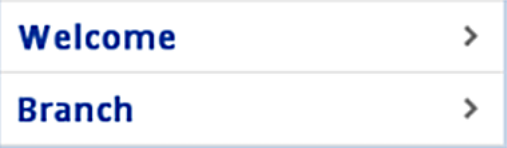

As illustrated in [Example 4–9](#), you do not have to include a `styleClass` attribute in the child `<tr:panelList>` component. For more information on using `<tr:panelList>` and `<tr:panelGroupLayout>`, see [Section 3.2.2, "Creating Lists"](#) and [Section 3.4, "Layout Components,"](#) respectively.

**Example 4–9 Adding Padding to panelList Components**

```
<tr:panelGroupLayout styleClass="af_m_panelBase">
  <tr:panelList>
    <tr:commandLink text="Welcome" action="welcome"/>
    <tr:commandLink text="Branch" action="branch"/>
  </tr:panelList>
</tr:panelGroupLayout>
```

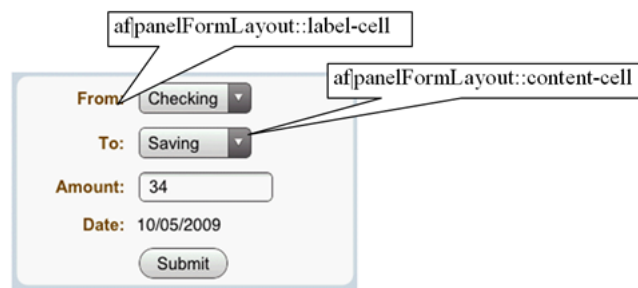
[Table 4–8](#) shows examples of padding in the `<tr:panelList>` component on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4–8** *Padding Applied to <tr:panelList> on Various Platforms*

Platform	Example
BlackBerry 4.6	
Windows Mobile	
Nokia Webkit	
BlackBerry 4.2	

### 4.3.4 PanelFormLayout

Defining the value of the `styleClass` attribute as `af_m_panelBase` within the `<tr:panelGroupLayout>` component applies padding to the child `<tr:panelFormLayout>` components, as shown in [Figure 4–9](#).

**Figure 4–9** *Padding Rendered in panelFormLayout on Apple iPhone*

As illustrated in [Example 4–10](#), you do not need to add `styleClass` to the `<tr:panelFormLayout>` component.

#### **Example 4–10** *Applying Padding to the PanelFormLayout Component*

```
<tr:panelGroupLayout styleClass="af_m_panelBase">
  <tr:panelFormLayout labelWidth="35%" fieldWidth="65%">
    <tr:selectOneChoice value="#{transferBean.transferFromAccount}"
      label="From:" showRequired="false">
      <f:selectItems value="#{bindings.AccountView1.items}" />
    </tr:selectOneChoice>
```

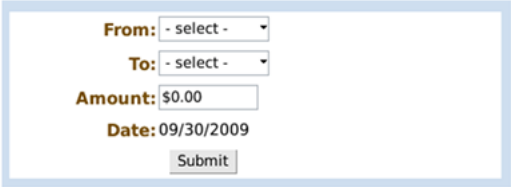

```

<tr:selectOneChoice value="#{transferBean.transferToAccount}"
    showRequired="false"
    unselectedLabel="- select -"
    label="To:">
    <f:selectItems value="#{bindings.AccountView1.items}" />
</tr:selectOneChoice>
<tr:inputText id="amount"
    columns="#{requestContext.agent.capabilities.narrowScreen ? '8' : '12'}"
    required="false" showRequired="false"
    value="#{transferBean.transferAmount}"
    label="Amount:">
    <f:converter converterId="Bank10.amountConverter" />
</tr:inputText>
<tr:panelLabelAndMessage label="Date: ">
    <tr:outputText value="#{transferBean.transferDate}" />
</tr:panelLabelAndMessage>
<f:facet name="footer">
    <tr:panelGroupLayout>
        <tr:spacer rendered="#{requestContext.agent.skinFamilyType eq
            'blackberryminimal'}"
            height="5"
            width="75" />
        <tr:commandButton text="Submit"
            action="#{transferBean.validateTransferRequest}" />
    </tr:panelGroupLayout>
</f:facet>
</tr:panelFormLayout>
</tr:panelGroupLayout>

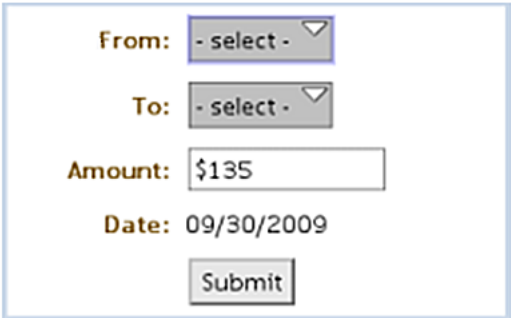

```

Table 4–9 shows examples of padding in the <tr:panelList> component on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4–9** *Padding Applied to <tr:panelFormLayout> Component on Various Platforms*

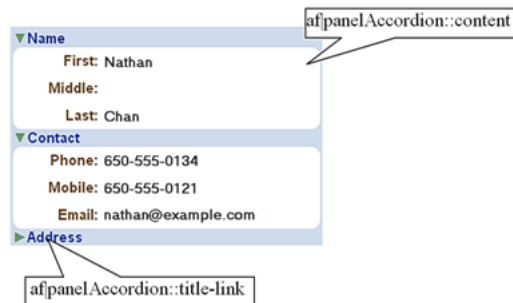
Platform	Example
BlackBerry 4.6	
Windows Mobile	

**Table 4–9 (Cont.) Padding Applied to <tr:panelFormLayout> Component on Various**

Platform	Example
Nokia Webkit	
BlackBerry 4.2	

### 4.3.5 Panel Accordion

Defining the value of the `styleClass` attribute as `af_m_panelBase` within the `<tr:panelGroupLayout>` component applies padding to its `<tr:panelAccordion>` component, as shown in [Figure 4–10](#).

**Figure 4–10 Padding Applied to the Panel Accordion on Apple iPhone**

As illustrated in [Example 4–11](#), you do not need to add the `styleClass` attribute to the `<tr:panelAccordion>` component.

**Example 4–11 Applying Padding to the <tr:panelAccordion> Component**

```
<tr:panelGroupLayout styleClass="af_m_panelBase">
  <tr:panelAccordion discloseMany="true">
    <tr:showDetailItem text="Name" disclosed="true">
      <tr:panelFormLayout fieldWidth="70%" labelWidth="30%">
        ...
      </tr:panelFormLayout>
    </tr:panelAccordion>
  </tr:panelGroupLayout>
```



```

</tr:showDetailItem>
<tr:showDetailItem text="Contact" disclosed="true">
  <tr:panelFormLayout fieldWidth="70%" labelWidth="30%">
    ...
  </tr:panelFormLayout>
</tr:showDetailItem>
<tr:showDetailItem text="Address">
  <tr:panelFormLayout fieldWidth="70%" labelWidth="30%">
    ...
  </tr:panelFormLayout>
</tr:showDetailItem>
</tr:panelAccordion>
</tr:panelGroupLayout>



```

Example 4-10 shows examples of padding in the `<tr:panelAccordion>` component on Windows Mobile devices, the Nokia Webkit, and BlackBerry smartphones.

**Table 4-10** `<tr:panelAccordion>` on Various Platforms

Platform	Example
BlackBerry 4.6	 <p>The screenshot shows a contact form on a BlackBerry 4.6 device. The form has a light blue header for each section. The 'Name' section is collapsed, showing 'First: Nathan', 'Middle:', and 'Last: Chan'. The 'Contact' section is expanded, showing 'Phone: 650-555-0134', 'Mobile: 650-555-0121', and 'Email: nathan@example.com'. The 'Address' section is collapsed, showing '[+]Address'.</p>
Windows Mobile	 <p>The screenshot shows the same contact form on a Windows Mobile device. The sections are expanded, showing 'First: Nathan', 'Middle:', 'Last: Chan', 'Phone: 650-555-0134', 'Mobile: 650-555-0121', and 'Email: nathan@example.com'. The 'Address' section is collapsed, showing '[+]Address'. The padding around the text is noticeably larger than on the BlackBerry device.</p>

**Table 4–10 (Cont.) <tr:panelAccordion> on Various Platforms**

Platform	Example
Nokia Webkit	 <p>The screenshot shows a vertical accordion on a Nokia Webkit device. It has three sections: '[-]Name', '[-]Contact', and '[+]Address'. The 'Name' section is expanded, showing 'First: Nathan', 'Middle:', and 'Last: Chan'. The 'Contact' section is collapsed, showing 'Phone: 650-555-0134', 'Mobile: 650-555-0121', and 'Email: nathan@example.com'. The 'Address' section is collapsed.</p>
BlackBerry 4.2	 <p>The screenshot shows a vertical accordion on a BlackBerry 4.2 device. It has four sections: '-Name', '-Contact', '+Address', and an unlabeled section. The 'Name' section is expanded, showing 'First: Nathan', 'Middle:', and 'Last: Chan'. The 'Contact' section is collapsed, showing 'Phone: 555-555-0134', 'Mobile: 555-555-0121', and 'Email: nathan@example.com'. The 'Address' section is collapsed.</p>





---

---

## 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

Because basic HTML browsers for mobile devices do not support JavaScript, they are less robust than the other browsers supported by ADF Mobile browser, such as those for BlackBerry smartphones or the Apple iPhone. Aside from the browsers listed in [Section 1.2, "Supported Mobile Browser Features,"](#) 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 browser's support is XHTML Basic, or the XHTML Mobile Profile that includes WAP2.*n* browsers.

---

---

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

---

---

### 5.2 Developing Applications for Basic HTML Mobile Browsers

Because the ADF Mobile browser 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. Although most of these browsers support CSS, ADF Mobile browser applications can even run on the browsers that do not support CSS. In these cases, however, the user interface may be difficult to use. As a precaution, you should test the ADF Mobile browser application on as many browsers as possible.

---

---

## 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:

- [Section 6.1, "About BlackBerry Browser Display Behavior"](#)
- [Section 6.2, "Formatting Tables to Prevent Wrapping"](#)
- [Section 6.3, "Formatting Label and Message Panels"](#)
- [Section 6.4, "Formatting Column Width"](#)
- [Section 6.5, "What You May Need to Know About Display Variations on BlackBerry Smartphones"](#)

### 6.1 About BlackBerry Browser Display Behavior

The BlackBerry browser behaves differently than other browsers in that it does not display pages using horizontal scrolling. Instead, it fits a page to the width of the screen.

### 6.2 Formatting Tables to Prevent Wrapping

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

---

---

**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 cannot fit within 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.

$3 * \text{Number of columns} + \text{the Sum of the size attributes in all columns} \leq X$ , when  $X=48$

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

$3 * \text{Number of Columns} +$   
 $\text{Sum of size attributes in all input field columns} +$   
 $\text{Sum of number of characters in longest words in all output field columns} \leq X$ , when  $X=48$

If the field 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 100%.

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

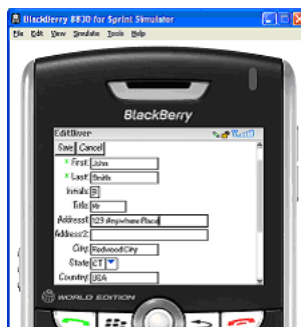
The same application can display differently on different devices because of such factors as:

- [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](#).

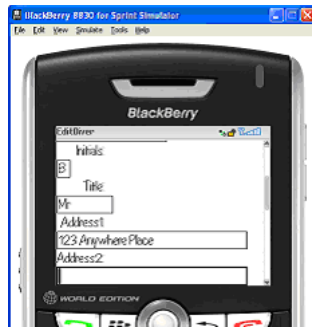
**Figure 6-1 Application Display Using the Default Font Size of 6 pt.**



Increasing the font size to 10 pt., however, 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 other 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.

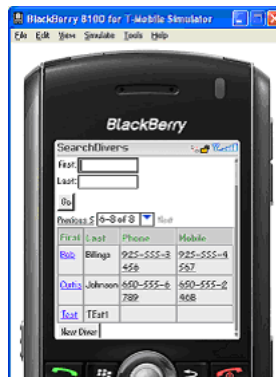
**Figure 6–3 Difficulty Displaying Input Fields and Labels with Font Size at 6 pt.**



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 accommodate the text, causing it to wrap.

**Figure 6–4 Wrapping Text**





---

---

# 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:

- [Section 7.1, "Determining Narrow Screen Support"](#)
- [Section 7.2, "Determining User Agent Capabilities Using EL Expressions"](#)

## 7.1 Determining Narrow Screen Support

Mobile devices have 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 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 determining 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. For example, 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` and `PlatformVersion` if it cannot parse them from the user agent string.

---

**Table 7-1 Browser Details Exposed through EL Expressions**

Detail Name	Description
<code>type</code>	Identifies a user agent type. For desktop and mobile browsers, the values are <i>desktop</i> and <i>PDA</i> , 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.
<code>agentName</code>	The name of the agent
<code>agentVersion</code>	The version of the agent
<code>platformName</code>	The platform on which the agent runs
<code>platformVersion</code>	The version of the platform on which the agent runs
<code>hardwareMakeModel</code>	The model of the mobile device

### 7.2.2 How to Determine Browser Capabilities

Trinidad sends its response to a user agent's request based on the capabilities it assigns to a user agent. These capabilities include a user agent's support for JavaScript and PPR. Some of these capabilities (listed in [Table 7-2](#)) are exposed 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 7–2 Browser Capabilities Exposed through EL Expressions**

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

---



---

---

# Extending ADF Mobile Browser Applications

This chapter describes how to add e-mail, telephony, and Google Maps functionality to ADF Mobile browser applications.

This chapter includes the following sections:

- [Section 8.1, "Introduction to Extending Applications for E-Mail, Telephony, and Google Maps"](#)
- [Section 8.2, "Integrating an E-Mail Client"](#)
- [Section 8.3, "Integrating Telephony"](#)
- [Section 8.4, "Integrating Google Maps"](#)
- [Section 8.5, "What You May Need to Know About Page Display Dimensions"](#)

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

In addition to using the style sheets described in [Chapter 4, "Skinning ADF Mobile Browser Applications"](#), you can further tailor an ADF Mobile browser application to include support for e-mail, telephony, and Google Maps by defining the `tr:goButton` and `tr:goLink` components with EL (Expression Language) expressions.

## 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	<code>subject =&lt;subject text&gt;</code>
cc Recipients	<code>cc=&lt;name@address.com&gt;</code>
bcc Recipients	<code>bcc=&lt;name@address.com&gt;</code>
Message Text	<code>body=&lt;Message Text&gt;</code>

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=hello
                                     &cc=myboss@oracle.com
                                     &bcc=me@oracle.com
                                     &body=good morning!"/>
```

## 8.3 Integrating Telephony

To invoke a call dialog box for a phone number:

1. Use either the `tr:goButton` or the `tr:goLink` component.
2. Prepend the phone number with the `tel:` protocol.

---

**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 an end user dials the primary phone number. Because Apple does not specify which portions of RFC 2086 that it supports, you must test each portion.

---

3. Set the `destination` property to the telephone number (represented as the EL expression, `#{sessionScope.empDetails.PhoneNumber}`) in [Example 8-3](#).

### 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, specify the `destination` property of the `tr:goLink` component as follows:

1. Use either the `tr:goButton` or the `tr:goLink` component.
2. Define `destination=` as the URL of Google Maps. (destination=`http://maps.google.com/maps`, as illustrated in [Example 8-4](#).)
3. To search for a location, append the Google Maps URL with `?q=`.
4. 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 for and locate. If multiple items are found, Google Maps drops multiple pins automatically.

---

**Note:** The address described in the `text` string must be well-formatted and include commas between words. For the `destination` string, 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,+USA"/>
```

[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},
              +#{sessionScope.empDetails.City},
              +#{sessionScope.empDetails.StateProvince},
              +#{sessionScope.empDetails.CountryName}"/>
```

You must join each EL expression in the address string with a plus sign (+), as illustrated in [Example 8-5](#). Do not include spaces between the EL expressions.

### 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, California, to 1 Telegraph Hill in San Francisco, California, are as follows:

```
http://maps.google.com/maps?
```

```
saddr=200+Oracle+Parkway,+Redwood+City,+CA,+USA
&daddr=1+Telegraph+Hill,+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 for 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 URL parameters supported by [maps.google.com](http://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 at the correct scale. [Example 8–6](#), illustrates setting the viewports for both iPhones and BlackBerry smartphones. For more information on the viewport, see *iOS Human Interface Guidelines*, available from the iOS Developer Library (<http://developer.apple.com/library/ios/navigation/>).

### 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 may appear too large. As a result, the iPhone shrinks pages until they are too small to read. The following line from [Example 8-6](#), illustrates how to 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;"/>
```

