

## **Oracle® Fusion Middleware**

Developing ADF Skins

12c (12.2.1.1)

**E67036-01**

June 2016

Documentation for Oracle Application Development Framework (Oracle ADF) developers and user interface designers that describes how to create and apply skins to an application.

Oracle Fusion Middleware Developing ADF Skins, 12c (12.2.1.1)

E67036-01

Copyright © 2014, 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author: Walter Egan

Contributing Authors: Laura Akel

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 END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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 or hardware and documentation may provide access to or information about 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 unless otherwise set forth in an applicable agreement between you and Oracle. 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, except as set forth in an applicable agreement between you and Oracle.

---

---

# Contents

Preface .....	vii
Audience .....	vii
Documentation Accessibility .....	vii
Related Documents.....	vii
Conventions.....	vii
<b>1 About Skinning a Web Application</b>	
1.1 Introduction to Skinning a Web Application .....	1-1
1.2 Overview of Developing an ADF Skin .....	1-2
1.3 Taking a Look at an ADF Skin .....	1-4
1.4 Inheritance Relationship of the ADF Skins Provided by Oracle ADF .....	1-6
<b>2 Working with the Theme Editor</b>	
2.1 About the Theme Editor .....	2-1
2.2 Setting Up and Starting the Theme Editor .....	2-2
2.2.1 How to Set Up the Theme Editor .....	2-3
2.2.2 How to Persist ADF Skins Created in the Theme Editor.....	2-3
2.2.3 How to Start the Theme Editor .....	2-4
2.3 Exporting an ADF Skin from the Theme Editor.....	2-4
<b>3 Working with ADF Skin Selectors</b>	
3.1 About ADF Skin Selectors .....	3-1
3.1.1 ADF Skin Selectors and Pseudo-Elements.....	3-3
3.1.2 ADF Skin Selectors and Icon Images .....	3-3
3.1.3 Grouped ADF Skin Selectors .....	3-5
3.1.4 Descendant ADF Skin Selectors .....	3-6
3.2 Pseudo-Classes in the ADF Skinning Framework .....	3-7
3.3 Properties in the ADF Skinning Framework .....	3-10
3.4 Accessing Selector Information from Within JDeveloper.....	3-12
<b>4 Working with ADF Skins in JDeveloper</b>	
4.1 About the Editors for ADF Skins in JDeveloper .....	4-1

4.2	Working with the ADF Skin Design Editor .....	4-1
4.2.1	How to Change the Browser that Renders the Design Editor's Sample Pages .....	4-2
4.3	Working with the ADF Skin Selectors Editor .....	4-3
4.3.1	About the Selector Tree .....	4-5
4.3.2	Interactive Preview in the Selectors Editor.....	4-7
4.4	Working with the Properties Window .....	4-8
4.5	Navigating ADF Skins .....	4-10
<b>5</b>	<b>Creating the Source Files for an ADF Skin</b>	
5.1	About Creating an ADF Skin .....	5-1
5.2	Creating an ADF Skin File.....	5-1
5.2.1	How to Create an ADF Skin in JDeveloper .....	5-2
5.2.2	What Happens When You Create an ADF Skin .....	5-3
5.3	Importing One or More ADF Skins Into the Current ADF Skin.....	5-5
5.4	Adding ADF Skins from an ADF Library JAR.....	5-5
5.4.1	How to Add an ADF Skin from an ADF Library JAR .....	5-6
5.4.2	What Happens When You Import an ADF Skin from an ADF Library JAR.....	5-6
<b>6</b>	<b>Working with Component-Specific Selectors</b>	
6.1	About Working with Component-Specific Selectors .....	6-1
6.2	Changing ADF Faces Components' Selectors.....	6-3
6.3	Changing ADF Data Visualization Components' Selectors .....	6-4
6.4	Changing a Component-Specific Selector .....	6-7
6.4.1	How to Change a Component-Specific Selector .....	6-8
6.4.2	What Happens When You Change a Component-Specific Selector .....	6-8
6.5	Configuring ADF Skin Properties to Apply to Messages .....	6-10
6.5.1	How to Configure an ADF Skin Property to Apply to a Message.....	6-11
6.5.2	What Happens When You Configure ADF Skin Properties to Apply to Messages .....	6-12
6.6	Configuring an ADF Skin for Accessibility .....	6-12
6.6.1	How to Configure an ADF Skin for Accessibility.....	6-13
<b>7</b>	<b>Working with Images and Color in Your ADF Skin</b>	
7.1	About Working with Images and Color in Your ADF Skin .....	7-1
7.2	Changing Images and Colors in the ADF Skin Design Editor .....	7-3
7.3	Working with Anchor Colors in an ADF Skin .....	7-4
7.4	Changing an Image for a Component Selector .....	7-8
7.4.1	How to Copy an Image into the Project.....	7-9
7.4.2	What Happens When You Copy an Image into the Project .....	7-10
<b>8</b>	<b>Working With Text in an ADF Skin</b>	
8.1	About Working with Text in an ADF Skin .....	8-1
8.2	Using Text From Your Own Resource Bundle .....	8-2
8.2.1	How to Specify an Additional Resource Bundle for an ADF Skin .....	8-2

8.2.2	What Happens When You Specify an Additional Resource Bundle for an ADF Skin .....	8-3
<b>9</b>	<b>Working With Global Selector Aliases</b>	
9.1	About Global Selector Aliases.....	9-1
9.2	Creating a Global Selector Alias .....	9-5
9.2.1	How to Create a Global Selector Alias .....	9-5
9.2.2	What Happens When You Create a Global Selector Alias .....	9-6
9.3	Modifying a Global Selector Alias.....	9-6
9.3.1	How to Modify a Global Selector Alias.....	9-7
9.4	Applying a Global Selector Alias .....	9-7
9.4.1	How to Apply a Global Selector Alias .....	9-8
9.4.2	What Happens When You Apply a Global Selector Alias .....	9-8
9.4.3	What You May Need to Know About Applying a Global Selector Alias .....	9-9
9.5	Referencing a Property Value from Another Selector.....	9-10
9.5.1	How to Reference a Property Value from Another Selector.....	9-10
9.5.2	What Happens When You Reference a Property Value from Another Selector.....	9-11
<b>10</b>	<b>Working with Style Classes</b>	
10.1	About Style Classes .....	10-1
10.2	Creating a Style Class.....	10-2
10.2.1	How to Create a Style Class.....	10-2
10.2.2	What Happens When You Create a Style Class.....	10-3
10.3	Modifying a Style Class .....	10-3
10.3.1	How to Modify a Style Class .....	10-3
10.4	Configuring a Style Class for a Specific Instance of a Component .....	10-3
10.4.1	How to Configure a Style Class for a Specific Instance of a Component.....	10-4
10.4.2	What Happens When You Configure a Style Class for a Specific Instance of a Component .....	10-4
<b>11</b>	<b>Working with At-Rules</b>	
11.1	About At-Rules in the ADF Skinning Framework.....	11-1
11.2	Working with Server-Side At-Rules.....	11-2
11.3	Working with Client-Side At-Rules .....	11-5
11.4	Creating At-Rules in an ADF Skin .....	11-7
11.4.1	How to Create an At-Rule .....	11-7
11.4.2	What Happens When You Create an At-Rule.....	11-8
11.4.3	What Happens at Runtime: How the ADF Skinning Framework Applies At-Rules .....	11-9
<b>12</b>	<b>Applying the Finished ADF Skin to Your Web Application</b>	
12.1	About Applying a Finalized ADF Skin to an Application.....	12-1
12.2	Testing Changes in Your ADF Skin .....	12-1

12.2.1	How to Set Parameters for Testing Your ADF Skin .....	12-4
12.2.2	What Happens When You Set Parameters for Testing Your ADF Skin.....	12-4
12.3	Packaging an ADF Skin into an ADF Library JAR .....	12-4
12.3.1	How to Package an ADF Skin into an ADF Library JAR .....	12-5
12.3.2	What Happens When You Package an ADF Skin into an ADF Library JAR .....	12-5
12.4	Applying an ADF Skin to Your Web Application .....	12-6
12.4.1	How to Apply an ADF Skin to an Application.....	12-7
12.4.2	What Happens When You Apply an ADF Skin to an Application.....	12-7
12.5	Applying an ADF Skin to a Running Web Application.....	12-7
12.5.1	How to Configure your Web Application to Accept an Updated ADF Skin .....	12-7
12.5.2	How to Deploy an ADF Library JAR to an MBean Server .....	12-8
12.5.3	What Happens When You Apply an ADF Skin to a Running Application.....	12-10

## 13 Advanced Topics

13.1	Referring to URLs in an ADF Skin's CSS File .....	13-1
13.2	Configuration Files for an ADF Skin .....	13-2
13.3	ADF Skins Provided by Oracle ADF.....	13-3
13.4	Versioning ADF Skins .....	13-3
13.4.1	How to Version an ADF Skin .....	13-4
13.4.2	What Happens When You Version ADF Skins.....	13-4

---

# Preface

Welcome to *Developing ADF Skins*.

## Audience

This document is intended for application developers and user interface designers who want to change the look and feel of their application by skinning ADF Faces Rich Client components.

## 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 that have purchased support 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.

## Related Documents

For more information, see the following documents:

- *Developing Web User Interfaces with Oracle ADF Faces*
- *Tag Reference for Oracle ADF Faces*
- *Tag Reference for Oracle ADF Faces Skin Selectors*
- *Tag Reference for Oracle ADF Faces Data Visualization Tools*
- *Tag Reference for Oracle ADF Data Visualization Tools Skin Selectors*

## 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.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

---



---

# About Skinning a Web Application

This chapter introduces you to developing an ADF skin. It provides an overview of the process of developing an ADF skin, takes a look at some of the changes that an ADF skin can implement, and describes the inheritance relationship of the ADF skins that Oracle ADF provides for you to extend from.

This chapter includes the following sections:

- [Introduction to Skinning a Web Application](#)
- [Overview of Developing an ADF Skin](#)
- [Taking a Look at an ADF Skin](#)
- [Inheritance Relationship of the ADF Skins Provided by Oracle ADF](#)

For definitions of unfamiliar terms found in this and other books, see the Glossary.

## 1.1 Introduction to Skinning a Web Application

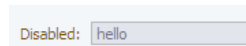
Skinning refers to the task of developing an ADF skin to apply to a web application that uses **ADF Faces** and **ADF Data Visualization components** in the user interface. An **ADF skin** uses the format, properties, and selectors of cascading style sheets (CSS) to allow you to customize the appearance of these components. Instead of providing a CSS file for each component, or inserting a style sheet on each page of the application, you create one ADF skin for the web application. Every component that renders in the user interface automatically uses the styles defined by the ADF skin. This means you do not have to make design-time changes to individual pages to change their appearance when you use an ADF skin.

Using an ADF skin also makes it easy for you to maintain a consistent appearance for all the pages that the application renders. Changes to the appearance of your application can easily be made should you decide to do so. You might decide, for example, to change colors to make your application adhere to your company's corporate brand. Additionally, you may want to define a style property for a component to make your application more usable. For example, [Figure 1-1](#) shows an ADF Faces `inputText` component.

**Figure 1-1** *Writable `inputText` Component*

Enter a value:

[Figure 1-2](#) shows another ADF Faces `inputText` component where the background color is grayed out by the ADF skin to indicate to the end user that the `inputText` component is read only.

**Figure 1-2 Read-Only inputText Component with Grayed-Out Background Color**

Other benefits of skinning include the ability to easily change the default text labels that ADF Faces components render at runtime. For example, the default text for the `dialog` component's labels are **OK** and **Cancel** if you set the component's `type` property to `okCancel`. You cannot modify the values of these labels by specifying properties for the `dialog` component. Instead, if you want to change **OK** to **Submit**, for example, you make changes in the ADF skin that references a **resource bundle** with the alternative string value. For more information, see [Working With Text in an ADF Skin](#).

The previous examples illustrate some of the use cases for ADF skins plus the benefits of creating an ADF skin. Note that you do not have to define all the changes that you want for your application in one ADF skin. You can create different ADF skins to serve different purposes. For example, you might create ADF skins with different color schemes to adhere to the corporate brand of different companies. In addition, you can configure an application so that the skin changes dynamically (for example, based on the role of the end user, or in response to a selection by the end user).

Oracle ADF provides a number of tools and resources to assist you in the task of creating an ADF skin. [Overview of Developing an ADF Skin](#) briefly describes the tools that you can use while [ADF Skins Provided by Oracle ADF](#) describes the ADF skins that Oracle ADF provides as a starting point for your ADF skin creation project. A new web application that you create in this release uses the Alta skin by default. Migrated web applications continue to use their existing ADF skin. To get the full benefit of the Oracle Alta UI system, Oracle recommends that you go beyond simply using the Alta skin and design your application around the Oracle Alta UI Design Principles. Designing your application using these principles enables you to make use of the layouts, responsive designs and components the Oracle Alta UI system incorporates to present content to your end users in a clean and uncluttered way. For more information about the Oracle Alta UI system and the Oracle Alta UI Design Principles, see <http://www.oracle.com/webfolder/ux/middleware/alta/index.html> and for information about Oracle Alta UI Patterns, see <http://www.oracle.com/webfolder/ux/middleware/alta/patterns/index.html>.

Note that this guide makes the following assumptions:

- You are familiar with the ADF Faces and ADF Data Visualization components that you can skin. The usage and functionality of these components is beyond the scope of this guide. For more information about these components, see *Developing Web User Interfaces with Oracle ADF Faces*.
- You are familiar with CSS. It is beyond the scope of this guide to explain CSS. For extensive information about CSS, including the official specification, visit the World Wide Web Consortium (W3C) web site at:

<http://www.w3.org/>

## 1.2 Overview of Developing an ADF Skin

Developing an ADF skin is an iterative process. Before you proceed, familiarize yourself with the concepts of CSS plus the ADF Faces and ADF Data Visualization components.

Oracle ADF provides two tools to help you to develop an ADF skin: the Theme Editor and JDeveloper's editor for ADF skins. The Theme Editor is a web application where

you can change the style properties of the most frequently-skinned ADF Faces components. You can view the results of the changes you make immediately within a preview pane of the same browser page where you make the style property changes.

The goal of the Theme Editor is to assist you accomplish the majority of your skinning tasks. For those skinning tasks that you cannot accomplish using the Theme Editor, we recommend that you export the ADF skin from the Theme Editor to an ADF Library JAR once you complete the tasks that can be completed in the Theme Editor. Using the exported ADF Library JAR, you can complete the remainder of your skinning project by creating an ADF skin that extends from the exported Theme Editor's ADF skin using the editors for ADF skins in JDeveloper. Examples of tasks that require you to use the editors for ADF skins in JDeveloper include styling DVT components, writing alternative text strings for the default labels that ADF Faces components render, and specifying at-rules in your ADF skin to determine the look and feel if the web application page renders in a specific browser or on a device.

---

---

**Note:** You cannot import an ADF skin into the Theme Editor and the Theme Editor user interface refers to an ADF skin as a "theme."

---

---

The high level steps to develop a theme (ADF skin) in the Theme Editor are:

1. Create an ADF skin using the Theme Editor.
2. Modify the style properties for the ADF Faces components that the Theme Editor enables you to customize using the provided controls.
3. Review your changes in the provided preview pages.
4. If satisfied or if you have completed all tasks that can be accomplished using the Theme Editor, export your completed ADF skin to an ADF Library JAR.

For more information about these steps, see [Working with the Theme Editor](#).

5. Choose the appropriate option:
  - Import the ADF Library JAR into a JDeveloper project from where you can extend the ADF skin to complete any remaining tasks you were unable to complete using the Theme Editor. For more information, see [Adding ADF Skins from an ADF Library JAR](#).
  - Distribute your completed ADF skin for use in the intended web applications. For more information, see [Applying an ADF Skin to Your Web Application](#).

The high level steps to develop an ADF skin in JDeveloper are:

1. Create an ADF skin in JDeveloper.

You create an ADF skin in JDeveloper where you write the declarations for the selectors that the **ADF skinning framework** exposes. When creating an ADF skin in JDeveloper, you must choose an existing ADF skin to extend from. If this ADF skin is the first ADF skin that you create, you choose from one of the ADF skins that Oracle ADF provides. For more information, see [ADF Skins Provided by Oracle ADF](#). For information about the inheritance relationship between these ADF skins, see [Inheritance Relationship of the ADF Skins Provided by Oracle ADF](#). If you create subsequent ADF skins, you can choose to extend from an ADF skin that you created previously.

For more information about creating an ADF skin, see [Creating an ADF Skin File](#).

2. Write declarations for the selectors, rules, and pseudo-elements in the ADF skin that you created.

The editor for ADF skins in JDeveloper provides a number of tabs that facilitate this task.

---

---

**Note:**

The Design tab only appears if you extend your ADF skin from the Skyros ADF skin. The Theme Editor (described previously) enables you to edit and preview changes that you make to an Alta or Skyros skin in a browser page. The selectors editor appears irrespective of the skin family that you extend from. For more information, see [Working with the ADF Skin Design Editor](#) and [Working with the ADF Skin Selectors Editor](#).

---

---

For a description of the different categories of selectors, rules, and pseudo-elements, see [Working with ADF Skin Selectors](#).

3. If applicable, import images that you want your ADF skin to reference at runtime in the **web application**. For more information, see [Working with Images and Color in Your ADF Skin](#).
4. If applicable, override the default text labels defined for the ADF Faces and ADF Data Visualization components by entering new values in a resource bundle. For more information, see [Working With Text in an ADF Skin](#).
5. Preview and test the changes that you made to the ADF skin to verify that the results are what you want. Modify the ADF skin as necessary. The Design tab described in Step 2 provides a number of sample pages where you can view your changes within the editor for ADF skins in JDeveloper or within a browser by clicking the **Preview in Browser** icon, as described in [Working with the ADF Skin Design Editor](#). For information about previewing and testing an ADF skin in a running web application, see [Testing Changes in Your ADF Skin](#).
6. Once you complete development of the ADF skin, you may want to package it for distribution. For more information, see [Packaging an ADF Skin into an ADF Library JAR](#).
7. Having completed the ADF skin and distributed it, you configure your web application so that it uses it. For more information, see [Applying an ADF Skin to Your Web Application](#).

## 1.3 Taking a Look at an ADF Skin

An ADF skin is a type of cascading style sheet. It differs from a cascading style sheet in a number of ways. One way it differs is that you can specify properties for the selectors that the ADF skinning framework exposes in the source file for the ADF skin. A selector exposed by the ADF skinning framework is similar to a CSS selector in that it identifies the ADF Faces and ADF Data Visualization components for which you want to change the appearance and allows you to specify one or more style properties for the component.

A selector exposed by the ADF skinning framework differs from a CSS selector in that it allows you to set values both for CSS properties and ADF skin properties exposed by the ADF skinning framework. CSS properties are interpreted directly by the end user's browser. ADF skin properties are prefaced by the characters `-tr-`. Some of

these ADF skin properties are read and interpreted by the web application. These properties are also known as server-side properties. A component that renders in the user interface may read these properties before it decides what to render. Other types of ADF skin properties, for example `-tr-rule-ref` or `-tr-property-ref`, enhance the capabilities of the ADF skinning framework, as described in [Properties in the ADF Skinning Framework](#).

The following example shows the selector for the `gauge` component that sets values for the ADF skin properties `-tr-graphic-antialiasing` and `-tr-animation-indicators`, plus the CSS properties `background-color` and `font-family`.

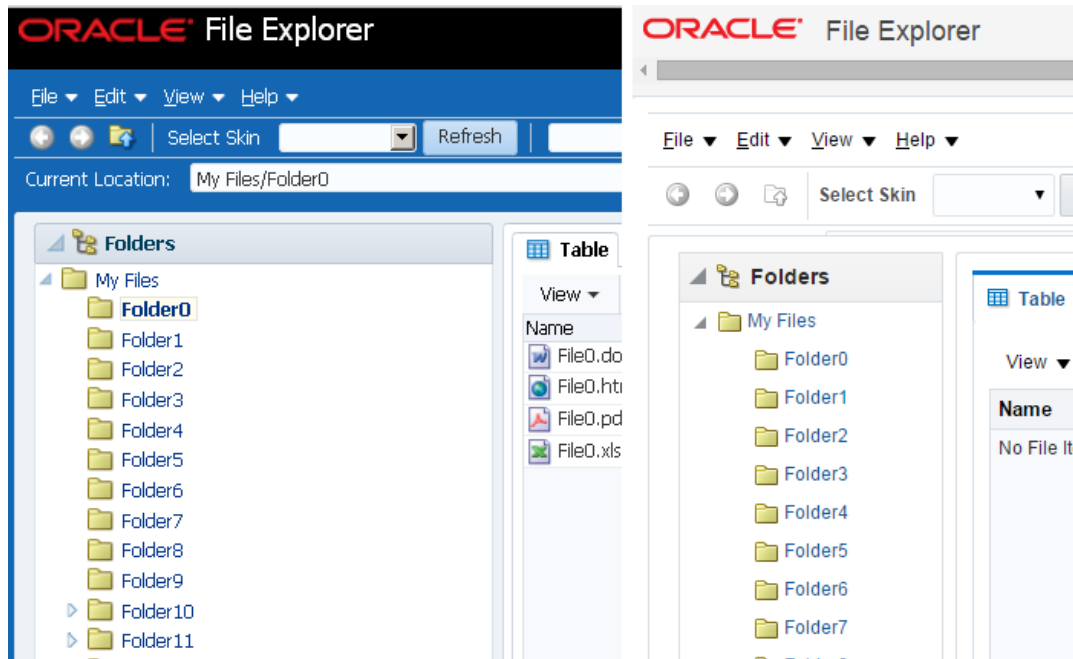
```
af|dvt-gauge
{
  /** ADF skin properties */
  -tr-graphic-antialiasing: false;
  -tr-animation-indicators: none;
  /** CSS properties */
  font-family: Geneva, Arial, Helvetica, sans-serif;
  background-color: rgb(243,255,185);
}
```

As the previous example demonstrates, you can set values for CSS properties and ADF skin properties within the declaration of a selector exposed by the ADF skinning framework. The ADF skinning framework exposes the ADF skin properties that you can define. In addition to ADF skin properties, the ADF skinning framework defines a number of pseudo classes and at-rules that you can specify in an ADF skin. Examples of supported at-rules and pseudo classes include `@platform`, `@agent`, `@accessibility-profile`, `:rtl`, and `@locale`. For more information, see [Working with ADF Skin Selectors](#).

At runtime, the web application creates a browser-specific CSS file from the ADF skin. The application then references this browser-specific CSS file as it would any CSS file.

[Figure 1-3](#) demonstrates the impact that an ADF skin can have on the appearance of an application's page. The page on the left renders using the `skyros` ADF skin. The page on the right renders using the `alta` skin. Each ADF skin defines different values for colors and fonts.

**Figure 1-3 File Explorer Application Using the Skyros ADF Skin and the Alta Skin**



**Note:**

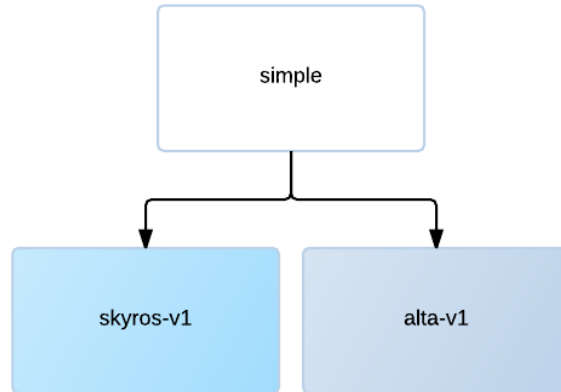
An ADF skin can affect the time it takes a client to render the user interface. The more styles that an ADF skin uses, the more the client has to load. This can affect performance in low bandwidth or high latency environments.

## 1.4 Inheritance Relationship of the ADF Skins Provided by Oracle ADF

Oracle ADF provides a number of ADF skin families that you can use in your application or extend when you create an ADF skin. The ADF skins provided by Oracle ADF offer increasing levels of customization for the appearance rendered by ADF Faces and ADF Data Visualization components at runtime.

Figure 1-4 shows the inheritance relationship between the different ADF skin families. The `skyros-v1` and `alta-v1` ADF skin families both extend from the `simple` ADF skin.

All ADF Faces components use, at a minimum, styles defined in the `simple` ADF skin as this is the skin from which the other ADF skins extend. The `simple` ADF skin defines the minimum style properties that ADF Faces components require to render in a web application. If you want to create an ADF skin with a minimal amount of customization, you create an ADF skin that extends from the `simple` ADF skin.

**Figure 1-4 Inheritance Relationship of ADF Skin Families Provided by Oracle ADF**

You can apply any of the ADF skins in [Figure 1-4](#) or an ADF skin that you create yourself to an application. For more information about applying an ADF skin to an application, see [Applying an ADF Skin to Your Web Application](#).

For a more detailed description of the ADF skins that Oracle ADF provides, see [ADF Skins Provided by Oracle ADF](#).





---

## Working with the Theme Editor

This chapter provides information to help you to configure the environment required to use the Theme Editor to create and modify ADF skins.

This chapter includes the following sections:

- [About the Theme Editor](#)
- [Setting Up and Starting the Theme Editor](#)
- [Exporting a Theme from the Theme Editor](#)

### 2.1 About the Theme Editor

The Theme Editor helps you to define the look and feel of your web application by selecting the colors, fonts, and other style properties that you want ADF Faces components to render at runtime. The end result of this process is the creation of an ADF skin.

---

---

**Note:**

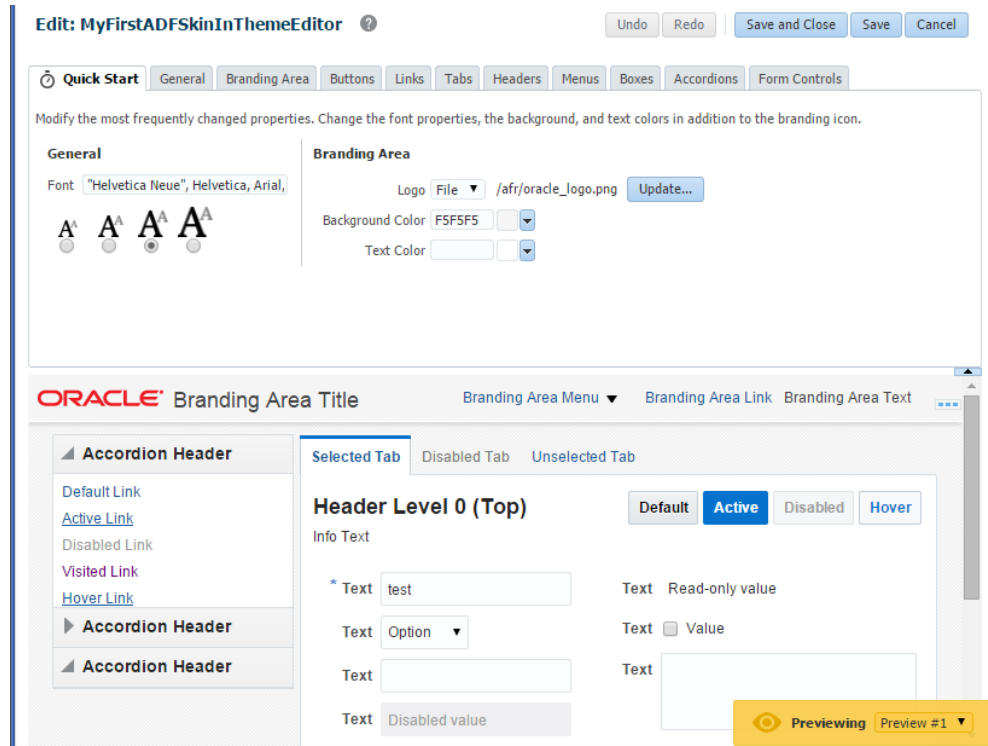
The Theme Editor's user interface refers to an ADF skin as a "theme".

---

---

Once you complete the creation of the theme, you can export it from the Theme Editor as an ADF skin to an ADF Library JAR file. Use this file to apply the ADF skin that you created in the Theme Editor to your web application. For more information, see [Applying the Finished ADF Skin to Your Web Application](#).

If you want to customize the ADF skin further and the Theme Editor does not provide the UI controls to achieve the result you want, import the ADF Library JAR into a project in JDeveloper where you can use the design-time that JDeveloper provides or JDeveloper's source editor for the CSS source file of an ADF skin to create an ADF skin that extends from the ADF skin you created in the Theme Editor. For more information, see [Adding ADF Skins from an ADF Library JAR](#).

**Figure 2-1 Theme Editor's Quick Start Page**

## 2.2 Setting Up and Starting the Theme Editor

Oracle ADF delivers the Theme Editor as a web application in the following EAR file in your JDeveloper installation:

```
\jdeveloper_install_dir\jdeveloper\skineditor\skin-editor-webapp.ear
```

To use the Theme Editor, you create a new application in JDeveloper using the **Application from EAR File** option in JDeveloper's New Gallery and deploy the Theme Editor to the Integrated WebLogic Server. For more information, see [How to Set Up the Theme Editor](#).

Once you create the new application with the Theme Editor in JDeveloper, you probably want to edit the application's `web.xml` file so that any ADF skins you create in the Theme Editor persist beyond a stop and restart of the Integrated WebLogic Server. For more information, see [How to Persist ADF Skins Created in the Theme Editor](#).

After you configure the Theme Editor to persist ADF skins, you can start it, as described in [How to Start the Theme Editor](#).

As the Theme Editor is packaged in an EAR file, you can deploy it to Integrated WebLogic Server that is installed with JDeveloper. You can also deploy it using Enterprise Manager, Oracle WebLogic Scripting Tool (WLST), or Oracle WebLogic Server Administration Console. For more information about deployment options, see the Deploying Fusion Web Applications chapter in *Developing Fusion Web Applications with Oracle Application Development Framework*.

## 2.2.1 How to Set Up the Theme Editor

You set up the Theme Editor by creating a new application in JDeveloper where you import the Theme Editor from the `skin-editor-webapp.ear` file in the `skineditor` directory of your JDeveloper installation.

How to set up the Theme Editor:

1. In the main menu, choose **File** and then **Application > New**.
2. In the New Gallery, in the **Items** list, double-click **Application from EAR file**.
3. In the **Application from EAR File** wizard, enter the location of the EAR file or click **Browse** to navigate to the `skin-editor-webapp.ear` file in the `skineditor` directory of your JDeveloper install directory.

For additional help with the wizard, click **Help**.

4. Click **Finish**.

## 2.2.2 How to Persist ADF Skins Created in the Theme Editor

The ready-to-use Theme Editor that Oracle ADF provides in the `skin-editor-webapp.ear` file in the `skineditor` directory of your JDeveloper install directory does not persist any ADF skins that you create if you stop and restart the Integrated WebLogic Server where you deploy the Theme Editor. You specify a file directory location in the `web.xml` file of the Theme Editor web application. The Theme Editor then saves ADF skins you create (and their resources) to this location so that they will be available after an Integrated WebLogic Server restart. You can view these resources in the file directory you specify, but you must export the ADF skin, as described in [Exporting an ADF Skin from the Theme Editor](#), if you want to extend the ADF skin in JDeveloper or distribute the ADF skin for use in a web application.

To persist ADF skins created in the Theme Editor:

1. In the Applications window, double-click the **web.xml** file in the `skin-editor` project.
2. In the source editor, add the following context initialization parameter entries:

```
<context-param>
  <description>Set this context parameter to file so that themes get saved to a
  temporary directory. Specify a directory location for
  oracle.adf.view.rich.SKIN_REPOSITORY_FILE_PATH to persist changes between server
  restarts.</description>
  <param-name>oracle.adf.view.rich.SKIN_REPOSITORY</param-name>
  <param-value>file</param-value>
</context-param>

<context-param>
  <description>Set this context parameter to a directory location where
  themes are saved. Use to persist changes between server restarts</description>
  <param-name>oracle.adf.view.rich.SKIN_REPOSITORY_FILE_PATH</param-name>
  <param-value>/home/user/themes</param-value>
</context-param>
```

3. Save and close the `web.xml` file.

### 2.2.3 How to Start the Theme Editor

Start the Theme Editor by running the `index.html` page in the `skin-editor` project.

To start the Theme Editor:

- In the Applications window, expand **skin-editor**, right-click `index.html` and choose **Run**.

The Create Default Domain dialog appears the first time you run an application and start a new domain in Integrated WebLogic Server. Use the dialog to define an administrator password for the new domain. Passwords you enter can be eight characters or more and must have a numeric character.

The `index.html` page within the `skin-editor` project is the run target. When you run the page, JDeveloper starts a browser and displays the launch page of the Theme Editor. Once the Theme Editor launch page appears, you can create new ADF skins or edit existing ADF skins.

---

---

**Note:**

The Theme Editor's user interface refers to an ADF skin as a "theme".

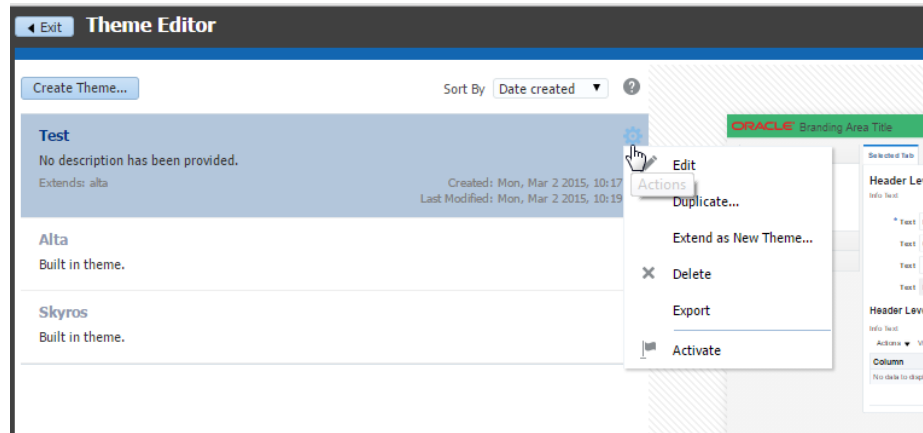
---

---

## 2.3 Exporting an ADF Skin from the Theme Editor

Once you complete the creation of an ADF skin in the Theme Editor, you may want to export it to an ADF Library JAR so that you can distribute it for use in a web application. Alternatively, you may want to edit it further using the design-time tools that JDeveloper provides for this purpose. This latter scenario may arise if you cannot achieve the look and feel you want using the controls provided by the Theme Editor. If you want to edit an ADF skin using JDeveloper's design-time tools, you import the ADF skin into JDeveloper, as described in [Adding ADF Skins from an ADF Library JAR](#), and create a new ADF skin that extends from the ADF skin you exported from the Theme Editor.

You select the theme (ADF skin) in the Theme Editor's launch page and choose **Export** from the menu that appears when you click the **Actions** button, as shown in [Figure 2-2](#). The Theme Editor exports the ADF skin to an ADF Library JAR. This ADF Library JAR contains all required resources for the ADF skin, such as a `.CSS` file with entries that reflect the changes you made in the Theme Editor, images that you imported to the ADF skin, and a `trinidad-skins.xml` file that contains metadata describing the ADF skin.

**Figure 2-2** *Exporting a Theme from the Theme Editor*



---

## Working with ADF Skin Selectors

This chapter describes the ADF skin selectors. These selectors along with pseudo-elements, pseudo-classes, ADF skin properties and ADF skinning framework rules allow you to customize the appearance of ADF Faces and ADF Data Visualization components.

This chapter includes the following sections:

- [About ADF Skin Selectors](#)
- [Pseudo-Classes in the ADF Skinning Framework](#)
- [Properties in the ADF Skinning Framework](#)
- [Accessing Selector Information from Within JDeveloper](#)

### 3.1 About ADF Skin Selectors

CSS uses selectors to determine the elements in a HTML page you that you define rules for. For example, in CSS the following selector defines a rule that determines the appearance of the content that renders in a <p> tag:

```
p { color: red }
```

Likewise, the **ADF skinning framework** defines selectors that allow you to specify rules with the style properties to render at runtime when the rule encounters the specified tag. The ADF skinning framework provides two types of selector: global selectors and component-specific selectors. A global selector defines style properties that you apply to one or more selectors. A component-specific selector defines style properties that apply to one component.

The ADF skins provided by Oracle ADF define many global selectors (**Global Selector Aliases** in the user interface of the selectors editor) that many ADF Faces components inherit. For example, many ADF Faces components use the `.AFDefaultFontFamily:alias` global selector to specify the font family. If you create an **ADF skin** that overrides this selector by specifying a different font family, that change affects all the components that have included the `.AFDefaultFontFamily:alias` selector in their selector definition.

**Figure 3-1** shows two instances of the same page. The instance of the page in the lower part of **Figure 3-1** renders using the default values specified for the `.AFDefaultFontFamily:alias` global selector in the skyros skin. The instance of the page in the upper part of **Figure 3-1** renders using an ADF skin that modifies the `.AFDefaultFontFamily:alias` and `.AFDefaultFont` global selectors as follows:

```
.AFDefaultFontFamily:alias {font-family: Georgia;}  
.AFDefaultFont:alias {font-size: 12pt;}
```

The components on the page that render text (for example, `af:showDetailItem` renders Welcome and an `af:link` component renders Login) do so using the font family specified by the `.AFDefaultFontFamily:alias` global selector in the ADF skin that the application uses.

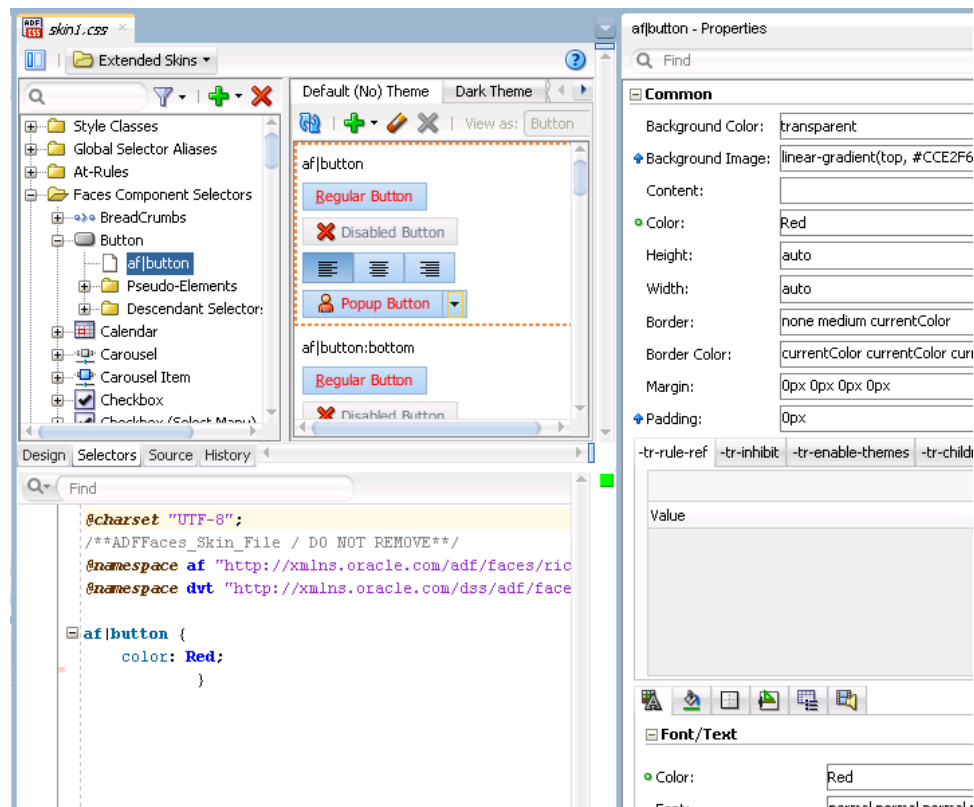
**Figure 3-1 Global Selector**



An ADF skin that you create inherits the global selector aliases defined in the ADF skin that it extends from. You can also create new global selector aliases in your ADF skin files. For more information, see [Working With Global Selector Aliases](#).

Component-specific selectors are selectors that the ADF skinning framework exposes that allow you to identify the corresponding **ADF Faces** and **ADF Data Visualization components** for which you can define style properties. For example, [Figure 3-2](#) shows the selector for the ADF Faces `button` component in the source editor and selectors editor. The value of the property that determines the color of the font to appear in the button has been changed to Red in the Properties window.

**Figure 3-2 Button Component's Skin Selector**



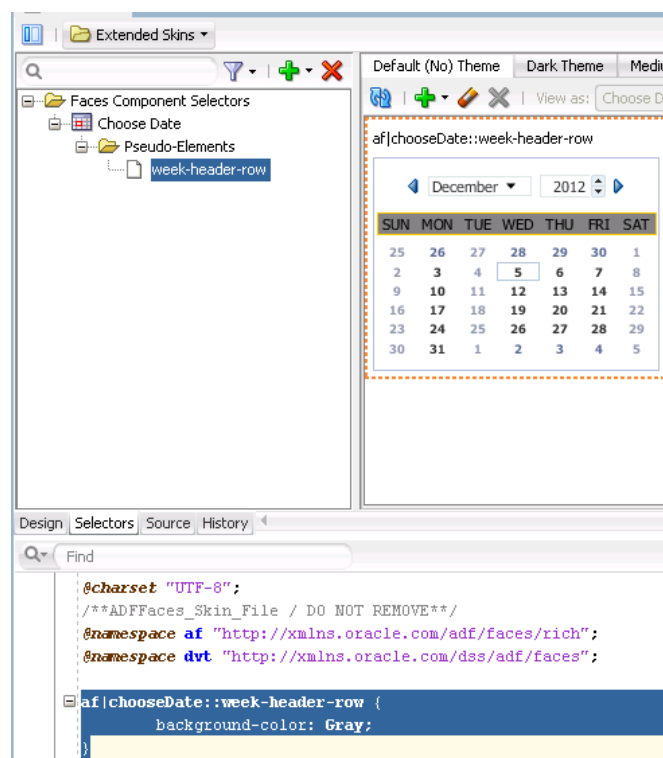


For more information about the component-specific selectors, see [Working with Component-Specific Selectors](#). For more information about global selector aliases, see [Working With Global Selector Aliases](#).

### 3.1.1 ADF Skin Selectors and Pseudo-Elements

Many ADF skin selectors expose pseudo-elements. A pseudo-element denotes a specific area of a component for which you can define style properties. Pseudo-elements are denoted by a double colon followed by the portion of the component the selector represents. For example, [Figure 3-3](#) shows how the `week-header-row` pseudo-element exposed by the `af|chooseDate` selector allows you to configure style properties for the appearance of the calendar grid. In [Figure 3-3](#), the `background-color` property of the `week-header-row` pseudo-element has been set to Gray.

**Figure 3-3 Pseudo-Elements for the Choose Date Component**



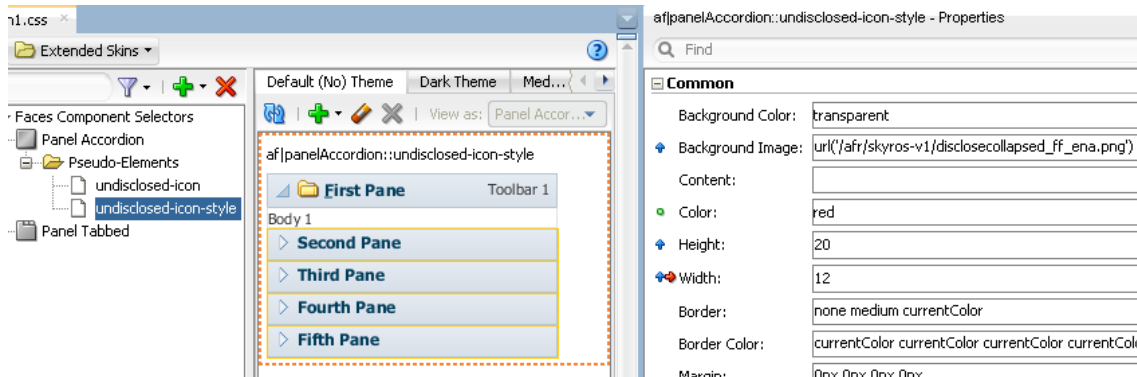
### 3.1.2 ADF Skin Selectors and Icon Images

ADF Faces components that render icons do so using a set of base icon images. No CSS code entries appear in the source file of the ADF skin for these icon images in contrast to, for example, the CSS code entries that appear in a source file when you specify an image as a value for the CSS `background-image` property. Instead, the ADF skinning framework registers the icon image for use with the renderer.

ADF skin selectors use two naming conventions for pseudo-elements that identify icon images that render in a component. The names of these pseudo-elements end in `-icon` or in `-icon-style`. [Figure 3-4](#) shows the example of the Panel Accordion selector's pseudo-elements. Pseudo-elements that end in `-icon-style` specify a background image, as shown in [Figure 3-4](#). In contrast, pseudo-elements that end in `-icon` do not specify a background image, but can reference `IMG` elements or text, as in the following examples:

```
af|panelAccordion::undisclosed-icon {content "X"}
af|panelAccordion::undisclosed-icon {content: url("http:server:port/img/img.png")}
```

**Figure 3-4 Panel Accordion Pseudo-Elements for Icon Images**



In [Figure 3-4](#), the `undisclosed-icon-style` pseudo-element styles the icon used for the undisclosed icon in the `panelAccordion` component. This pseudo-element specifies the icon as a background image. Use the `:hover` and `:active` pseudo-classes to customize the appearance. For example, write the following syntax to make the background red if the end user hovers the mouse over the icon:

```
af|panelAccordion::undisclosed-icon-style:hover {
    background-color: Red;
}
```

**Tip:**

Many browsers do not render background images when in accessibility mode. The following example demonstrates how you can work around this behavior if you want your application to display an image when in accessibility mode.

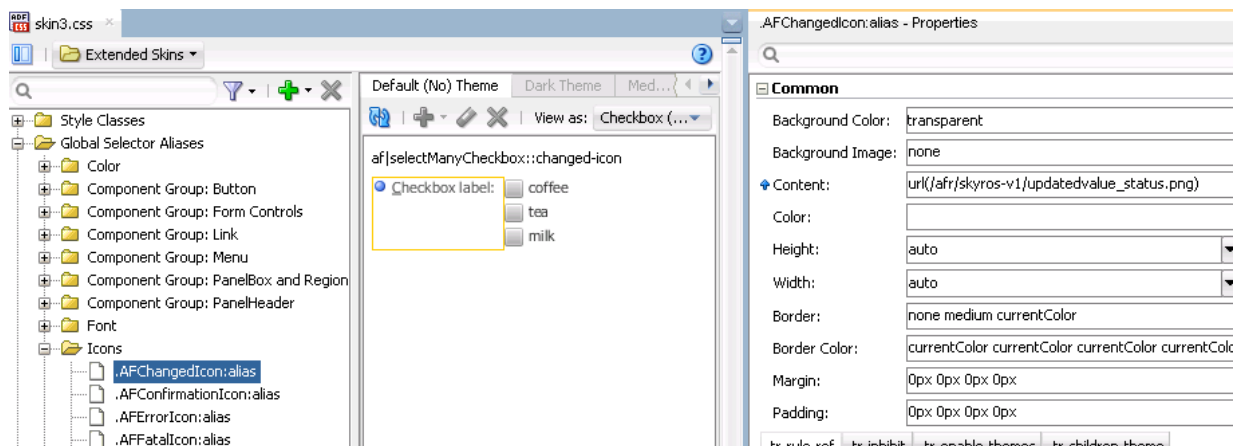
If you want to use your own image rather than the icon specified as a background image, you need to first prevent the background image from rendering. Do this by specifying the `-tr-inhibit` ADF skin property for the component's selector pseudo-element as follows:

```
af|panelAccordion::undisclosed-icon-style
{
    -tr-inhibit: background-image;
}
```

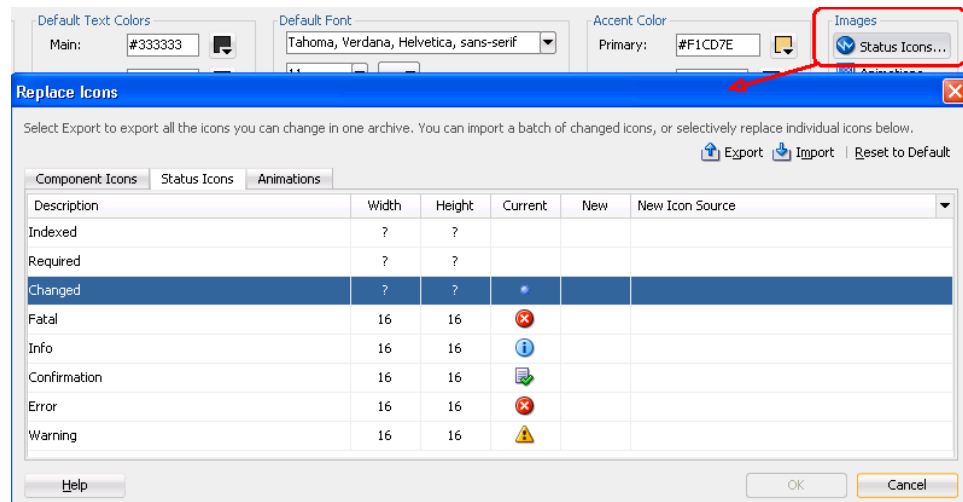
Next you specify the text or image that you want to render as a value for the `content` property of the `undisclosed-icon` selector. For example, write syntax similar to the following to specify an alternative image:

```
af|panelAccordion::undisclosed-icon
{
    content:url("images/undisclosed.png");
    width: 10px;
    height: 10px;
}
```

The ADF skinning framework also defines a number of global selector aliases that specific icon images to use in different scenarios. These global selector aliases appear under the **Icons** node in the **Selector Tree** of the selectors editor, as shown in [Figure 3-5](#). The `.AFChangedIcon:alias` shown in [Figure 3-5](#) enables you to globally set the changed icon for all components using that icon.

**Figure 3-5 Global Selector Aliases for Icons**

These icons can also be viewed and changed using the Replace Icons dialog that you invoke from the design editor, as described in [Changing Images and Colors in the ADF Skin Design Editor](#), if your ADF skin extends from the Skyros ADF skin. [Figure 3-6](#) shows the dialog that appears for an ADF skin that extends from the Skyros ADF skin. Using the dialog, you can export or import multiple icons or replace an individual icon by double-clicking in the **New Icon Source** field.

**Figure 3-6 Design Editor's Replace Icons Dialog for Status Icons**

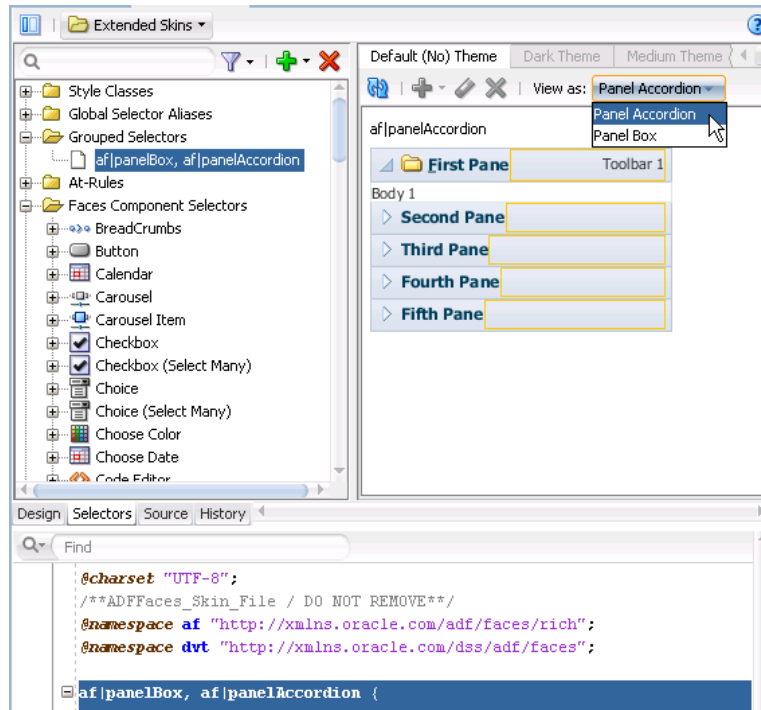
For more information, see [Working with Images and Color in Your ADF Skin](#).

### 3.1.3 Grouped ADF Skin Selectors

You can group ADF skin selectors and global selector aliases to minimize the number of entries in the source file of the ADF skin. The selectors that you group render under the Grouped Selectors node in the Selector Tree of the selectors editor, as shown in [Figure 3-7](#). The **View as** list in the Preview Pane displays all the selectors that you grouped.

As the selectors editor does not provide a way to specify grouped selectors, you use the source editor to specify the selectors and/or global selector aliases that you want to put in a grouped selector. Separate each selector by a comma (,) to include in the grouped selector.

Figure 3-7 Grouped Selectors in the Selector Tree



### 3.1.4 Descendant ADF Skin Selectors

A descendant selector defines style properties for one ADF skin selector (the descendant selector) to render when the selector's component is a descendant of another component in the page that renders the components. For example, assume that you want the content area of an `inputText` component to render using a background color of `Green` when the component renders inside a `table` component. In this scenario, you specify the descendant selector shown in the following example.

```

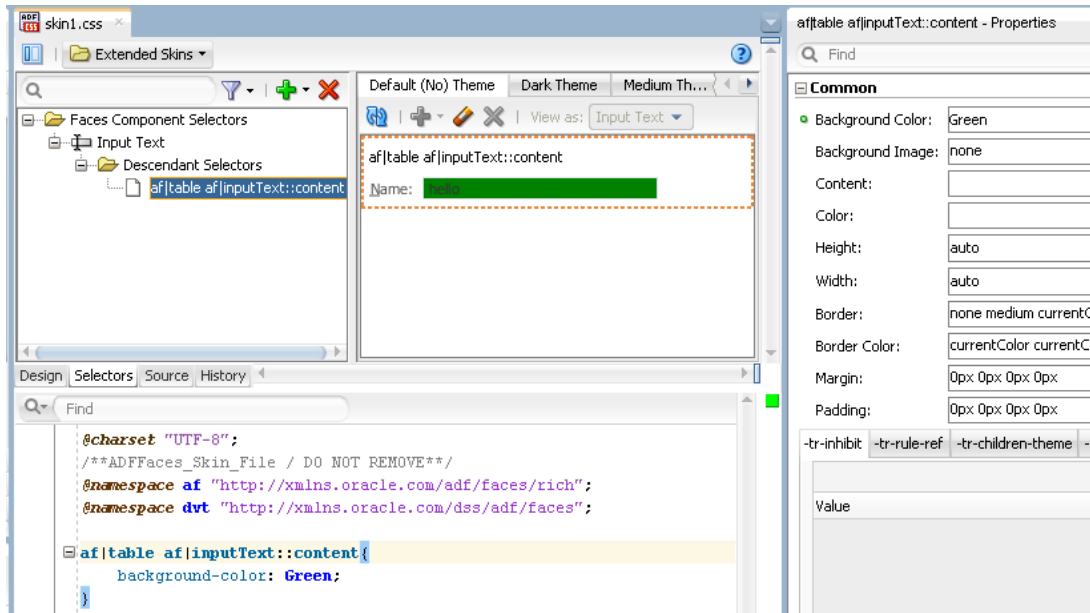
af|table af|inputText::content {
    background-color: Green;
}

af|inputText::content {
    background-color: Red;
}

```

At runtime, when the `inputText` component renders in a `table` component, the background color of the content area is `Green`. The appearance of other `inputText` components that render outside of `table` components is determined by the style properties defined elsewhere in the ADF skin (for example, `Red`).

A descendant selector is made up of two or more selectors separated by white space. When you configure a descendant selector, the selectors editor displays a Descendant Selectors node under the selector included in the descendant selector, as shown in [Figure 3-8](#).

**Figure 3-8** Descendant Selectors in the Selector Tree

As the selectors editor does not provide a way to specify descendant selectors, you use the source editor to specify the selectors and/or global selector aliases that you want to specify in a descendant selector. Separate each selector by a white space.

## 3.2 Pseudo-Classes in the ADF Skinning Framework

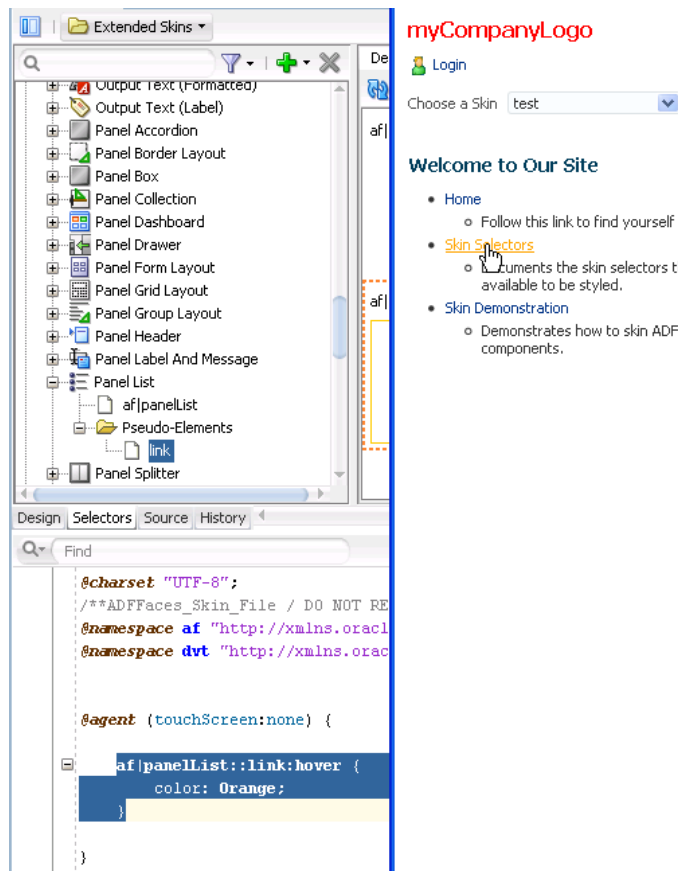
The CSS specification defines pseudo-classes, such as `:hover` and `:active`, which are used to define style properties for when a selector is in a particular state. You can apply these pseudo-classes to almost every ADF Faces component. In addition, the ADF skinning framework provides additional pseudo-classes for specialized functions. Examples include pseudo-classes to apply when a browser's locale is a right-left language (`:rtl`) or for drag and drop operations (`:drag-target` and `:drag-source`). The syntax that appears in the source file of an ADF skin to denote a pseudo-class uses the following format(s):

```
adfskinselector:pseudo-class
```

```
adfskinselector::pseudo-element:pseudo-class
```

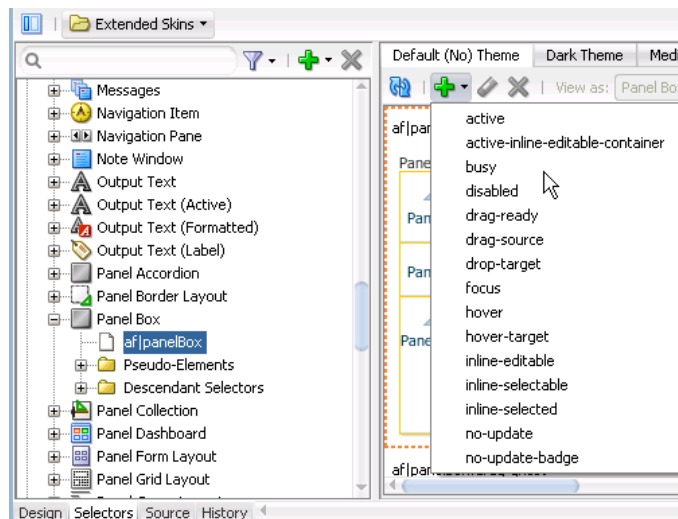
[Figure 3-9](#) shows the syntax that you write (`af|panelList::link:hover { color: Orange; }`) in the source file of an ADF skin for the `:hover` pseudo-class so that a `panelList` component's link renders orange when the end user hovers a cursor over the link in [Figure 3-9](#).

**Figure 3-9 Pseudo-Class Syntax and Runtime Behavior for a Panel List Link**



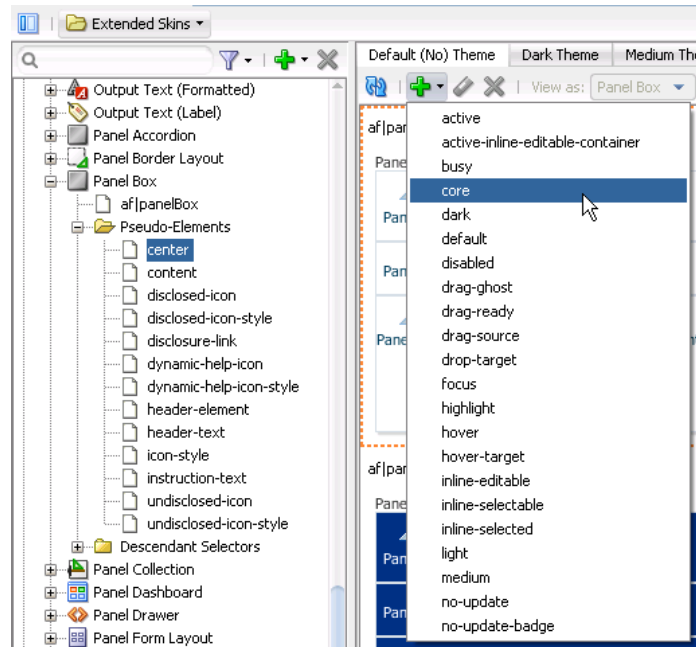
Some components make more use of pseudo-classes than other components. The `panelBox` component's selector, for example, makes extensive use of pseudo-classes to define its appearance when it is in various states (for example, active, disabled, or busy). [Figure 3-10](#) shows the list of available pseudo-classes that renders when you select the `panelBox` component's selector in the Selector Tree of the selectors editor and click the **Add Pseudo-Class** icon to display the list of available pseudo-classes in an ADF skin that extends from the Skyros ADF skin.

**Figure 3-10 Pseudo-classes for the panelBox Component's Selector**



Pseudo-classes can also be applied to pseudo-elements that selectors expose. The `panelBox` component selector's pseudo elements are a good example. Figure 3-11, of the Selector Tree in the selectors editor, shows the list of pseudo-classes that the `center` pseudo-element exposed by the `panelBox` component selector accepts. Many of these pseudo-classes allow you to define the appearance for the `panelBox` component depending on the value that the application developer sets for its attributes. For example, the `core` and `highlight` pseudo-classes define the corresponding appearance when the application developer sets the `panelBox` component's `ramp` attribute to `core` or `highlight`.

**Figure 3-11 Pseudo-classes for the center Pseudo-element**



The following are common pseudo-classes used by ADF Faces selectors.

- **Drag and drop:** The two pseudo-classes available are `:drag-source` applied to the component initiating the drag and removed once the drag is over, and `:drop-target` applied to a component willing to accept the drop of the current drag.
- **Standard:** In CSS, pseudo-classes like `:hover`, `:active`, and `:focus` are considered states of the component. This same concept is used in applying skins to components. Components can have states like `read-only` or `disabled`. When states are combined in the same selector, the selector applies only when all states are satisfied.
- **Right-to-left:** Use this pseudo-class to set a style or icon definition when the browser is in a right-to-left language. Another typical use case is asymmetrical images. You will want the image to be flipped when setting skin selectors that use the image in a right-to-left reading direction. Be sure to append the `:rtl` pseudo-class to the very end of the selector and point it to a flipped image file. The skin editor's preview pane does not render changes that you make to a flipped image file. The following example from the Skyros skin shows the image that the `calendar` component's `toolbar-day-hover-icon` pseudo-element references when it renders in a browser that uses a right-to-left language:

```
af|calendar::toolbar-day-hover-icon:rtl {
    content: url(/afr/cal_day_ovr_rtl.png);
}
```

```
width: 16px;  
height: 16px;  
}
```

You can also use `:rtl` to apply to skin icons. For more information, see [Working with Images and Color in Your ADF Skin](#).

- **Inline editing:** This pseudo-class is applied when the application activates a component subtree for editing in the browser. For example, `:inline-selected` is a pseudo-class applied to currently selected components in the active inline-editable subtree.
- **Message:** This pseudo-class is used to set component-level message styles using CSS pseudo-classes of `:fatal`, `:error`, `:warning`, `:confirmation`, and `:info`. For more information, see [Configuring ADF Skin Properties to Apply to Messages](#).

---

---

**Note:**

The global selector aliases that appear in the Selector Tree are a special type of pseudo-class (`:alias`). For more information, see [Working With Global Selector Aliases](#).

---

---

### 3.3 Properties in the ADF Skinning Framework

The ADF skinning framework defines a number of ADF skin properties. The **web application**, rather than the user's browser, interprets ADF skin properties. When configured, ADF skin properties enable you to do the following:

- Reference styles from other selectors with the `-tr-rule-ref` property.  
Create your own global selector alias and combine it with other selectors using the `-tr-rule-ref` property. For more information, see [Creating a Global Selector Alias](#), [Modifying a Global Selector Alias](#), and [Applying a Global Selector Alias](#).
- Suppress styles defined in an ADF skin with the `-tr-inhibit` skin property.  
Suppress or reset CSS properties inherited from a base skin with the `-tr-inhibit` skin property. For example, the `-tr-inhibit:padding` property removes any inherited padding. Remove (clear) all inherited properties with the `-tr-inhibit:all` property. The suppressed property name must be matched exactly with the property name in the base skin.
- Reference the value of a property defined in another selector using the `-tr-property-ref` property.  
For more information, see [Referencing a Property Value from Another Selector](#).
- Configure a theme for child components with the `-tr-children-theme` property.  
The Alta skin (and skins that extend from the Alta skin) do not use themes.
- ADF skin selectors with style properties.  
Skin style properties allow you to customize the rendering of a component throughout the application. A CSS property is stored with a value in the Skin object and is available when the component is being rendered. For example, in `af|breadcrumbs{-tr-show-last-item: false}`, the skin property `-tr-show-`



`last-item` is set to hide the last item in the `breadcrumbs` component's navigation path.

As already noted, ADF skin properties allow you to customize the rendering of the component throughout the application. This means that you cannot use ADF skin properties to customize specific instances of a component in your application by, for example, configuring an ADF skin property in a style class that an instance of a component then references using its `styleClass` attribute.

The ADF skinning framework also provides the `+` and `-` operators that allow you to set a selector's color or font properties relative to the value that you specify for the color or font properties of another selector. This is useful if you want to apply a range of colors to selectors or maintain a relative size between fonts.

[Example 3-1](#) demonstrates the syntax that you write to make use of this feature for a color property. A global selector alias defines the background color that another global selector alias (`.fooBackgroundColorTest`) applies using the `-` operator. [Example 3-1](#) also demonstrates the syntax that you write to make use of this feature for a font property. A global selector alias (`.FontSizeTest:alias`) defines the font size and `.fooFontTestIncrease` increases this font size by using the `+` operator.

[Figure 3-12](#) shows how the style classes defined in [Example 3-1](#) effect the runtime appearance of instances of the `af:outputLabel` components to which you apply the `fooFontTestIncrease` and `fooBackgroundColorTest` style classes by specifying these style classes as values for the component's `styleClass` attribute, as illustrated in the following example.

```
<af:outputLabel value="Increase font-size" id="ol2"
                styleClass="fooFontTestIncrease"/>
```

**Figure 3-12 Using Operators to Apply Color and Change Font Size**

Base background-color and font-size

Change background-color

Increase font-size

For more information about style classes, see [Working with Style Classes](#).

**Example 3-1 Using Operators to Apply Color and Change Font Size**

```
.FontSizeTest:alias {
    font-size: 30px;
}

.BaseBackgroundColor:alias {
    background-color: #0099ff;
}

.fooFontTestIncrease {
    -tr-rule-ref: selector(".FontSizeTest:alias");
    font-size: +20px;
}

.fooBackgroundColorTest {
    -tr-rule-ref: selector(".BaseBackgroundColor:alias");
    background-color: -#333333;
}
```

```
af|outputLabel {
  -tr-rule-ref: selector(".BaseBackgroundColor:alias");
  -tr-rule-ref: selector(".FontSizeTest:alias");
  color: Red;
}
```

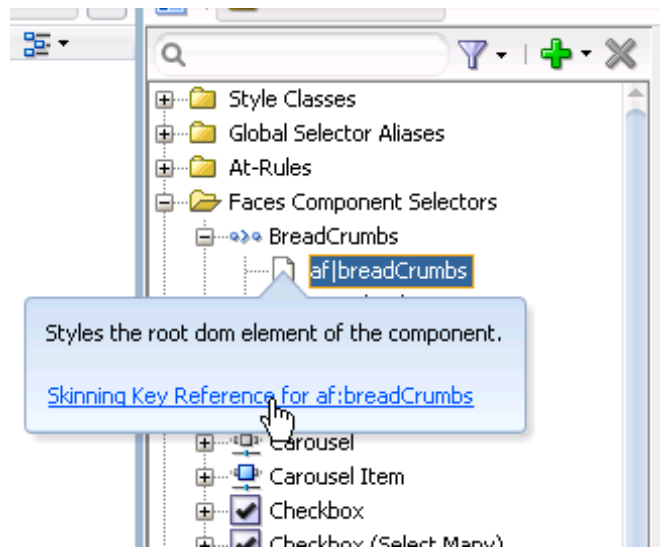
### 3.4 Accessing Selector Information from Within JDeveloper

You can access reference information for the ADF skin selectors and CSS properties that you configure in your ADF skin in a number of ways within the editor for ADF skins in JDeveloper. The reference information that you can access includes the following reference documents for ADF skin selectors:

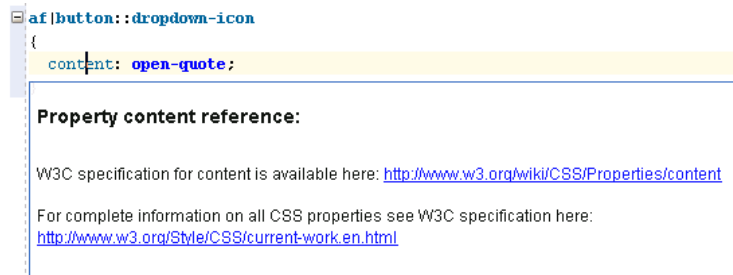
- *Tag Reference for Oracle ADF Faces Skin Selectors*
- *Tag Reference for Oracle ADF Data Visualization Tools Skin Selectors*

You can access these reference documents in the documentation library or in a Help Center window if you click the link in the information text that appears when you hover over a selector in the Selector Tree of the selectors editor, as shown in [Figure 3-13](#).

**Figure 3-13** Reference Documentation for ADF Skin Selectors



In addition to referencing information for the ADF skin selectors, you can access information for CSS selectors. You do this from the Source tab of the editor by selecting the CSS property and pressing Control + D or choosing **Show Quick Reference** from the context menu that appears when you right-click the selector, as illustrated in [Figure 3-14](#).

**Figure 3-14 Quick Reference Documentation for CSS Properties**

The screenshot shows a code editor with the CSS selector `af|button::dropdown-icon` and the property `content: open-quote;`. A yellow highlight is under the property. Below the code, a box titled "Property content reference:" contains two lines of text with blue hyperlinks: "W3C specification for content is available here: <http://www.w3.org/wiki/CSS/Properties/content>" and "For complete information on all CSS properties see W3C specification here: <http://www.w3.org/Style/CSS/current-work.en.html>".

```
af|button::dropdown-icon
{
  content: open-quote;
```

**Property content reference:**

W3C specification for content is available here: <http://www.w3.org/wiki/CSS/Properties/content>

For complete information on all CSS properties see W3C specification here: <http://www.w3.org/Style/CSS/current-work.en.html>



---

## Working with ADF Skins in JDeveloper

This chapter describes the editors for ADF skins that JDeveloper provides to create ADF skins. Key features of these editors, such as the Selector Tree that you use to browse the selectors that you can configure in an ADF skin, the Properties window that you use to set properties, and how you navigate to an ADF skin that you extend, are also described.

This chapter includes the following sections:

- [About the Editors for ADF Skins in JDeveloper](#)
- [Working with the ADF Skin Design Editor](#)
- [Working with the ADF Skin Selectors Editor](#)
- [Working with the Properties Window](#)
- [Navigating ADF Skins](#)

### 4.1 About the Editors for ADF Skins in JDeveloper

The editor for ADF skins in JDeveloper is a tool that creates ADF skins for applications built using Oracle ADF. It provides a number of visual and source editors where you edit the selectors exposed by the ADF skinning framework, preview your changes, and package the final **ADF skin** into an ADF Library JAR.

Key features of the editors for ADF skins in JDeveloper include the:

- ADF Skin Design Editor (design editor) where you can declaratively modify an ADF skin that extends from the Skyros ADF skin using the provided controls.
- ADF Skin Selector Editor (selectors editor) where you can view all of the selectors exposed by the ADF skinning framework in the **Selector Tree**.
- Properties window where you can modify the properties of the selectors that you choose in the Selector Tree.

### 4.2 Working with the ADF Skin Design Editor

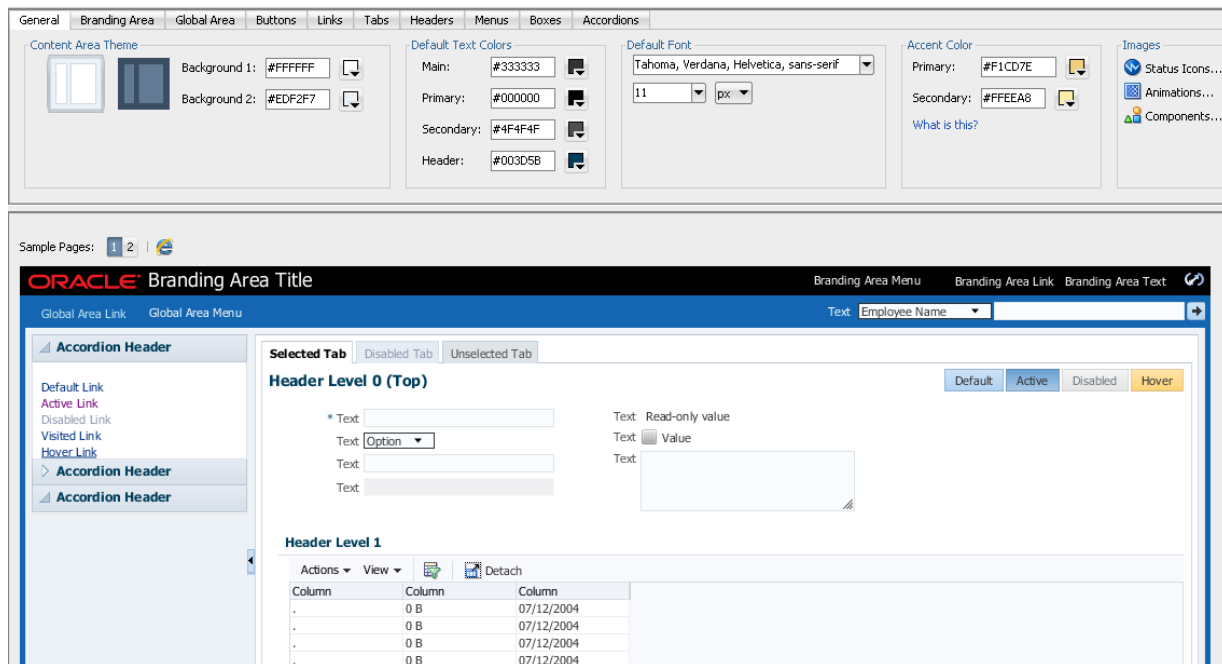
By default, the design editor opens when you create an ADF skin that extends from the Skyros ADF skin, as described in [Creating an ADF Skin File](#). This editor provides a variety of controls to change the most commonly styled parts of applications.

The lower part of the design editor displays a number of sample pages that render a wide variety of the commonly used **ADF Faces** components, such as buttons, links, and panel accordions. These sample pages refresh to reflect the changes that you make using the various controls in the upper part of the editor. A **Preview in Browser** icon renders the sample page in a browser when clicked. In [Figure 4-1](#), for example, clicking this icon renders the sample page in Internet Explorer. You can choose to

render the sample page in another browser, as described in [How to Change the Browser that Renders the Design Editor's Sample Pages](#).

The upper part of the design editor displays a variety of tabs that group together controls to modify the selectors for various areas of an application page, such as the branding area, the global area, buttons, links, and menus. Within each tab, user interface controls such as color pickers, input text components and links to invoke dialogs appear. [Figure 4-1](#) shows the General tab in the design editor that appears when you extend an ADF skin from the Skyros ADF skin. This tab renders color pickers that you can invoke when you click the dropdown arrows beside the fields that display the current color values, dropdown lists where you can select different fonts and font size and links to invoke dialogs where you can replace the images that the ADF skin references for status icons, animations and components.

**Figure 4-1 ADF Skin Design Editor that Appears for a Skyros-Extended ADF Skin**



Any changes that you make using the controls in the design editor result in the generation of CSS syntax that appears in the source file of the ADF skin. The design editor is useful for changing the commonly styled parts of an application. For example, one click in the Branding Area tab invokes a dialog where you can select a new image to render as the logo in the branding area of your application's page. Consider using the selectors editor, described in [Working with the ADF Skin Selectors Editor](#), when you need to go beyond changing the most commonly styled parts.

For more information about how you can use the design editor to change colors and images, see [Changing Images and Colors in the ADF Skin Design Editor](#).

## 4.2.1 How to Change the Browser that Renders the Design Editor's Sample Pages

You can change the browser that renders the design editor's sample pages when you click the **Preview in Browser** icon.

To change the browser that renders the design editor's sample pages:

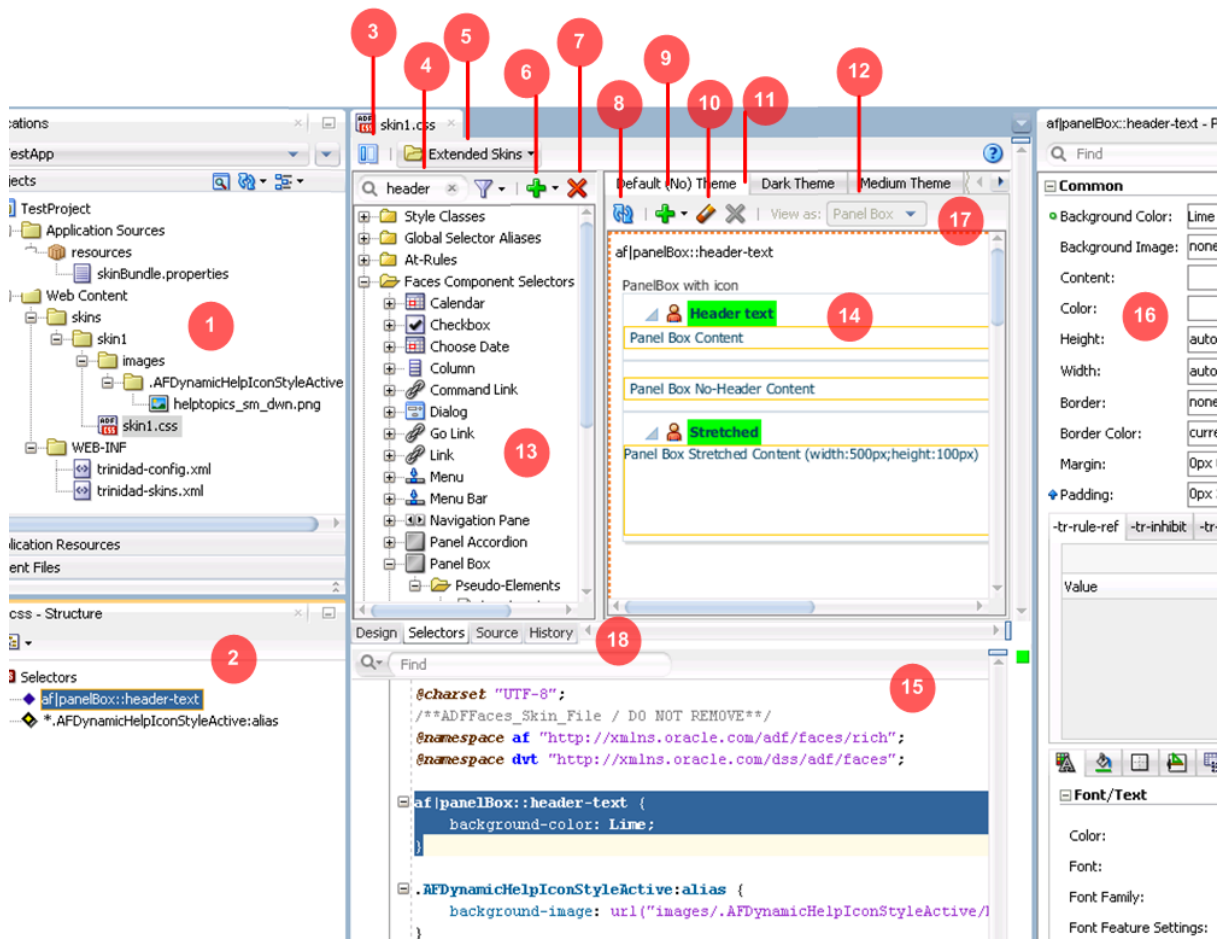
1. From the main menu, choose **Tools > Preferences**.

2. In the Preferences dialog, select the **Web Browser and Proxy** page.
3. Choose the browser that you want to use in the Web Browsers list.
4. Click OK.

### 4.3 Working with the ADF Skin Selectors Editor

Figure 4-2 shows the selectors editor. Each label number corresponds to a description in the list that follows Figure 4-2. The selectors editor opens by default if the ADF skin that you create extends from the Alta skin family. If your ADF skin extends from the Skyros skin family, you can access the selectors editor by clicking the Selectors tab.

Figure 4-2 ADF Skin Selectors Editor



1. The Projects node in the Applications window displays the source files for the ADF skins that you create. It also displays associated configuration and image files. By default, JDeveloper saves an ADF skin to a directory named skins. You can specify an alternative directory name to store the source files. For more information about creating ADF skins, see [Creating the Source Files for an ADF Skin](#).
2. The Structure window lists the selectors, global selector aliases, style classes, and at-rules that you added to the ADF skin file.
3. Click the **Hide/Show Divider** icon to hide or show the Selector Tree.

4. Filter the selectors that appear in the Selector Tree.

You can enter text in the input text field to filter the list of selectors that appear in the Selector Tree or you can use the filter icon to display:

- Available Selectors: all selectors in the Selector Tree.
- Updated Selectors: only those selectors that you modified in the ADF skin.
- Selectors with At-Rules: only those selectors that have an associated at-rule.

5. The Extended Skins list displays the list of ADF skins from which the current ADF skin extends. It also identifies imported ADF skins.

For more information, see [Navigating ADF Skins](#).

6. Use the **Add** icon to create a new style class, alias selector, or at-rule.

For information about creating a new style class, see [Working with Style Classes](#). For information about creating an alias selector, see [Working With Global Selector Aliases](#). For information about creating an at-rule, see [Working with At-Rules](#).

7. Use the **Delete** icon to remove a selector that you added to the ADF skin.

8. Click the **Refresh** icon to update the Preview Pane after you make changes to the properties of a selector in the Properties window.

9. Click the **Add Pseudo-Class** icon to apply a pseudo-class to the item that you selected in the Selector Tree.

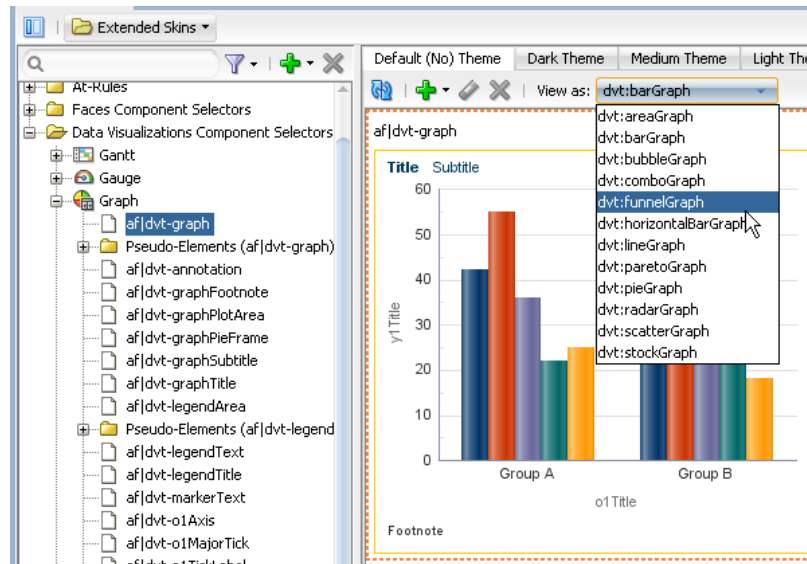
For more information about pseudo-classes, see [Pseudo-Classes in the ADF Skinning Framework](#).

10. Click the **Clear Property Settings** icon to undo any change that you made to the item selected in the Selector Tree.

11. Click the **Delete Pseudo-Class from Skin File** icon to delete any pseudo-classes that you specified in the ADF Skin.

12. The View as list allows you to preview how changes you make to a global selector alias in the Selector Tree affect the components that reference the global selector alias. The View as list displays all components that reference the global selector alias. The View as list also allows you to preview how changes you make to the properties of one component-specific selector impact all sub-types of that component. For example, [Figure 4-3](#) shows the ADF Data Visualization component selector for the `graph` component (`af|dvt-graph`) that exposes a single set of component-specific selectors that apply changes to all graph types. Use the **View as** list to preview a change that you make to a selector in one of the other types of graph (for example, Bar, Funnel, Pareto, and so on).



**Figure 4-3 View as List for a Component**

For more information about global selector aliases, [Working With Global Selector Aliases](#).

13. The Selector Tree displays the list of selectors, global selector aliases, style classes, and at-rules that you can configure values for in an ADF skin.

For more information, see [Working with the ADF Skin Selectors Editor](#).

14. The Preview Pane renders a preview of the changes that you make to a selector in an ADF skin after you click the **Refresh** icon (8).
15. You can also view the source of an ADF skin file.

**Tip:**

Select **Split Document** from a context menu that you can invoke from the Preview Pane to render the source and design views of an ADF skin side by side.

16. The Properties window identifies properties that you can configure for the ADF skin.

For more information, see [Working with the Properties Window](#).

17. The tabs for themes allow you to preview changes that you make for supported themes.

The Alta skin (and skins that extend from the Alta skin) do not use themes.

### 4.3.1 About the Selector Tree

The Selector Tree displays a list of the style classes, global selector aliases, and selectors for which you can configure properties to change the appearance of ADF Faces and **ADF Data Visualization components**.

[Figure 4-4](#) shows the nodes that the Selector Tree in the selectors editor exposes:

- Style Classes

A style class defines one or more style properties that you can apply to specific instances of a component. The selectors editor categorizes the inherited style classes into style classes defined for general usage, note windows, and popups. For more information, see [Working with Style Classes](#).

- Global Selector Aliases

A global selector alias defines style properties that you apply to one or more selectors. The selectors editor categorizes the inherited global selector aliases into selector aliases defined for general usage, icons, and messages. For more information, see [Working With Global Selector Aliases](#).

- Grouped Selectors

Identifies style properties grouped into one declaration to apply to more than one selector. For example, [Figure 4-4](#) shows a grouped selector for the `button` and `link` component's selectors.

- At-Rules

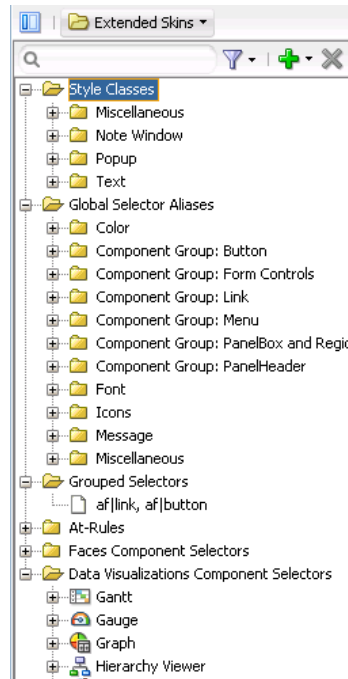
At-rules are a way to define style properties for when an application's page renders in a particular environment such as, for example, when using a specific browser. For more information, see [Working with At-Rules](#).

- Faces Component Selector

Selectors identify the ADF Faces components for which you can configure properties. The selectors editor displays subcategories for pseudo-elements, component selector aliases, and descendant selectors. For brevity, the ADF Faces components node is not expanded. For more information, see [Working with Component-Specific Selectors](#).

- Data Visualizations Component Selectors

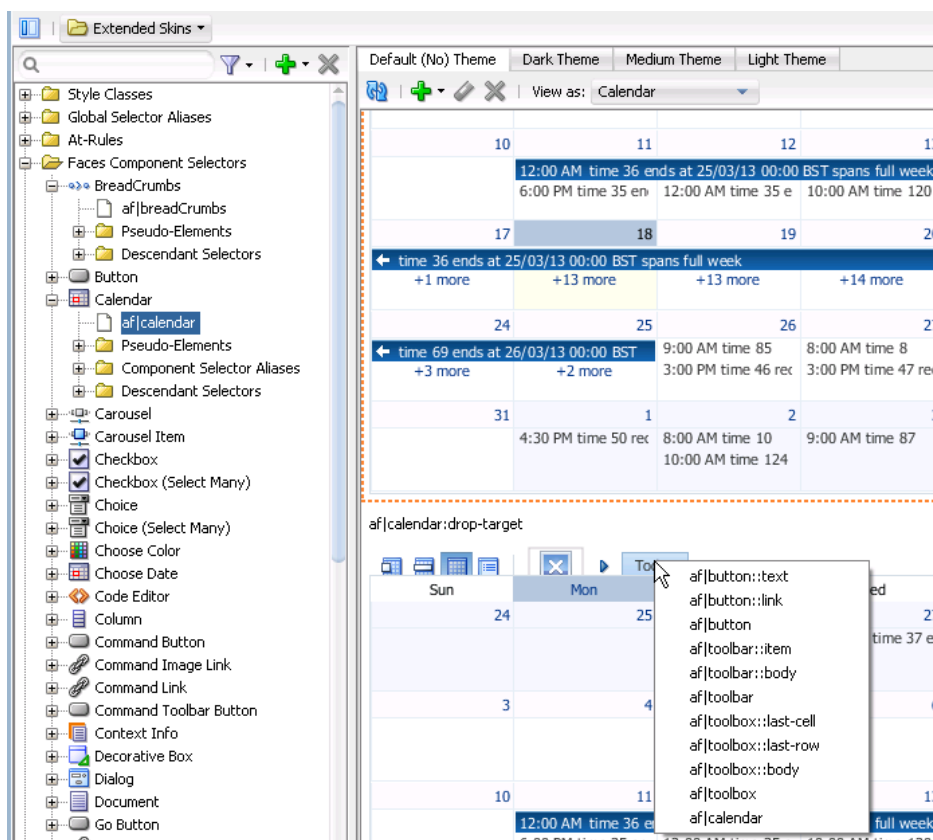
Selectors identify the ADF Data Visualization components for which you can configure properties. The selectors editor displays subcategories for pseudo-elements, component selector aliases, and descendant selectors. For more information, see [Working with Component-Specific Selectors](#).

**Figure 4-4 Selector Tree**

### 4.3.2 Interactive Preview in the Selectors Editor

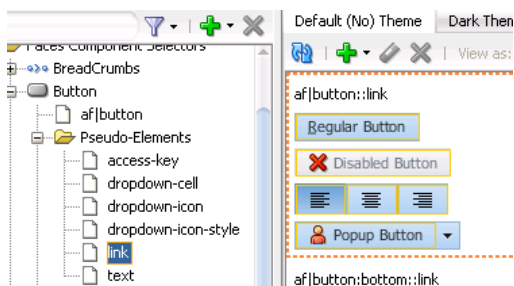
The preview pane in the selectors editor displays an interactive preview of the component that is currently selected in the Selector Tree. Hover your mouse over this preview to view text that identifies the specific pseudo-element that you need to customize to change the appearance of the component. Clicking on parts of this preview navigates you to the location where you can configure properties to change the appearance of what you have just clicked on. You can also right-click a pseudo-element to invoke a context menu that displays a hierarchical list of the selector pseudo-elements that the current pseudo-element contains, as shown in [Figure 4-5](#).

**Figure 4-5 Interactive Preview for the Calendar Component**



Clicking an entry in the context menu that appears or clicking a part of the calendar component that uses properties defined in the pseudo-element of another component selector navigates you to that pseudo-element in the Selector Tree. For example, if you click `af|button::link` in the context menu in Figure 4-5, the component preview navigates you to the location for the `button` component selector's pseudo-element in the Selector Tree of the selectors editor, as shown in Figure 4-6.

**Figure 4-6 Button Component's link Pseudo-Element**



## 4.4 Working with the Properties Window

The Properties window serves a number of functions apart from its primary role of allowing you to set values for CSS properties and ADF skin properties for the selectors that the ADF skinning framework exposes. These functions are the ability to:

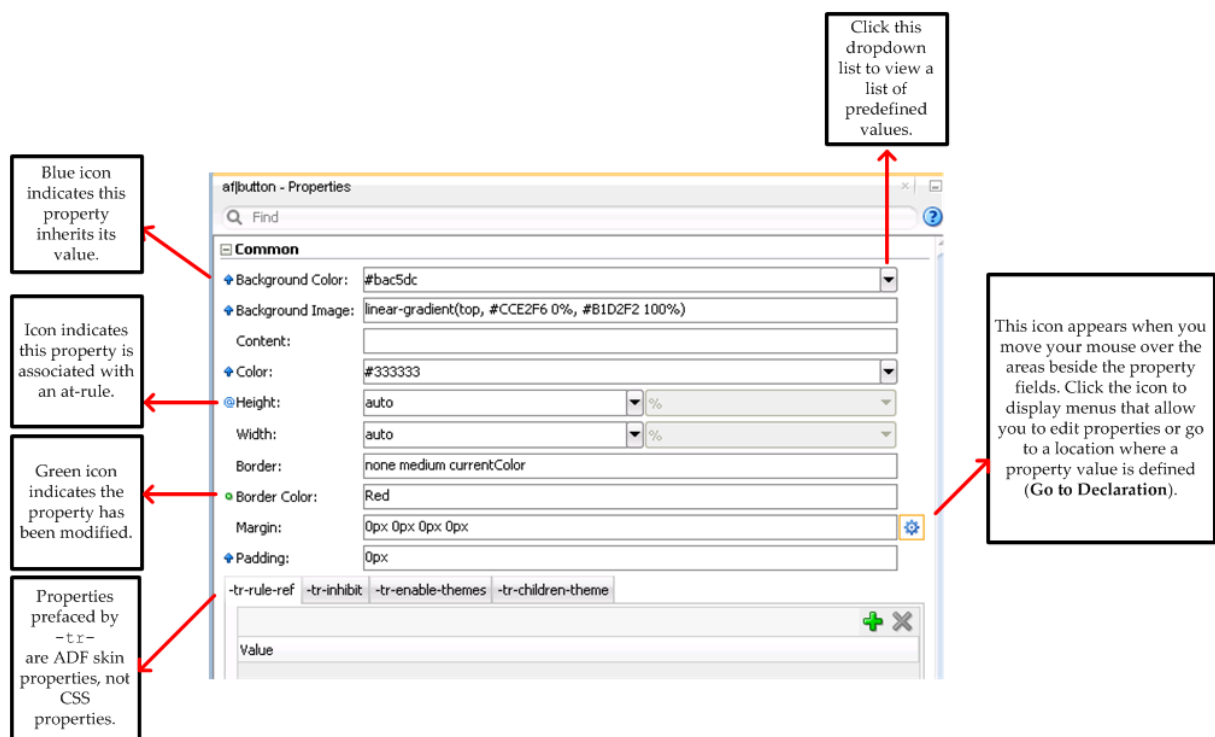
- Copy an image into the project where you develop the ADF skin.

For more information, see [Working with Images and Color in Your ADF Skin](#).

- Identify the properties that inherit their values from an extended ADF (blue icon) skin and identify the properties that you configured (green icon) in the ADF skin, as shown in [Figure 4-7](#).
- Identify the properties that are associated with at-rules, as shown in [Figure 4-7](#).  
For more information about at-rules, see [Working with At-Rules](#).
- Present ADF skin properties that you can configure for a selector.  
For more information, see [Properties in the ADF Skinning Framework](#).
- Navigate to the selector in an extended ADF skin that defines an inherited property in your ADF skin (**Go to Declaration**).  
For more information, see [Navigating ADF Skins](#).
- Invoke a dialog where you can define the colors for properties that support color value.

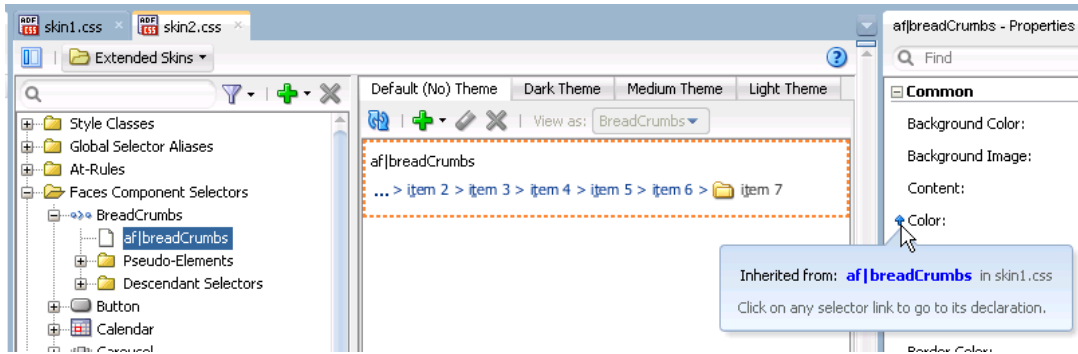
[Figure 4-7](#) presents an overview of the various controls that the Properties window exposes when you edit an ADF skin.

**Figure 4-7 Controls in the Properties Window for ADF Skins**



Hover your mouse over the icons that indicate a property associated with an at-rule or a property that inherits its value in order to display an information tip, as shown in [Figure 4-8](#). Clicking the link in this information tip navigates you to the source file of the ADF skin where the at-rule or inherited property value is defined.

**Figure 4-8 Information Tip Showing Link to Navigate to Source Declaration**



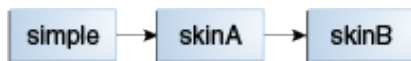
## 4.5 Navigating ADF Skins

When you create an ADF skin, as described in [Creating an ADF Skin File](#), you choose an ADF skin from which to extend. The ADF skin that you choose to extend from defines properties that your newly created ADF skin inherits. When you create your first ADF skin, you must choose one of the ADF skins that Oracle ADF provides.

Subsequent ADF skins that you create can extend an ADF skin that you created or one of the ADF skins provided by Oracle ADF. For example, you create your first ADF skin named `skinA` that extends the `simple` ADF skin provided by Oracle ADF. You then create a second ADF skin named `skinB`. When creating `skinB`, you have the choice of extending from `skinA` or from any of the ADF skins provided by Oracle ADF. If you choose to extend `skinB` from `skinA`, the inheritance relationship between the ADF skins is illustrated in [Figure 4-9](#).

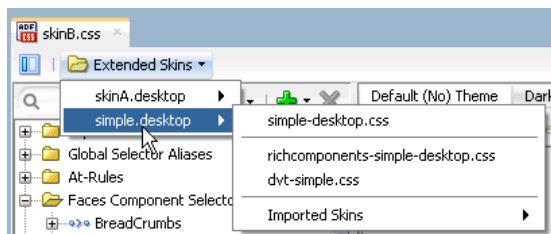
For more information about the ADF skins that Oracle ADF provides, see [Inheritance Relationship of the ADF Skins Provided by Oracle ADF](#), and [ADF Skins Provided by Oracle ADF](#).

**Figure 4-9 Example Inheritance Relationship Between ADF Skins**



The Extended Skins list in the selectors editor displays the list of ADF skins that the current ADF skin extends. The list also identifies if any of the ADF skins that your skin extends include imported skins. [Figure 4-10](#) shows the list of ADF skins that appears if you implement the inheritance relationship described in [Figure 4-9](#). You open an extended ADF skin that you want to view by clicking it in the Extended Skins list.

**Figure 4-10 Extended Skins List**

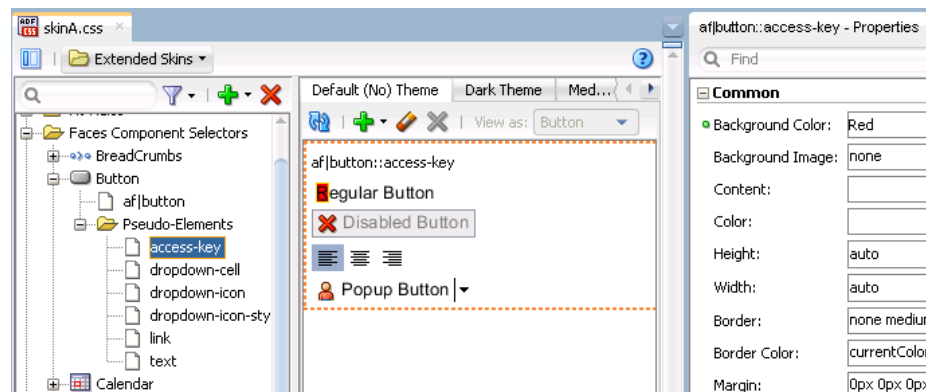


**Note:**

You cannot edit the properties of the selectors in the ADF skins provided by Oracle ADF. You can only edit the properties of selectors in extended ADF skins that you created.

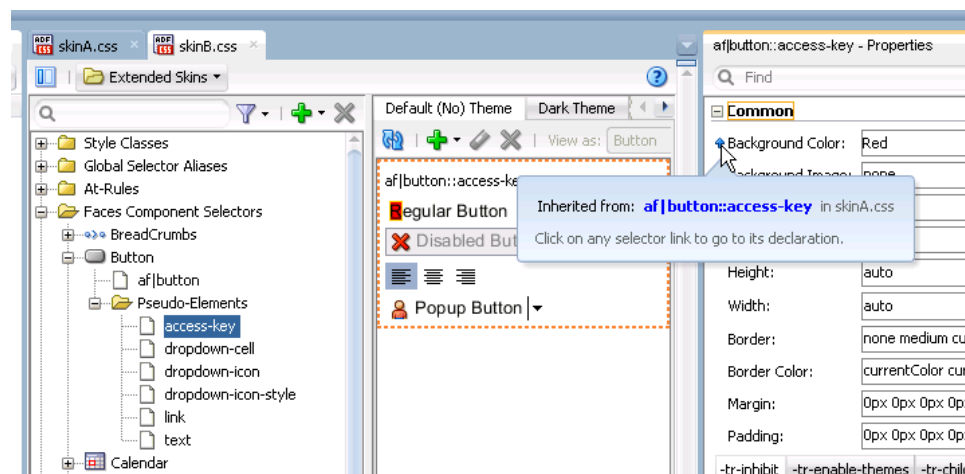
Using the **Go to Declaration** menu that the Properties window exposes, you can navigate to the location in an extended ADF skin where the extended ADF skin declares style properties inherited by other ADF skins. For example, assume that the `skinA` ADF skin defines a background color of Red for the `af|button` selector's `access-key` pseudo-element, as shown in [Navigating ADF Skins](#).

**Figure 4-11 Declaration of a Property Value**



The `skinB` ADF skin that extends from `skinA` ADF skin inherits the property values that are defined in the `skinA` ADF skin. [Navigating ADF Skins](#) shows the `skinB` ADF skin in the selectors editor with a value of Red for the `background-color` property.

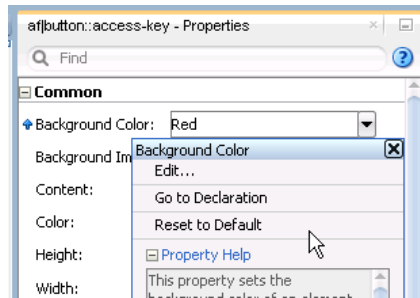
**Figure 4-12 Inheriting a Property Value from an Extended Skin**



To go to the declaration of a property:

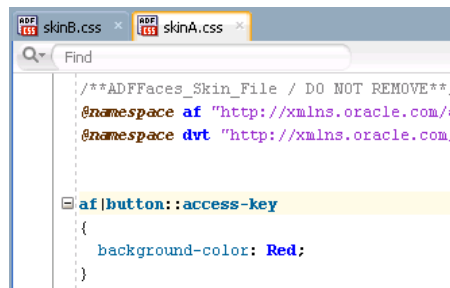
1. Identify a property in your ADF skin that inherits its values from an extended ADF skin. A blue upward-pointing arrow, as shown in [Figure 4-12](#), identifies these properties.
2. Click the icon that appears when you hover over the property field to invoke a context menu where you select **Go to Declaration**, as shown in [Figure 4-13](#).

**Figure 4-13 Go to Declaration Context Menu**



The extended ADF skin opens in the source view, as shown in [Figure 4-14](#). If the extended ADF skin is one that you created, you can modify the property values defined in it. The ADF skins provided by Oracle ADF, described in [ADF Skins Provided by Oracle ADF](#), are read-only.

**Figure 4-14 Property Value Defined in Extended ADF Skin**





---

## Creating the Source Files for an ADF Skin

This chapter describes how to create the source files for an ADF skin in JDeveloper. Information on how to import an ADF skin from an ADF Library JAR file is also provided.

This chapter includes the following sections:

- [About Creating an ADF Skin](#)
- [Creating an ADF Skin File](#)
- [Importing One or More ADF Skins Into the Current ADF Skin](#)
- [Adding ADF Skins from an ADF Library JAR](#)

### 5.1 About Creating an ADF Skin

An **ADF skin** defines the properties for the selectors that ADF Faces and ADF Data Visualization components expose. Using the editor in JDeveloper, you can create a source file for an ADF skin. As a source file for an ADF skin is a type of CSS file, you could create it without using an editor. However, when you use the editor, associated configuration files get created (the first time that you create an ADF skin) or modified (when you create subsequent ADF skins). For more information about these configuration files, see [Configuration Files for an ADF Skin](#).

### 5.2 Creating an ADF Skin File

You can create an ADF skin file in JDeveloper that defines how ADF Faces and ADF Data Visualization components render at runtime. The ADF skin that you create must extend either one of the ADF skins that Oracle ADF provides or from an existing ADF skin that you created. The ADF skins that Oracle ADF provides vary in the level of customization that they define for ADF Faces and ADF Data Visualization components. For information about the inheritance relationship between the ADF skins that Oracle ADF provides, see [Inheritance Relationship of the ADF Skins Provided by Oracle ADF](#). For information about the levels of customization in the ADF skins provided by Oracle ADF and for a recommendation about the ADF skin to extend, see [ADF Skins Provided by Oracle ADF](#).

By default, the editors in JDeveloper create ADF skins for the `org.apache.myfaces.trinidad.desktop` render kit. A **render kit** defines how ADF Faces components map to component tags that are appropriate for a particular client.

After you create an ADF skin, you can use the design editor and the other provided editors to modify the values for the selectors that the ADF Faces and ADF Data Visualization components expose. Otherwise, the ADF skin that you create defines the same appearance as the ADF skin from which it extends. For more information, see [Working with Component-Specific Selectors](#).

If you create an ADF skin that extends from the Skyros ADF skin, you enable the design editor. This tab provides an overview pane where you can use controls to set properties for many frequently-used selectors. For more information about using the design editor, see [Working with the ADF Skin Design Editor](#).

## 5.2.1 How to Create an ADF Skin in JDeveloper

You can create an ADF skin in JDeveloper.

To create an ADF skin in JDeveloper:

1. In the Applications window, right-click the project that contains the code for the user interface and choose **New**.
2. In the New Gallery, expand **Web Tier**, select **JSF/Facelets** and then **ADF Skin**, and click **OK**.
3. In the Skin File page of the Create ADF Skin dialog, enter the following:
  - **File Name:** Enter a file name for the new ADF skin.
  - **Directory:** Enter the path to the directory where you store the CSS source file for the ADF skin or accept the default directory proposed by the editor.
  - **Family:** Enter a value for the family name of your skin.

You can enter a new name or specify an existing family name. If you specify an existing value, you may need to version ADF skins, as described in [Versioning ADF Skins](#), to distinguish between ADF skins that have the same value for family.

The value you enter is set as the value for the `<family>` element in the `trinidad-skins.xml` where you register the ADF skin that you create. At runtime, the `<skin-family>` element in an application's `trinidad-config.xml` file uses this value to identify the ADF skin that an application uses. For more information, see [Applying an ADF Skin to Your Web Application](#).

- **Use as the default skin family for this project:** Deselect this checkbox if you do not want to make the ADF skin the default for your project immediately. If you select the checkbox, the `trinidad-config.xml` file is updated, as described in [What Happens When You Create an ADF Skin](#).
4. In the Base Skin page of the Create ADF Skin dialog, specify the following:
    - **Show Latest Versions Only:** Clear this checkbox to show all available versions of ADF skins.
    - **Available Skins:** Select the ADF skin that you want to extend. ADF Faces provides a number of ADF skins that you can extend. The list also includes any ADF skins that you created previously in this project. For more information and a recommendation on the ADF skin to extend, see [ADF Skins Provided by Oracle ADF](#).
    - **Skin Id:** A read-only field that displays a concatenation of the value you enter in **File Name** and the ID of the render kit (`desktop`) for which you create your ADF skin. You select this value from the **Extends** list if you want to create another ADF skin that extends from this one.

JDeveloper writes the value to the `<id>` element in the `trinidad-skins.xml` file.

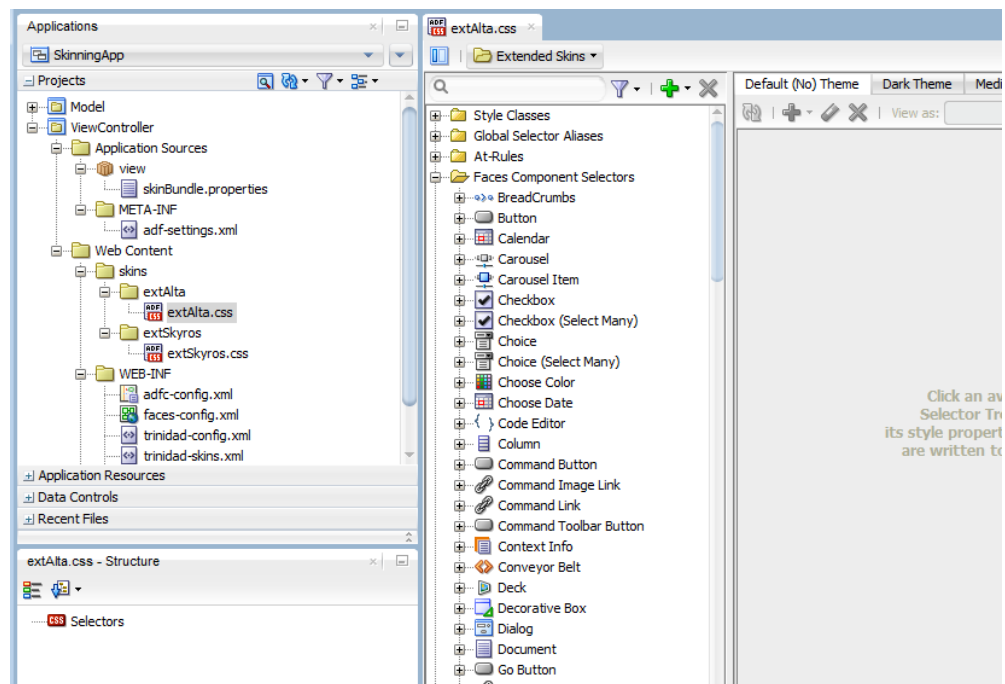
5. Click **Finish**.

## 5.2.2 What Happens When You Create an ADF Skin

If you accepted the default value proposed for the Directory field, a file with the extension `.css` is generated in a subdirectory of the `skins` directory in your project.

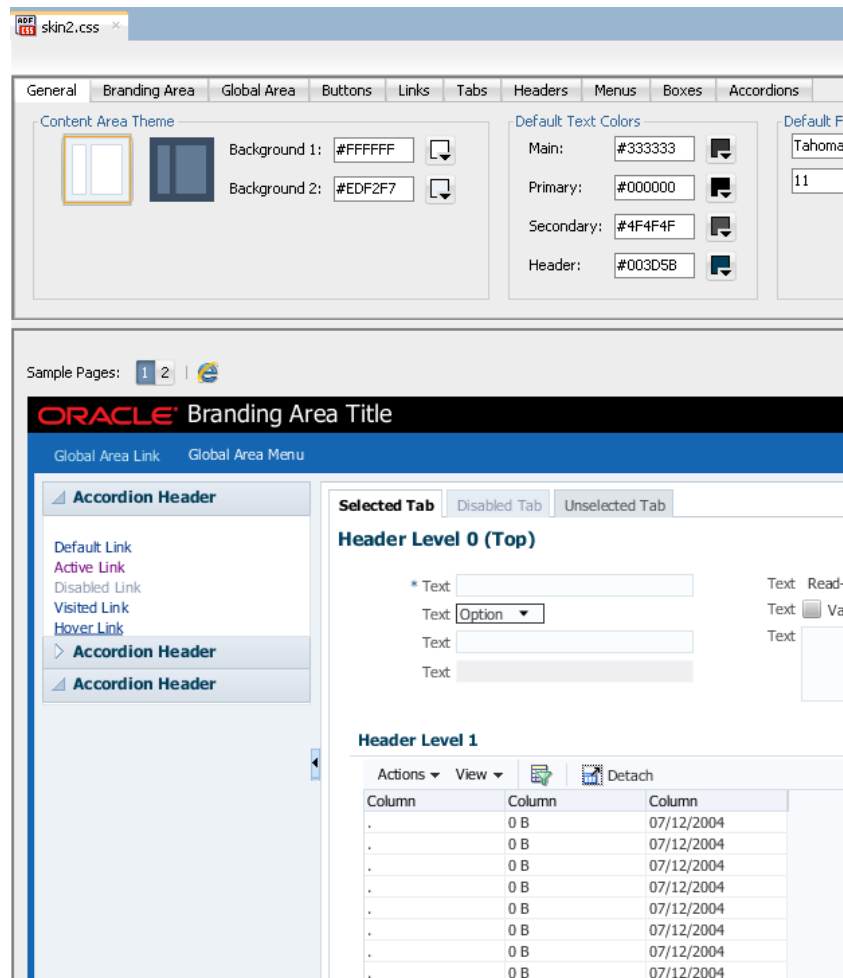
An ADF skin that extends the Alta skin opens in the selectors editor, as illustrated in [Figure 5-1](#). This selectors editor is also available if you create an ADF skin that extends from the Skyros ADF skin.

**Figure 5-1** *Newly-Created ADF Skin that Extends from Alta in the Selectors Editor*



An ADF skin that extends the Skyros ADF skin opens in the design editor, as illustrated in [Figure 5-2](#).

**Figure 5-2** Newly-Created ADF Skin that Extends from Skryos in the Design Tab



The `trinidad-skins.xml` file is modified to include metadata for the ADF skin that you create, as illustrated in [Example 5-1](#), which shows entries for an ADF skin that extends from the Skyros family of ADF skins. [Example 5-1](#) also contains values that specify the render kit and the **resource bundle** associated with this ADF skin. For more information about resource bundles, see [Working With Text in an ADF Skin](#).

If you select the **Use as the default skin family for this project** check box in the Create ADF Skin dialog, the `trinidad-config.xml` file is modified to make the new ADF skin the default skin for your application. This means that if you run the application from JDeveloper, the application uses the newly-created ADF skin. For more information, see [Applying an ADF Skin to Your Web Application](#). The following example shows a `trinidad-config.xml` file that makes the ADF skin in [Example 5-1](#) the default for an application.

```
<?xml version="1.0" encoding="windows-1252"?>
<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">
  <skin-family>skin2</skin-family>
</trinidad-config>
```

The source file for the ADF skin contains a comment and namespace references, as illustrated in the following example. These entries in the source file for the ADF skin distinguish the file from non-ADF skin files with the `.css` file extension. A source file for an ADF skin requires these entries in order to open in the design or selectors editors for the ADF skin.

```

/**ADFFaces_Skin_File / DO NOT REMOVE**/
@namespace af "http://xmlns.oracle.com/adf/faces/rich";
@namespace dvt "http://xmlns.oracle.com/dss/adf/faces";

```

The first time that you create an ADF skin in your project, a resource bundle file (`skinBundle.properties`) is generated, as illustrated in [Figure 5-1](#). For more information about using resource bundles, see [Working With Text in an ADF Skin](#).

#### Example 5-1 *trinidad-skins.xml* File

```

<?xml version="1.0" encoding="windows-1252"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  ....
  <skin>
    <id>skin2.desktop</id>
    <family>skin2</family>
    <extends>skyros-v1.desktop</extends>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
    <style-sheet-name>skins/skin2/skin2.css</style-sheet-name>
    <bundle-name>resources.skinBundle</bundle-name>
  </skin>
</skins>

```

## 5.3 Importing One or More ADF Skins Into the Current ADF Skin

You can import another ADF skin that is in your JDeveloper project into your ADF skin using the `@import` rule. This makes the style properties defined in the latter ADF skin available to your ADF skin. The following examples demonstrate the valid syntax to import an ADF skin (`skinA`) into the current ADF skin:

```

/** Use the following syntax if skinA.css is in the same directory **/
@import "skinA.css";
@import url("skinA.css");

/** Use the following syntax if skinA.css is in another directory **/
@import "../skinA/skinA.css";
@import url("../skinA/skinA.css");

```

The `@import` rule(s) must follow all `@charset` rules and precede all other at-rules and rule sets in an ADF skin, as shown in the following example that imports two ADF skins into the current ADF skin:

```

@charset "UTF-8";

@import url("../skinA/skinA.css");
@import url("../skinB/skinB.css");

/**ADFFaces_Skin_File / DO NOT REMOVE**/
@namespace af "http://xmlns.oracle.com/adf/faces/rich";
@namespace dvt "http://xmlns.oracle.com/dss/adf/faces";

af|inputText{
  background-color: Green;
}
...

```

## 5.4 Adding ADF Skins from an ADF Library JAR

You can add ADF skins that have been packaged in a JAR file into your JDeveloper project. When you add an ADF skin from a JAR file into your project, the imported

ADF skin is available to extend from when you create a new ADF skin, as described in [Creating an ADF Skin File](#).

The recommended type of JAR file to use to package an ADF skin is an ADF Library JAR file. For information about how to package an ADF skin into this type of JAR file, see [Packaging an ADF Skin into an ADF Library JAR](#).

### 5.4.1 How to Add an ADF Skin from an ADF Library JAR

You can add ADF skins into your project that have been packaged in a JAR file.

To add an ADF skin from an ADF Library JAR:

1. From the main menu, choose **Application > Project Properties**.
2. In the Project Properties dialog, select the **Libraries and Classpath** page and click **Add JAR/Directory**.
3. In the Add Archive or Directory dialog, navigate to the JAR file that contains the ADF skin you want to add and click **Select**.

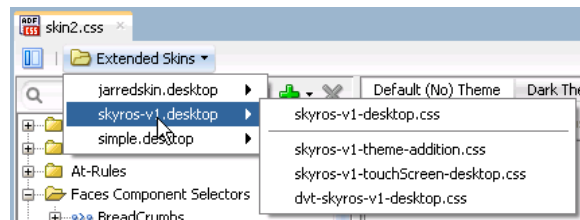
The JAR file appears in the Classpath Entries list.

4. When finished, click **OK**.

### 5.4.2 What Happens When You Import an ADF Skin from an ADF Library JAR

The ADF skin(s) that you add from the JAR file appear in the Extends list when you create a new ADF skin, as described in [Creating an ADF Skin File](#). After you create a new ADF skin by extending an ADF skin that you added from a JAR file, the Extended Skins list in the selectors editor's Preview Pane displays the name of the ADF skin that you added. For example, in [Figure 5-3](#) the `skin2.css` ADF skin has been created by extending the ADF skin, `jarredskin.css`, that was added into the project from a JAR file.

**Figure 5-3 Imported ADF Skin in the Extended Skins List**



Properties that have been defined in the ADF skin that you added appear with a blue upward pointing arrow in the Properties window. An information tip about the inheritance relationship displays when you hover the mouse over the property, as illustrated in [Figure 5-4](#).

**Figure 5-4 Property Inherited from an Imported ADF Skin**



---

# Working with Component-Specific Selectors

This chapter describes how to change the appearance of ADF Faces and ADF Data Visualization components by specifying properties for the selectors that the ADF skinning framework exposes for these components. Features such as the ability to configure ADF skin properties to apply to messages, themes that you can apply to ADF Faces components, and how to configure an ADF skin for accessibility are also described.

This chapter includes the following sections:

- [About Working with Component-Specific Selectors](#)
- [Changing ADF Faces Components' Selectors](#)
- [Changing ADF Data Visualization Components' Selectors](#)
- [Changing a Component-Specific Selector](#)
- [Configuring ADF Skin Properties to Apply to Messages](#)
- [Configuring an ADF Skin for Accessibility](#)

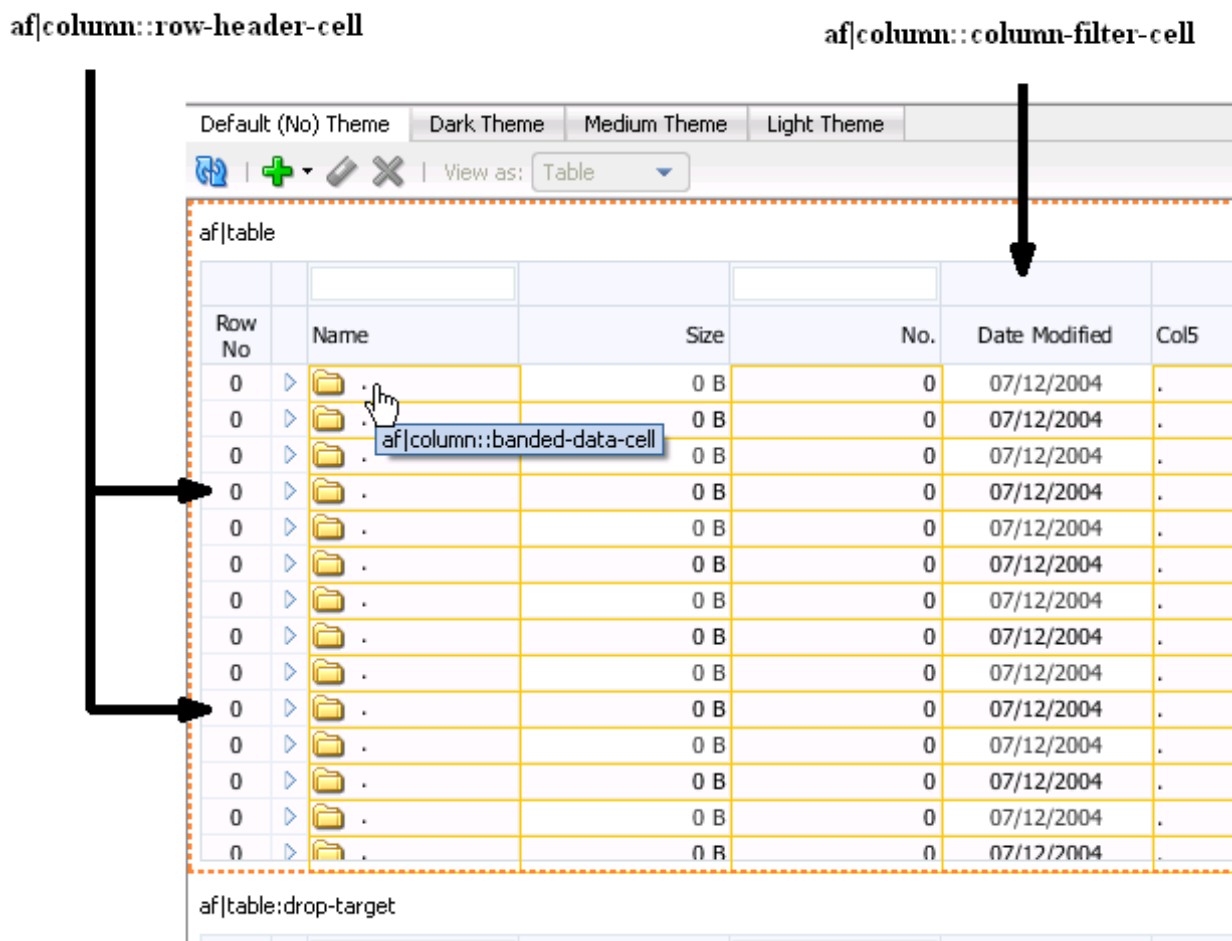
## 6.1 About Working with Component-Specific Selectors

You customize the appearance of **ADF Faces** or **ADF Data Visualization components** by defining style properties for the selectors that the components expose. To achieve the appearance you want, you need to become familiar with the component-specific selectors that the ADF Faces and ADF Data Visualization components expose, plus the global selector aliases and descendant selectors that a component-specific selector may reference. The ADF skins that you extend from when you create an **ADF skin** define many global selector aliases and descendant selectors. You also need to become familiar with the component itself and how it relates to other components. For example, customizing the appearance of the ADF Faces `table` component shown in [Figure 6-1](#) requires you to define style properties for the `row-header-cell` and `column-filter-cell` selectors exposed by the `af:column` component in addition to selectors exposed by the ADF Faces `table` component. You may also need to modify the style properties for one or more of the icon or message global selector aliases that the ADF skin you extend defines.

**Note:**

Consider using the design editor, as described in [Working with the ADF Skin Design Editor](#), if you want to change the properties of some of the most frequently used selectors in an ADF skin that extends from the Skyros ADF skin. This editor appears by default if your ADF skin extends from the Skyros ADF skin. The design editor provides a variety of controls to quickly change your ADF skin.

**Figure 6-1** Selectors for an ADF Faces table Component



Use the tools that the selectors editor for ADF skins provides to customize the appearance of the ADF Faces components and ADF Data Visualization components. For more information, see [Working with ADF Skins in JDeveloper](#).

Other sources of information that may help you as you change the selectors of ADF Faces and ADF Data Visualization components include the following:

- Images: An ADF skin can reference images that render icons and logos, for example, in a page. For more information about how to work with images in an ADF skin, see [Working with Images and Color in Your ADF Skin](#).



- Text: An ADF skin does not include text strings that render in your page. However, you can specify a **resource bundle** that defines the text strings you want to appear in the page. For more information, see [Working With Text in an ADF Skin](#).
- Global selector aliases: A global selector alias specifies style properties that you can apply to multiple ADF Faces components simultaneously. For more information about global selector aliases, see [Working With Global Selector Aliases](#).
- Style Classes: A style class in an ADF skin specifies a number of style properties that an ADF Faces component can reference as a value if it exposes a style-related attribute (`styleClass` and `inlineStyle`). For more information about style classes, see [Working with Style Classes](#).
- ADF Faces Rich Client Components Hosted Demo: The Oracle Technology Network (OTN) web site provides a link to an application that demonstrates how ADF skins change the appearance of ADF Faces and ADF Data Visualization components. For more information, navigate to <http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html>

## 6.2 Changing ADF Faces Components' Selectors

ADF Faces components render user interface controls, such as buttons, links and check boxes in your web application. ADF Faces components also include components that render calendars, panels to arrange other user interface controls and tables in your web application. For more information about ADF Faces components and the functionality that they provide, see *Developing Web User Interfaces with Oracle ADF Faces*.

You can change the runtime appearance of ADF Faces components by editing the properties that each ADF Faces skin selector exposes. The number of selectors that an ADF Faces component exposes varies by component. For example, the ADF Faces components, `af:image` and `af:popup`, expose one selector each. In contrast, the ADF Faces component, `af:panelHeader`, exposes a variety of selectors that enable you to change the appearance of different parts of the user interface of that component. There are, for example, selectors that allow you to change the `af:panelHeader` component's instruction text, help icons, and titles.

The process to follow to change the runtime appearance of an ADF Faces component is the same for each component; the only difference is the number of selectors that each ADF Faces component exposes. [Figure 6-2](#) and [Figure 6-3](#) take the `button` component as an example and illustrate how you can customize the appearance of this component using pseudo-elements and the component's selector. [Figure 6-2](#) shows the application of the `skyros` skin on the `button` component and the component icon.

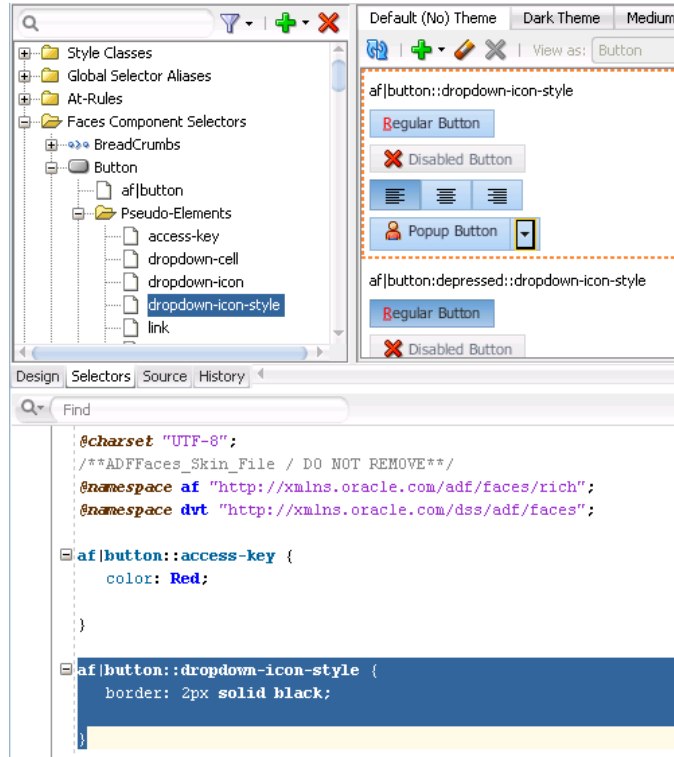
**Figure 6-2 Button Component Default Appearance with Skyros ADF Skin**



[Figure 6-3](#) shows the appearance of the component in the selectors editor after you set values for the following pseudo-elements on the component's selector:

- access-key: The Color property is set to red
- dropdown-icon-style: The Border property is set to `2px solid black`

**Figure 6-3 Button Component with Modified Selectors**



Reference information about the selectors that ADF Faces components expose can be found in the *Tag Reference for Oracle ADF Faces Skin Selectors*.

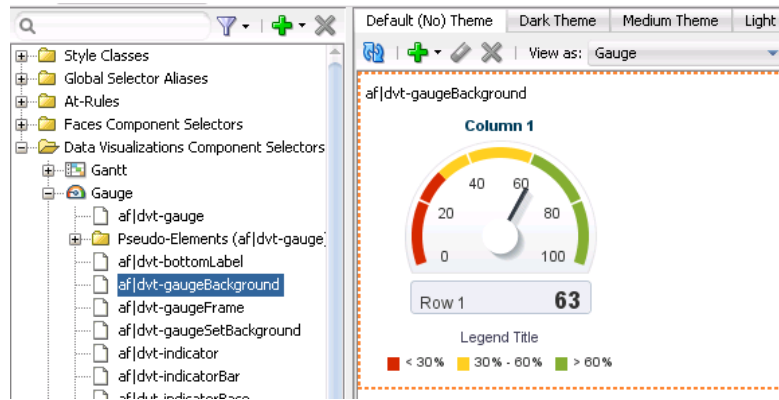
## 6.3 Changing ADF Data Visualization Components' Selectors

The ADF Data Visualization components are a set of components that provide functionality to represent data in graphical and tabular formats. Examples of the ADF Data Visualization components include the following: graph, gantt, pivot table, and hierarchy viewer. For more information about ADF Data Visualization components and the functionality that they provide, see *Developing Web User Interfaces with Oracle ADF Faces*.

You can change the runtime appearance of ADF Data Visualization components by editing the properties that each ADF Data Visualization component selector exposes. The number of selectors exposed by an ADF Data Visualization component varies by component.

Figure 6-4 shows an ADF skin in the selectors editor with the nodes expanded to show the selectors that you can customize for the ADF Data Visualization gauge component.

**Figure 6-4 ADF Data Visualization Component Selectors**



You customize the appearance of ADF Data Visualization components by defining style properties for the selectors that each ADF Data Visualization component exposes. Using the tools provided by JDeveloper's selectors editor for ADF skins, you customize the appearance of the ADF Data Visualization components. For more information, see [Working with ADF Skins in JDeveloper](#).

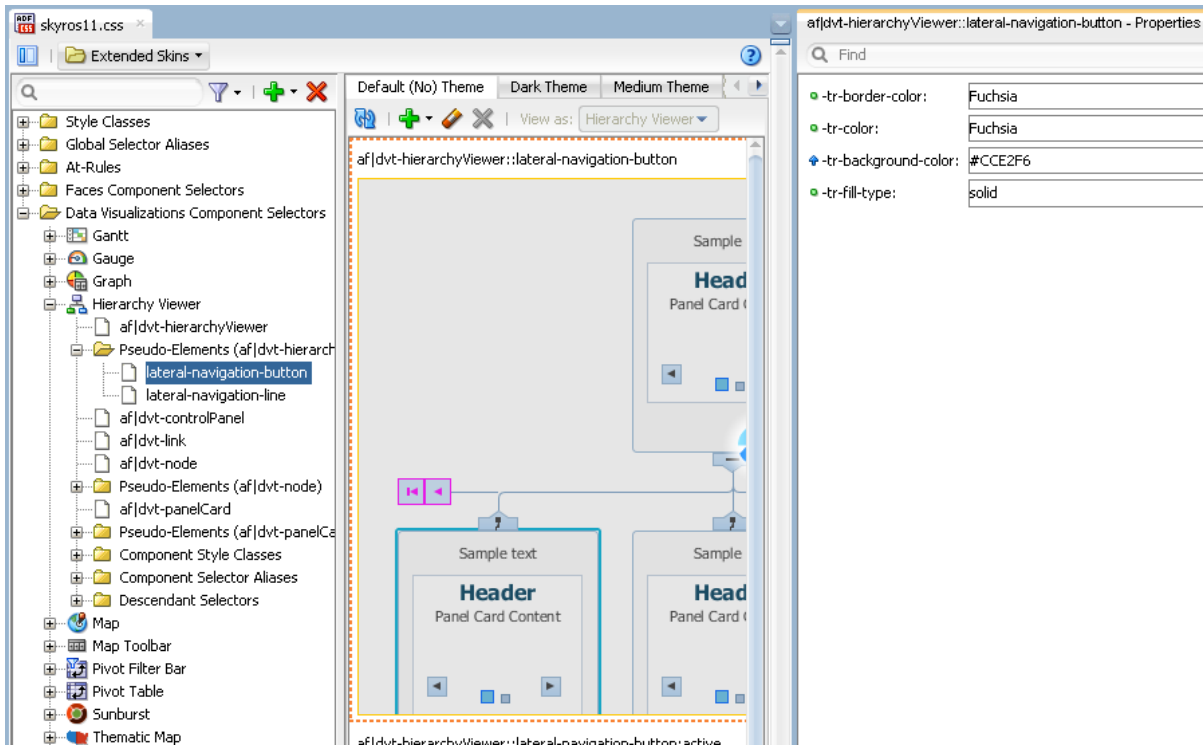
To achieve the appearance you want, you need to become familiar with the selectors that the ADF Data Visualization component exposes, the global selector aliases that the component may reference and which are defined in the ADF skin that you extend when you create an ADF skin. You also need to become familiar with the component itself and how it relates to other components. For example, customizing the appearance of the ADF Data Visualization `pivotTable` component shown in [Figure 6-5](#) requires you to define style properties for this selector's pseudo-elements. You may also need to modify the style properties for one or more of the global selector aliases that the ADF skin you extend defines.

**Figure 6-5 ADF Data Visualization `pivotTable` Component**

		Sales		Units	
		All Channels		All Channels	
		World	Boston	World	Boston
2007	Tents	20,000	500	200	50
	Canoes	15,000	1,500	75	8
2006	Tents	10,000	250	100	25
	Canoes	7,500	750	40	4
2005	Tents	5,000	125	50	15
	Canoes	3,750	375	20	2

Many ADF Data Visualization component selectors, such as the selectors for the `graph` and `hierarchyViewer` components, expose pseudo-elements for which you configure ADF skin properties. These ADF skin properties modify the appearance of the area specified by the pseudo-element. The characters `-tr-` preface the names of ADF skin properties. For example, [Figure 6-6](#) shows the properties of the `hierarchyViewer`'s `lateral-navigation-button` selector, all of which are prefaced by `-tr-`.

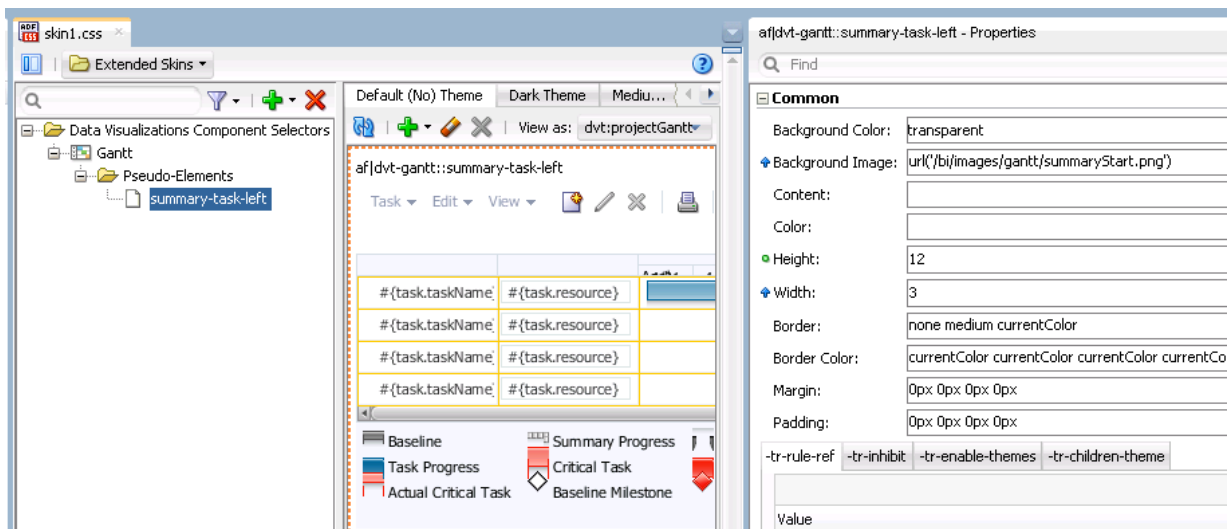
**Figure 6-6** Properties for the hierarchyViewer Component lateral-navigation-button Pseudo-Element



In contrast, the gantt component's summary-task-left selector, shown in [Figure 6-7](#), exposes only four ADF skin properties (-tr-rule-ref, -tr-inhibit-, -tr-enable-themes, and -tr-children-theme) as the majority of the properties that you configure for this selector are CSS properties.

For more information about ADF skin properties, see [Properties in the ADF Skinning Framework](#).

**Figure 6-7** Properties for the gantt Component summary-task-left Pseudo-Element

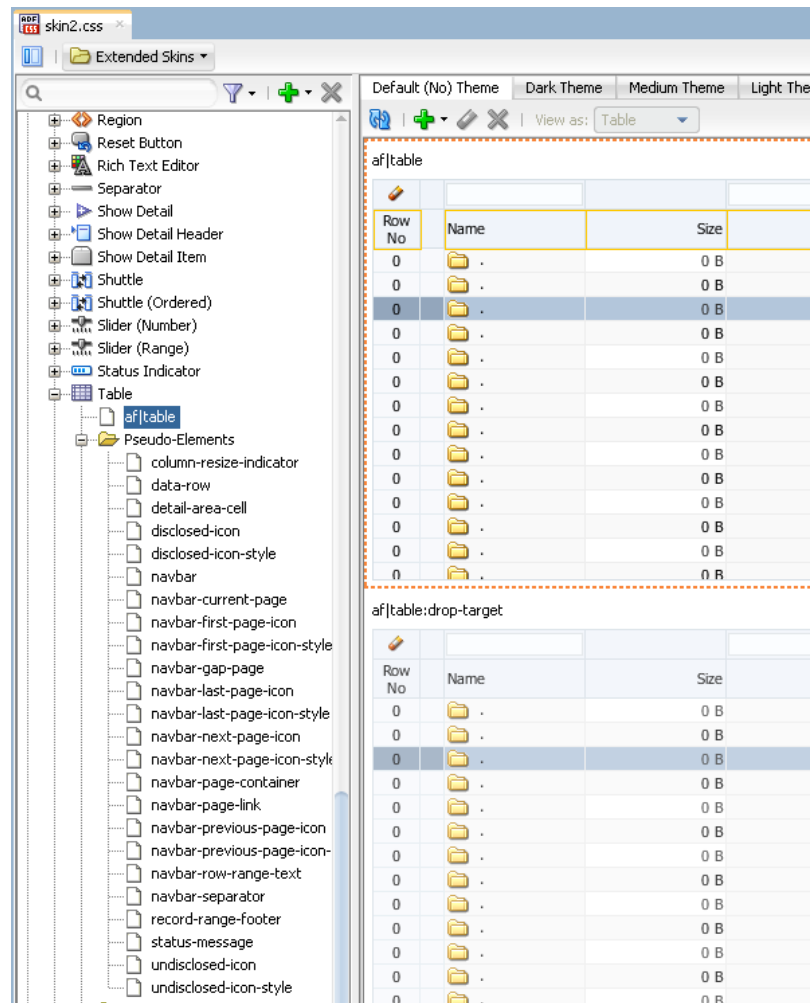


Reference information about the selectors, pseudo-elements, and pseudo-classes that ADF Data Visualization components expose can be found in the *Tag Reference for Oracle ADF Data Visualization Tools Skin Selectors*.

## 6.4 Changing a Component-Specific Selector

The process to change a component-specific selector is the same for both the ADF Faces and ADF Data Visualization components. In the **Selector Tree** of the selectors editor, you expand the Faces Components Selectors or Data Visualization Selectors node to select the selector of the component you want to modify. You then set values for this selector using the Properties window. You can also set properties for any pseudo-elements, component style classes, component selector aliases, or descendant selectors that the selector you select references. In addition, you can add pseudo-classes that the component-specific supports. For more information about pseudo-classes, see [Pseudo-Classes in the ADF Skinning Framework](#). [Figure 6-8](#) shows a view of the skin selector for the ADF Faces `table` component in the Selector Tree of the selectors editor with the different pseudo-elements that you can configure for this skin selector.

**Figure 6-8** Selector for the table Component



[Figure 6-9](#) shows a runtime view of an ADF Faces `table` component that renders data using the style properties provided by the ADF Faces `simple` skin.

**Figure 6-9 ADF Faces table Component Rendered By the simple Skin**

PersonId	PrincipalName	Title	FirstName	LastName
108	NGREENBE		Nancy	Greenberg
109	DFAVIET		Daniel	Faviet
110	JCHEN		John	Chen
111	ISCIARRA		Ismael	Sciarra
112	JMURMAN		Jose Manuel	Urman
113	LPOPP		Luis	Popp
114	DRAPHEAL		Den	Raphaely
115	AKHOO		Alexander	Khoo
116	SBAIDA		Shelli	Baida
117	STOBIAS		Sigal	Tobias

## 6.4.1 How to Change a Component-Specific Selector

You change a component-specific selector by selecting the selector in the Selector Tree and setting values for the selector, its pseudo-elements, or descendant selectors in the Properties window. In addition, you can add a pseudo-class if the component-specific selector supports one.

To change a component-specific selector:

1. In the Selector Tree of the selectors editor, choose the appropriate option:
  - Expand the **Faces Component Selectors** node if you want change a selector for an ADF Faces component.
  - Expand the **Data Visualization Selectors** node if you want to change a selector for an ADF Data Visualization component.

For example, expand the Faces Component Selectors node, the Column node, the Pseudo-Elements node, and select the **column-header-cell** selector.

2. In the Properties window, specify values for the properties that the selector you selected in Step 1 supports.

For example, in the Common section of the Properties window, specify values for the following attributes:

- **Background Color:** Specify the background color that you want to appear in the header row of the table.
  - **Color:** Specify the color that you want to apply to text that appears in the header row of the table's column.
3. In the Preview Pane, click the **Add Pseudo-Class** icon to choose a supported pseudo-class from the displayed list of supported pseudo-classes that appears.

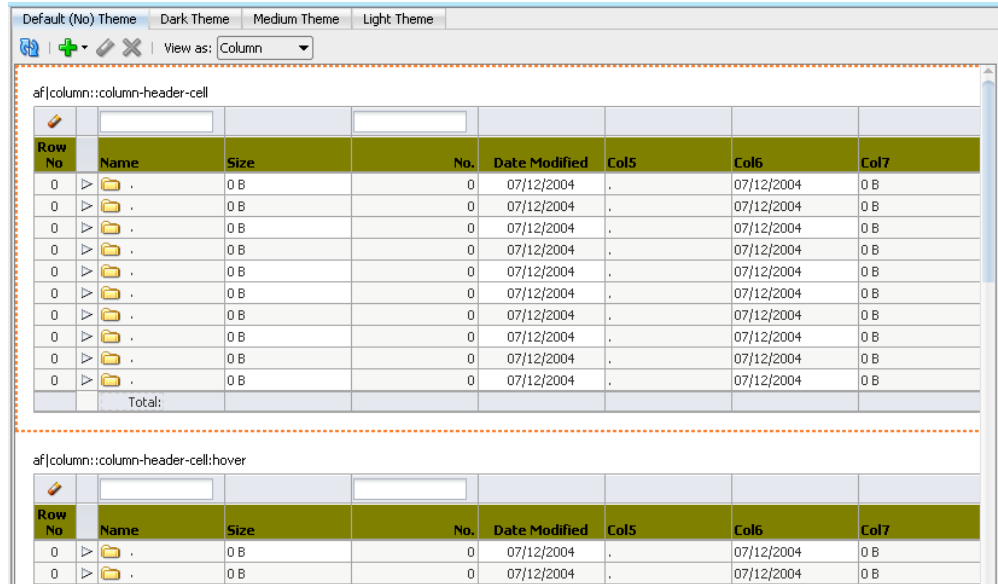
## 6.4.2 What Happens When You Change a Component-Specific Selector

The selectors editor displays the changes that you make to the selector after you click the **Refresh** icon in the Preview Pane. If you add a pseudo-class to the selector, the Preview Pane also displays an entry for the selector with the added pseudo-class. For example, [Figure 6-10](#) shows an entry for a selector with the `:hover` pseudo-class added.

**Note:**

The Preview Pane for the af | document selector only displays one entry even if you add a pseudo-class to this selector.

**Figure 6-10 Preview Pane with a Component Specific Selector and a Pseudo-Class**



The selectors editor also writes the values that you specify for the selectors in the Properties window to the source file for the ADF skin. The following example shows the changes that appear in the source file after making some of the changes described in [How to Change a Component-Specific Selector](#).

```
af|column::column-header-cell
{
  color: Black;
  background-color: Olive;
  font-weight: bold;
}
```

When a web application uses an ADF skin that contains the values shown in the previous example, header rows in the columns of a table rendered by the ADF Faces table component appear as illustrated by [Figure 6-11](#) where the table uses a skin that extends the simple skin.

**Figure 6-11 ADF Faces table with a Header Row Skinned**

PersonId	PrincipalName	Title	FirstName	LastName
108	NGREENBE		Nancy	Greenberg
109	DFAVIET		Daniel	Faviet
110	JCHEN		John	Chen
111	ISCIARRA		Ismael	Sciarra
112	JMURMAN		Jose Manuel	Urman
113	LPOPP		Luis	Popp
114	DRAPHEAL		Den	Raphaely
115	AKHOO		Alexander	Khoo
116	SBAIDA		Shelli	Baida
117	STOBIAS		Sigal	Tobias

## 6.5 Configuring ADF Skin Properties to Apply to Messages

You can apply styles to ADF Faces input components based on whether or not the input components have certain types of message associated with them. When a message of a particular type is added to a component, the styles of that component are automatically modified to reflect the new status. If you do not define styles for the type of message in question, the component uses the default styles defined in the ADF skin.

The types of message property are:

- `:fatal`
- `:error`
- `:warning`
- `:confirmation`
- `:info`

Figure 6-12 shows an `inputText` component rendered using the `simple` ADF skin. In Figure 6-12, the `simple` ADF skin defines style values for the `:warning` message property to apply to the `inputText` component when an end user enters values that generate a warning.

**Figure 6-12** *inputText Component Displaying Style for :warning Message Property*

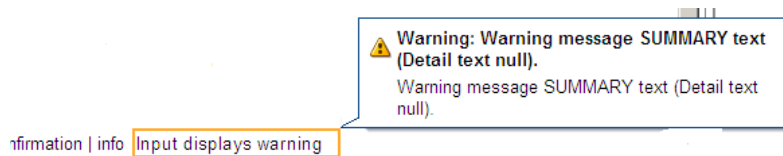
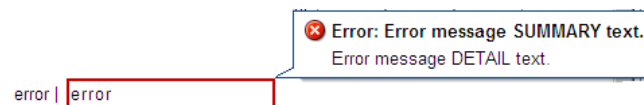


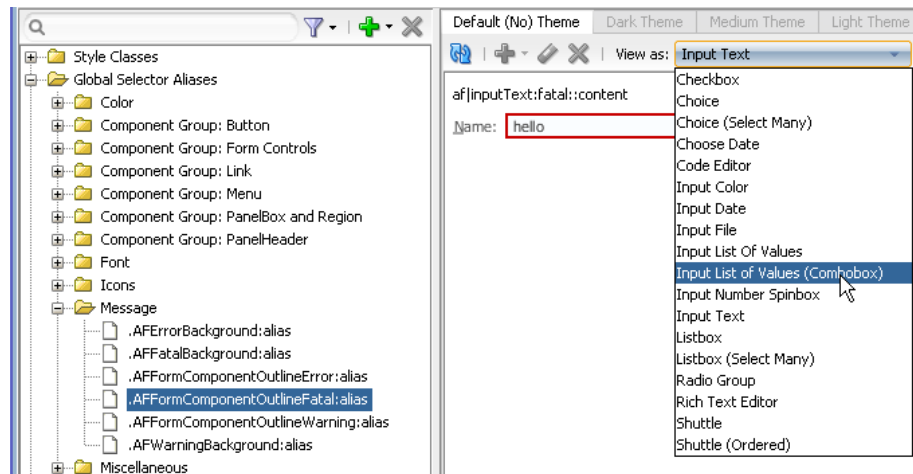
Figure 6-13 shows the same `inputText` component as in Figure 6-12. In Figure 6-13, the end user entered a value that generated an error. As a result, the `inputText` component renders using the style properties configured for the `:error` message property.

**Figure 6-13** *inputText Component Displaying Style for :error Message Property*



The **ADF skinning framework** defines a number of global selector aliases that define style properties to apply to messages. Figure 6-14 shows a list of global selector aliases under the `Message` node in the Selector Tree of the selectors editor. The Preview Pane, on the right of Figure 6-14, shows how the style properties defined for the global selector alias currently selected in the Selector Tree render the component selected from the View as list.



**Figure 6-14 Global Selector Aliases for Messages**

You can customize the global selector aliases that the ADF skinning framework provides for messages by defining style properties in your ADF skin. The style properties that you define for the global selector alias affect all ADF Faces components that reference the global selector alias. For example, if you change the border color for the global selector alias shown in [Figure 6-14](#) to green, all the ADF Faces components shown in the View as list render with a border that is green. For more information about global selector aliases, see [Working With Global Selector Aliases](#).

The `af|message` and `af|messages` selectors also expose pseudo-elements that enable you to change the icons that appear in the message dialogs at runtime. In addition, these selectors define resource strings that determine the text to appear in tool tips when an end user hovers over a message dialog. You can override these resource strings by specifying alternative values in a resource bundle, as described in [Working With Text in an ADF Skin](#). For more information about configuring messages for ADF Faces components, see the "Displaying Tips, Messages, and Help" chapter in *Developing Web User Interfaces with Oracle ADF Faces*.

### 6.5.1 How to Configure an ADF Skin Property to Apply to a Message

You add a pseudo-class to the component's selector for the message type that you want to configure. You then define style properties for the pseudo-class using the Properties window.

To configure an ADF skin property to apply to a message:

1. In the Selector Tree of the selectors editor, expand the Faces Component Selectors section and select the selector for the ADF Faces component for which you want to configure the style properties to apply to a message.

For example, select the `af|inputText` selector to configure the style properties to apply to the ADF Faces `inputText` component.

2. Click the **Add Pseudo- Class** icon to display a list of the available pseudo-classes for the selector that you selected in Step 1.
3. Select the pseudo-class that corresponds to the message for which you want to configure style properties. The following pseudo-classes are available for the ADF Faces components:

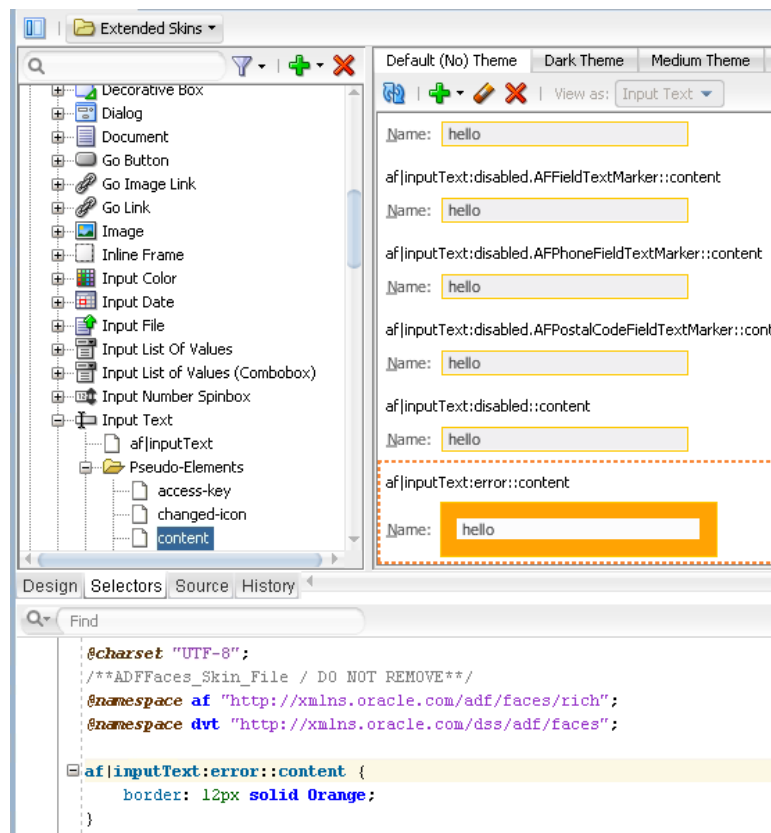
- `fatal`

- error
  - warning
  - confirmation
  - info
4. Configure the style properties that you want to apply to the component at runtime when the application displays a message with the component.

## 6.5.2 What Happens When You Configure ADF Skin Properties to Apply to Messages

The selectors editor writes the values that you specify for the selector's pseudo-class in the Properties window to the source file for the ADF skin. For example, assume that you set the value of the Border property to orange for the `content` pseudo-element of the `af|inputText` selector's `error` pseudo-class. Figure 6-15 shows the syntax entries that appear in the source file of the ADF skin and the change in the Preview Pane of the selectors editor.

**Figure 6-15** Style Properties for an `inputText` Component's Error Message



## 6.6 Configuring an ADF Skin for Accessibility

Oracle ADF provides application accessibility support to make applications developed using ADF Faces components usable for persons with disabilities. You can define style properties in your ADF skin specifically for persons with disabilities as part of efforts to make your application accessible. You preface these style properties with the `@accessibility-profile` rule.

The `@accessibility-profile` rule allows you to define style properties for the `high-contrast` and `large-fonts` accessibility profile settings that you can specify in the `trinidad-config.xml` file. For more information about the `trinidad-config.xml` file, see [Configuration Files for an ADF Skin](#).

Define style properties for the `high-contrast` accessibility profile where you want background and foreground colors to contrast highly with each other. Define style properties for the `large-fonts` accessibility profile for cases where the user must be allowed to increase or decrease the text scaling setting in the web browser. Defining `large-fonts` does not mean that the fonts are large, but rather that they are scalable fonts or dimensions instead of fixed pixel sizes.

[Example 6-1](#) shows style properties that get applied to the `af|column::sort-ascending-icon` pseudo-element when an application renders using the `high-contrast` accessibility profile.

For more information about developing accessible ADF Faces pages and accessibility profiles, see the "Developing Accessible ADF Faces Pages" chapter in *Developing Web User Interfaces with Oracle ADF Faces*.

**Example 6-1 Style Properties Defined Using the `@accessibility-profile`**

```
@accessibility-profile high-contrast {
  af|calendar::calendar-icon-reminder-style,
  af|calendar::calendar-icon-recurring-style,
  af|calendar::calendar-icon-recurring-change-style {
    -tr-inhibit: all;
  }
}
```

## 6.6.1 How to Configure an ADF Skin for Accessibility

You define style properties for the selector or selectors pseudo-elements that you want to configure and preface these style properties with the `@accessibility-profile` rule.

To configure an ADF skin for accessibility:

1. Define style properties for the selectors and selectors' pseudo-elements that you want to configure, as described in [Changing a Component-Specific Selector](#).
2. In the source file for the ADF skin, preface the skinning keys that you configured with the `@accessibility-profile` rule, as illustrated in [Example 6-1](#).



---

# Working with Images and Color in Your ADF Skin

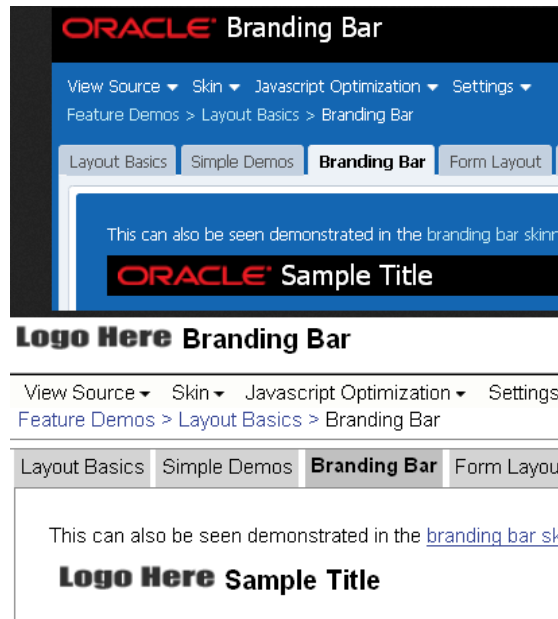
This chapter describes how to work with images and color in an ADF skin. Features, such as how you change images using the provided editors, are described in addition to how to work with the color categories in a Skyros-extended skin to quickly change the color palette that your ADF skin defines.

This chapter includes the following sections:

- [About Working with Images and Color in Your ADF Skin](#)
- [Changing Images and Colors in the ADF Skin Design Editor](#)
- [Working with Anchor Colors in an ADF Skin](#)
- [Changing an Image for a Component Selector](#)

## 7.1 About Working with Images and Color in Your ADF Skin

Apart from the `simple` skin which contains only minimal formatting, the ADF skins provided by **Oracle ADF** define color schemes and reference images to provide a colorful look and feel for applications. Changing these colors and the images that your **ADF skin** references is a task that will make a significant difference to the appearance of the application that uses your ADF skin. [Figure 7-1](#) illustrates this point by showing the same page from an application that renders using two different ADF skins (`skyros` and `simple`). The `skyros` skin defines the look and feel of the application page in the upper part of [Figure 7-1](#). It uses more color and images than the application page in the lower part that uses the `simple` skin.

**Figure 7-1 ADF Skin Using Images and Color**

Among the selectors in the ADF skins provided by Oracle ADF that reference images are those in the following list. A short description of the role that the referenced images performs in skinning the **web application** also appears.

- `af|document::splash-screen-icon`  
This component-specific selector specifies the icon that appears within a splash screen when a web applications loads in a browser.
- `af|column::sorted-descending-icon-style`  
This component-specific selector specifies the icon that renders for the sorted descending indicator in a column.
- `.AFFatalIcon:alias`  
This global selector alias specifies the icon to appear if a fatal error occurs on a page

One example of color that the ADF skins provided by Oracle ADF define is the `.AFHoverPrimaryColor:alias` global selector alias. This global selector alias defines the background color when, for example, a user hovers a cursor over a button component. Another example is the `.AFBackgroundColor:alias` global selector alias that defines the background color used for the main content area of your page.

The editor for ADF skins in JDeveloper provides features to help change the colors and images that your ADF skin uses. The availability of some or all of these features depends on the ADF skin that you extend, as described in the following list:

- If your ADF skin extends the Skyros skin  
JDeveloper enables the design editor where you can use various color pickers and other controls to change some of the more frequently used colors and images in an ADF skin. For more information, see [Changing Images and Colors in the ADF Skin Design Editor](#).
- If your ADF Skin extends from the Alta skin

Use the selectors editor to change images, as described in [Changing an Image for a Component Selector](#).

---



---

**Note:**

Alternatively, use the Theme Editor described in [Working with the Theme Editor](#), to change the colors and images in an ADF skin that extends from the Alta or Skyros skins.

---



---

## 7.2 Changing Images and Colors in the ADF Skin Design Editor

The design editor appears when you create an ADF skin that extends from the Skyros ADF skin. You access it by clicking the Design tab of the open ADF skin. For an overview of the design editor, see [Working with the ADF Skin Design Editor](#).

Examples of tasks that you can carry out using this editor include the following:

- Change the default text color in ADF skins that extend from Skyros
- Change the background color that appears to highlight when you hover over components such as the `button` component
- Replace icons

You can change all or individual icons for components, status, and animation icons using the Replace Icons dialog that you invoke when you click one of the **Status Icon**, **Animations**, or **Component** buttons in the Images area of the General tab. For more information, click **Help** on the Replace Icons dialog.

[Figure 7-2](#) shows an ADF skin that extends from the Skyros ADF skin where the following changes have been made:

- In the General tab

---



---

**Note:**

Red rectangles in [Figure 7-2](#) identify the controls used to make the changes in the General tab. Red arrows point to a corresponding result in the sample page.

---



---

- Change the main default text color to `Fuchsia`

This changes the color value of the `AFTextColor` global selector alias which is an anchor color. This change also affects the global selector aliases (for example, `AFTextPrimaryColor` and `AFTextSecondaryColor`) that set color properties which derive their hue value from the `AFTextColor` global selector alias. For more information about this relationship, see [Working with Anchor Colors in an ADF Skin](#).

- Change the primary accent color to `Black`

This changes the color that renders when a cursor hovers over a component such as a `button` component. The global selector aliases that sets this color property are `AFHoverPrimaryColor` and `AFButtonGradientStartHoverColor`. Other global selector aliases use the `AFButtonGradientStartHoverColor` global selector alias to derive the hue

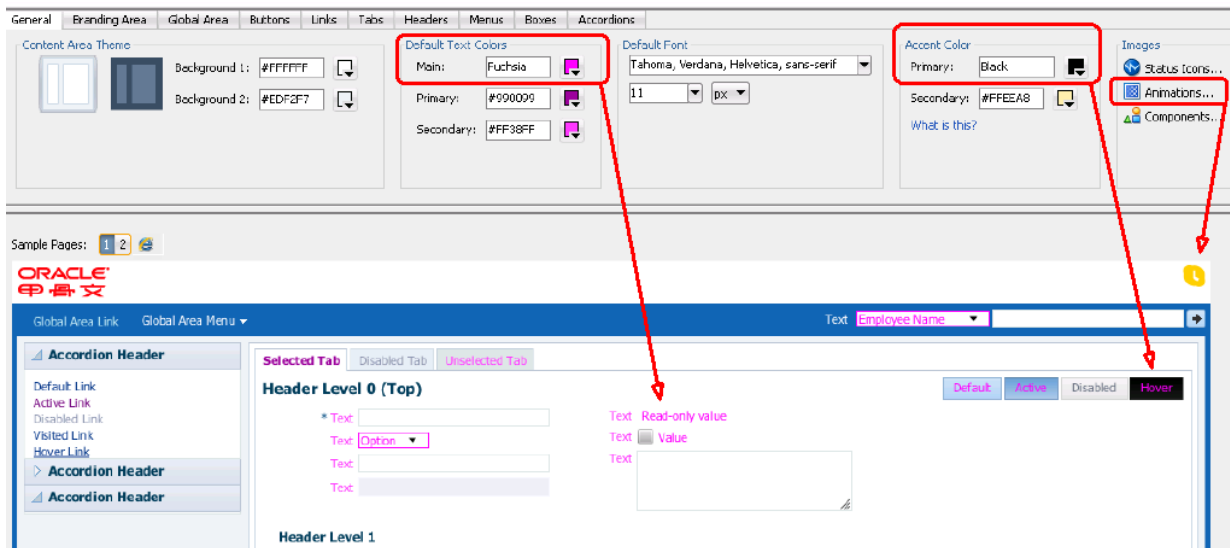
value of the color properties that they set. Examples of global selector aliases that derive their color property from the `AFButtonGradientStartHoverColor` global selector alias include `AFButtonBorderBottomHoverColor` and `AFButtonBorderHoverColor`. For more information about this relationship, see [Working with Anchor Colors in an ADF Skin](#).

- Change one of the animated icons that indicate connection status

In this example, the animation icon referenced by the `af | statusIndicator :: idle-icon` was changed.

- In the Branding Area tab
  - Change the color property that determines the background color for the branding area (`AFBrandingBackgroundColor` global selector alias) to transparent.
  - Change the image file that is used to render the logo in the branding area.

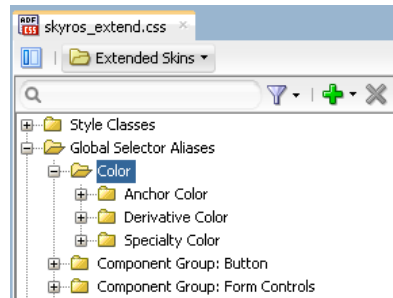
**Figure 7-2** Changing Colors and Icons in ADF Skin Design Editor



## 7.3 Working with Anchor Colors in an ADF Skin

An ADF skin that extends from the Alta or Skyros families of ADF skin defines global selector aliases that group colors into one of three categories, as illustrated in [Figure 7-3](#). Changing the value of `color` properties for global selector aliases categorized as anchor colors can help you to quickly change the color palette that your ADF skin defines.

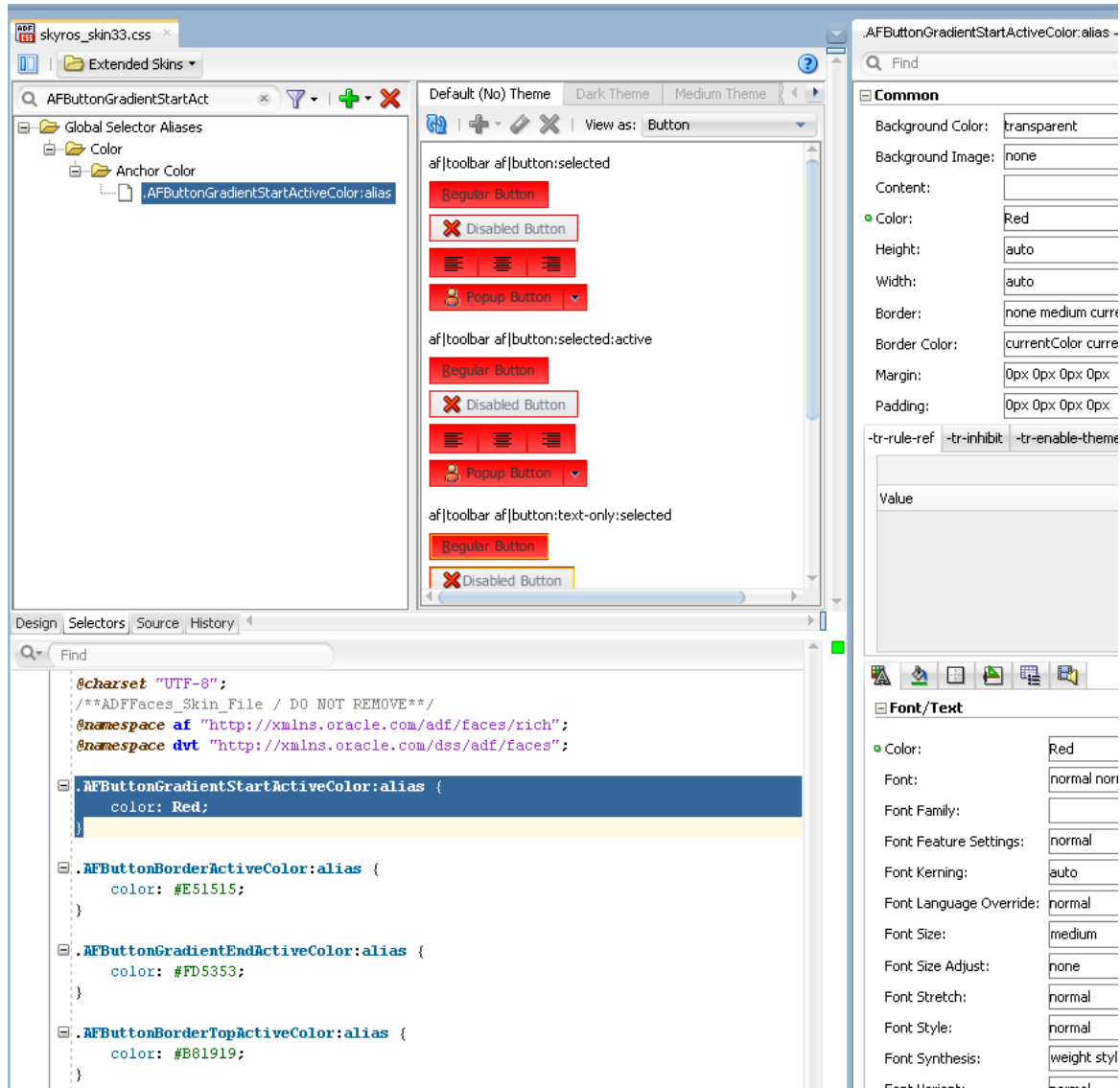


**Figure 7-3 Color Categories Skyros's Global Selector Aliases**

- **Anchor Color:** These global selector aliases define the base colors for your ADF skin. For example, the `AFButtonGradientStartColor` global selector alias defines the start gradient color for a button.
- **Derivative Color:** These global selector aliases derive the hue value for their color properties from anchor colors. The global selector aliases in [Example 7-1](#) all derive their hue value from the `AFButtonGradientStartActiveColor` global selector alias. JDeveloper propagates any change that you make to the anchor color to the derivative color. The derivative colors inherit any change that you make to an anchor color using the editor for ADF skins in JDeveloper.
- **Specialty Color:** These global selector aliases define color properties that do not derive their hue value from anchor colors and are not anchor colors for other colors. For example, the `AFCarousel1FocusBorderColor` global selector alias that defines the border color when the `carousel1` component has focus.

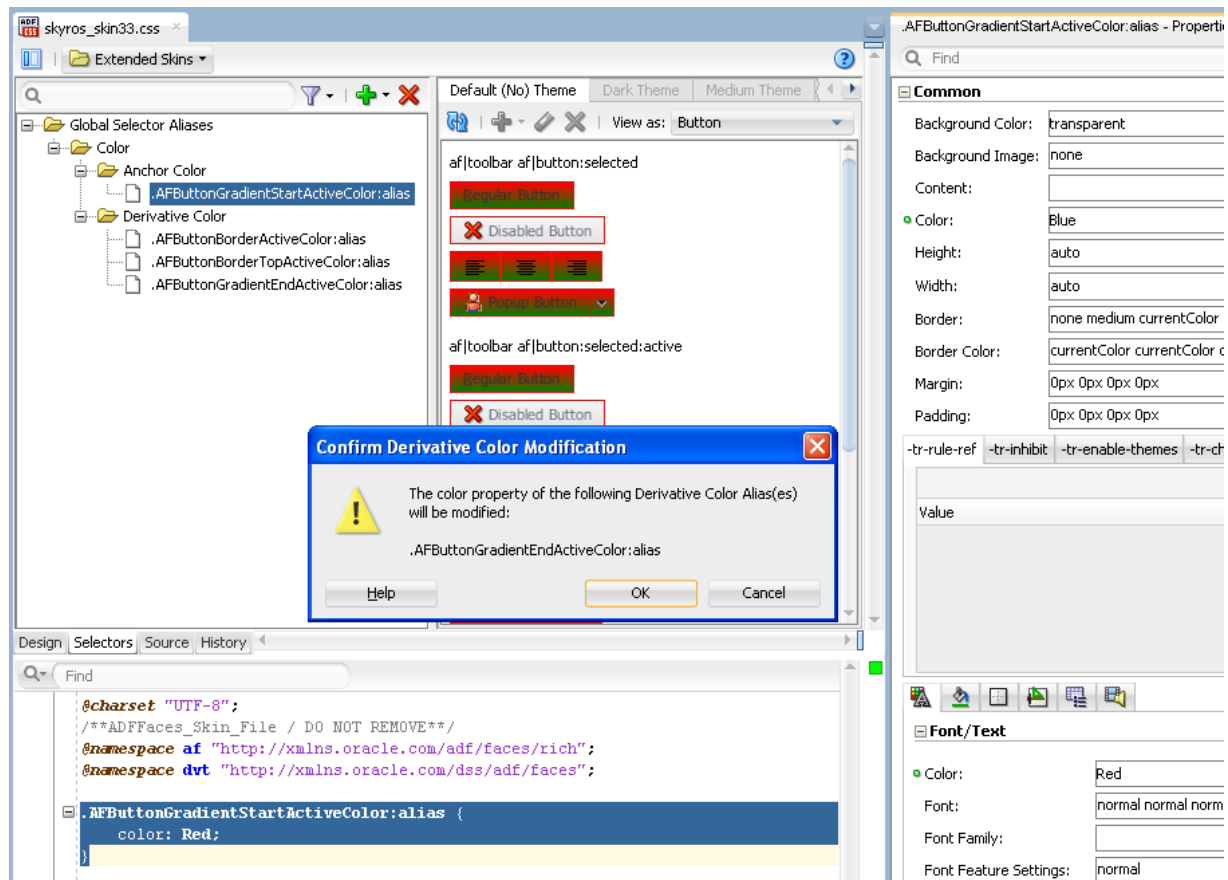
[Figure 7-4](#) shows the result of changing the default value of the `AFButtonGradientStartActiveColor` global selector alias. The editor for ADF skins in JDeveloper also updates the values of the derivative colors that derive their hue value from the anchor color.

Figure 7-4 Modified Anchor Color and Effect on Derivative Colors



If you change the color property of a derivative color and later make a change to the associated anchor color, the editor for ADF skins in JDeveloper displays a Confirm Derivative Color Modification dialog to warn you that the change you make to the anchor color will override the change that you made to the derivative color, as illustrated in Figure 7-5. Click **OK** to make the change to the anchor color and to override the already-defined value for the derivative color.

Figure 7-5 Overriding a Derivative Color



Example 7-1 shows entries from the Skyros ADF skin (`skyros-v1-desktop.css`) that define the default values for the `AFButtonGradientStartActiveColor` global selector alias and its associated derivative colors. These global selector aliases share the same hue value (209) but specify different values for the saturation and lightness values.

Example 7-2 shows the same global selector aliases referenced in Example 7-1. In Example 7-2, an ADF skin extends from Skyros and changes the value of the `color` property of the `AFButtonGradientStartActiveColor` global selector alias to `#6CD5A1`. The editor for ADF skins in JDeveloper modifies the color properties of the global selector aliases that derive their color value from the anchor color.

#### Example 7-1 Global Selector Aliases with Anchor and Derivative Colors in Skyros

```
/* Anchor, hsl(209, 56%, 63%), #6AA1D5 */
.AFButtonGradientStartActiveColor:alias {
color: #6AA1D5;
}

/* Derivative of AFButtonGradientStartActiveColor, hsl(209, 32%, 54%),
#648BAF */
.AFButtonBorderTopActiveColor:alias {
color: #648BAF;
}

/* Derivative of AFButtonGradientStartActiveColor, hsl(209, 39%, 62%),
#789FC4 */
.AFButtonBorderActiveColor:alias {
```

```

color: #789FC4;
}

/* Derivative of AFButtonGradientStartActiveColor, hsl(209, 54%, 79%),
#ACCAE6 */
.AFButtonGradientEndActiveColor:alias {
color: #ACCAE6;
}

```

**Example 7-2 Modified Anchor and Derivative Colors**

```

.AFButtonGradientStartActiveColor:alias {
color: #6CD5A1;
}

.AFButtonBorderTopActiveColor:alias {
color: #64AF8A;
}

.AFButtonGradientEndActiveColor:alias {
color: #ADE6CA;
}

.AFButtonBorderActiveColor:alias {
color: #79C39E;
}

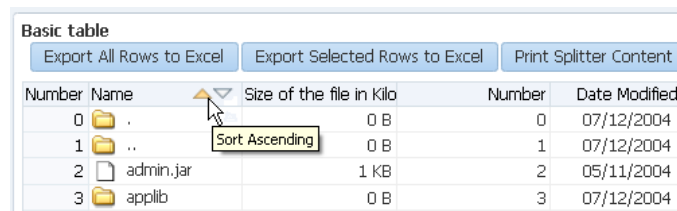
```

## 7.4 Changing an Image for a Component Selector

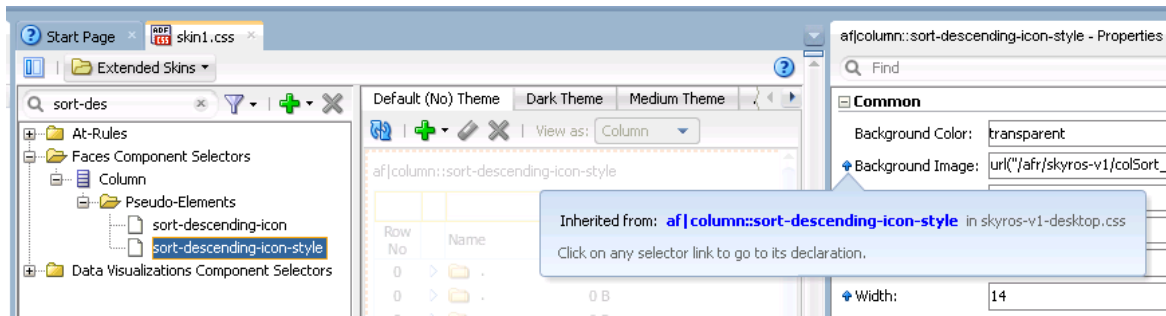
Many **ADF Faces** and **ADF Data Visualization components** reference images using selectors. These images display in the background of the component or render as icons or controls on the component. When you create an ADF skin, the ADF skin that you extend from provides the values for these selectors, such as the relative path to an image and the sizes for height and width.

**Figure 7-6** shows a runtime view of the table component rendering a control that sorts the data in a table column in ascending order. The image that renders this control is referenced by the ADF Faces column component's `sort-ascending-icon-style` selector.

**Figure 7-6 Image Referenced by the sort-ascending-icon-style Selector**



**Figure 7-7** shows a design-time view where an ADF skin inherits values for the column component's `sort-descending-icon-style` selector from the extended ADF skin. The values inherited include the file name for the image used as an icon (`colSort_asc_ena.png`), the height, and the width for the image.

**Figure 7-7** Inherited Values for the *sort-descending-icon-style* Selector

Other examples of ADF Faces and ADF Data Visualization components that expose selectors which reference images associated with the component include the following:

- ADF Faces `progressIndicator` component exposes the `determinate-empty-icon-style` selector.
- ADF Faces `panelAccordion` component exposes the `disclosed-icon-style` selector.
- ADF Data Visualization `mapToolBar` component exposes the `zoomin-enable-icon` selector.

If you decide that you want to modify the image that is associated with a component selector, you need to modify the selector in your ADF skin and copy the image into the project for your ADF skin. You can copy images individually using the procedure in [How to Copy an Image into the Project](#).

After you import an image into your project, the selector that references the image uses a URL in the source file of the ADF skin to refer to this image. Note that this URL is updated when you deploy your ADF skin (and associated files) in an ADF Library JAR, as described in [Packaging an ADF Skin into an ADF Library JAR](#).

**Tip:**

Associate an image with a global selector alias. If multiple component selectors reference the global selector alias, you only need to make one change if you want to use a different image at a later time (change the image associated with the global selector alias). For more information about global selector aliases, see [Creating a Global Selector Alias](#).

If your ADF skin extends the Skyros ADF skin, you can change some of the more frequently used images in the design editor, as described in [Changing Images and Colors in the ADF Skin Design Editor](#).

### 7.4.1 How to Copy an Image into the Project

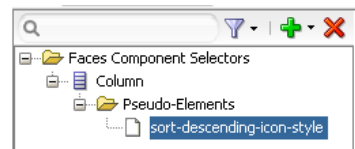
You use a context menu to copy an image that an extended ADF skin references into a directory of the project for your ADF skin. You then make the changes that you want to the image.

To copy an ADF skin image into your project:

1. In the **Selector Tree** of the selectors editor, select the selector that references the image you want to change.

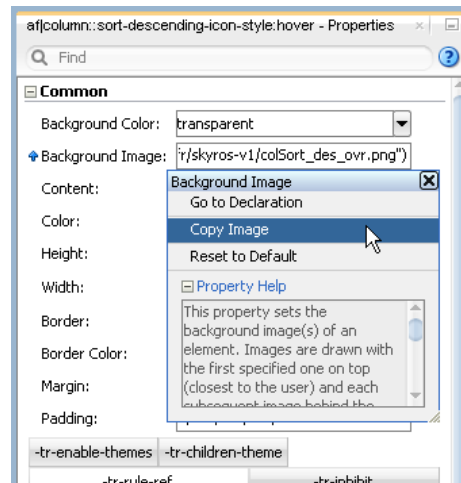
For example, select the ADF Faces `column` component's `sort-descending-icon-style` selector to change the sort ascending icon, as shown in [Figure 7-8](#).

**Figure 7-8** Column Component's `sort-descending-icon-style` Selector



- In the Properties window, expand the **Common** section and select **Copy Image** from the Background Image list, as shown in [Figure 7-9](#).

**Figure 7-9** Copy Image Menu to Import an Image into a JDeveloper Project



This copies the image into the project for your ADF skin.

## 7.4.2 What Happens When You Copy an Image into the Project

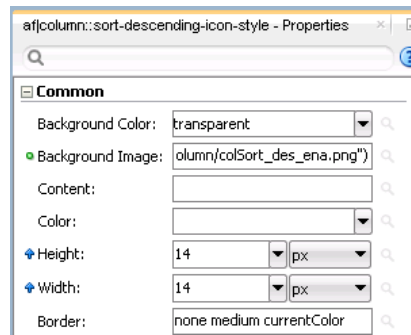
The image is copied into a subdirectory that is generated in the project of your ADF skin. For example, if you decided to copy the image that the ADF Faces `column` component's `sort-ascending-icon-style` selector references, the `colSort_asc_ena.png` file is copied to the following directory:

```
/public_html/skins/skin1/images/af_column
```

where `af_column` refers to the ADF Faces `column` component.

The relative URL value of the property in the Properties window is modified to reference the new location of the image. [Figure 7-10](#) shows an example.

In addition, the Properties window indicates that the selector no longer inherits the image from the extended ADF skin by displaying a green icon to the left of the property label. [Figure 7-10](#) shows the Properties window after importing the `colSort_asc_ena.png` file into the project for the ADF skin. Note that the ADF skin still inherits the values for the Height and Width properties from the extended ADF skin.

**Figure 7-10 Properties Window After Importing an Image into an ADF Skin**

Finally, CSS syntax appears in the source file of your ADF skin. [Example 7-3](#) shows the CSS syntax that corresponds to the values shown in [Figure 7-10](#).

**Example 7-3 CSS Syntax in Source File of ADF Skin After Importing an Image**

```
af|column::sorted-ascending-icon-style
{
    background-image: url("images/af_column/colSort_des_ena.png");
}
```





## Working With Text in an ADF Skin

This chapter describes how to work with text in an ADF skin. Key concepts, such as how the resource strings that ADF Faces components render at runtime are stored in resource bundles, are described in addition to how you can specify additional resource bundles with different resource strings.

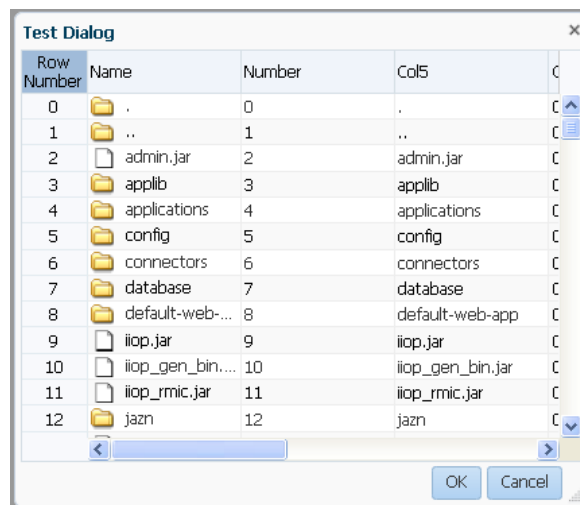
This chapter includes the following sections:

- [About Working with Text in an ADF Skin](#)
- [Using Text From Your Own Resource Bundle](#)

### 8.1 About Working with Text in an ADF Skin

The source file for an ADF skin does not store any text that ADF Faces components render in the user interface of your application. Applications that render ADF Faces components abstract the text that these components render as resource strings and store the resource strings in a **resource bundle**. For example, [Figure 8-1](#) shows an ADF Faces dialog component that renders buttons with **OK** and **Cancel** labels.

**Figure 8-1** ADF Faces dialog Component



The text that appears as the labels for these buttons (**OK** and **Cancel**) is defined in a resource bundle and referenced by a resource string. If you want to change the text that appears in the button labels, you create a resource bundle where you define the values that you want to appear by specifying alternative text for the following resource strings:

- `af_dialog.LABEL_OK`
- `af_dialog.LABEL_CANCEL`

---

**Note:**

By default, a resource bundle (`skinBundle.properties`) is created in your project when you create a new ADF skin, as described in [Creating an ADF Skin File](#).

---

In addition to the resource strings that define the text to appear in the user interface for specific components, there are a range of resource strings that define text to appear that is not specific to any particular component. These resource strings are referred to as global resource strings. For more information about the resource strings for ADF Faces components and global resource strings, see the *Tag Reference for Oracle ADF Faces Skin Selectors*.

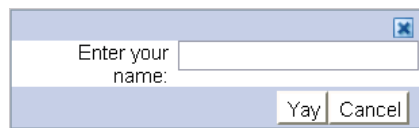
ADF Faces components provide automatic translation. The resource bundle used for the ADF Faces components' skin is translated into 28 languages. If, for example, an end user sets the browser to use the German (Germany) language, any text contained within the components automatically displays in German. For this reason, if you create a resource bundle for a new ADF skin, you must also create localized versions of that resource bundle for any other languages your web application supports.

For more information about creating resource bundles, resource strings, and localizing ADF Faces components, see the "Internationalizing and Localizing Pages" chapter in *Developing Web User Interfaces with Oracle ADF Faces*.

## 8.2 Using Text From Your Own Resource Bundle

If you enter alternative text in a resource bundle to override the default text values that render in the user interface of the ADF Faces components in your application, you need to specify this resource bundle for your application. At runtime, the application renders the alternative text in your resource bundle for the resource strings that you override. For resource strings that you do not override, the application renders the text defined in the base resource bundle. For example, [Figure 8-4](#) shows an ADF Faces dialog component where the application developer overrides the default value for the `af_dialog.LABEL_OK` resource string from OK to Yay while the default value for the `af_dialog.LABEL_CANCEL` resource string remains unchanged. That is, the application developer did not define a value for the `af_dialog.LABEL_CANCEL` resource string in a resource bundle; the application references the base resource bundle for this resource string's value.

**Figure 8-2** Overridden and Default Values Resource Strings



For more information about how to create a resource bundle and how to define string key values, see the "Internationalizing and Localizing Pages" chapter in *Developing Web User Interfaces with Oracle ADF Faces*.

### 8.2.1 How to Specify an Additional Resource Bundle for an ADF Skin

You specify a resource bundle for your ADF skin by adding its name and location as a value to the `bundle-name` property in the `trinidad-skins.xml` file.

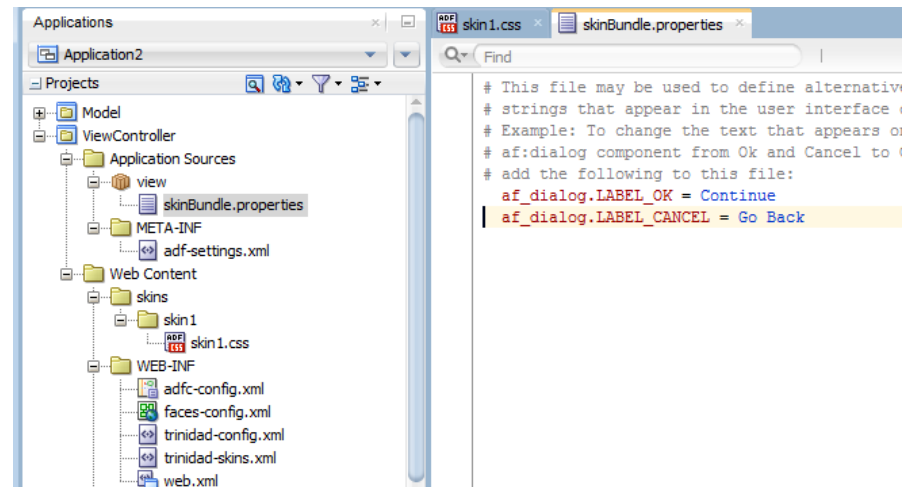
To specify an additional resource bundle for an ADF skin:

1. In the Applications window, double-click the `trinidad-skins.xml` file for your application. By default, this is under the Web Content/WEB-INF node.
2. In the Structure window, right-click the skin node for which you want to add an additional resource bundle and choose **Insert inside skin > bundle-name**.
3. In the Properties window, specify the name and location for your resource bundle as a value for the `bundle-name` property.

For example, the resource bundle that is created by default after you create the first ADF skin in your project, as illustrated in [Figure 8-3](#), specifies the following value for the `<bundle-name>` element:

```
<bundle-name>resources.skinBundle</bundle-name>
```

**Figure 8-3** Default Resource Bundle for an ADF Skin



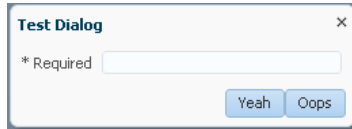
## 8.2.2 What Happens When You Specify an Additional Resource Bundle for an ADF Skin

The `trinidad-skins.xml` file references the resource bundle that you specified as a value for the `bundle-name` property, as shown in the following example.

```
<skin>
  <id>skin1.desktop</id>
  <family>skin1</family>
  <extends>skyros-v1.desktop</extends>
  <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
  <style-sheet-name>skins/skin1/skin1.css</style-sheet-name>
  <bundle-name>resources.skinBundle</bundle-name>
</skin>
```

At runtime, the application renders text values that you specified in your resource bundle to override the default text values. For example, assume that you defined a resource bundle where you specified Yeah as the value for the `af_dialog.LABEL_OK` resource string and Oops as the value for the `af_dialog.LABEL_CANCEL`. [Figure 8-4](#) shows a dialog component that renders labels using these values.

**Figure 8-4** *Dialog Rendering Labels Defined in a Custom Resource Bundle*



---

## Working With Global Selector Aliases

This chapter describes how to work with global selector aliases. Information on how to create, modify, and apply a global selector alias is provided in addition to describing how to reference a property value from another selector.

This chapter includes the following sections:

- [About Global Selector Aliases](#)
- [Creating a Global Selector Alias](#)
- [Modifying a Global Selector Alias](#)
- [Applying a Global Selector Alias](#)
- [Referencing a Property Value from Another Selector](#)

### 9.1 About Global Selector Aliases

A global selector alias defines style properties in one location in the **ADF skin** that you can apply to multiple **ADF Faces** and **ADF Data Visualization components**. A global selector alias may also be referred to as a selector alias, or simply a selector. The ADF skins provided by Oracle ADF, described in [Inheritance Relationship of the ADF Skins Provided by Oracle ADF](#) and [ADF Skins Provided by Oracle ADF](#) make extensive use of global selector aliases to define common style properties for text, messages, icons, colors and different groups of components. Many component-specific selectors inherit the styles defined for these global selector aliases. For example, the `.AFDefaultFontFamily:alias` global selector alias defines a default font family for all ADF Faces components in your application that display text. Any ADF skin that you create by extending from one of the ADF skins provided by Oracle ADF inherits the properties defined in the `.AFDefaultFontFamily:alias` global selector alias. [Figure 9-1](#) shows how the selectors editor displays that the `af|button` selector inherits the value for font family from the `.AFDefaultFontFamily:alias` global selector alias.

Figure 9-1 Component Selector Inheriting Value from Global Selector Alias

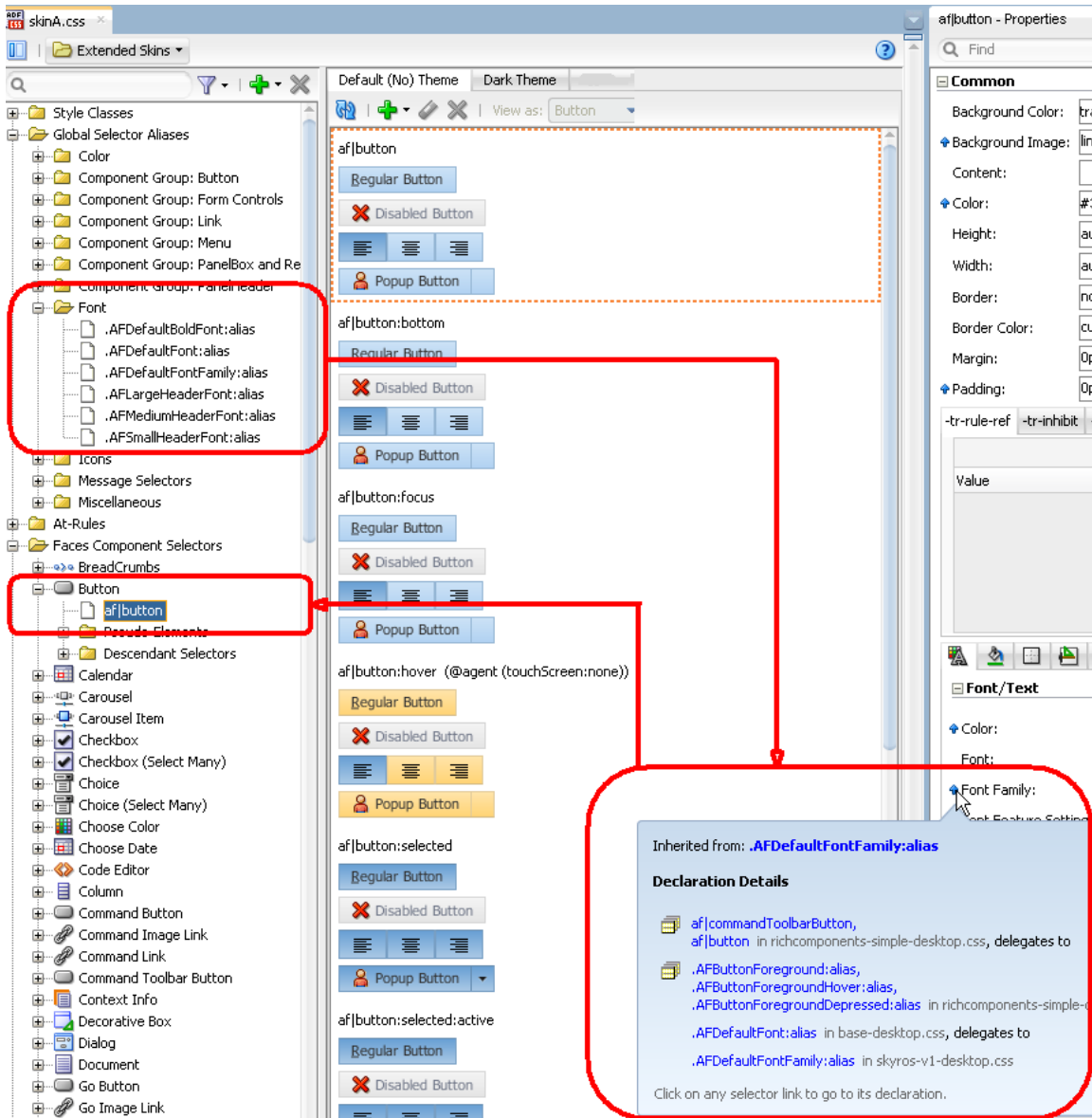


Figure 9-1 also shows the different categories of global selector aliases. Each category groups global selector aliases according to their purpose:

- Color:** Defines colors used by the ADF skins provided by Oracle ADF. Many global selector aliases that you may want to override appear in this category because they determine most of the colors that appear in a web application. Changes that you make to these global selector aliases have the most effect if you extend the Alta or Skyros ADF skins described in [ADF Skin Selectors and Icon Images](#). You can change the color palette of an ADF skin that extends from these ADF skins relatively quickly by changing the global selector aliases that are categorized as anchor colors. For more information, see [Working with Anchor Colors in an ADF Skin](#).

**Tip:**

As with other global selector aliases, you can view which component-specific selectors inherit the values defined in a specific global selector using the **View as** list.

- **Component Group: Button:** Defines style properties inherited by selectors for many of the ADF Faces components that render buttons. For example, the `.AFButtonAccessKeyStyle:alias` global selector alias defines style properties for the access key rendered by the ADF Faces button and dialog components among others.
- **Component Group: Form Controls:** Defines style properties for form controls.
- **Component Group: Link:** Defines style properties for many of the components that render links.
- **Component Group: Menu:** Defines style properties for many of the components that render menus.
- **Component Group: PanelBox and Region:** Defines style properties for the `panelBox` and `region` components.
- **Component Group: PanelHeader:** Defines style properties for the `panelHeader` components.
- **Font:** Defines style properties for fonts. For example, the `.AFDefaultFontFamily:alias` global selector alias defines the style properties inherited by many of the ADF Faces component selectors.
- **Icons:** Defines the style properties that apply to icons that render in multiple components.
- **Message Selectors:** Defines style properties for messages that ADF Faces input components display when they render different types of messages. For more information, see [Configuring ADF Skin Properties to Apply to Messages](#).
- **Miscellaneous:** Defines global selector aliases that do not fit in the other categories. For example, the `.AFDynamicHelpIconStyle:alias` global selector alias defines the style to use for the dynamic help icon.

For detailed descriptions of the global selector aliases, see the *Tag Reference for Oracle ADF Faces Skin Selectors*. Global selector aliases that you define appear under the Global Selector Aliases node, as shown by the entry for the `.UserDefined:alias` in [Figure 9-1](#).

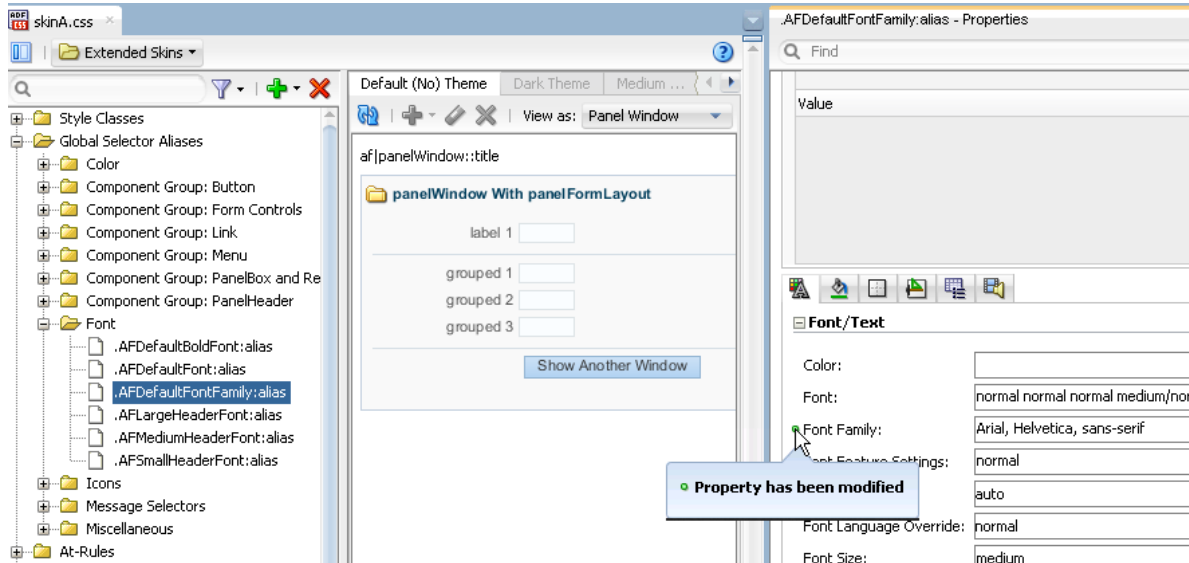
The **View as** list displays the list of components that reference a global selector alias when you select a global selector alias in the **Selector Tree**. In [Figure 9-2](#), the user selected Panel Window from the list because the `panelWindow` component references the global selector alias.

**Note:**

Sometimes components appear in the **View as** list for which the style properties defined in the global selector alias do not render in the component. This may be because the component initially referenced the global selector alias in an extended ADF skin and your ADF skin overrides the global selector alias for that component. Alternatively, it may be because the component itself overrides the global selector alias using one of its style-related attributes (`styleClass` or `inlineStyle`).

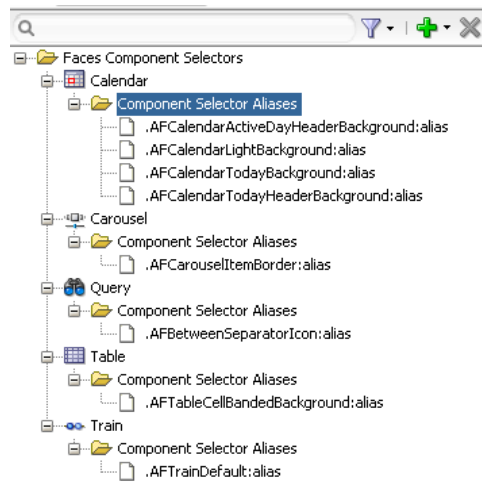
In [Figure 9-2](#), the user has changed the inherited value for the `.AFDefaultFontFamily:alias` global selector alias and viewed the resulting change as it applies to the `panelWindow` component. All selectors that inherit the value of the `.AFDefaultFontFamily:alias` global selector alias will render at runtime using the font family defined in the ADF skin. For example, both the `dialog` and `panelWindow` components render using this font family.

**Figure 9-2 ADF Skin Changing a Global Selector Alias**



In addition to the global selector aliases already described, a number of component selectors define selector aliases that are specific to these components only. These selector aliases appear under the nodes for the component selectors in the Selector Tree. [Figure 9-3](#) shows examples from a number of the component selectors that expose these types of selector aliases.



**Figure 9-3 Component Selector Aliases**

## 9.2 Creating a Global Selector Alias

You can create a global selector alias to define in one location the style properties that you want a number of selectors to reference. You enter the name of the new global selector alias in the Create Alias Selector dialog. The editor for ADF skins in JDeveloper appends the keyword `:alias` and prepends `.` to the name that you enter in the dialog. For example, if you enter `myGlobalSelector` as the name in the dialog, the resulting name that appears in the user interface and in the source file of the ADF skin is:

```
.myGlobalSelector:alias
```

The keyword `:alias` identifies your global selector alias as a CSS pseudo-class and serves as a syntax aid to organize the CSS code in the source file of your ADF skin.

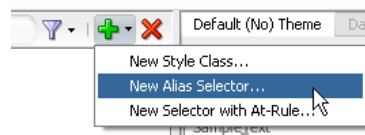
After you create a global selector alias, you modify it to define the style properties that you want it to contain. For more information, see [Modifying a Global Selector Alias](#).

### 9.2.1 How to Create a Global Selector Alias

You can create a global selector alias that defines the style properties that you want a number of user interface components to use.

To create a global selector alias:

1. In the Selector Tree of the selectors editor, select **New Alias Selector** from the dropdown list, as illustrated in [Figure 9-4](#).

**Figure 9-4 New Alias Selector Option in the Selector Tree**

The Create Alias Selector dialog opens.

2. Enter a name for the global selector alias in the Alias Selector Name field.

**Tip:**

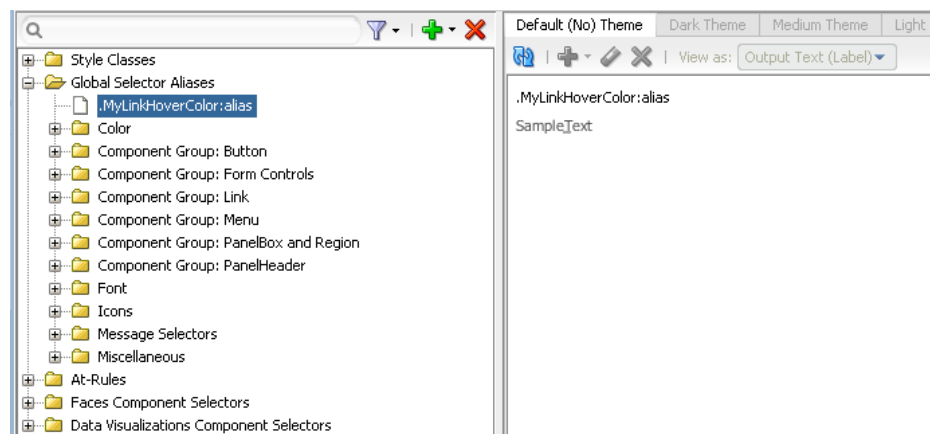
Enter a name for the global selector alias that indicates the purpose it serves. For example, `MyLinkHoverColor` for a global selector alias that is to change the color of a link when an end user hovers over the link.

3. Click **OK**.
4. In the Properties window, set values for the properties that you want to configure in the global selector alias.

## 9.2.2 What Happens When You Create a Global Selector Alias

The global selector alias appears under the Global Selector Aliases node in the Selector Tree and a visual representation as it applies to a component appears in the Preview Pane, as illustrated in [Figure 9-5](#).

**Figure 9-5** Newly-Created Global Selector Alias



CSS syntax for the global selector alias that you create appears in the source file of the ADF skin. The following example shows the entries that appear in the source file of the ADF skin in [Figure 9-5](#).

```
.MyLinkHoverColor:alias{
}
```

## 9.3 Modifying a Global Selector Alias

You can modify any of the categories of global selector alias described in [About Global Selector Aliases](#). Modifying a global selector alias that appears under the Global Selector Aliases node in the Selector Tree when you first create the ADF skin means that you override the inherited style properties defined in the parent ADF skin of your ADF skin. The parent ADF skin is the ADF skin from which your ADF skin extends. You chose the ADF skin from which to extend when you created an ADF skin, as described in [Creating an ADF Skin File](#). After you modify a global selector alias, the component-specific selectors that inherit the style properties defined in the global selector alias use the modified values.

Modifying a global selector alias that you create in your ADF skin does not override any style properties inherited from the parent ADF skin.

### 9.3.1 How to Modify a Global Selector Alias

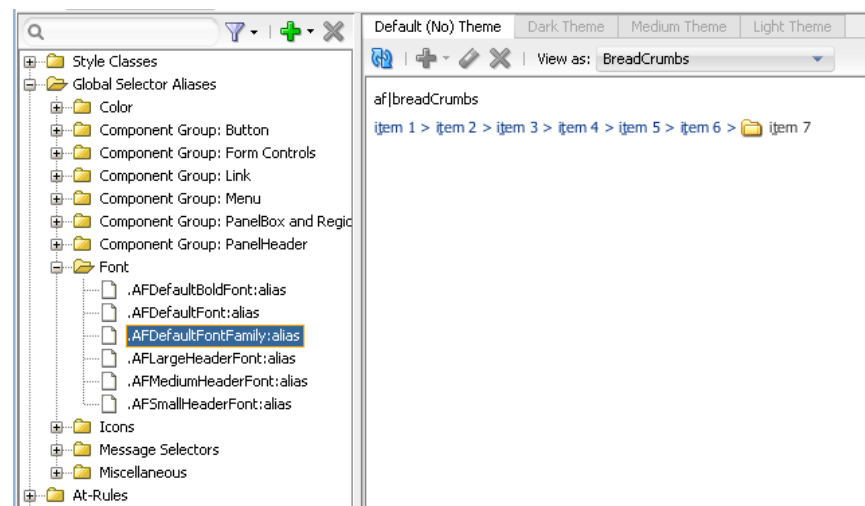
You modify a global selector alias by setting values for it in the Properties window. You then verify that the changes you make apply to the component-specific selectors as you intend.

To modify a global selector alias:

1. In the Selector Tree of the selectors editor, select the global selector alias that you want to modify.

For example, if you want to modify the global selector alias that defines the default font family, select it as illustrated in [Figure 9-6](#).

**Figure 9-6** *Modifying a Global Selector Alias*



2. In the Properties window, set values for the properties that you want to modify.
3. In the selectors editor, click the **View as** list to select a component-specific selector that inherits the property values defined in the global selector alias that you have just modified.
4. In the selectors editor, verify that the changes render for the component-specific selector as you intend. Repeat Steps 1 to 3 until you achieve the changes you want for the component-specific selectors that inherit from the global selector alias.

## 9.4 Applying a Global Selector Alias

After you create a global selector alias in your ADF skin, you need to specify the ADF Faces and ADF Data Visualization components that you want to render at runtime using the style properties that you defined in the global selector alias.

Applying a global selector alias to an ADF Faces or ADF Data Visualization component requires you to:

- Select the selector, pseudo-element, or pseudo-class for each component that you want to apply the style properties defined in the global selector alias. If you want to apply the style properties defined in your global selector alias to another global selector alias, select the target global selector alias.

- Set the global selector alias as a value for the `-tr-rule-ref` ADF skin property.

### 9.4.1 How to Apply a Global Selector Alias

You apply a global selector alias by specifying it as a value for the `-tr-rule-ref` ADF skin property.

To apply a global selector alias:

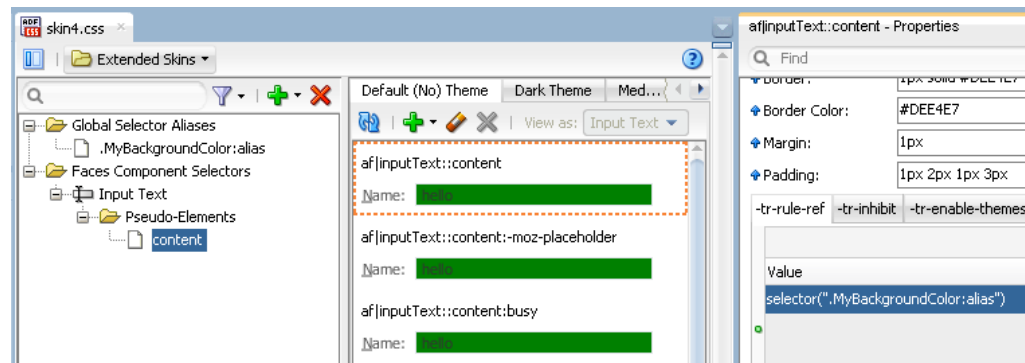
1. In the Selector Tree of the selectors editor, select the item to which you want to apply the global selector alias.

For example, select the `inputText` component's **content** pseudo-element if you want to apply the style properties defined in your global selector alias to the label for that component, as shown in [Figure 9-7](#).

2. In the Properties window, expand the **Common** section and then click the Add icon next to the `-tr-rule-ref` field.
3. Enter the name of the global selector alias. Enter the name between quotes that you preface with the `selector` keyword in the **Value** field.

For example, if the name of the global selector alias is `.MyBackgroundColor:alias`, enter `selector(".MyBackgroundColor:alias")`, as illustrated in [Figure 9-7](#).

**Figure 9-7** *inputText Component's content Pseudo-Element*

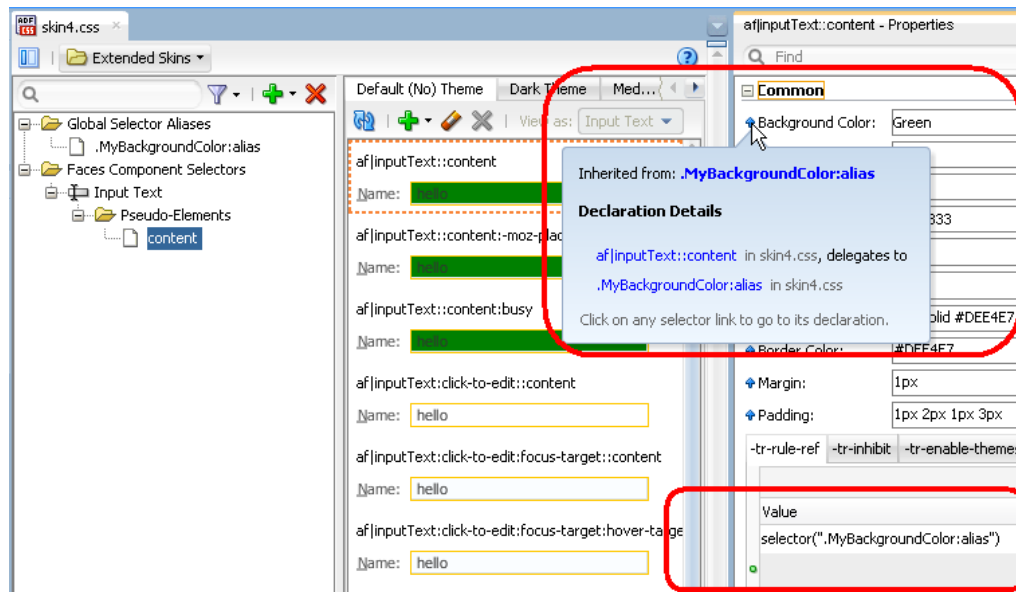


4. Click the Refresh icon in the Preview Pane to view the changes.

### 9.4.2 What Happens When You Apply a Global Selector Alias

The selector to which you applied the global selector alias inherits the style properties defined in the global selector alias. [Figure 9-8](#) shows the `content` pseudo-element for the `inputText` component's selector that inherits the style properties defined in the `.MyBackgroundColor:alias` global selector alias. The properties that inherit their values from a global selector alias when you specify the global selector alias as a value for the `-tr-rule-ref` ADF skin property update to use the inheritance icon, as shown for the **Background Color** and **Color** fields in [Figure 9-8](#).

At runtime, the `inputText` component's content area renders using the style properties defined in the global selector alias.

**Figure 9-8 Global Selector Alias Applied to inputText Component**

### 9.4.3 What You May Need to Know About Applying a Global Selector Alias

If you override a global selector alias in an extended ADF skin, component selectors that used the `-tr-rule-ref` ADF skin property to determine the value of a style property in the parent ADF skin use the overridden value of the global selector alias. The following example shows ADF skin B that extends ADF skin A. At runtime, the top of a `decorativeBox` component renders red for the `background-color` CSS property because the global selector alias in ADF skin B overrides ADF skin A.

```

/** Skin A */
/** ----- */
.MyBackColor:alias
{
    background-color: blue
}

af|decorativeBox::top
{
    -tr-rule-ref: selector(".MyBackColor:alias");
}

/** Skin B */
/** ----- */

.MyBackColor:alias
{
    background-color: Red
}

```

If you specify a style property value in an extended ADF skin where the parent ADF skin also specifies a value for the style property, the **ADF skinning framework** applies the value in the extended ADF skin. [Example 9-1](#) shows ADF skin C where the `.myClass` style class specifies Red as the value for the `background-color` CSS property. If an application uses ADF skin D (that extends ADF skin C), components that apply the `.myClass` style class apply Lime for the `background-color` CSS property. This is because the ADF skinning framework calculates the values of

statements that include values in an ADF skin (like `-tr-rule-ref`) first. The ADF skinning framework then calculates specific properties (for example, `background-color`) next. As a result, the value for the `background-color` CSS property in ADF skin D (Lime) overrides the value for the `-tr-rule-ref` ADF skin property (Blue) or inherited values from ADF skin C (Red).

---

**Note:**

If you subsequently override the `.myClass` style class as follows in ADF skin D, the value that the ADF skinning framework applies for the `background-color` CSS property is Blue:

```
.myClass {-tr-rule-ref: selector(".MyBlueColor:alias")}
```

---

Consider using tools, such as Firebug for the Mozilla Firefox browser (or similar for your browser), when you run your application to determine what style property value the ADF skinning framework applies to a component selector at runtime. For more information, see [Testing Changes in Your ADF Skin](#).

**Example 9-1 Overriding a Local Global Selector Alias**

```
/** ADF skin C */
/** ----- */
.myClass {
    background-color: Red
}

/** ADF skin D */
/** ----- */
.MyBackColor:alias {
    background-color: Blue;
}

.myClass {
    background-color: Lime;
    -tr-rule-ref: selector(".MyBackColor:alias")
}
```

## 9.5 Referencing a Property Value from Another Selector

Rather than set a specific style property for each selector to which you want to apply the style property, you can reference the value of a property using the `-tr-property-ref` ADF skin property. You can configure this ADF skin property for global selector aliases and component-specific selectors. For example, you could define a value for the `background-color` property in a global selector alias and reference this value from multiple other selectors. If you decide at a later time to change the value of the `background-color` property, you change the value in the global selector alias. All selectors that reference the `background-color` property using the `-tr-property-ref` ADF skin property update to use the change you make. The `-tr-property-ref` ADF skin property can also be used with compact CSS properties like, for example, `border`.

### 9.5.1 How to Reference a Property Value from Another Selector

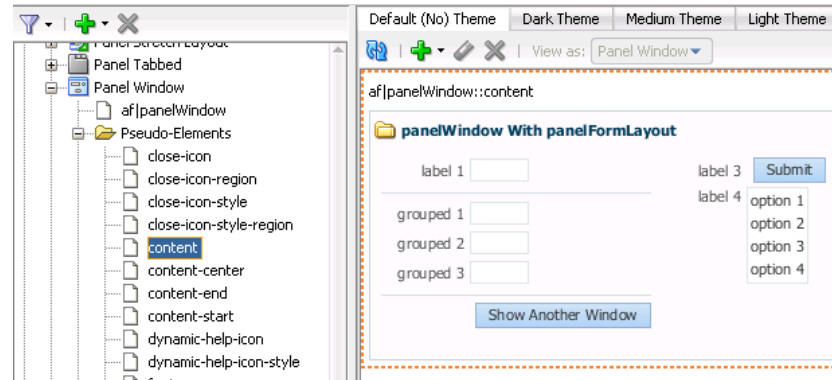
You reference the property value that you want to use for a selector using the `-tr-property-ref` ADF skin property.

To reference a property value from another selector:

1. In the Selector Tree of the selectors editor, select the selector that you want to reference a property value from another selector.

For example, if you want the content area of the `panelWindow` component to reference a style property defined in another selector, select **content** under the Pseudo-Elements node of the `panelWindow` component, as illustrated in [Figure 9-9](#).

**Figure 9-9 Panel Window Component's content Pseudo-Element**



2. In the Properties window, specify the property value that you want to reference as a value for the selector's property using the `-tr-property-ref` ADF skin property.

For example, assume that you created the following global selector alias:

```
.MyColor:alias {
  color: rgb(255,181,99);
  font-weight: bold;
}
```

and that you want to reference the `color` property from this global selector alias for the `background-color` property of the `content` pseudo-element that you selected in Step 1. In this scenario, enter the following value for the `background-color` property of the `content` pseudo-element,

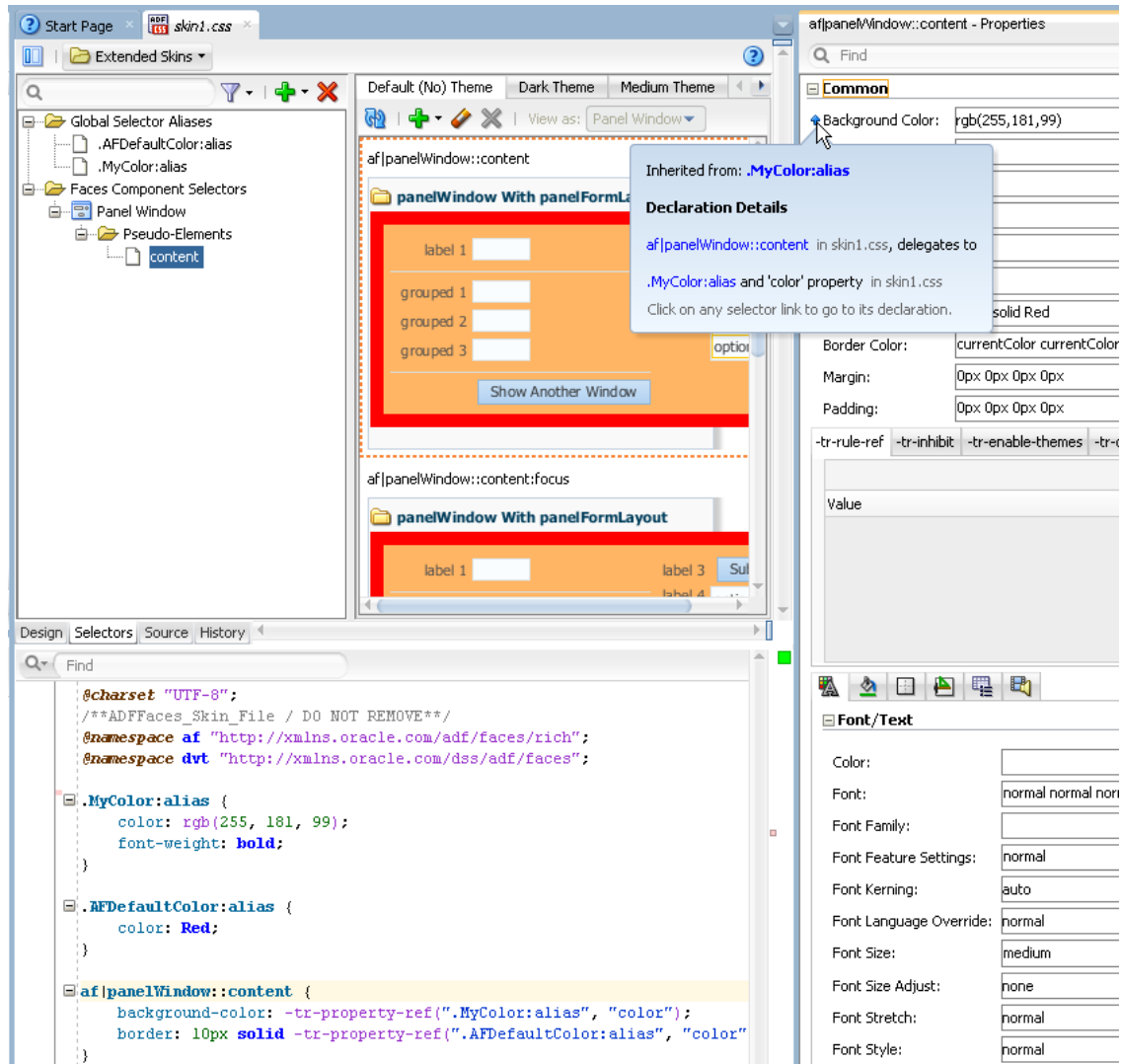
```
-tr-property-ref(".MyColor:alias", "color");
```

If you want to use the `-tr-property-ref` in compact values, enter syntax similar to the following:

```
border: 10px solid -tr-property-ref(".AFDefaultColor:alias", "color");
```

## 9.5.2 What Happens When You Reference a Property Value from Another Selector

The Properties window shows that the property for which you set a value using the `-tr-property-ref` ADF skin property to reference a value from another selector inherits its value, as illustrated in [Figure 9-10](#).

**Figure 9-10 Selector Property Referencing a Property Value from Another Selector**

Syntax similar to the following example appears in the source file of the ADF skin.

```

@charset "UTF-8";
/**ADFFaces_Skin_File / DO NOT REMOVE**/
@namespace af "http://xmlns.oracle.com/adf/faces/rich";
@namespace dvt "http://xmlns.oracle.com/dss/adf/faces";

.MyColor:alias {
  color: rgb(255, 181, 99);
  font-weight: bold;
}

.AFDefaultColor:alias {
  color: Red;
}

af|panelWindow::content {
  background-color: -tr-property-ref(".MyColor:alias", "color");
  border: 10px solid -tr-property-ref(".AFDefaultColor:alias", "color");
}

```



---

## Working with Style Classes

This chapter describes how to work with style classes. Information on how to create, modify, and apply a style class is provided in addition to describing how to configure a style class for a specific instance of a component.

This chapter includes the following sections:

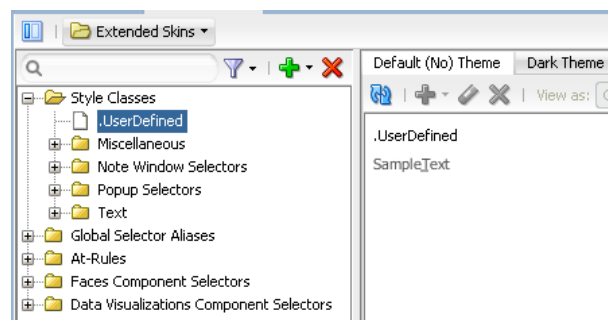
- [About Style Classes](#)
- [Creating a Style Class](#)
- [Modifying a Style Class](#)
- [Configuring a Style Class for a Specific Instance of a Component](#)

### 10.1 About Style Classes

A style class allows you to specify a number of style properties in one location in an **ADF skin** that you want to apply to specific instances of **ADF Faces** or **ADF Data Visualization components**. The style properties that you define for a style class take precedence over the style properties that you define for the component's selectors. Application developers can specify a style class as a value for the `styleClass` and `inlineStyle` attributes that many ADF Faces components expose. At runtime, the style properties that you defined in the style class get applied to the ADF Faces component rather than other style properties defined in the ADF skin. Style classes differ from the global selector aliases, described in [Working With Global Selector Aliases](#), which enable you to define style properties that you want to apply to multiple ADF Faces components.

[Figure 10-1](#) shows an ADF skin with the nodes expanded for the different categories of style classes.

**Figure 10-1** Categories of Style Class



Each category of style class serves a purpose:

- **Miscellaneous:** Miscellaneous style classes inherited from the extended ADF skins. For example, this category includes the `.AFBrandingBar` style class that can be used for a branding bar containers.
- **Note Window Selectors:** Style classes inherited from the extended ADF skins that affect the `noteWindow` component.
- **Popup:** Style classes inherited from the extended ADF skins that affect the `popup` component.
- **Text:** Style classes inherited from the extended ADF skins that determine the appearance of various types of text (for example, address fields and instruction text).

Style classes that you or other users define appear under the Style Classes node as shown by the entry for the `.UserDefined` style class in [Figure 10-1](#). For detailed descriptions of the style classes in the ADF skins that Oracle ADF provides, see the *Tag Reference for Oracle ADF Faces Skin Selectors*.

## 10.2 Creating a Style Class

You can create a new style class in your ADF skin or override a style class that your ADF skin inherits from the ADF skin that it extends.

After you create a style class, you modify it to define the style properties that you want it to contain. For more information, see [Modifying a Style Class](#).

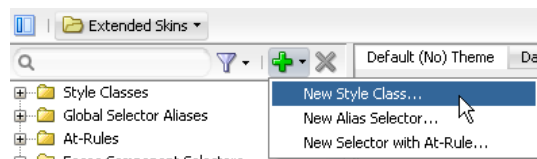
### 10.2.1 How to Create a Style Class

You can create a style class that defines the style properties you want an application developer to apply to an ADF Faces or ADF Data Visualization component using the component's `styleClass` or `inlineStyle` attribute.

To create a style class:

1. In the **Selector Tree** of the selectors editor, select **New Style Class** from the dropdown list, as shown in [Figure 10-2](#).

**Figure 10-2** *New Style Class Option in the Selector Tree*



The Create Style Class dialog opens.

2. Choose the appropriate option:
  - Enter a new name if you want to create a new style class that does not inherit style properties from an ADF skin that your ADF skin extends.
 

**Tip:**

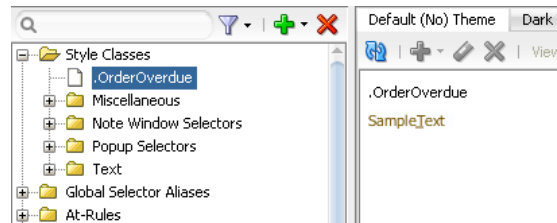
Enter a name for the style class that indicates the purpose it serves.
  - Enter the name of a style class that inherits style properties from an ADF skin that your ADF skin extends and for which you want to override style properties in your ADF skin.

3. Click **OK**.

## 10.2.2 What Happens When You Create a Style Class

The style class appears under the Style Classes node in the Selector Tree and a visual representation as it applies to a component appears in the Preview Pane, as shown in [Figure 10-3](#).

**Figure 10-3** Newly-Created Style Class



CSS syntax for the style class that you create appears in the source file of the ADF skin. The following example shows the entries that appear in the source file for the ADF skin in [Figure 10-3](#).

```
.OrderOverdue
{
}
```

## 10.3 Modifying a Style Class

The process to modify a style class is the same for the different categories of style class that appear in the selectors editor. You select the style class in the Selector Tree and use the menus in the Preview Pane to add or remove pseudo-classes to the style class or use the Properties window to set or override style properties for the style class.

### 10.3.1 How to Modify a Style Class

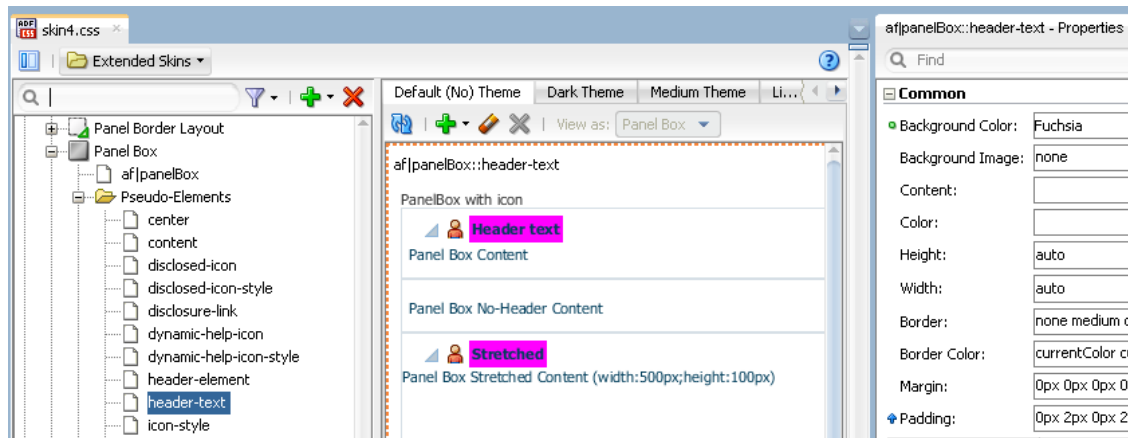
You select the style class under the Style Classes node in the Selector Tree and modify its properties using the Properties window.

To modify a style class:

1. In the Selector Tree, navigate to the style class that you want to modify.
2. In the Properties window, make changes to the properties that you want to configure for the style class.
3. Click the **Refresh** icon to update the Preview Pane after you make changes to the style class.

## 10.4 Configuring a Style Class for a Specific Instance of a Component

You can define a style class where you define style properties to apply to a specific instance of a component. Consider, for example, a `panelBox` component that application developers use to show or hide content on a page. One page can render multiple instances of a `panelBox` component. You decide to make fuchsia the default background color for the header text that `panelBox` components render, as shown in [Figure 10-4](#).

**Figure 10-4** Setting Background Color for a panelBox Component

However, you decide that you want to render one or more instances of the `panelBox` component without the disclosure link control that allows end users to show and hide the content in the component. Additionally, you decide that you want the header text of these instances of the `panelBox` component to render with the background color set to red. To achieve this, you define style properties for a style class in the ADF skin. You then specify the style class as the value for the `styleClass` attribute for each instance of the `panelBox` component that you want to render using these style properties. The following example shows the syntax that appears in the source file of the ADF skin to achieve the outcome just described.

```
.panelBoxInstanceClass af|panelBox::disclosure-link{display:none;}
.panelBoxInstanceClass af|panelBox::header-text {background-color: Red;}
```

---

**Note:** You cannot configure ADF skin properties in the style classes that you create for specific instances of a component. ADF skin properties allow you to customize the rendering of a component throughout the application, not specific instances of a component. For more information about ADF skin properties, see [Properties in the ADF Skinning Framework](#).

---

### 10.4.1 How to Configure a Style Class for a Specific Instance of a Component

You specify the style class as the value for the `styleClass` attribute for each instance of a component that you want to render using the style class.

To configure a style class for a specific instance of a component:

1. Create a style class, as described in [Creating a Style Class](#).
2. In JDeveloper, set the component's `styleClass` attribute to the name of the style class you created in step 1.

For more information about setting the component's `styleClass` attribute, see *Developing Web User Interfaces with Oracle ADF Faces*.

### 10.4.2 What Happens When You Configure a Style Class for a Specific Instance of a Component

At runtime, instances of the component for which you do not specify instance-specific style properties using a style class render using the style properties defined in the component-specific selectors and global selector aliases. In [Figure 10-5](#), this is the

panelBox component labeled **First Panel Box**. Instances of the component for which you specify a style class as a value for the `styleClass` attribute render using the style properties defined in this style class. In [Figure 10-5](#), this is the panelBox component labeled **Second Panel Box**.

**Figure 10-5** *Component Rendering with Style Properties Defined in Style Class*

---

▶ First Panel Box

---

Second Panel Box



---

## Working with At-Rules

This chapter describes how to work with at-rules. Information on how to create, modify, and apply an at-rule is provided in addition to describing the various types of at-rule that the ADF skinning framework supports.

This chapter includes the following sections:

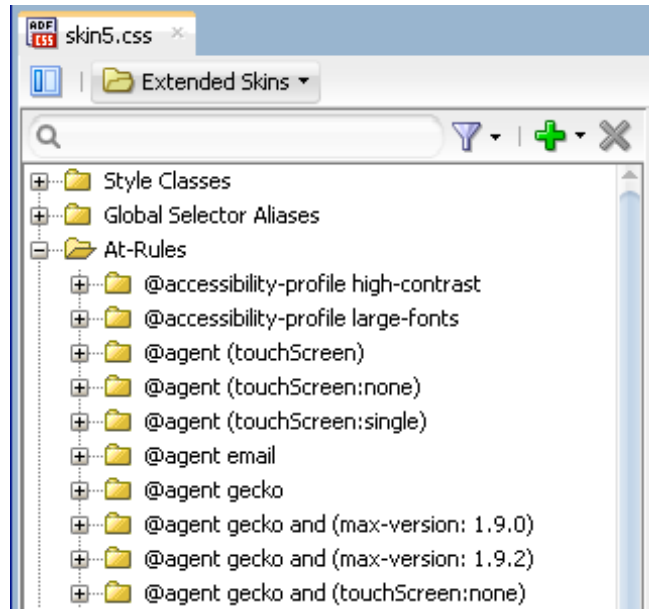
- [About At-Rules in the ADF Skinning Framework](#)
- [Working with Server-Side At-Rules](#)
- [Working with Client-Side At-Rules](#)
- [Creating At-Rules in an ADF Skin](#)

### 11.1 About At-Rules in the ADF Skinning Framework

CSS at-rules (at-rules) are a way to define style properties for when an application's page renders in a particular environment such as, for example, a browser, platform, locale or device. The **ADF skinning framework** supports a number of at-rules that allow you to define properties for selectors that you want to apply to a particular environment. For example, you may need to add some padding in Internet Explorer that you do not need on any other browser or perhaps you want to increase the size of icons if a page renders on a touch device. To style a selector for these scenarios, put the style properties inside an at-rule.

The ADF skinning framework divides the at-rules that it supports into two categories. It categorizes any at-rules that it passes directly to the user agent to interpret as a **client-side at-rule** and any at-rules that the ADF skinning framework itself interprets as a **server-side at-rule**. For more information about these categories, see [Working with Server-Side At-Rules](#) and [Working with Client-Side At-Rules](#).

You can use the selectors editor in JDeveloper to create at-rules in your **ADF skin**, as described in [Creating At-Rules in an ADF Skin](#). At-rules that your ADF skin inherits or at-rules that you define in your ADF skin appear in the **Selector Tree** under the At-Rules node, as illustrated in [Figure 11-5](#).

**Figure 11-1 At-Rules in the Selector Tree**

Apart from the at-rules described in this chapter, you can also use the `@import` at-rule to import another ADF skin into your ADF skin. For more information, see [Importing One or More ADF Skins Into the Current ADF Skin](#).

## 11.2 Working with Server-Side At-Rules

Table 11-1 lists a number of the server-side at-rules that the ADF skinning framework supports. The ADF skinning framework interprets these rules and determines the style properties to render, as described in [What Happens at Runtime: How the ADF Skinning Framework Applies At-Rules](#).

**Table 11-1 Server-Side At-Rules Supported by the ADF Skinning Framework**

Name	Description
@accessibility-profile	<p>Defines styles for high-contrast and large-fonts accessibility profile settings when enabled in the <code>trinidad-config.xml</code> file.</p> <p>For more information about the <code>@accessibility-profile</code> rule, see <a href="#">Configuring an ADF Skin for Accessibility</a>.</p>
@locale	<p>Specify a locale to define styles only for a particular language and country. You can specify either only the language or both the language and the country.</p> <p>Note that the ADF skinning framework does not support the <code>:lang</code> pseudo class.</p>
@mode	<p>Defines styles for when a page renders in a particular mode. This at-rule supports the following values:</p> <ul style="list-style-type: none"> <li>quirks</li> <li>standards</li> </ul>



**Table 11-1 (Cont.) Server-Side At-Rules Supported by the ADF Skinning Framework**

Name	Description
@platform	Define platform styles. Supported values are: <ul style="list-style-type: none"> <li>• android</li> <li>• blackberry</li> <li>• genericpda</li> <li>• iphone</li> <li>• linux</li> <li>• macos</li> <li>• nokia_s60</li> <li>• ppc (Pocket PC)</li> <li>• solaris</li> <li>• unix</li> <li>• windows</li> </ul>

Apart from the rules listed in [Table 11-1](#), one of the most frequently used server-side at-rules is @agent. The @agent at-rule enables you to define styles to apply to one or more user agents. The following list specifies the supported values to set an agent-specific style using the @agent at-rule.

- blackberry
- email
- gecko
- genericDesktop
- genericpda
- googlebot
- ie
- konqueror
- mozilla
- msnbot
- nokia\_s60
- opera
- oracle\_ses
- unknown
- webkit (maps to Safari and Google Chrome)

Using the @agent at-rule, you can:

- Specify styles for any version of Internet Explorer:
  - @agent ie

- Optionally, specify a specific version of the agent using the `and` keyword. For example, to specify version 9 of Internet Explorer:

```
@agent ie and (version: 9)
```

- Use comma-separated rules to specify styles for a number of agents. For example, use the following rule to specify styles for Versions 15 and 17 of Mozilla Firefox and for Internet Explorer 9.x:

```
@agent mozilla and (version: 15.*), mozilla and (version: 17.*), ie and (version: 9.*)
```

- Note that the following two syntax examples specify the same rule:

```
@agent ie and (version: 9.*)
```

```
@agent ie and (version: 9)
```

To specify a rule for styles to apply only to Internet Explorer 9.0.x, write the following:

```
@agent ie and (version: 9.0.*)
```

- Use the `max-version` and `min-version` keywords to specify a range of versions. For example, you can rewrite the following rule:

```
@agent ie and (version: 9), ie and (version: 10)
```

as:

```
@agent ie and (min-version: 9) and (max-version: 10)
```

to apply styles that you define to all versions of Internet Explorer 9 and 10.

You can also use the `@agent` rule to determine styles to apply to agents that are touch devices. The following examples show the syntax that you write in an ADF skin file to configure this capability.

```
@agent (touchScreen) {  
    /* Touchscreen specific styles for all touch devices: both single and multiple  
    touch. */  
}
```

```
@agent (touchScreen:single) {  
    /* Styles specific for a touch device with single touch. */  
}
```

```
@agent (touchScreen:multiple) {  
    /* Styles specific for a touch device with multiple touch. */  
}
```

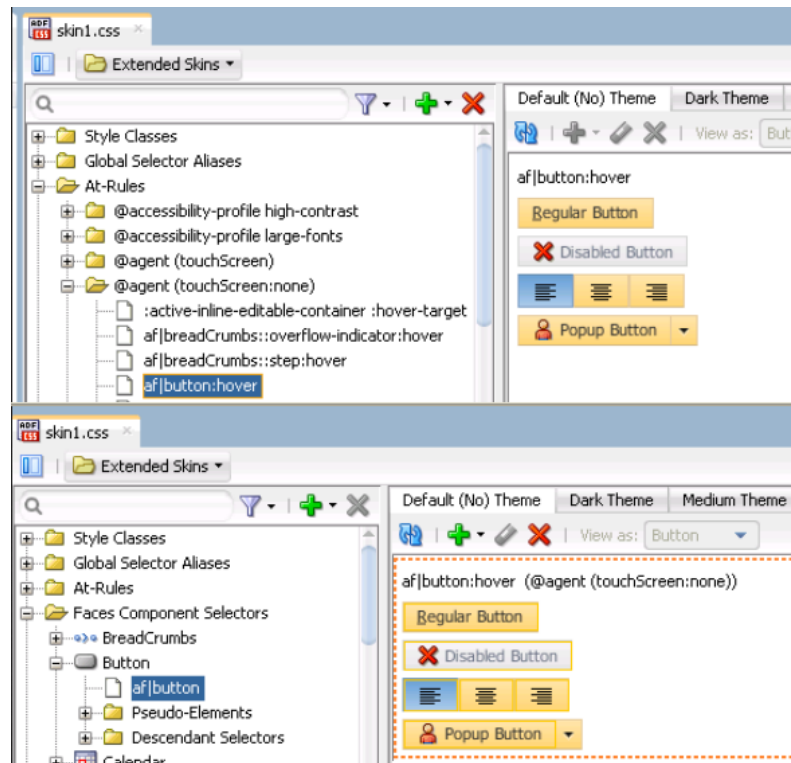
Use the `@agent (touchScreen:none)` at-rule to specify styles that you do not want to render on a touch device. For example, the Alta skin applies this at-rule to selectors configured to use the `:hover` pseudo class. This is because the `:hover` pseudo-class is not appropriate for use on a touch device. The `@agent (touchScreen:none)` at-rule wraps selectors that use the `:hover` pseudo-class, as in the following example:

```
@agent (touchScreen:none){  
    .AFBrandingBar af|link:hover,  
    .AFBrandingBar af|goLink:hover,
```

```
.AFBrandingBar af|commandLink:hover,
...
```

Figure 11-2 shows how the Selector Tree displays selectors to which the `@agent (touchScreen:none)` at-rule is applied.

**Figure 11-2** `@agent (touchScreen:none)` at-rule in the Selector Tree



For more information about creating applications to render in touch devices, see the "Creating Web Applications for Touch Devices Using ADF Faces" appendix in *Developing Web User Interfaces with Oracle ADF Faces*.

For information about how to create an at-rule in an ADF skin, see [Creating At-Rules in an ADF Skin](#).

## 11.3 Working with Client-Side At-Rules

The ADF skinning framework does not evaluate the following list of at-rules:

- `@charset`
- `@document`
- `@font-face`
- `@import`
- `@keyframes`
- `@media`
- `@page`
- `@supports`

Instead, it passes the at-rule, and the style properties within the at-rule, directly to the user agent. The user agent evaluates the at-rule and applies the style properties within the at-rule if the condition that the at-rule specifies is satisfied.

Because the style properties inside client-side at-rules get passed directly to the user agent, you cannot use ADF skin properties or global selector aliases inside client-side at-rules. The ADF skinning framework needs to evaluate these items to determine their runtime values. [Example 11-1](#) demonstrates a number of valid usages of client-side at-rules in an ADF skin. In [Example 11-1](#), the `@media` at-rule specifies the style properties to render for an `af:button` component when a screen has a maximum width of 1680px. The example also specifies style properties to apply for the `af:button` component when this condition is not met.

---



---

**Note:**

Do not insert ADF skin properties or global selector aliases inside a client-side at-rule. Unexpected behavior may result when you render a page using the ADF skin. The name of an ADF skin property is prefaced by `-tr-` and a global selector alias appends `:alias`. For more information, see [Properties in the ADF Skinning Framework](#) and [About Global Selector Aliases](#).

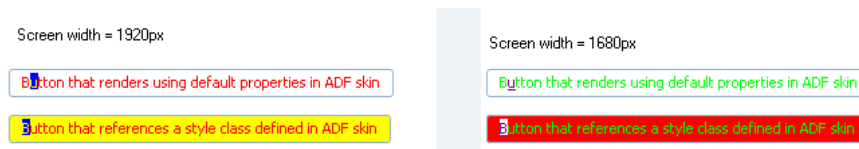
---



---

[Figure 11-3](#) shows instances of the `af:button` component that render using the appropriate style properties defined in [Example 11-1](#) based on the maximum width of the screen where the components display.

**Figure 11-3 Client-Side At-Rule Applied to a Button Component**



Client-side at-rules can nest within server-side at-rules. Server-side at-rules can nest within client-side at-rules. [Example 11-2](#) demonstrates instances where client-side and server-side at-rules nest within each other.

The `@page` and `@font-face` client-side at-rules are exceptions. These client-side at-rules cannot contain a server-side at-rule because they contain CSS properties whereas other client-side at-rules contain complete styles.

**Example 11-1 Client-Side At-Rules in an ADF Skin**

```
...
.myStyleClass {
    background-color: Yellow;
}

af|button {
    -tr-inhibit: background-image;
    color: Red;
}

af|button::access-key {
    background-color: Blue;
    color: Yellow;
}

@media screen and (max-width:1680px) {
```

```

.myStyleClass {
    background-color: Red;
}
af|button {
    color: Lime;
}
af|button::access-key {
    background-color: White;
    color: Purple;
}
}
...

```

### Example 11-2 Nested Client-Side and Server-Side At-Rules

```

@agent gecko {
    @page :first {
        margin: 2in ;
    }
}

@keyframes mymove {
    @agent gecko {
        0% { top: 0; left: 0; }
        30% { top: 50px; }
        68%, 72% { left: 50px; }
        100% { top: 100px; left: 100%; }
    }

    @agent ie {
        0% { top: 1; left: 1; }
        30% { top: 100px; }
        100% { top: 200px; left: 100%; }
    }
}

```

## 11.4 Creating At-Rules in an ADF Skin

You can create a new at-rule in your ADF skin or override an at-rule that your ADF skin inherits from the ADF skin that it extends. After you create an at-rule, you modify it to define the style properties that you want it to contain.

### 11.4.1 How to Create an At-Rule

You can create an at-rule to specify that style properties for one or more selectors render in a particular way when a condition specified by the at-rule is met.

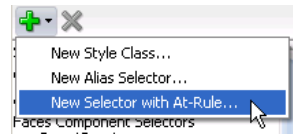
To create an at-rule:

1. In the Selector Tree of the selectors editor, select **New Selector with At-Rule** from the dropdown list, as illustrated in [Figure 11-4](#).

**Tip:**

If you know the name of the selector for which you want to configure an at-rule, right-click it in the Selector Tree and, from the context menu, choose **New Selector with At-Rule**. This populates the Selector field in the Create At-Rule Declaration dialog with the name of the selector that you right-clicked.

**Figure 11-4 New Selector with At-Rule Menu in the Selector Tree**



2. In the Create At-Rule Declaration dialog, select the at-rule that you want to configure from the **Rule** dropdown list.

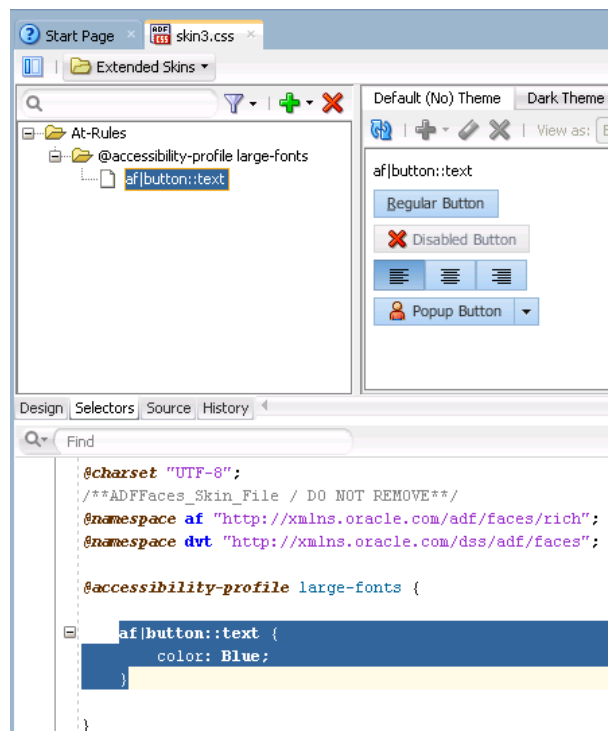
For more information about the at-rules that the ADF skinning framework supports, see [Working with Server-Side At-Rules](#) and [Working with Client-Side At-Rules](#).

3. Click **OK**.
4. In the Selector Tree, select the newly-created at-rule and, in the Properties window, configure the properties that you want this at-rule to apply.

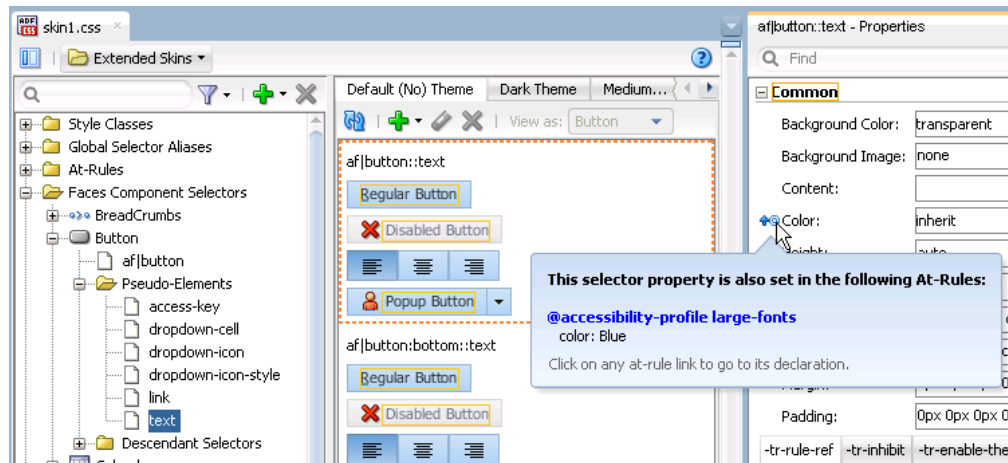
## 11.4.2 What Happens When You Create an At-Rule

The at-rule appears under the At-Rules node in the Selector Tree and a visual representation as it applies to a component appears in the Preview Pane, as shown in [Figure 11-5](#). CSS syntax for the at-rule that you create and any properties that you modified also appear in the source file of the ADF skin, as shown in [Figure 11-5](#).

**Figure 11-5 At-Rule in the Selector Tree and Source Editor**



In the Properties window for the selector property on which you set an at-rule, an icon appears to indicate that an at-rule is set, as illustrated in [Figure 11-6](#).

**Figure 11-6 Properties Window with an At-Rule set on a Property**

### 11.4.3 What Happens at Runtime: How the ADF Skinning Framework Applies At-Rules

At runtime, the ADF skinning framework picks the styles with at-rules based on the HTTP request information, such as agent and platform, and merges them with the styles without rules. Those style properties that match the rules get merged with those outside of any rules. The most specific rules that match a user's environment take precedence.

[Example 11-3](#) shows several selectors in the source file of an ADF skin that will be merged together to provide the final style.

#### **Example 11-3 Merging of Style Selectors in an ADF Skin**

```

/** For IE and Gecko on Windows, Linux and Solaris, make the color pink. */
@platform windows,linux,solaris{
  @agent ie, gecko
  {
    af|inputText::content {background-color:pink}
  }
}

/** Define default properties for the af|panelFormLayout selector. */
af|panelFormLayout {
  color: red;
  width: 10px;
  padding: 4px
}

/** Define at-rule for af|panelFormLayout on Internet Explorer (IE). We need
to increase the width, so we override the width. We still want the color
and padding; this gets merged in. We want to add height in IE. */
@agent ie{
  af|panelFormLayout {width: 25px; height: 10px}
}

/** For IE 9 and 10, we also need some margins.*/
@agent ie( version:9)and( version:10){
  af|panelFormLayout {margin: 5px;}
}

/** For Firefox 10 (Gecko 10) use a smaller margin.*/
@agent gecko( version:10){
  af|panelFormLayout {margin: 4px;}
}

```





---

# Applying the Finished ADF Skin to Your Web Application

This chapter describes how to complete tasks that you need to do once you finish your ADF skin. Information is provided on how to test your ADF skin, package the completed ADF skin in an ADF Library JAR, and configure a web application so that it uses the completed ADF skin.

This chapter includes the following sections:

- [About Applying a Finalized ADF Skin to an Application](#)
- [Testing Changes in Your ADF Skin](#)
- [Packaging an ADF Skin into an ADF Library JAR](#)
- [Applying an ADF Skin to Your Web Application](#)
- [Applying an ADF Skin to a Running Web Application](#)

## 12.1 About Applying a Finalized ADF Skin to an Application

After you create an **ADF skin** where you define style properties for one or more ADF skin selectors, you may want to test the changes that you make in the ADF skin. Once you complete testing the changes in your ADF skin and are satisfied with the final ADF skin, you can package the ADF skin and associated files (**resource bundle**, images, and configuration files) into an ADF Library JAR to distribute for inclusion to the application projects that use the final ADF skin. Once you have distributed the final ADF skin, you configure the application to apply the ADF skin to it.

## 12.2 Testing Changes in Your ADF Skin

Once you have created an ADF skin and defined style properties that you want for one or more selectors, you may want to test how these style properties render at runtime in a browser. To do this, you apply the ADF skin to your application and run a page that renders the **ADF Faces** component which exposed the selector.

Consider using tools, such as Firebug for the Mozilla Firefox browser (or similar for your particular browser), when you run your application. These tools provide useful information that can help you as you iteratively develop your ADF skin. For example, in addition to inspecting changes that you have already made, these tools can help you identify the ADF skin selectors that correspond to a particular DOM element.

You can also configure context initialization parameters in the `web.xml` file of your application that allow you to:

- View changes in an ADF skin without having to restart the application

Set the value of the following context initialization parameter to `true`:

```
org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION
```

- Display the full uncompressed CSS style class name at runtime

Set the value of the following context initialization parameter to true:

```
org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION
```

Note that not all changes that you make to an ADF skin in your **web application** appear immediately if you set the `CHECK_FILE_MODIFICATION` to true. You must restart the web application to view changes that you make to icon and ADF skin properties.

For more information about context initialization parameters, see the "ADF Faces Configuration" appendix in *Developing Web User Interfaces with Oracle ADF Faces*.

Figure 12-1 demonstrates how the name of a component selector (for the ADF Faces button component) is suppressed. In Figure 12-1, the style class (`fndGlobalSearchCategory`) that is defined in an ADF skin is applied to the button component using the component's `styleClass` attribute.

**Figure 12-1 Compressed CSS from an ADF Skin**

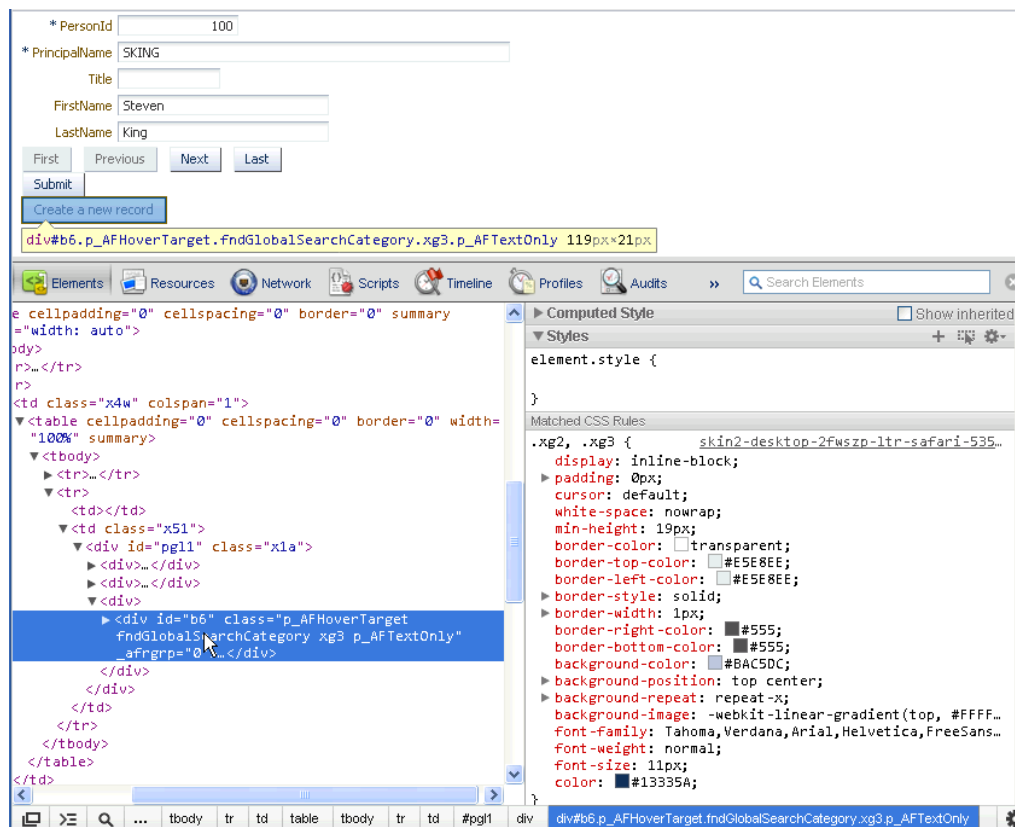


Figure 12-2 shows how the browser renders the full uncompressed name of the ADF Faces component when you set the `DISABLE_CONTENT_COMPRESSION` parameter to true. In Figure 12-2, the uncompressed style class `af|button` corresponds to the `af|button` selector documented in the *Tag Reference for Oracle ADF Faces Skin Selectors*.

The uncompressed style classes that correspond to the pseudo-elements that an ADF skin selector exposes can also be identified. For example, the `tab-end` pseudo-

element exposed by the `af|panelTabbed` selector (`af|panelTabbed::tab-end`) translates to the uncompressed `af_panelTabbed_tab-end` style class at runtime.

Similarly, changes that you make to the appearance of a component when it is in a specific state can also be identified or inspected using browser tools. For example, the following entry in the source file of an ADF skin allows you to define the style for the ADF Faces `panelTabbed` component when a user selects the right-hand side of the component:

```
af|panelTabbed::tab:selected af|panelTabbed::tab-end
```

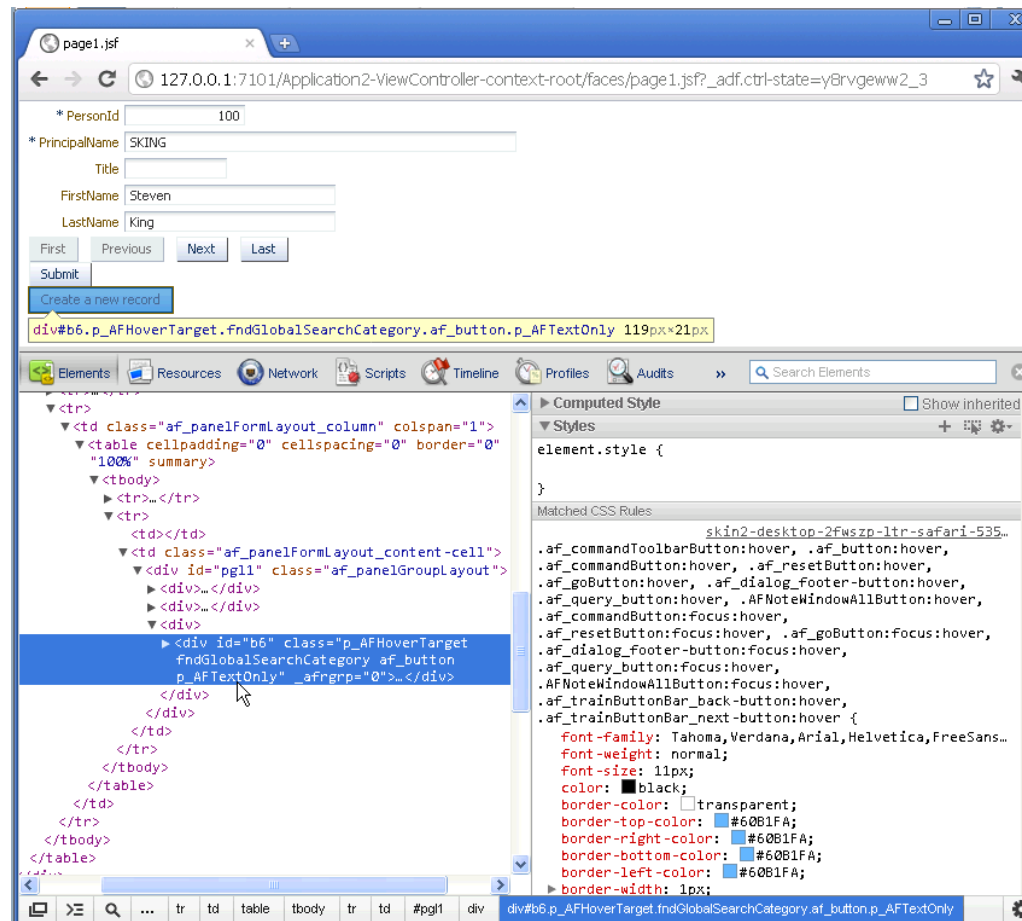
At runtime, the uncompressed style class name translates to the following:

```
.af_panelTabbed_tab.p_AFSelected .af_panelTabbed_tab-end
```

Note that `:selected` translates to `p_AFSelected` although sometimes the generated CSS does not have a `p_AFSelected` equivalent because some browsers have built-in support for that particular state, as is the case for other pseudo-classes like `:hover`.

It is recommended that you only customize the ADF skin selectors, pseudo-elements, and pseudo-classes documented in the *Tag Reference for Oracle ADF Faces Skin Selectors* and the *Tag Reference for Oracle ADF Data Visualization Tools Skin Selectors*. Customizing other ADF skin selectors may result in unexpected or inconsistent behavior for your application.

**Figure 12-2 Uncompressed CSS from an ADF Skin**



## 12.2.1 How to Set Parameters for Testing Your ADF Skin

You set the `CHECK_FILE_MODIFICATION` and `DISABLE_CONTENT_COMPRESSION` context initialization parameters to `true` in the `web.xml` file of your application.

To set parameters for testing your ADF skin:

1. In the Applications window, double-click the `web.xml` file.
2. In the source editor, add the following context initialization parameter entries and set to `true`:
  - `org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION`
  - `org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION`
3. Save and close the `web.xml` file.

## 12.2.2 What Happens When You Set Parameters for Testing Your ADF Skin

Entries appear in the `web.xml` file for your application, as illustrated in the following example.

```
<context-param>
  <param-name>org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION</param-name>
  <param-value>true</param-value>
</context-param>
<context-param>
  <param-name>org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION</param-name>
  <param-value>true</param-value>
</context-param>
```

Changes that you make to a selector for an ADF Faces component (other than changes to icon and skin properties) render immediately when you refresh a web application's page that renders the ADF Faces component. Using Firebug if your browser is Mozilla Firefox or Google Chrome's developer tools, you can see the uncompressed style class names that render at runtime and establish what ADF skin selector it corresponds to. Remember that setting `org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION` to `true` incurs a performance cost for your web application so set it to `false` when you finish testing your changes.

## 12.3 Packaging an ADF Skin into an ADF Library JAR

You can deploy an ADF skin and associated files (for example, image files, configuration files, and resource bundles) in an ADF Library JAR. This enables you to package files required to apply an ADF skin to an application. The benefits of packaging ADF skins into an ADF Library JAR as compared to bundling them into the application are the following:

- An ADF skin can be deployed and developed separately from the application. This also helps to reduce the number of files to be checked in case some changes must be applied to the ADF skin.
- The source files for an ADF skin and images can be separated into their own ADF Library JARs. Therefore, you can partition the image base into separate ADF Library JARs, so that not all files have to be deployed with all applications.

- An ADF skin in an ADF Library JAR can be applied to an application that is running without requiring a restart, as described in [Applying an ADF Skin to a Running Web Application](#).

### 12.3.1 How to Package an ADF Skin into an ADF Library JAR

Create an ADF Library JAR file deployment profile to package the ADF skin into an ADF Library JAR.

To create an ADF Library JAR file deployment profile:

1. In the Applications window, right-click the project that contains the ADF skins and choose **Deploy > New Deployment Profile**.
2. In the Create Deployment Profile dialog, choose **ADF Library JAR File** in the Profile Type dropdown list.
3. Enter a name for the deployment profile in the Deployment Profile Name input field and click **OK**.
4. Review the options in the Edit ADF Library JAR Deployment Profile Properties dialog that appears. For more information at any time, click **Help**.
5. Click **OK**.

To package an ADF skin into an ADF Library JAR:

1. In the Applications window, right-click the project that contains the ADF skin and choose **Deploy > deployment**, where **deployment** is the name of the ADF Library JAR file deployment profile.
2. In the Deploy dialog Deployment Action page, click **Deploy to ADF Library JAR file**, click **Next** and then click **Finish**.

### 12.3.2 What Happens When You Package an ADF Skin into an ADF Library JAR

An ADF Library JAR file is written to the directory specified by the deployment profile. This ADF Library JAR contains the source file for the ADF skin, the `trinidad-skins.xml` file, image files, and any resource bundles that you created to define resource strings or to override the default strings defined for ADF Faces components. The ADF Library JAR file also contains other files from the ADF skin's project not related to skinning.

[Example 12-1](#) shows the directory structure for a project that contains the following items for an ADF skin:

- The `trinidad-skins.xml` file
- An image file (`sort_des_ena.png`) copied into the JDeveloper project
- The source file for an ADF skin (`skin1.css`)
- An `.sva` file (`oracle.adf.common.services.ResourceService.sva`) that is used to inspect the content of the ADF Library JAR when you import it into a project, as described in [Adding ADF Skins from an ADF Library JAR](#).
- A resource bundle (`skinBundle.properties`) that contains string values to override strings from the default resource bundle

For information about how to specify resource bundles that contain string values you define, see [How to Specify an Additional Resource Bundle for an ADF Skin](#).

The directory paths for images in the ADF skin that appear in the ADF Library JAR are modified to include the directory path from the JDeveloper project. [Example 12-2](#) demonstrates an example of the changes that occur:

**Example 12-1 Directory Structure for an ADF Library JAR Containing an ADF Skin**

```
ADFLibraryJARRootDirectory
+---META-INF
|   |   MANIFEST.MF
|   |   oracle.adf.common.services.ResourceService.sva
|   |   trinidad-skins.xml
|
|   +---adf
|   |   \---skins
|   |       \---skin1
|   |           \---images
|   |               \---af_column
|   |                   colSort_des_ena.png
|   |
|   |   \---skins
|   |       \---skin1
|   |           skin1.css
|   |
|   \---resources
|       skinBundle.properties
```

**Example 12-2 Modified Directory Path for Images in a Deployed ADF Skin**

```
// Reference to an image in an ADF skin prior to deployment to an ADF Library JAR
af|column::sorted-descending-icon-style
{
  background-image: url("images/af_column/colSort_des_ena.png");
}

// Reference to an image in an ADF skin after deployment to an ADF Library JAR
af|column::sorted-descending-icon-style
{
  background-image: url("/adf/skins/skin1/images/af_column/colSort_des_ena.png");
}
```

## 12.4 Applying an ADF Skin to Your Web Application

You configure an application to use an ADF skin by specifying values in the application's `trinidad-config.xml` file. You specify a value for the `<skin-family>` element that identifies the ADF skin family the application uses at runtime. If you created more than one ADF skin in the ADF skin family, you can version these ADF skins. If you versioned multiple ADF skins in the same ADF skin family, use the `<skin-version>` element in the `trinidad-config.xml` file to identify the specific version that you want the application to use.

If you do not identify a specific ADF skin from an ADF skin family by entering a value for the `<skin-version>` element in the `trinidad-config.xml` file or using the `<default>true</default>` element in the `trinidad-skins.xml` file, the application uses the last skin defined in the `trinidad-skins.xml` file. For more information about versioning ADF skins and how this can determine the ADF skin that your application chooses, see [Versioning ADF Skins](#).

Note that you can configure an application page for your end users to dynamically select the ADF skin that they want the application to use. For more information, see *Developing Web User Interfaces with Oracle ADF Faces*.

### 12.4.1 How to Apply an ADF Skin to an Application

You apply an ADF skin to an application by modifying the application's `trinidad-config.xml` file. You do this by editing the application's `trinidad-config.xml` file to specify the ADF skin family to use. Alternatively, you can select the ADF skin family from a list in the ADF View options of JDeveloper's Project Properties dialog.

To apply an ADF skin to an application:

1. In the Applications window, double-click the `trinidad-config.xml` file. By default, this file is in the **Web Content/WEB-INF** node.
2. In the source editor, write entries to specify the value of the `<skin-family>` element and, optionally, the `<skin-version>` element as shown in [Example 12-3](#).

### 12.4.2 What Happens When You Apply an ADF Skin to an Application

The values that you specify for the `<skin-family>` element and, optionally, the `<skin-version>` element in the `trinidad-config.xml` file determine the ADF skin that the web application uses at runtime, as shown in [Example 12-3](#).

#### Example 12-3 `trinidad-config.xml` File

```
<?xml version="1.0" encoding="windows-1252"?>
<trinidad-config xmlns="http://myfaces.apache.org/trinidad/config">
  <skin-family>skyros</skin-family>
  <skin-version>v1</skin-version>
</trinidad-config>
```

## 12.5 Applying an ADF Skin to a Running Web Application

Using **Java Management Extensions (JMX)**, you can apply an ADF skin that is packaged in an ADF Library JAR to a web application without having to restart the application. To do this, you must configure the web application's source files, as described in [How to Configure your Web Application to Accept an Updated ADF Skin](#). You then use JDeveloper to connect to the MBean server and deploy the ADF Library JAR containing the ADF skin(s). For more information, see [How to Deploy an ADF Library JAR to an MBean Server](#). This makes all ADF skins contained in the ADF Library JAR available to the web application.

### 12.5.1 How to Configure your Web Application to Accept an Updated ADF Skin

You make the following changes to the web application's ViewController project in JDeveloper so that the application can apply a new ADF skin deployed by a MBean server without requiring a restart of the web application:

- Select the **Enable Runtime Skin Updates** checkbox in the ADF View page of the application's ViewController project
- Add ADF Faces JMX Runtime 11 to the application's classpath
- (Optional) Add a context initialization parameter to the application's `web.xml` file  
The context initialization parameter enables you to specify a user friendly name to identify the web application rather than use the application's context root.

---

---

**Note:**

The web application must be deployed in an exploded format, as is the case when you run the application in the Integrated WebLogic Server.

---

---

To configure your web application to accept an updated ADF skin:

1. In JDeveloper's Applications window, select the ViewController project.
2. From the main menu, choose **Application > Project Properties**.
3. In the Project Properties dialog, select the ADF View page and then select the **Enable Runtime Skin Updates** checkbox.
4. Select the Libraries and Classpath page and verify that ADF Faces JMX Runtime 11 appears in the **Classpath Entries** list. If it is not, click **Add Library**.
5. In the **Add Library** dialog, select **ADF Faces JMX Runtime 11** and click **OK**.
6. Click **OK**.
7. Optionally, in the Applications window, double-click the **web.xml** file located in the **WEB-INF** directory and add a context initialization parameter where you can specify an easy to remember name for your application. Otherwise, the default behavior is to use the context root of the application.

In the overview editor, click the **Application** navigation tab and then click the **Add** icon next to the **Context Initialization Parameters** table to add an entry for the `oracle.adf.view.rich.SKINNING_MBEAN_NAME` parameter and set its value to a name that you will use to identify the web application to the MBean server.

## 12.5.2 How to Deploy an ADF Library JAR to an MBean Server

You deploy the ADF Library JAR that packages the ADF skin(s) to the MBean server. For information about how to create an ADF Library JAR file deployment profile to package the ADF skin(s), see [How to Package an ADF Skin into an ADF Library JAR](#).

To deploy an ADF Library JAR to an MBean Server:

1. In the Applications window, right-click the project that contains the ADF skin(s) and choose **Deploy > deployment**, where **deployment** is the name of the ADF Library JAR file deployment profile.
2. In the Deployment Action page, select **Deploy to ADF Skin Managed Bean** and then click **Next**.
3. In the Skin Connection page, choose the appropriate option:
  - Click **Add** to create a new connection to the MBean server and go to Step 4.
  - Choose an existing connection from the **Connection** dropdown list and go to Step 5.
4. In the Create JMX Connection dialog, complete the fields to connect to the MBean server:



- **Connection Name:** Enter a name for the connection. The connection name must be a valid Java identifier, and as the name and connection are global across your installation, choose an appropriate and unique name.
  - **Server Type:** The default of Weblogic Server is preselected for connections where you deploy to an ADF Skin Managed Bean.
  - **Username:** Enter the user name to be authorized for access to the MBean server.
  - **Password:** Enter the password to be associated with the specified user name. An asterisk (\*) appears for each character you type in this field.
  - **Protocol:** JDeveloper uses the t3 protocol to communicate with the MBean server.
  - **Hostname:** Enter a value to identify the machine running the MBean server. Use an IP address or a host name that can be resolved by TCP/IP, for example if the MBean server is on the local machine, use `localhost`, or `127.0.0.1`.
  - **Port:** Enter the listen port for the MBean server. The default is whatever the default port number of the Integrated Weblogic Server is (often 7101).
  - **URL Provider Path:** Enter the absolute JNDI name of the MBean server. It must start with `/jndi/` and be followed by one of:
    - `weblogic.management.mbeanservers.domainruntime`
    - `weblogic.management.mbeanservers.runtime`
    - `weblogic.management.mbeanservers.edit`
  - **Server Install Location:** Displays the server install location.
  - **Test Connection:** Click to test the connection.
  - **Status:** A `Success!` message indicates that JDeveloper has been able to connect to the MBean server. Any other message indicates that the connection has failed. Amongst the things you should check before trying Test Connection again are:
    - Whether the network is working correctly when the MBean server is not local.
    - The values entered in this dialog.
5. In the **Application Name** field, select the name of the web application that you want to deploy the ADF Library JAR containing the ADF skin(s) to. Click **Find Running Applications** to retrieve the list of available applications and to make sure that the application is running.

The name of the application's root context appears unless you specified the name of the application to be the value that you entered for the `oracle.adf.view.rich.SKINNING_MBEAN_NAME` parameter, as described in [How to Configure your Web Application to Accept an Updated ADF Skin](#).

6. Click **Next** and then click **Finish**.

### 12.5.3 What Happens When You Apply an ADF Skin to a Running Application

JDeveloper deploys the ADF Library JAR containing the ADF skin(s) to the web application. This ADF Library JAR contains the ADF skin and other associated files (for example, any images that the ADF skin requires). For more information about the contents of the ADF Library JAR, see [What Happens When You Package an ADF Skin into an ADF Library JAR](#). The ADF skins are installed in the root directory of the web application. The `trinidad-skins.xml` file of the web application is updated to reference the newly-added ADF skins.

To make the web application use the newly-available ADF skin, you need to update the value that the `trinidad-config.xml` file's `<skin-family>` element references. You can do this manually, as described in [How to Apply an ADF Skin to an Application](#), or you can specify an EL expression for the element that updates the value programmatically. For more information about this latter option, see the "Customizing the Appearance Using Styles and Skins" chapter in *Developing Web User Interfaces with Oracle ADF Faces*.

---

## Advanced Topics

This chapter provides information to help you if you make changes in the source file of an ADF skin or in the configuration files that control the usage of ADF skins. The chapter also lists and describes the ADF skins provided by Oracle ADF.

This chapter includes the following sections:

- [Referring to URLs in an ADF Skin's CSS File](#)
- [Configuration Files for an ADF Skin](#)
- [ADF Skins Provided by Oracle ADF](#)
- [Versioning ADF Skins](#)

### 13.1 Referring to URLs in an ADF Skin's CSS File

An ADF skin's CSS file typically uses a URL to refer to a resource that is external to the file. For example, an image that an application uses to render with an error message. You can refer to a URL from an ADF skin's CSS file in a number of different formats. The supported formats are:

- Absolute

You specify the complete URL to the resource. For example, a URL in the following format:

```
http://www.mycompany.com/WebApp/Skin/skin1/img/errorIcon.gif
```

- Relative

You can specify a relative URL if the URL does not start with / and no protocol is present. A relative URL is based on the location of the ADF skin's CSS file. For example, if the ADF skin's CSS file directory is `WebApp/Skin/skin1/` and the specified URL is `img/errorIcon.gif`, the final URL is `/WebApp/Skin/mySkin/img/errorIcon.gif`

- Context relative

This format of URL is resolved relative to the context root of your web application. You start a context relative root with `/`. For example, if the context relative root of a web application is:

```
/WebApp
```

and the specified URL is:

```
/img/errorIcon.gif
```

the resulting URL is:

```
/WebApp/img/errorIcon.gif
```

- Server relative

A server relative URL is resolved relative to the web server. This differs to the context relative URL in that it allows you reference a resource located in another application on the same web server. You specify the start of the URL using `//`. For example, write a URL in the following format:

```
//WebApp/Skin/mySkin/img/errorIcon.gif
```

The format of URL that you use may be important if you create a Java Archive (JAR) file to package and distribute your **ADF skin** and its associated files. For more information, see [Packaging an ADF Skin into an ADF Library JAR](#).

## 13.2 Configuration Files for an ADF Skin

The following list describes the configuration files associated with the project for an ADF skin. You modify values in these files while you develop your ADF skin or when you finish development and want to apply the finished ADF skin to an application.

- `trinidad-skins.xml`

This file registers the ADF skins that you create, as described in [Creating an ADF Skin File](#). The following example demonstrates how to register a number of ADF skins that extends from a sample of the ADF skins described in [Table 13-1](#).

```
<!-- Use the following values in the trinidad-skins.xml file if you want to
extend the alta-v1 skin. -->
<skin>
  <id>yourSkin.desktop</id>
  <family>yourSkinFamily</family>
  <extends>alta-v1.desktop</extends>
  ...
</skin>

<!-- Use the following values in the trinidad-skins.xml file if you want to
extend the skyros-v1 skin. -->
<skin>
  <id>yourSkin.desktop</id>
  <family>yourSkinFamily</family>
  <extends>skyros-v1.desktop</extends>
  ...
</skin>
```

For more information about this file, see the "Configuration in `trinidad-skins.xml`" section in *Developing Web User Interfaces with Oracle ADF Faces*.

- `trinidad-config.xml`

You configure the `<skin-family>` element in this configuration file to tell the application what ADF skin to use, as described in [Applying an ADF Skin to Your Web Application](#). The following example demonstrates how to configure your web application to use some of the ADF skins listed in [Table 13-1](#).

```
<!-- Use the following value in the trinidad-config.xml file if you want your
application to use the Alta skin. -->
<skin-family>alta</skin-family>
  <skin-version>v1</skin-version>

<!-- Use the following value in the trinidad-config.xml file if you want your
application to use the skyros skin. -->
```

```
<skin-family>skyros</skin-family>
<skin-version>v1</skin-version>
```

For more information about this file, see the "Configuration in trinidad-config.xml" section in *Developing Web User Interfaces with Oracle ADF Faces*.

- web.xml

You can configure context initialization parameters in this file to facilitate the development and testing of your ADF skin, as described in [Testing Changes in Your ADF Skin](#). You can also configure a context initialization parameter (`org.apache.myfaces.trinidad.skin.MAX_SKINS_CACHED`) to specify the maximum number of unique ADF skins for which you store information in memory about the generated CSS files. Using this context initialization parameter can help maintain the performance of your application if you use many different skins.

For more information about the web.xml file and context initialization parameters, see the "Configuration in web.xml" section in *Developing Web User Interfaces with Oracle ADF Faces*.

### 13.3 ADF Skins Provided by Oracle ADF

Oracle ADF provides a number of ADF skins from which you can extend when you create a new ADF skin. [Table 13-1](#) describes the differences between each of these ADF skins.

The Base Skin page of the Create ADF Skin dialog that appears when you create an ADF skin, as described in [Creating an ADF Skin File](#), recommends the appropriate ADF skin to extend from.

You can apply any of the ADF skins listed in [Table 13-1](#) to your web application. For more information, see [Configuration Files for an ADF Skin](#). For a diagram that illustrates the inheritance relationship between the ADF skins, see [Inheritance Relationship of the ADF Skins Provided by Oracle ADF](#).

**Table 13-1 ADF Skins Provided by Oracle ADF**

ADF Skin	Description
simple	Contains only minimal formatting.
skyros-v1	Extends the simple skin. It provides a colorful look and feel to applications that use it.
alta-v1	The alta-v1 skin is the skin that permits web applications to take advantage of the Oracle Alta UI system. Use this skin or extend from it if you want your web application to take advantage of the enhancements introduced in the Oracle Alta UI system. The alta-v1 skin is the default skin for web applications that you create using this release. For more information about the Oracle Alta UI system, see <a href="http://www.oracle.com/webfolder/ux/middleware/alta/index.html">http://www.oracle.com/webfolder/ux/middleware/alta/index.html</a> .

### 13.4 Versioning ADF Skins

You can specify version numbers for your ADF skins in the trinidad-skins.xml file using the <version> element. Use this optional capability if you want to distinguish between ADF skins that have the same value for the <family> element in

the `trinidad-skins.xml` file. Note that when you configure an application to use a particular ADF skin, you do so by specifying values in the `trinidad-config.xml` file, as described in section [Applying an ADF Skin to Your Web Application](#).

### 13.4.1 How to Version an ADF Skin

You specify a version for your ADF skin by entering a value for the `<version>` element in the `trinidad-skins.xml` file.

To version an ADF skin:

1. In the Applications window, double-click the `trinidad-skins.xml` file. By default, this is in the **Web Content/WEB-INF** node.
2. In the Structure window, right-click the **skin** node for the ADF skin that you want to version and choose **Insert inside skin > version**.
3. In the Insert version dialog, select **true** from the default list if you want your application to use this version of the ADF skin when no value is specified in the `<skin-version>` element of the `trinidad-config.xml` file, as described in [Applying an ADF Skin to Your Web Application](#).
4. Enter a value in the name field. For example, enter `v1` if this is the first version of the ADF skin.
5. Click **OK**.

### 13.4.2 What Happens When You Version ADF Skins

[Example 13-1](#) shows an example `trinidad-skins.xml` that references three ADF skins (`skin1.desktop`, `skin2.desktop`, and `skin3.desktop`). Each of these ADF skins have the same value for the `<family>` element (`test`). The values for the child elements of the `<version>` elements distinguish between each of these ADF skins. At runtime, an application that specifies `test` as the value for the `<skin-family>` element in the application's `trinidad-config.xml` file uses `skin1.desktop` because this ADF skin is configured as the default skin in the `trinidad-skins.xml` file (`<default>true</default>`). You can override this behavior by specifying a value for the `<skin-version>` element in the `trinidad-config.xml` file, as described in [Applying an ADF Skin to Your Web Application](#). For example, if you specify `v2` as a value for the `<skin-version>` element in the `trinidad-config.xml` file, the application uses `skin2.desktop` instead of `skin1.desktop` that is defined as the default in the `trinidad-skins.xml` file.

If you do not specify the skin version to pick (using the `<skin-version>` element in the `trinidad-config.xml` file), then the application uses the skin that is defined as the default using the `<default>true</default>` element in the `trinidad-skins.xml` file. If you do not specify a default skin, the application uses the last ADF skin defined in the `trinidad-skins.xml` file. In [Example 13-1](#), the last ADF skin to be defined is `skin3.desktop`.

#### **Example 13-1** *trinidad-skins.xml File with Versioned ADF Skin Files*

```
<?xml version="1.0" encoding="windows-1252"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  <skin>
    <id>skin1.desktop</id>
    <family>test</family>
    <extends>skyros-v1.desktop</extends>
    <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
```

```
<style-sheet-name>skins/skin1/skin1.css</style-sheet-name>
<version>
  <default>true</default>
  <name>v1</name>
</version>
</skin>
<skin>
  <id>skin2.desktop</id>
  <family>test</family>
  <extends>skin1.desktop</extends>
  <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
  <style-sheet-name>skins/skin2/skin2.css</style-sheet-name>
  <version>
    <name>v2</name>
  </version>
</skin>
<skin>
  <id>skin3.desktop</id>
  <family>test</family>
  <extends>skin2.desktop</extends>
  <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
  <style-sheet-name>skins/skin3/skin3.css</style-sheet-name>
  <version>
    <name>v3</name>
  </version>
</skin>
</skins>
```

