

Oracle® JavaScript Extension Toolkit (Oracle JET)

Developing Applications with Oracle JET



9.2.0
F34549-02
October 2020



Copyright © 2014, 2020, Oracle and/or its affiliates.

Primary Author: Ralph Gordon

Contributing Authors: Walter Egan

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, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. 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 Inside 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, Epyc, and the AMD 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

Audience	xvi
Documentation Accessibility	xvi
Related Resources	xvi
Conventions	xvii

What's New in This Guide for Release 9.2.0

1 Getting Started with Oracle JavaScript Extension Toolkit (JET)

The Oracle JET Architecture	1-1
What's Included in Oracle JET	1-5
Third Party Libraries Used by Oracle JET	1-5
Typical Workflow for Getting Started with Oracle JET Application Development	1-6
Choose a Development Environment for Oracle JET	1-7
Choose a Development Environment	1-7
Install Oracle JET Tooling	1-9
Install Node.js	1-9
Install the Oracle JET Command-Line Interface	1-9
Configure Oracle JET Applications for TypeScript Development	1-10
Work with the Oracle JET Starter Templates	1-12
About the Starter Templates	1-12
About Modifying Starter Templates	1-15
Modify Starter Template Content	1-16
Work with the Oracle JET Base Distribution	1-19
About the Oracle JET Base Distribution	1-20
Add Oracle JET to an Existing JavaScript Application	1-20
Optimize Application Startup Using Oracle CDN and Oracle JET Libraries	1-21

2 Understanding the Web Application Workflow

Scaffold a Web Application	2-1
----------------------------	-----

About ojet create Command Options for Web Applications	2-3
About Scaffolding a Web Application	2-4
About the Web Application File Structure	2-5
Modify the Web Application's File Structure	2-6
Add Hybrid Mobile Features to Web Applications	2-7
Build a Web Application	2-9
About ojet build Command Options for Web Applications	2-9
Serve a Web Application	2-10
About ojet serve Command Options for Web Applications	2-11
Serve a Web Application to a HTTPS Server Using a Self-signed Certificate	2-13
Serve a Web Application Using Path-based Routing	2-16
Customize the Web Application Tooling Workflow	2-19
About the Script Hook Points for Web Applications	2-20
About the Process Flow of Script Hook Points	2-22
Change the Hooks Subfolder Location	2-24
Create a Hook Script for Web Applications	2-25
Pass Arguments to a Hook Script for Web Applications	2-27

3 Understanding the Hybrid Mobile Application Workflow

Install the Mobile Tooling	3-1
Install Apache Cordova	3-2
Install Android Development Tools	3-2
Install an Emulator Accelerator	3-3
Create an Android Virtual Device	3-3
Set Up Your Android Device to Install an App from Your Development Machine	3-4
Install Gradle and Configure Gradle Proxy Settings	3-4
Configure Environment Variables to Reference JDK and Android SDK Installations	3-5
Install iOS Development Tools	3-7
Install Windows Development Tools	3-7
Enable Developer Mode on Windows 10	3-7
Install Visual Studio	3-8
Configure Environment Variable to Reference Microsoft Build Tools	3-9
Install a Personal Information Exchange File in Your Computer's Certificate Store	3-9
Create a Hybrid Mobile Application Using the Oracle JET Command-Line Interface	3-11
Scaffold a Hybrid Mobile Application	3-11
About ojet create Command Options for Mobile Hybrid Applications	3-14
About the Hybrid Mobile Application File Structure	3-15
Modify the Hybrid Mobile Application's File Structure	3-16

Add Web Browser Capability to Hybrid Mobile Applications	3-17
Build a Hybrid Mobile Application	3-18
About ojet build Command Options for Hybrid Mobile Applications	3-19
Serve a Hybrid Mobile Application	3-21
About ojet serve Command Options for Hybrid Mobile Applications	3-22
Review Your Application Settings in the config.xml File	3-25
Change the Splash Screen and App Launcher Icon	3-26
Customize the Hybrid Mobile Application Tooling Workflow	3-28
About the Script Hook Points for Hybrid Mobile Applications	3-28
About the Process Flow of Script Hook Points	3-31
Change the Hooks Subfolder Location	3-33
Create a Hook Script for Hybrid Mobile Applications	3-34
Pass Arguments to a Hook Script for Hybrid Mobile Applications	3-36
Use Cordova Plugins to Access Mobile Device Services	3-37
About Apache Cordova and Cordova Plugins	3-38
Use a Plugin in Your App	3-39
Cordova Plugins Recommended by Oracle JET	3-40
Use a Different Web View in Your JET Hybrid Mobile App	3-40

4 Designing Responsive Applications

Typical Workflow for Designing Responsive Applications in Oracle JET	4-1
Oracle JET and Responsive Design	4-2
Media Queries	4-2
Oracle JET Flex, Grid, Form, and Responsive Helper Class Naming Convention	4-4
Oracle JET Flex Layouts	4-5
About Modifying the flex Property	4-6
About Wrapping Content with Flex Layouts	4-7
About Customizing Flex Layouts	4-8
Oracle JET Grids	4-8
About the Grid System	4-8
The Grid System and Printing	4-10
Grid Convenience Classes	4-12
Responsive Layout and Content Design Patterns	4-13
Responsive Form Layouts	4-19
Add Responsive Design to Your Application	4-19
Use Responsive JavaScript	4-20
The Responsive JavaScript Classes	4-20
Change a Custom Element's Attribute Based on Screen Size	4-21
Conditionally Load Content Based on Screen Size	4-22
Create Responsive Images	4-23

Use the Responsive Helper Classes	4-24
Create Responsive CSS Images	4-25
Change Default Font Size	4-25
Change Default Font Size Across the Application	4-26
Change Default Font Size Based on Device Type	4-26
Control the Size and Generation of the CSS	4-27

5 Using RequireJS for Modular Development

Typical Workflow for Using RequireJS	5-1
About Oracle JET and RequireJS	5-2
About Oracle JET Module Organization	5-2
About RequireJS in an Oracle JET Application	5-7
Use RequireJS in an Oracle JET Application	5-8
Add Third-Party Tools or Libraries to Your Oracle JET Application	5-9
Troubleshoot RequireJS in an Oracle JET Application	5-12
About JavaScript Partitions and RequireJS in an Oracle JET Application	5-12

6 Creating Single-Page Applications

Typical Workflow for Creating Single-Page Applications in Oracle JET	6-1
Design Single-Page Applications Using Oracle JET	6-1
Understand Oracle JET Support for Single-Page Applications	6-2
Create a Single-Page Application in Oracle JET	6-2
Use the oj-module Element	6-3
Work with oj-module's ViewModel Lifecycle	6-4

7 Understanding Oracle JET User Interface Basics

Typical Workflow for Working with the Oracle JET User Interface	7-1
About the Oracle JET User Interface	7-2
Identify Oracle JET UI Components, Patterns, and Utilities	7-2
About Common Functionality in Oracle JET Components	7-2
About Oracle JET Reserved Namespaces and Prefixes	7-6
About Binding and Control Flow	7-6
Use oj-bind-text to Bind Text Nodes	7-6
Bind HTML attributes	7-7
Use oj-bind-if to Process Conditionals	7-8
Use oj-bind-for-each to Process Loop Instructions	7-10
Bind Style Properties	7-11
Bind Event Listeners to JET and HTML Elements	7-12
Bind Classes	7-14

Add an Oracle JET Component to Your Page	7-17
Add Animation Effects	7-18
Manage the Visibility of Added Component	7-19

8

Working with Oracle JET User Interface Components

Work with Oracle JET UI Components - A Typical Workflow	8-1
Work with Collections	8-2
Choose a Table, Data Grid, or List View	8-2
About DataProvider Filter Operators	8-5
Work with Data Grids	8-6
Work with CubeDataSource	8-8
Work with List Views	8-13
Understand the Data Requirements for List Views	8-14
Work with List Views and Inline Templates	8-17
Work with Pagination	8-19
Work with Row Expanders	8-21
Work with Tables	8-24
Understand the Data Requirements for Table	8-25
Understand oj-table and Sorting	8-28
Work with Tables and Inline Templates	8-29
Work with Tree Views	8-31
Understand the Data Requirements for Tree Views	8-32
Specifying Initial Expansion in Tree Views	8-35
Work with Controls	8-35
Work with Buttons	8-36
Work with Button Sets	8-37
Work with Conveyor Belts	8-39
Work with File Picker	8-40
Work with Film Strips	8-43
Configure Film Strips	8-44
Work with Menus	8-46
Work with oj-menu	8-46
Work with Menu Buttons	8-48
Work with Context Menus	8-49
Work with Submenus	8-51
Work with Menu Select Many	8-53
Work with Progress Indicators	8-55
Work with Tags	8-56
Work with Toolbars	8-56
Work with Trains	8-57

Work with Forms	8-58
Work with Checkbox and Radio Sets	8-59
Work with Color Pickers	8-61
Work with oj-color-palette	8-61
Work with oj-color-spectrum	8-63
Work with Comboboxes	8-65
Understand oj-combobox-one Search	8-68
Work with Form Controls	8-69
Work with Form Layouts	8-71
Work with Input Components	8-73
Work with Labels	8-74
Work with Select	8-76
Work with Sliders	8-79
About the oj-slider Component	8-79
Create Sliders	8-80
Tips on Formatting for oj-slider	8-81
Work with Switches	8-82
Work with Validation and User Assistance	8-83
Work with Layout and Navigation	8-84
Work with Accordions	8-84
Work with Collapsibles	8-85
Work with Dialogs	8-86
Work with Masonry Layouts	8-87
Configure Masonry Layouts	8-88
About the oj-masonry-layout Layout Process	8-90
oj-masonry-layout Size Style Classes	8-90
Work with Nav Lists	8-90
Understand the Data Requirements for Nav Lists	8-91
Work with Nav Lists and Knockout Templates	8-94
Work with offCanvasUtils	8-96
Configure an Off-Canvas Partition	8-96
Work with Panels	8-96
Work with Popups	8-98
Work with oj-popup	8-98
Work with the Oracle JET Popup Framework	8-100
Work with Tab Bars	8-102
Add Content in Tab Bars	8-104
Add Interactivity in Tab Bars	8-106
Understand the Data Requirements for Tab Bars	8-108
Work with Visualizations	8-109
Choose a Data Visualization Component for Your Application	8-109

9 Working with Oracle JET Web Components

Typical Workflow for Working with Oracle JET Web Components	9-1
Design Custom Web Components	9-2
About Web Components	9-4
Web Component Files	9-8
Web Component Slotting	9-9
Web Component Template Slots	9-10
Web Component Events	9-11
Web Component Examples	9-12
Best Practices for Web Component Creation	9-12
Recommended Standard Patterns and Coding Practices	9-13
CSS and Theming Standards	9-17
Version Numbering Standards	9-18
Create Web Components	9-20
Create Standalone Web Components	9-21
Create JET Packs	9-29
Create Resource Components for JET Packs	9-34
Create Reference Components for Web Components	9-37
Test Web Components	9-40
Add Web Components to Your Page	9-40
Build Web Components	9-43
Package Web Components	9-44
Create a Project to Host a Shared Oracle Component Exchange	9-44
Publish Web Components to Oracle Component Exchange	9-47
Upload and Consume Web Components on a CDN	9-48

10 Using the Common Model and Collection API

Typical Workflow for Binding Data in Oracle JET	10-1
About Oracle JET Data Binding	10-1
About the Oracle JET Common Model and Collection Framework	10-2
About the Oracle JET Common Model and Collection API	10-2
About Oracle JET Data Binding and Knockout	10-3
Use the Oracle JET Common Model and Collection API	10-3
Integrate REST Services	10-5
About Oracle JET Support for Integrating REST Services	10-5
Pass Custom AJAX Options in Common Model CRUD API calls	10-6
Supply a customURL Callback Function	10-6

Replace sync or ajax Functions	10-7
Create a CRUD Application Using Oracle JET	10-8
Define the ViewModel	10-8
Read Records	10-15
Create Records	10-16
Update Records	10-19
Delete Records	10-23

11 Validating and Converting Input

Typical Workflow for Validating and Converting Input	11-1
About Oracle JET Validators and Converters	11-2
About Validators	11-2
About the Oracle JET Validators	11-3
About Oracle JET Component Validation Attributes	11-3
About Oracle JET Component Validation Methods	11-4
About Converters	11-4
About Oracle JET Component Converter Options	11-5
About Oracle JET Converters	11-7
Use Oracle JET Converters with Oracle JET Components	11-8
About Oracle JET Converters Lenient Parsing	11-11
Understand Time Zone Support in Oracle JET	11-12
Use Custom Converters in Oracle JET	11-15
Use Oracle JET Converters Without Oracle JET Components	11-18
About Oracle JET Validators	11-20
Use Oracle JET Validators with Oracle JET Components	11-20
Use Custom Validators in Oracle JET	11-24
About Asynchronous Validators	11-26

12 Working with User Assistance

Typical Workflow for Working with User Assistance	12-1
Understand Oracle JET's Messaging APIs on Editable Components	12-2
About Oracle JET Editable Component Messaging Attributes	12-3
About Oracle JET Component Messaging Methods	12-4
Understand How Validation and Messaging Works in Oracle JET Editable Components	12-4
Understand How an Oracle JET Editable Component Performs Normal Validation	12-5
About the Normal Validation Process When User Changes Value of an Editable Component	12-6

About the Normal Validation Process When Validate() is Called on Editable Component	12-6
Understand How an Oracle JET Editable Component Performs Deferred Validation	12-7
About the Deferred Validation Process When an Oracle JET Editable Component is Created	12-7
About the Deferred Validation Process When value Property is Changed Programmatically	12-7
Use Oracle JET Messaging	12-8
Notify an Oracle JET Editable Component of Business Validation Errors	12-8
Use the messages-custom Attribute	12-8
Use the showMessages() Method on Editable Components	12-10
Understand the oj-validation-group Component	12-10
Track the Validity of a Group of Editable Components Using oj-validation-group	12-11
Create Page Level Messaging	12-13
Create Messages with oj-message	12-15
Configure an Editable Component's oj-label Help Attribute	12-17
Configure an Editable Component's help.instruction Attribute	12-18
Control the Display of Hints, Help, and Messages	12-20

13 Developing Accessible Applications

Typical Workflow for Developing Accessible Oracle JET Applications	13-1
About Oracle JET and Accessibility	13-1
About the Accessibility Features of Oracle JET Components	13-2
Create Accessible Oracle JET Pages	13-3
Configure WAI-ARIA Landmarks	13-3
Configure High Contrast Mode	13-5
Understand Color and Background Image Limitations in High Contrast Mode	13-6
Add High Contrast Mode to Your Oracle JET Application	13-6
Add High Contrast Images or Icon Fonts	13-7
Test High Contrast Mode	13-8
Hide Screen Reader Content	13-8
Use ARIA Live Region	13-8

14 Internationalizing and Localizing Applications

Internationalize and Localize Oracle JET Applications - A Typical Workflow	14-1
About Internationalizing and Localizing Oracle JET Applications	14-1
Internationalize and Localize Oracle JET Applications	14-4
Use Oracle JET's Internationalization and Localization Support	14-4
Enable Bidirectional (BiDi) Support in Oracle JET	14-5

Set the Locale Dynamically	14-6
Work with Currency, Dates, Time, and Numbers	14-9
Work with Oracle JET Translation Bundles	14-10
About Oracle JET Translation Bundles	14-10
Add Translation Bundles to Oracle JET	14-13

15 Using CSS and Themes in Applications

About the Redwood Theme Included with Oracle JET	15-1
Typical Workflow for Working with Themes in Oracle JET Applications	15-2
CSS Files Included with the Redwood Theme	15-3
Create an Application with the Redwood Theme	15-4
Best Practices for Using CSS and Themes	15-6
DOCTYPE Requirement	15-9
ThemeUtils	15-9
Set the Text Direction	15-9
Work with Images	15-10
Image Considerations	15-10
Icon Font Considerations	15-10
Work with Custom Themes	15-11
Add Custom Theming Support	15-11
Disable JET Styling for Unused JET Components	15-13
Disable JET Styling of Base HTML Tags	15-14
Experimental: Work with CSS Variables	15-15
About CSS Variables and Custom Themes in Oracle JET	15-16
Experimental: Client-side Theming Using CSS Variables	15-17
Experimental: Build-time Theming with JET Tooling	15-20

16 Securing Applications

Typical Workflow for Securing Oracle JET Applications	16-1
About Securing Oracle JET Applications	16-1
Oracle JET Components and Security	16-2
Oracle JET Security and Developer Responsibilities	16-2
Oracle JET Security Features	16-2
Oracle JET Secure Response Headers	16-4
Content Security Policy Headers	16-5
Use OAuth in Your Oracle JET Application	16-9
Initialize OAuth	16-10
Verify OAuth Initialization	16-10
Obtain the OAuth Header	16-11

Use OAuth with Oracle JET Common Model	16-11
Embed OAuth in Your Application's ViewModel	16-11
Add OAuth as a Plugin in Your ViewModel	16-12
Integrate OAuth with Oracle Identity Management (iDM) Server	16-13
About Securing Hybrid Mobile Applications	16-14
Manage Authentication in JET Hybrid Mobile Apps	16-14
Manage App Configuration for JET Hybrid Mobile Apps	16-14
About Cross-Origin Resource Sharing (CORS)	16-15

17 Configuring Data Cache and Offline Support

About the Oracle Offline Persistence Toolkit	17-1
Install the Offline Persistence Toolkit	17-2

18 Optimizing Performance

Typical Workflow for Optimizing Performance of Oracle JET Applications	18-1
About Performance and Oracle JET Applications	18-1
Add Performance Optimization to an Oracle JET Application	18-2
About Configuring the Application for Oracle CDN Optimization	18-7
Configure Bundled Loading of Libraries and Modules	18-8
Configure Individual Loading of Libraries and Modules	18-9
Understand the Path Mapping Script File and Configuration Options	18-10

19 Auditing Application Files

20 Testing and Debugging

Typical Workflow for Testing and Debugging an Oracle JET Application	20-1
Test Oracle JET Applications	20-1
Test Applications	20-2
Test Hybrid Mobile Applications	20-2
Use BusyContext API in Automated Testing	20-3
Debug Oracle JET Applications	20-7
Debug Web Applications	20-8
Debug Hybrid Mobile Applications	20-9

21 Packaging and Deploying Applications

Typical Workflow for Packaging and Deploying Applications	21-1
Package and Deploy Web Applications	21-1
Package Web Applications	21-2
Deploy Web Applications	21-2
Package and Publish Hybrid Mobile Applications	21-2
About Packaging and Publishing Hybrid Mobile Applications	21-3
Package a Hybrid Mobile App on Android	21-3
Package a Hybrid Mobile App on iOS	21-4
Package a Hybrid Mobile App on Windows	21-6
Create the Build Configuration File to Package Your Application on Windows	21-7
Build Your Application for Windows	21-7
Remove and Restore Non-Source Files from Your JET Application	21-8

A Troubleshooting

B Oracle JET Application Migration for Release 9.2.0

Migrate Application Source to Release 9.2.0	B-1
Migrate a v9.x.0 Application to v9.2.0	B-6
Migrate to the Redwood Theme CSS	B-7

C Alta Theme in Oracle JET v9.0.0 and Later

Consider Using the Redwood Theme in Applications	C-1
CSS Files Included in Alta	C-2
Understand the Color Palette in Alta	C-3
Customize Alta Themes Using the Tooling	C-3
Work with Sass	C-5
SCSS Variables	C-6
SCSS File Organization and Naming Convention	C-7
Use Variables to Control CSS Content	C-7
Work with Icon Fonts in Alta	C-8
Work with Image Files in Alta	C-10
Use Tag Selectors or Style Classes with Alta	C-10
Use Normalize with Alta	C-12

D Oracle JET References

Oracle Libraries and Tools	D-1
----------------------------	-----

Preface

Developing Applications with Oracle JET describes how to build responsive web and hybrid mobile applications using Oracle JET.

Topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Resources](#)
- [Conventions](#)

Audience

Developing Applications with Oracle JET is intended for intermediate to advanced JavaScript developers who want to create pure client-side, responsive web or hybrid mobile applications based on JavaScript, HTML5, and CSS3.

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 Resources

For more information, see these Oracle resources:

- [Oracle JET Web Site](#)
- [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#)
- [Oracle® JavaScript Extension Toolkit \(JET\) Keyboard and Touch Reference](#)
- [Oracle® JavaScript Extension Toolkit \(JET\) Styling Reference](#)

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<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.

What's New in This Guide for Release 9.2.0

For Oracle JET release 9.2.0 this guide has been updated in several ways.

The following table lists the sections that are new or changed.

For changes made to Oracle JET for this release, see the product [Release Notes](#).

Chapter	Changes Made
Chapter 1, Getting Started with Oracle JavaScript Extension Toolkit (JET)	Configure Oracle JET Applications for TypeScript Development updated to describe the interface type naming convention change that determines how to specify JET component types within a TypeScript project.
Chapter 2, Understanding the Web Application Workflow	<p>Serve a Web Application to a HTTPS Server Using a Self-signed Certificate added to describe how you configure the <code>before_serve</code> hook point to serve your web application to a HTTPS server using a self-signed certificate. Content describing how you customized a web application's serve behavior by editing the <code>oraclejet-serve.js</code> file has been removed, as this latter approach is no longer recommended.</p> <p>Serve a Web Application Using Path-based Routing added to describe how you configure the <code>before_serve</code> hook point to serve a web application that uses path segments (path-based routing) in the URL that appears in the browser rather than the default parameter-based routing.</p>
Chapter 7, Understanding Oracle JET User Interface Basics	Manage the Visibility of Added Component added to highlight the <code>subtreeHidden</code> and <code>subtreeShown</code> methods that you should call when you hide or display certain types of component using the CSS <code>display</code> property set to <code>none</code> or <code>display:block</code> . Use these methods to ensure that the component works correctly in your application after you change its display status.
Chapters 19, Auditing Application Files	Chapter contents moved to new JET publication, <i>Using and Extending the Oracle JET Audit Framework</i> .

Chapter	Changes Made
Appendix B: Oracle JET Application Migration for Release 9.2.0	<p>Appendix updated with steps to migrate to the latest release.</p> <ul style="list-style-type: none">• Migrate Application Source to Release 9.2.0 updated to reflect migration steps from releases prior to release 9.1.0 that allow migrating applications to the latest release.• Migrate a v9.x.0 Application to v9.2.0 updated to describe the steps to upgrade to a minor release version within the same major release.• Migrate to the Redwood Theme CSS updated to describe how to migrate an Alta-themed application to the Redwood theme. As the topic describes, starting in release 9.0.0 it is possible to migrate and either remain on the Alta theme or to migrate to the Redwood theme.

1

Getting Started with Oracle JavaScript Extension Toolkit (JET)

Oracle JET is a collection of Oracle and open source JavaScript libraries engineered to make it as simple and efficient as possible to build client-side web and hybrid mobile applications based on JavaScript, HTML5, and CSS.

To begin using Oracle JET, you do not need more than the basics of JavaScript, HTML, and CSS. Many developers learn about these related technologies in the process of learning Oracle JET.

Oracle JET is designed to meet the following application needs:

- Add interactivity to an existing page.
- Create a new end-to-end client-side web application using JavaScript, HTML5, CSS, and best practices for responsive design.
- Create a hybrid mobile application that looks and feels like a native iOS, Android or Windows application.

Topics:

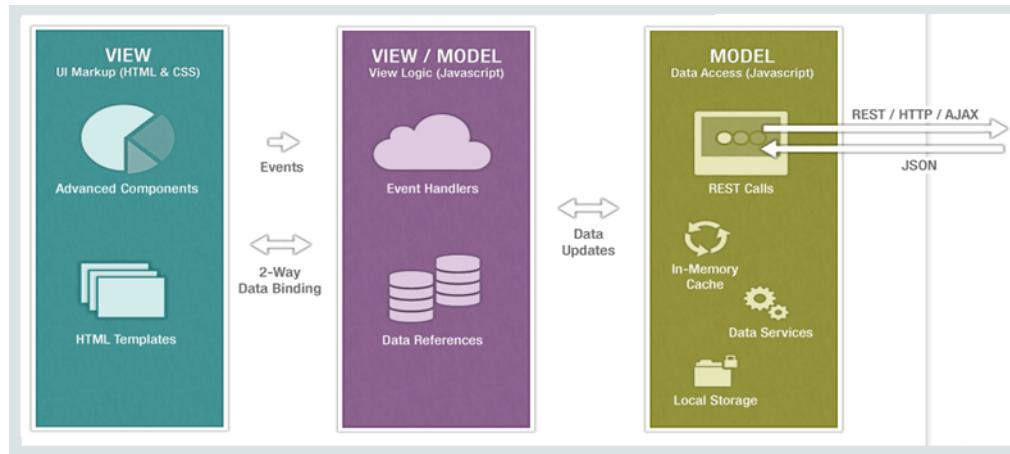
- [The Oracle JET Architecture](#)
- [What's Included in Oracle JET](#)
- [Third Party Libraries Used by Oracle JET](#)
- [Typical Workflow for Getting Started with Oracle JET Application Development](#)
- [Choose a Development Environment for Oracle JET](#)
- [Work with the Oracle JET Starter Templates](#)
- [Work with the Oracle JET Base Distribution](#)
- [Optimize Application Startup Using Oracle CDN and Oracle JET Libraries](#)

You can also view videos that provide an introduction to Oracle JET in the [Oracle JET Videos](#) collection.

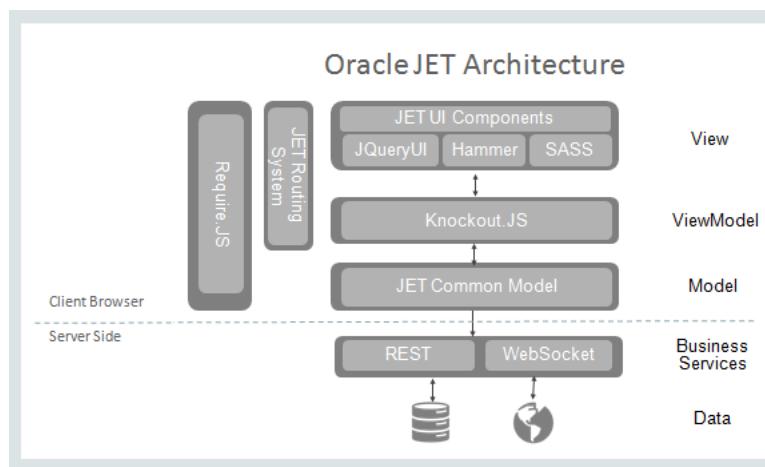
The Oracle JET Architecture

Oracle JET supports the Model-View-ViewModel (MVVM) architectural design pattern.

In MVVM, the Model represents the application data, and the View is the presentation of the data. The ViewModel exposes data from the Model to the view and maintains the application's state.



To support the MVVM design, Oracle JET is built upon a modular framework that includes a collection of third-party libraries and Oracle-provided files, scripts, and libraries.



The Oracle JET Common Model and Collection Application Programming Interface (API) implements the model layer. The API includes the following JavaScript objects:

- **Model:** Represents a single record data from a data service such as a RESTful web service
- **Collection:** Represents a set of data records and is a list of `Model` objects of the same type
- **Events:** Provides methods for event handling
- **KnockoutUtils:** Provides methods for mapping the attributes in an `Model` or `Collection` object to Knockout observables for use with component view models.

To implement the View layer, Oracle JET provides a collection of UI components implemented as HTML5 custom elements, ranging from basic buttons to advanced data visualization components such as charts and data grids.

Knockout.js implements the ViewModel and provides two-way data binding between the view and model layers.

Oracle JET Features

Oracle JET features include:

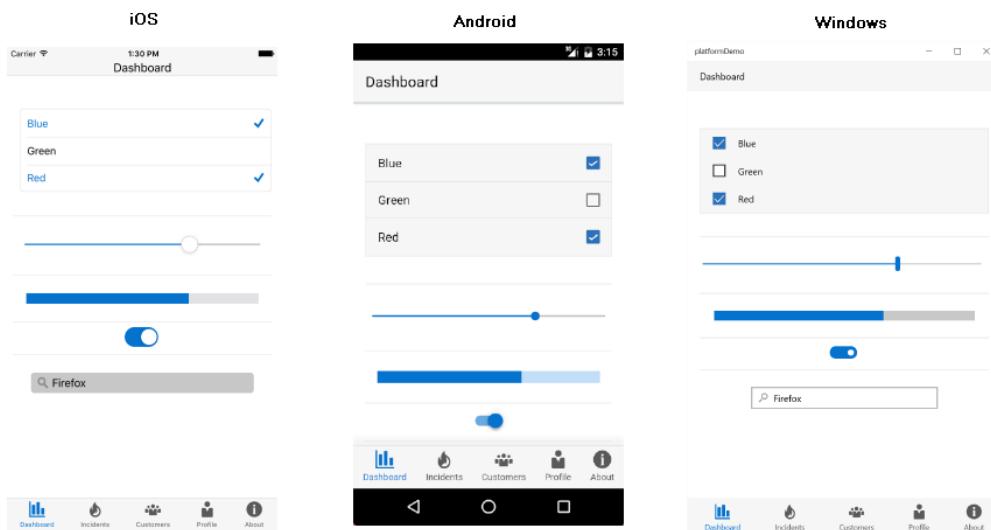
- Messaging and event services for both Model and View layers
- Validation framework that provides UI element and component validation and data converters
- Caching services at the Model layer for performance optimization of pagination and virtual scrolling
- Filtering and sorting services provided at the Model layer
- Connection to data sources through Web services, such as Representational State Transfer (REST) or WebSocket
- Management of URL and browser history using Oracle JET `CoreRouter` and `obj-module` components
- Integrated authorization through OAuth 2.0 for data models retrieved from REST Services
- Resource management provided by RequireJS
- API compatibility with Backbone.js Model, Collection, and Events classes, except for Backbone.js Underscore methods.
- JavaScript logging
- Popup UI handling

Hybrid Mobile Application Development Toolkit Features

Oracle JET includes support for hybrid mobile applications that run on iOS, Android, and Windows mobile devices within the Apache Cordova container.

[Apache Cordova](#) enables you to use web technologies such as HTML5, CSS, and JavaScript to develop applications that you can deploy to mobile devices. Using the Cordova JavaScript APIs to access native device services, major mobile platforms such as Android, iOS, and Windows can be supported from a common code base.

The following image shows the same application code rendered on an iOS, Android, and Windows mobile device. Oracle JET automatically applies the native theme when you configure the application for the desired platform.



To support hybrid mobile development, Oracle JET includes the following features:

- Native themes that you can use with Cordova to create hybrid mobile applications that emulate the look and feel of native iOS, Android, and Windows mobile devices
- Tooling that enables you to scaffold and build Cordova-based hybrid applications using Oracle JET directly from the command line
- Code samples, applications, and demos that you can use to create hybrid mobile applications using Oracle JET best practices
- Mobile UI behaviors and interactive gestures such as animation, sticky headers, pull to refresh, and swipe to reveal
- An `Config.getDeviceType()` function that returns the type of device your application runs on. This can be useful if you want to configure different behavior for different devices or different types of application (hybrid mobile application versus web applications).

Oracle JET Visual Component Features

Oracle JET visual components include the following features and standards compliance:

- Compliance with Oracle National Language Support (NLS) standards (i18n) for numeric, currency, and date/time formatting
- Built-in theming supporting the Oracle Redwood theme style specifications and implementing the Oracle Redwood Design System
- Support for Oracle software localization standards, l10n, including:
 - Lazy loading of localized resource strings at run time
 - Oracle translation services formats
 - Bidirectional locales (left-to-right, right-to-left)
- Web Content Accessibility Guidelines (WCAG) 2.1. In addition, components provide support for high contrast and keyboard-only input environments.
- Gesture functionality by touch, mouse, and pointer events where appropriate

- Support for Oracle test automation tooling
- Responsive layout framework

What's Included in Oracle JET

The Oracle JET zip distribution includes Oracle JET libraries and all third party libraries that the toolkit uses.

Specifically, Oracle JET includes the following files and libraries:

- CSS and CSS files for the Redwood theme (starting with JET release 9.0.0)
- CSS and SCSS files for the Alta theme (support for releases before JET release 9.0.0)
- Minified and debug versions of the Oracle JET libraries
- Data Visualization Tools (DVT) CSS and JavaScript
- Knockout and Knockout Mapping libraries
- jQuery libraries
- RequireJS, RequireJS text plugin, and RequireJS CSS plugin
- js-signals
- es6-promise polyfill
- Hammer.js

Oracle JET components use Hammer.js internally for gesture support. Do not add to Oracle JET components or their associated DOM nodes.

- Oracle JET dnd-polyfill HTML5 drag and drop polyfill
- proj4js library
- webcomponentsjs polyfill

Third Party Libraries Used by Oracle JET

To begin using Oracle JET, you do not need to understand more than the basics of JavaScript, HTML, and CSS or the third party libraries and technologies that Oracle JET uses. In fact, many developers learn about these related technologies in the process of learning Oracle JET.

Name	Description	More Information
CSS	Cascading Style Sheets	http://www.w3.org/Style/CSS
HTML5	Hypertext Markup Language 5	http://www.w3.org/TR/html5
JavaScript	Programming language	https://developer.mozilla.org/en/About_JavaScript
TypeScript	Typed superset of JavaScript that enables you to support typechecking against the TypeScript API of JET elements and non-element classes.	http://www.typescriptlang.org

Name	Description	More Information
jQuery	JavaScript library designed for HTML document traversal and manipulation, event handling, animation, and Ajax. jQuery includes an API that works across most browsers.	http://jquery.com
Knockout	JavaScript library that provides support for two-way data binding	http://www.knockoutjs.com
RequireJS	JavaScript file and module loader used for managing library references and lazy loading of resources. RequireJS implements the Asynchronous Module Definition (AMD) API.	RequireJS: http://www.requirejs.org AMD API: http://requirejs.org/docs/whyamd.html
SASS	SASS (Syntactically Awesome Style Sheets) extends CSS3 and enables you to use variables, nested rules, mixins, and inline imports to customize your application's themes. Oracle JET uses the SCSS (Sassy CSS) syntax of SASS.	http://www.sass-lang.com

If you will be using Oracle JET tooling to create web or hybrid mobile applications, you may also want to familiarize yourself with the following technologies.

Name	Description	More Information
Apache Cordova (Hybrid only)	Open source mobile development framework that allows you to use HTML5, CSS3, and JavaScript for cross-platform development targeted to multiple platforms with one code base	http://cordova.apache.org/
Node.js	Open source, cross-platform runtime environment for developing server-side web applications, used by Oracle JET for package management. Node.js includes the npm command line tool.	https://nodejs.org

Typical Workflow for Getting Started with Oracle JET Application Development

Familiarize yourself with the third party tools that Oracle JET uses before you start development. Depending on your installation method, you may also need to install prerequisite packages. After you've created your application, you can customize your configuration or load Oracle JET from the Oracle Content Delivery Network (CDN).

To get started developing Oracle JET applications, refer to the typical workflow described in the following table. After you verify your prerequisites, you can choose to create a web or hybrid mobile application.

Task	Description	More Information
Verify prerequisites	Verify that you meet the prerequisite knowledge and choose a development environment. If you will be using the recommended tooling, install the prerequisite packages.	Choose a Development Environment for Oracle JET
Create a web application	Create a web application using the tooling. Note: You must not use more than one version of Oracle JET to create your application. Oracle JET does not support running multiple versions of Oracle JET components in the same HTML document.	Understanding the Web Application Workflow or Add Oracle JET to an Existing JavaScript Application
Create a hybrid mobile application	Install Cordova and optional Android, iOS, and Windows tools. Scaffold, build, and serve development versions of Android, iOS, and Windows hybrid mobile applications.	Understanding the Hybrid Mobile Application Workflow
Modify a starter template	Understand the structure of Oracle JET starter templates and how to modify them with examples from the Oracle JET cookbook.	Work with the Oracle JET Starter Templates
Load Oracle JET from CDN	Load Oracle JET files and libraries from the Oracle Content Delivery Network (CDN).	Optimize Application Startup Using Oracle CDN and Oracle JET Libraries

Choose a Development Environment for Oracle JET

You can decide what development environment you want to use before you start developing Oracle JET applications. If you will use Oracle JET tooling to develop web or hybrid mobile applications, you must install the Oracle JET packages.

Topics:

- [Choose a Development Environment](#)
- [Install Oracle JET Tooling](#)
- (Optional) [Configure Oracle JET Applications for TypeScript Development](#)

Choose a Development Environment

You can develop Oracle JET applications in virtually any integrated development environment (IDE) that supports JavaScript (or TypeScript), HTML5, and CSS3. However, an IDE is not required for developing Oracle JET applications, and you can use any text editor to develop your application.

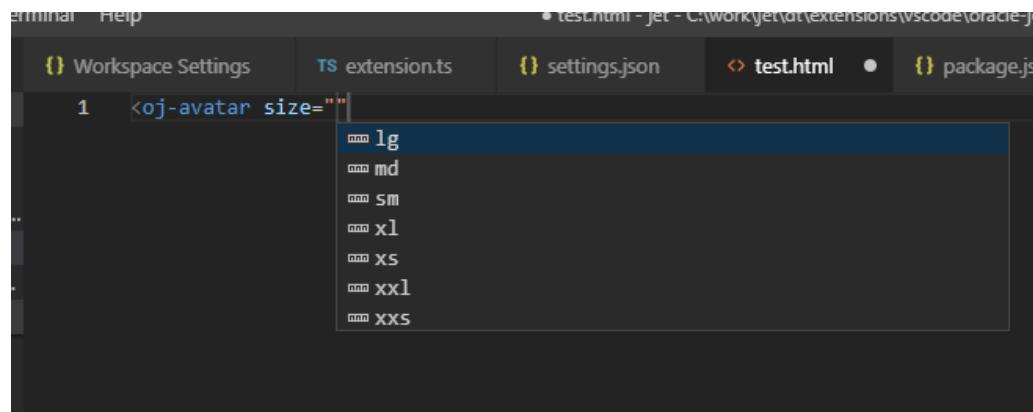
You can use an IDE in conjunction with the Oracle JET command-line tooling, where you scaffold web applications by using one of the provided starter templates. You can proceed to develop the scaffolded application in the IDE of your choice by opening the project that was created using the JET tooling, in that IDE. After saving changes

your application files in the IDE, you use the JET tooling to build and run the JET application.

If you are using Microsoft Visual Studio Code (VS Code) as your editor, you can add the Visual Studio Code Extension of Oracle JET Core to support developing Oracle JET applications. Specifically, the Oracle JET extension for VS Code improves developer productivity for creating clientside JavaScript or TypeScript web applications by providing:

- Code completion against the JET API and JET component metadata.
- Ability to work with code snippets for the most commonly used Oracle JET components.
- Capability to diagnose application source (JavaScript, HTML, CSS, and JSON files) by running Oracle JET audit reports.

This custom HTML data support for JET components support means that when you are editing HTML files, VS Code will prompt you with Oracle JET tags and attributes. As you start typing your Oracle JET HTML tag, a dropdown will show a list of matching choices:



For more examples of Oracle JET support for VS Code, visit the [Oracle JET Core Extension](#) download page in the Visual Studio Marketplace.

If you want to develop hybrid mobile applications using Oracle JET tooling, you must also install a platform-specific SDK for the platform (Android, iOS, or Windows) where you want to run the hybrid mobile application. However, you can still use your favorite editor for editing your application. For details about developing hybrid applications using Oracle JET tooling, see [Understanding the Hybrid Mobile Application Workflow](#).

Install Oracle JET Tooling

If you plan to use Oracle JET tooling to develop web or hybrid mobile applications, you must install Node.js and the Oracle JET command-line interface (CLI), ojet-cli.

 **Note:**

If you already have Oracle JET tooling installed on your development platform, check that you are using the minimum versions supported by Oracle JET and upgrade as needed. For the list of minimum supported versions, see [Oracle JET Support](#).

To install the prerequisite packages:

1. [Install Node.js](#)
2. [Install the Oracle JET Command-Line Interface](#)

Install Node.js

Install Node.js on your development machine.

From a web browser, download and install one of the installers appropriate for your OS from the [Node.js download page](#). Oracle JET recommends that you install the latest LTS version. Node.js is pre-installed on macOS, but is likely an old version, so upgrade to the latest LTS version if necessary.

After you complete installation and setup, you can enter `npm` commands from a command prompt to verify that your installation succeeded. For example, to configure a proxy server, use `npm config`.

```
npm config set proxy http-proxy-server-URL:proxy-port
npm config set https-proxy https-proxy-server-URL:proxy-port
```

Include the complete URL in the command. For example:

```
npm config set proxy http://my.proxyserver.com:80
npm config set https-proxy http://my.proxyserver.com:80
```

Install the Oracle JET Command-Line Interface

Use `npm` to install the Oracle JET command-line interface (`ojet-cli`).

- At the command prompt of your development machine, enter the following command as Administrator on Windows or use `sudo` on Macintosh and Linux machines:

```
[sudo] npm install -g @oracle/ojet-cli
```

It may not be obvious that the installation succeeded. Enter `ojet help` to verify that the installation succeeded. If you do not see the available Oracle JET commands, scroll through the install command output to locate the source of the failure.

- If you receive an error related to a network failure, verify that you have set up your proxy correctly if needed.
- If you receive an error that your version of `npm` is outdated, type the following to update the version: `[sudo] npm install -g npm`.

You can also verify the Oracle JET version with `ojet --version` to display the current version of the Oracle JET command-line interface. If the current version is not displayed, please reinstall by using the `npm install` command for your platform.

Configure Oracle JET Applications for TypeScript Development

If you plan to build an Oracle JET application or Oracle JET Web Component in TypeScript, your application project requires the TypeScript type definitions that Oracle bundles with the Oracle JET npm package.

When you install Oracle JET from npm, the TypeScript type definitions get installed with the JET bundle and are available for use when you develop applications. To begin application development using TypeScript, Oracle JET tooling supports scaffolding your application by using a variety of Oracle JET Starter Templates that have been optimized for TypeScript development, with the default ES6 implementation. For details, see [Scaffold a Web Application](#) and [Scaffold a Hybrid Mobile Application](#).

If you have already created an application and you want to switch to developing with TypeScript, you can use the Oracle JET tooling to add support for type definitions and compiler configuration. To add TypeScript to an existing application, use `ojet add typescript` from your application root.

```
ojet add typescript
```

When you add TypeScript support to an existing application, Oracle JET tooling installs TypeScript locally with an `npm install`. The tooling also creates the `tsconfig.json` compiler configuration file at your application root.

You can use the compiler configuration file, for example, to specify compiling ES6 application source files into ES5 JavaScript code and AMD format modules. During the build process JET tooling overwrites compiler options that you may have modified. If needed, you can revert overwritten options by customizing a JET hook script template, `before_app_typescript` and `before_component_typescript`, for invocation during the JET tooling build process. For details about hook scripts, see [Create a Hook Script for Web Applications](#).

When you begin development with TypeScript, you can import TypeScript definition modules for Oracle JET custom elements, as well as non-element classes, namespaces, and interfaces. For example, in this Oracle JET application, the `oj-chart` import statement supports typechecking against the element's TypeScript API.

```

1 import ArrayDataProvider = require("ojs/ojarraydataProvider");
2 import { ojChart } from "ojs/ojchart";
3 import "ojs/ojchart";
4
5 interface ChartItem {
6   id: number;
7   series: string;
8   group: string;
9   value: number;
10 }
11
12 class ViewModel {
13   private readonly data: Array<ChartItem> = [
14     {
15       "id": 0,
16       "series": "Series 1",
17       "group": "Group A",
18       "value": 42
19     },
20     {
21       "id": 2,
22       "series": "Series 2",
23       "group": "Group A",
24       "value": 55
25     }
26   ];
27   readonly dataProvider = new ArrayDataProvider<ChartItem["id"], ChartItem>(this.data, {
28     keyAttributes: "id"
29   })
30   setType = () => {
31     const chartElement = document.getElementById('chart') as ojChart<ChartItem["id"], ChartItem, null, null>;
32     chartElement.type = "pie";
33   }
34 }
35 export = ViewModel;

```

And, your editor can leverage the definition files for imported modules to display type information.

```

1 import ArrayDataProvider = require("ojs/ojarraydataProvider");
2 import { ojchart } from "ojs/ojchart";
3 import "ojs/ojchart";
4
5 interface ChartItem {
6   id: number;
7   series: string;
8   group: string;
9   value: number;
10 }
11
12 class ViewModel {
13   private readonly data: Array<ChartItem> = [
14     {
15       "id": 0,
16       "series": "Series 1",
17       "group": "Group A",
18       "value": 42
19     },
20     {
21       "id": 2,
22       "series": "Series 2",
23       "group": "Group A",
24       "value": 55
25     }
26   ];
27   readonly dataProvider = new ArrayDataProvider<ChartItem["id"], ChartItem>(this.data, {
28     keyAttributes: "id"
29   })
30   setType = () => (property) ojChart<number, ChartItem, null, null>.type: "line" | "area" | "lineWithArea" | "stock" | "boxPlot" | "combo" | "pie" | "scatter" | "bubble" | "funn
31   const chartEl = el | "pyramid" | "bar"
32   chartElement.type = "pie";
33 }
34
35 export = ViewModel;

```

Note that the naming convention for JET custom element types is changing. The type name that you specify within your TypeScript project to import a JET component's exported interface will follow one of these two naming conventions:

- *componentName + Element* (new "suffix" naming convention)

For example, `oj-input-search` and the `oj-stream-list` have the type name `InputSearchElement` and `StreamListElement`, respectively.

or

- `oj + componentName` ("oj" prefix naming convention of not yet migrated components)

For example, `oj-chart` and `oj-table` continue to adhere to the old-style type naming with "oj" prefix: `ojChart` and `ojTable`, respectively.

Until all JET component interface type names have been migrated to follow the new standard, suffix naming convention, some JET core components will continue to follow the old "oj" prefix naming convention (without the "Element" suffix). To find out the type name to specify in your TypeScript project, view the Module Usage section of the API documentation for the component.

For more information about working with TypeScript in JET, see [API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\) - JET In Typescript Overview](#).

Work with the Oracle JET Starter Templates

The Oracle JET Starter Templates provide everything you need to start working with code immediately. Use them as the starting point for your own application or to familiarize yourself with the JET components and basic structure of an Oracle JET application.

Each template is designed to work with the [Oracle JET Cookbook](#) examples and follows current best practice for application design.

Topics:

- [About the Starter Templates](#)
- [Modify Starter Template Content](#)

You can also view a video that shows how to work with the Oracle JET Starter Templates in the [Oracle JET Videos](#) collection.

About the Starter Templates

Each template in the Starter Template collection is a single page application that is structured for modular development. The collection of available Starter Templates supports JavaScript or TypeScript development and will depend on the template type you add to your application.

Instead of storing all the application markup in the `index.html` file, the application uses the `oj-module` component to bind either a view template containing the HTML markup for the section or both the view template and JavaScript or TypeScript file that contains the `viewModel` for any components defined in the section.

The following code shows a portion of the `index.html` file in the Web Nav Drawer Starter Template that highlights the `oj-module` component definition. For the sake of brevity, most of the code and comments are omitted. Comments describe the purpose of each section, and you should review the full source code for accessibility and usage tips.

```
<!DOCTYPE html>
<html lang="en-us">
  <head>
    <title>Oracle JET Starter Template - Web Nav Drawer</title>
    ... contents omitted
```

```
</head>
<body class="oj-web-applayout-body">

    ... contents omitted

    <oj-module role="main" class="oj-web-applayout-max-width oj-web-applayout-
content" config="[[moduleAdapter.koObservableConfig]]">
        </oj-module>

    ... contents omitted

        <script type="text/javascript" src="js/libs/require/require.js"></
script>
        <script type="text/javascript" src="js/main.js"></script>
    </body>
</html>
```

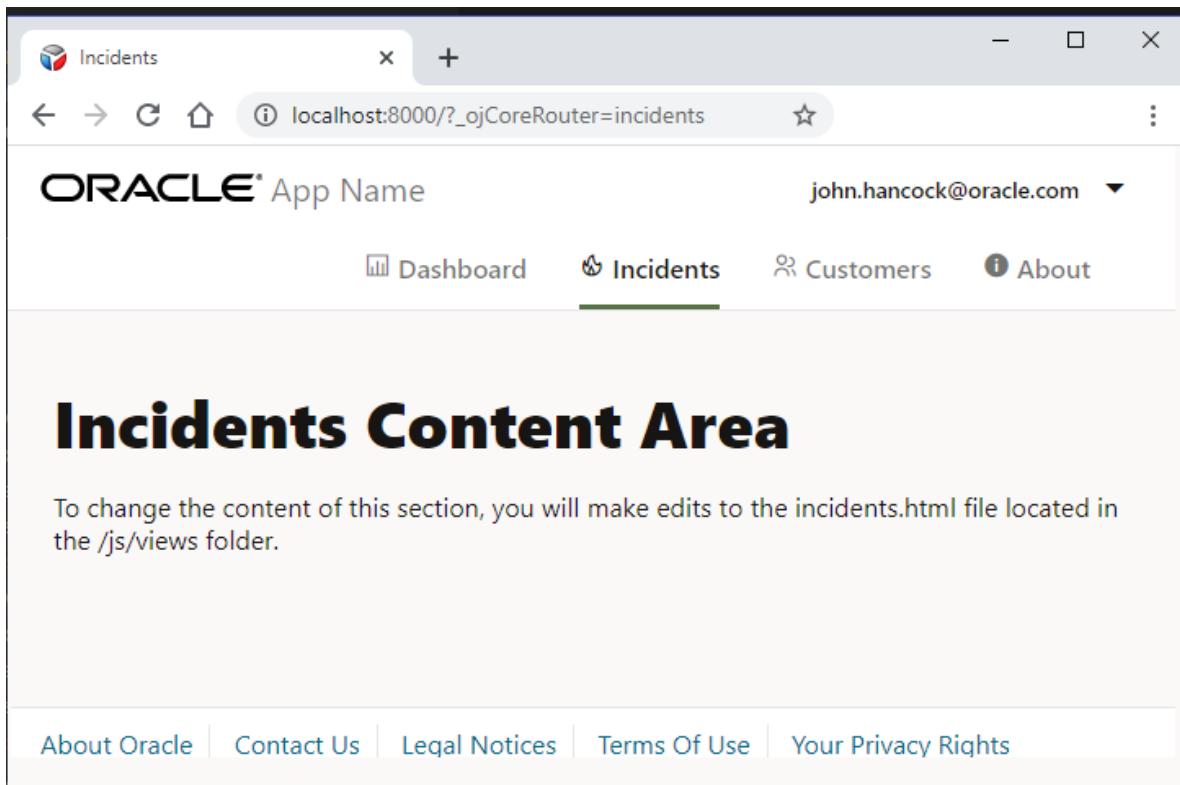
The main page's content area uses the Oracle JET `oj-web-applayout-*` CSS classes to manage the responsive layout. The main page's content uses the HTML `oj-module` element with its role defined as `main` (`role="main"`) for accessibility.

The `oj-module` component's `config.view` attribute tells Oracle JET that the section is only defining a view template, and the view will be bound to the existing `viewModel`. When the `oj-module` element's `config.view-model` attribute is defined, the application will load both the `viewModel` and view template with the name corresponding to the value of the `config.view-model` attribute.

When the `oj-module` element's `view` and `view-model` attributes are missing, as in this example, the behavior will depend on the parameter specified in the `config` attribute's definition.

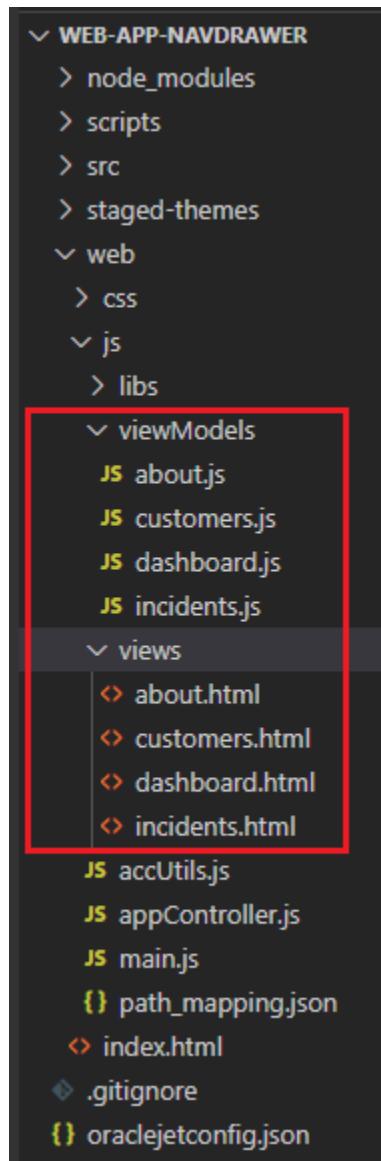
- If the parameter is an Oracle JET router's `moduleConfig` object as in the above example, then `oj-module` will automatically load and render the content of the `viewModel` script and view template corresponding to the router's current state.

The Web Nav Drawer Starter Template uses `CoreRouter` to manage navigation when the user clicks one of the application's navigation items. The routes include `dashboard`, `incidents`, `customers`, and `about`. If the user clicks **Incidents**, for example, the main content area changes to display the content in the `incidents` view template.



- If the parameter is a Knockout observable containing the name of the viewModel, the application will load both the viewModel and view template with the indicated name.

The /js/views folder contains the view templates for the application and the /js/viewModels contains the viewModel scripts. The image below shows the Web Nav Drawer Starter Template file structure.



For additional information about working with single page applications, `oj-module`, `CoreRouter`, and Knockout templates, see [Creating Single-Page Applications](#).

For details about the `oj-web-applayout-*` CSS classes, see [Web Application Patterns](#). For additional information about working with responsive design, see [Designing Responsive Applications](#).

About Modifying Starter Templates

The Starter Template is the starting point for creating your applications. You can modify any Oracle JET starter template to provide a customized starting point.

You can obtain the Starter Template from the Oracle JET application that you create using one of the following methods.

- [Scaffold a Web Application](#)

- [Scaffold a Hybrid Mobile Application](#)

Load the starter template into your favorite IDE, or extract the zip file into a development folder.

 **Tip:**

If you used the command line tooling to scaffold your application, you can still use an IDE like Visual Studio Code for editing. For example, in Visual Studio Code, choose **File -> Open Folder** and select the folder containing the application you created. Edit your application as needed, but use the tooling commands in a terminal window to build and serve your application.

To modify the template you can remove unneeded content and add new content. Content that you add can be your own or you can reuse content from Oracle JET Cookbook samples. When you copy markup from a Cookbook sample, you copy the desired HTML and the supporting JavaScript.

Included in the code you add will be the RequireJS module dependency for the code. The application's `main.js` file contains the list of RequireJS modules currently included in the application. If you are using the Cookbook sample, you can determine modules that you need to add by comparing list of libraries in the application's `main.js` file to the list in the Cookbook sample. You will add any missing modules to the `define()` function in the JavaScript file for your application. For example, to add the `oj-input-date-time` component from the Cookbook, you would need to add the `ojs/ ojdatetimepicker` module to the `dashboard.js` viewModel file since it's not already defined in `dashboard.js`.

To familiarize yourself with the RequireJS module to add for a Cookbook sample or for your own code, see the table at [About Oracle JET Module Organization](#).

If you add content to a section that changes its role, then be sure to change the role associated with that section. Oracle JET uses the role definitions for accessibility, specifically WAI-ARIA landmarks. For additional information about Oracle JET and accessibility, see [Developing Accessible Applications](#).

Modify Starter Template Content

To add content, modify the appropriate view template and ViewModel script (if it exists) for the section that you want to update. Add any needed RequireJS modules to the ViewModel's `define()` definition, along with functions to define your ViewModel.

The example below uses the Web Nav Drawer Starter Template, but you can use the same process on any of the Starter Templates.

Before you Begin:

- See the [Date and Time Pickers](#) demo in the Oracle JET Cookbook. This task uses code from this sample.

To modify the Starter Template content:

1. In your application's `index.html` file, locate the `oj-module` element for the section you want to modify and identify the template and optional ViewModel script.

In the Web Nav Drawer Starter Template, the `oj-module` element is using the `config` attribute. The following code sample shows the `mainContent` HTML `oj-module` definition in `index.html`, where the `moduleAdapter` observable, a `ModuleAdapterClass` object, obtains the configuration from its `koObservableConfig` field.

```
<oj-module role="main" class="oj-web-applayout-max-width oj-web-applayout-content"
           config="[[moduleAdapter.koObservableConfig]]">
</oj-module>
```

The return value of the `[[moduleAdapter.koObservableConfig]]` observable is set to the current state of the `CoreRouter` object. The `CoreRouter` object is defined with an initial value of `dashboard` in the application's `appController.js` script, where the page initially loads and no path is yet specified, as shown in the `navData` array below for the empty path case. The `router` object is created from the array and then passed to the `moduleAdapter` declaration.

```
let navData = [
    { path: '', redirect: 'dashboard' },
    { path: 'dashboard', detail: { label: 'Dashboard', iconClass: 'oj-ux-ico-bar-chart' } },
    { path: 'incidents', detail: { label: 'Incidents', iconClass: 'oj-ux-ico-fire' } },
    { path: 'customers', detail: { label: 'Customers', iconClass: 'oj-ux-ico-contact-group' } },
    { path: 'about', detail: { label: 'About', iconClass: 'oj-ux-ico-information-s' } }
];

// Router setup
let router = new CoreRouter(navData, {
    urlAdapter: new UrlParamAdapter()
});
router.sync();

this.moduleAdapter = new ModuleRouterAdapter(router);
this.selection = new KnockoutRouterAdapter(router);
```

The navigation data provider for `oj-navigation-list` element is created as an `ArrayDataProvider` object that associates the available `navData` routes by using the `slice(1)` function to remove the first path definition in the `navData` array that specifically handles the "empty path" case.

```
// Setup the navDataProvider with the routes, excluding the first redirected route.
this.navDataProvider = new ArrayDataProvider(navData.slice(1),
{keyAttributes: "path"});
```

To modify the starter templates, for example, the Dashboard Content Area, you will modify both `dashboard.html` and `dashboard.js`.

2. To modify the view template, remove unneeded content, and add the new content to the view template file.

For example, if you are working with an Oracle JET Cookbook sample, you can copy the markup into the view template you identified in the previous step (`dashboard.html`). Replace everything after the `<h1>Dashboard Content Area</h1>` markup in the template with the markup from the sample.

The following code shows the modified markup if you replace the existing content with a portion of the content from the Date and Time Pickers demo.

```
<div id="div1">
  <ojs/oj-label for="dateTime">Default</ojs/oj-label>
  <ojs/oj-input-date-time id="dateTime" value='{{value}}'>
  </ojs/oj-input-date-time>

<br/><br/>

  <span class="oj-label">Current component value is:</span>
  <span><ojs/oj-bind-text value="[[value]]"></ojs/oj-bind-text></span>

</div>
```

3. To modify the ViewModel, remove unneeded content, and add the new content as needed. Include any additional RequireJS modules that your new content may need.

The application's `main.js` file contains the list of RequireJS modules currently included in the application. Compare the list of libraries with the list you need for your application, and add any missing modules to your `define()` function in the ViewModel script. For example, to use the `oj-input-date-time` element shown in the demo and to use the `IntlConverterUtils` namespace API, add `ojs/ojdatetimetypepicker` and add `ojs/ojconverterutils-i18n` modules to the `dashboard.js` ViewModel script since it's not already defined in `dashboard.js`.

The sample below shows a portion of the modified `dashboard.js` file, with the changes highlighted in bold.

```
define(['knockout', 'ojs/ojconverterutils-i18n', 'ojs/ojknockout', 'ojs/ojdatetimetypepicker', 'ojs/ojlabel'],
  [], function(ko, ConverterUtilsI18n) {
  /**
   * The view model for the main content view template
   */
  function DashboardViewModel() {
    var self = this;
    self.value =
      ko.observable(ConverterUtilsI18n.IntlConverterUtils.dateToLocalIso(new
      Date(2020, 0, 1)));
  }

  return DashboardViewModel;
});
```

Note:

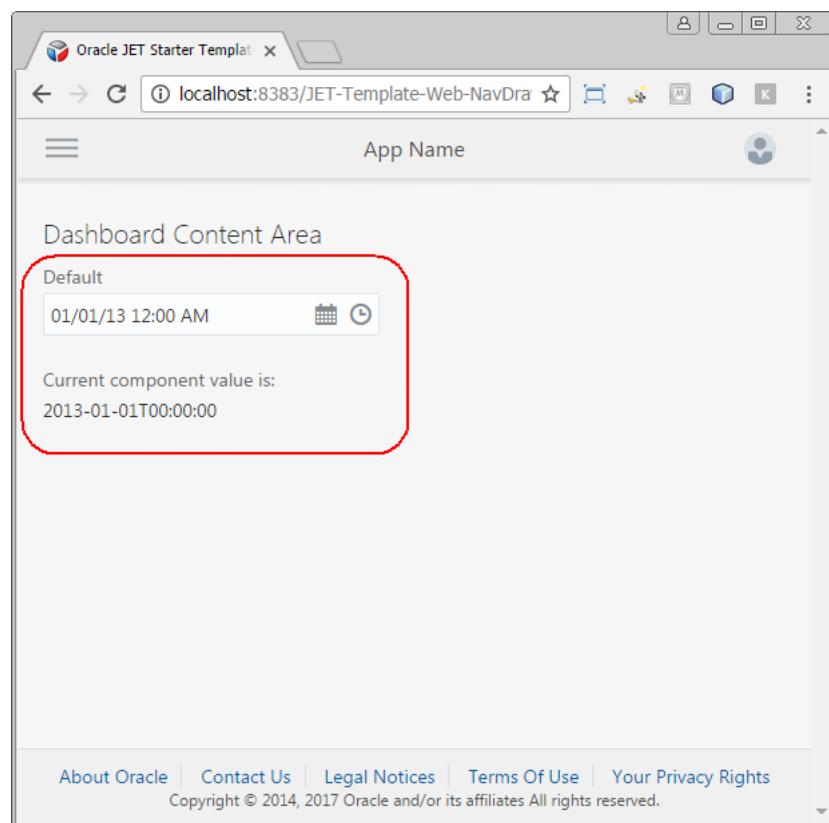
In this example, you are not copying the entire code section. The Cookbook uses a `require()` call to load and use the needed libraries in a single bootstrap file. The Starter Template that you are pasting uses `define()` to create a RequireJS module that can be used by other parts of your application.

4. If you want to add, change, or delete modules or templates in the application, modify the `main.js` RequireJS bootstrap file and `appController.js` file as needed.

The `appController.js` file also contains the event handler that responds when a user clicks one of the navigation items. Depending upon your modifications, you may need to update this method as well.

5. Verify the changes in your favorite browser.

The following image shows the runtime view of the Web Nav Drawer Starter Template with the new Dashboard Content Area content showing `oj-input-datetime` with its current value.



Work with the Oracle JET Base Distribution

The Oracle JET Base Distribution provides an alternative to creating your application by using an Oracle JET Starter Template. It also provides the open source libraries for Oracle JET, the Oracle JET components, and the Oracle JET stylesheets. You can also use the base distribution to add Oracle JET functionality to your existing JavaScript application.

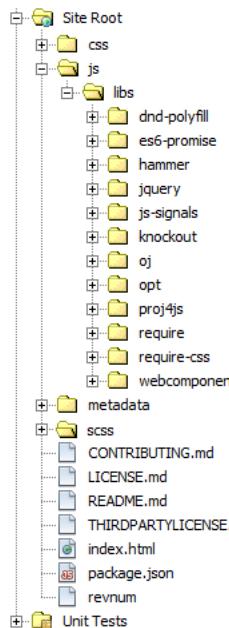
Topics:

- [About the Oracle JET Base Distribution](#)
- [Add Oracle JET to an Existing JavaScript Application](#)

About the Oracle JET Base Distribution

Oracle makes the Oracle JET base distribution available as a zip file that you can download. You can add files from the base distribution to your existing application when you want to use Oracle JET libraries and theming.

You add Oracle JET to your existing JavaScript application by extracting the `oraclejet.zip` file into your application's Site Root. After extraction, your site root folder will contain the following Oracle JET files and folders.



The `css` folder contains the themes included with Oracle JET, and the `scss` folder contains the SASS source files for the Oracle JET themes, as described in [Using CSS and Themes in Applications](#).

The `js` folder contains the Oracle JET libraries and third party libraries, included in Oracle JET, as described in [What's Included in Oracle JET](#). The `buildnum` and `revnum` files identify the Oracle JET build number and version.

If you want to use the Oracle JET libraries, you can use RequireJS, a third party JavaScript library provided with the base distribution, to manage the Oracle JET library, link, and script references. For a list of available Oracle JET modules and additional details about using RequireJS to manage library references in your Oracle JET application, see [Using RequireJS for Modular Development](#).

Add Oracle JET to an Existing JavaScript Application

You can add Oracle JET to your existing JavaScript application by extracting the `oraclejet.zip` file into the site root of your application and including references to the Oracle JET libraries and CSS as needed.

Before you begin:

- Oracle JET can be downloaded at: [Oracle JET Downloads](#).

To add Oracle JET to an existing JavaScript application:

1. Navigate to the Oracle JET download location.
2. Choose **Accept License Agreement**.
3. Download **Oracle JavaScript Extension Toolkit: Base Distribution**.
4. Extract `oraclejet.zip` into the site root of your application.
5. If you want to use one of the themes included with Oracle JET, add the appropriate link to the CSS.
 - a. Create a new `index.html` file in the project root.
 - b. In the application's main page, `index.html`, add the HTML `link` element and point it to the CSS theme that you will use.

For example, to use the Oracle JET Alta web theme:

```
<!-- Oracle JET CSS files -->
<link rel="stylesheet" href="css/libs/oj/version/alta/oj-alta-min.css"
type="text/css" />
```

- a. If you will be using the CSS included with Oracle JET, add the following line to the top of your application's main HTML page:

```
<!DOCTYPE html>
```

This line is needed for the Oracle JET CSS to function as expected.

6. If you want to use the Oracle JET libraries, you can use RequireJS to manage the Oracle JET library, link, and script references.
 - a. Copy `js/libs/oj/version/main-template.js` to the `js` folder.
 - b. In the `js` folder, rename `main-template.js` to `main.js`.
 - c. Add the following script reference to your `index.html` file:

```
<script data-main="js/main" src="js/libs/require/require.js"></script>
```
 - d. Update `main.js` as needed to reference Oracle JET modules or your own scripts.

Optimize Application Startup Using Oracle CDN and Oracle JET Libraries

You can configure the Oracle JET application to minimize the network load at application startup through the use of Oracle Content Delivery Network (CDN) and the Oracle JET distributions that the CDN supports.

When your production application supports users who access the application from diverse geographical locations, you can perform a significant performance optimization by configuring the Oracle JET application to access Oracle CDN as its source for loading the required Oracle JET libraries and modules. Oracle maintains its CDN with the libraries and modules that are specific to a given Oracle JET release. The CDN support for each release is analogous to the way Oracle JET tooling also supports copying these files into the local `src` folder of the application for a particular release. In both cases, access to the appropriate libraries and modules is automated for the

application developer. You configure the application to determine where you want the application to load the libraries and modules from.

After you create your application, the application is configured by default to load the needed libraries and modules from the local `src` folder. This allows you to create the application without the requirement for network access. Then, when you are ready to test in a staging environment or to move to production, you can configure the Oracle JET application to use CDN server replication to reduce the network load that occurs when users access the application at the start of a browser session. When the user initially starts the application in their browser, Oracle CDN ensures a distributed server closest to the geographic location of the user is used to deliver the application's needed third party libraries and Oracle JET modules to the user's browser.

Configuring the application to load from CDN offers these advantages over loading from the application `src` folder:

- Once loaded from a CDN distribution server, the required libraries and modules will be available to other applications that the user may run in the same browser session.
- Enables the option to load bundled libraries and modules using a bundles configuration file that Oracle maintains on CDN. The bundles configuration file groups the most commonly accessed libraries and modules into content packages that are specific to the release and makes them available for delivery to the application as a bundle.

 **Tip:**

Configuring your application to reference the bundles configuration on Oracle CDN is recommended because Oracle maintains the configuration for each release. By pointing your application to the current bundles configuration, you will ensure that your application runs with the latest supported library and module versions. For information about how to enable this bundle loading optimization, see [About Configuring the Application for Oracle CDN Optimization](#).

2

Understanding the Web Application Workflow

Developing client-side web applications with Oracle JET is designed to be simple and efficient using the development environment of your choice and Starter Templates to ease the development process.

Oracle JET supports creating web applications from a command-line interface:

- Before you can create your first Oracle JET web application using the CLI, you must install the prerequisite packages if you haven't already done so. For details, see [Install Oracle JET Tooling](#).
- Then, use the Oracle JET command-line interface package (`ojet-cli`) to scaffold a web application containing either a blank template or a complete pre-configured sample application that you can modify as needed.
- After you have scaffolded the application, use the `ojet-cli` to build the application, serve it in a local web browser, and create a package ready for deployment. This approach also supports easily creating hybrid mobile applications.

Note:

For additional information about creating hybrid mobile applications, see [Understanding the Hybrid Mobile Application Workflow](#).

You must not use more than one version of Oracle JET to add components to the same HTML document of your web application. Oracle JET does not support running multiple versions of Oracle JET components in the same web page.

Topics:

- [Scaffold a Web Application](#)
- [Build a Web Application](#)
- [Serve a Web Application](#)
- [Customize the Web Application Tooling Workflow](#)
- [Serve a Web Application to a HTTPS Server Using a Self-signed Certificate](#)

Scaffold a Web Application

Use the Oracle JET command-line interface (CLI) to scaffold an application that contains a blank template or one pre-configured Starter Template with a basic layout, navigation bar, or navigation drawer. Each Starter Template is optimized for responsive web or hybrid mobile applications. Additionally, Starter Templates support

TypeScript development should you wish to create your application in TypeScript. After scaffolding, you can modify the application as needed.

Before you can create your first Oracle JET web application using the CLI, you must also install the prerequisite packages if you haven't already done so. For details, see [Install Oracle JET Tooling](#).

To scaffold an Oracle JET web application:

1. At a command prompt, enter `ojet create` with optional arguments to create the Oracle JET application and scaffolding.

```
ojet create [directory]
    [--template={template-name:[web|hybrid]|template-url|
    template-file}]
    [--typescript]
    [--help]
```

 **Tip:**

You can enter `ojet help` at a terminal prompt to get additional help with the Oracle JET CLI.

For example, the following command will create a web application in the `web-app-navbar` directory using the web version of the `navbar` template:

```
ojet create web-app-navbar --template=navbar
```

To scaffold the web application using the same Starter Template but with support for TypeScript development, add the `--typescript` argument to the command:

```
ojet create web-app-navbar --template=navbar --typescript
```

To scaffold the web application with the hybrid mobile version of the `navbar` template, enter the following command:

```
ojet create web-app-navbar --template=navbar:hybrid
```

2. Wait for confirmation.

The scaffolding will take some time to complete. When successful, the displays:

```
Oracle JET: Your app is ready! Change to your new app directory
web-app-navbar and try ojet build and serve...
```

3. In your development environment, update the code for your application.

 **Tip:**

If you selected the blank template during scaffolding, you can still follow the same process to add cookbook samples or other content to your application. However, it will be up to you to create the appropriate view templates or viewModel scripts.

About `ojet create` Command Options for Web Applications

Use `ojet create` with optional arguments to create the Oracle JET web application and scaffolding.

The following table describes the available `ojet create` command options and provides examples for their use.

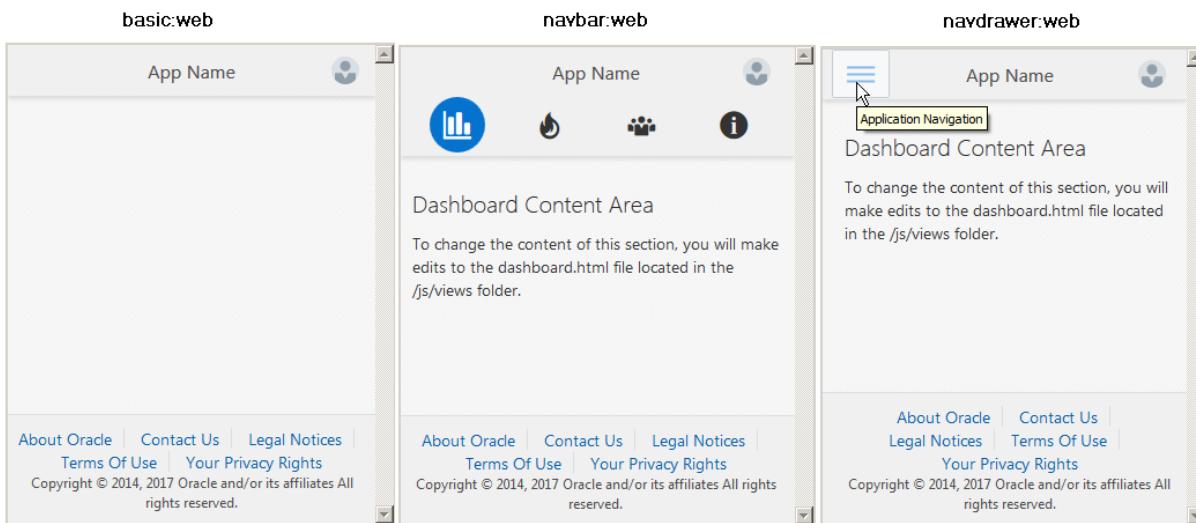
Option	Description
<code>directory</code>	Application location. If not specified, the application is created in the current directory. The directory will be created during scaffolding if it doesn't already exist.
<code>template</code>	<p>Template to use for the application. Specify one of the following:</p> <ul style="list-style-type: none">• <code>template-name</code> Predefined template. You can enter blank, basic, navbar or navdrawer . Defaults to blank if not specified.• <code>template-URL</code> URL to zip file containing the name of a zipped application: <code>http://path-to-app/app-name.zip</code>.• <code>template-file</code> Path to zip file on your local file system containing the name of a zipped application: "<code>path-to-app/app-name.zip</code>". For example: <code>--template="C:\Users\SomeUser\app.zip"</code> <code>--template="/home/users/SomeUser/app.zip"</code> <code>--template("~/projects/app.zip")</code> <p>If the <code>src</code> folder is present in the zip file, then all content will be placed under the <code>src</code> directory of the application, except for the <code>script</code> folder which remains in the root. If no <code>src</code> folder is present, the contents of the zip file will be placed at the root of the new application.</p>

Option	Description
help	Displays a man page for the <code>ojet create</code> command, including usage and options: <code>ojet create --help</code> .

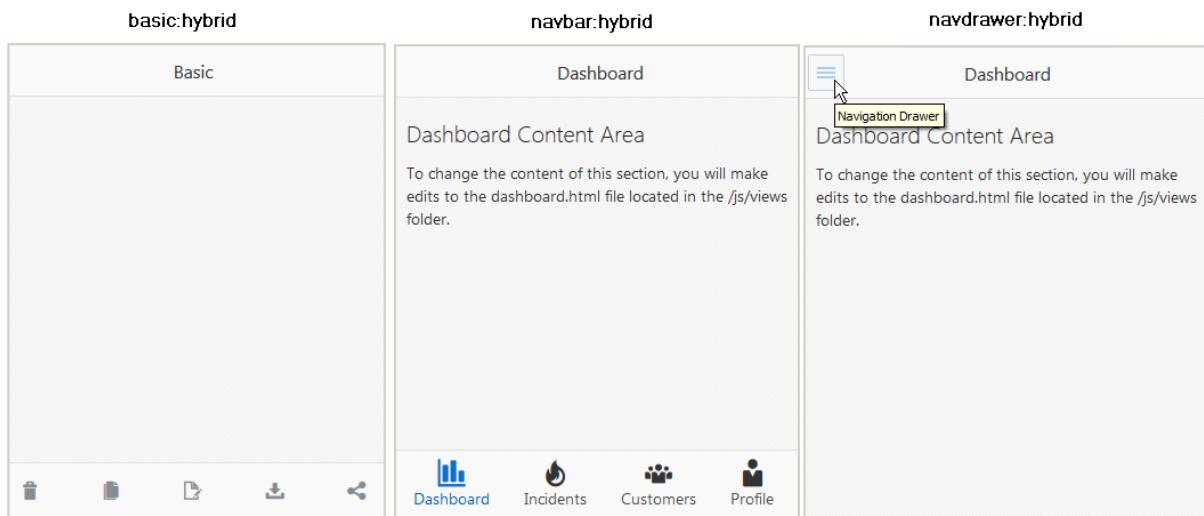
About Scaffolding a Web Application

Scaffolding is the process you use in the Oracle JET command-line interface (CLI) to create an application that contains a blank template or one pre-configured with a basic layout, navigation bar, or navigation drawer. Each pre-configured template is optimized for responsive web or hybrid mobile applications. After scaffolding, you can modify the application as needed.

The following image shows the differences between the pre-configured Starter Templates. The blank template contains an `index.html` file but no UI features. The `basic:web` template is similar to the blank template but adds responsive styling that will adjust the display when the screen size changes. The `navbar:web` and `navdrawer:web` templates contain sample content and follow best practices for layout, navigation, and styling that you can also modify as needed.



If you want your web application to look more like a mobile application, you can scaffold your web application with a hybrid mobile version of the basic, navbar, or navdrawer template: `basic:hybrid`, `navbar:hybrid`, or `navdrawer:hybrid`.



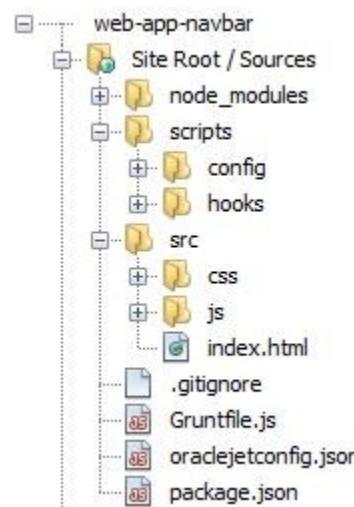
After scaffolding, you can perform the following tasks to customize your application:

- [Modify Starter Template Content](#)
- [Modify the Web Application's File Structure](#)
- [Add Hybrid Mobile Features to Web Applications](#)

About the Web Application File Structure

The Oracle JET scaffolding process creates files and folders that you modify as needed for your application.

The new application will have a directory structure similar to the one shown in the following image.



The application folders contain the application and configuration files that you will modify as needed for your own application.

Directory or File	Description
node_modules	Contains the Node.js modules used by the tooling.
scripts	Contains template hook scripts that you can modify to define new build and serve steps for your application. See Customize the Web Application Tooling Workflow
src	<p>Site root for your application. Contains the application files that you can modify as needed for your own application and should be committed to source control.</p> <p>The content will vary, depending upon your choice of template. Each template, even the blank one, will contain an <code>index.html</code> file and a <code>main.js</code> RequireJS bootstrap file.</p> <p>Other templates may contain view templates and <code>viewModel</code> scripts pre-populated with content. For example, if you specified the navbar template during creation, the <code>js/views</code> and <code>js/viewModels</code> folders will contain the templates and scripts for a web application that uses a nav bar for navigation.</p>
.gitignore	Defines rules for application folders to ignore when using a GIT repository to check in application source. Users who do not use a GIT repository can use <code>ojet strip</code> to avoid checking in content that Oracle JET always regenerates. Note this file must not be deleted since the <code>ojet strip</code> command depends on it.
oraclejetconfig.json	Contains the default source and staging file paths that you can modify if you need to change your application's file structure.
package.json	Defines npm dependencies and project metadata.

After scaffolding, you can perform the following tasks to customize your application:

- [Modify Starter Template Content](#)
- [Modify the Web Application's File Structure](#)
- [Add Hybrid Mobile Features to Web Applications](#)

Modify the Web Application's File Structure

You can modify your scaffolded application's file structure if the default structure doesn't meet your needs.

The `oraclejetconfig.json` file in your application's top level directory contains the default source and staging file paths that you can modify.

```
{
  "paths": {
    "source": {
      "common": "src",
      "web": "src-web",
      "hybrid": "src-hybrid",
      "javascript": "js",
      "styles": "css",
      "themes": "themes"
    }
  }
}
```

```
        } ,
        "staging": {
            "web": "web",
            "hybrid": "hybrid",
            "themes": "themes"
        }
    },
    "defaultBrowser": "chrome",
    "sassVer": "4.13.0",
    "defaultTheme": "redwood",
    "defaultCssvars": "disabled",
    "generatorVersion": "9.2.0"
}
```

To change the web application's file structure:

1. In your application's top level directory, open `oraclejetconfig.json` for editing.
2. In `oraclejetconfig.json`, change the paths as needed and save the file.

For example, if you want to change the default styles path from `css` to `app-css`, edit the following line in `oraclejetconfig.json`:

```
"styles": "app-css"
```

3. Rename the directories as needed for your application, making sure to change only the paths listed in `oraclejetconfig.json`.

For example, if you changed `styles` to `app-css` in `oraclejetconfig.json`, change the application's `css` directory to `app-css`.

4. Update your application files as needed to reference the changed path.

For example, if you modified the path to the CSS for your application to `app-css`, update the links appropriately in your application's `index.html`.

```
<link rel="icon" href="app-css/images/favicon.ico" type="image/x-icon" />

<!-- This is the main css file for the default Alta theme -->
<!-- injector:theme -->
<link rel="stylesheet" href="app-css/libs/obj/v9.2.0/redwood/obj-redwood-min.css" type="text/css"/>
<!-- endinjector -->
```

5. At the command prompt, from the application root directory, build your application to use the new paths.

```
ojet build
```

Add Hybrid Mobile Features to Web Applications

Add Cordova hybrid mobile platforms to your web application using the `ojet add hybrid` command. When you run this command, the tooling creates a Cordova project

and new `src-hybrid` and `src-web` directories, enabling you to create both web and hybrid mobile applications from the same source.

Since mobile applications typically provide a very different user experience to web applications, taking this approach allows you to add view and `viewModel` files that you create specifically for a mobile device user experience. Then when you build the original application by using the command `ojet build hybrid`, the tooling will copy your added files from the `src-hybrid` folder and merge them with the contents of the application's `src` folder. Finally, serving the application, by using the command `ojet serve hybrid`, populates the `web staging` folder with the merged source files and the tooling runs the application in the desired environment (browser, simulator, or mobile device) from this location.

Before you begin:

- If needed, install the mobile tooling packages, as described in [Understanding the Hybrid Mobile Application Workflow](#).
- Familiarize yourself with the folder structure of your web application. The file locations will vary in this procedure if you modified your directory structure, as described in [Modify the Web Application's File Structure](#) or [Modify the Hybrid Mobile Application's File Structure](#).

To add hybrid mobile platforms to your web application:

1. At a terminal prompt, in your application's top level directory, enter the following command to add hybrid mobile features to your web application:

```
ojet add hybrid [--platforms=android,ios,windows]
                [--appid=application-id]
                [--appname=application-name]
                [--help]
```

You should specify at least one platform for the `platforms` value. Add additional platforms as needed, separated by commas. If you don't specify a platform, the command will attempt to locate available platforms and prompt for your selection.

```
--platforms=android
--platforms=ios,android
```

When you run the command, Oracle JET tooling creates the Cordova project and two new empty source directories that you can use for platform specific content: `src-hybrid` and `src-web`.

2. To make changes to content that apply to both web and hybrid platforms, edit content in `src`.
3. To make changes to content that apply only to the web or hybrid mobile platform, add the new content to `src-web` or `src-hybrid` as needed.

When you build your application, the content in the platform specific file takes precedence over content in `src`. For example, if you create a new `src-hybrid/js/viewModels/dashboard.html` file, content in that file will take precedence over the `src/js/viewModels/dashboard.html` file when you build the application as a hybrid mobile application.

 **Note:**

It is important that files you add to the `src-web` or `src-hybrid` folder maintain the same folder structure as the `src` merge target folder. For example, based on the original application's default folder structure, view or viewModel files specific to the hybrid mobile platform that you add should appear in the `src-hybrid/js/view` and `src-hybrid/js/viewModels` directories.

Build a Web Application

Use the Oracle JET command-line interface (CLI) to build a development version of your web application before serving it to a browser. This step is optional.

Change to the application's root directory and use the `ojet build` command to build your application.

```
ojet build [--cssvars=enabled|disabled  
           --theme=themename  
           --themes=theme1,theme2,...  
           --sass]
```

 **Tip:**

You can enter `ojet help` at a terminal prompt to get help for specific Oracle JET CLI options.

The command will take some time to complete. If it's successful, you'll see the following message:

Done.

The command will also create a web folder in your application's root to contain the built content.

 **Note:**

You can also use the `ojet build` command with the `--release` option to build a release-ready version of your application. For information, see [Package and Deploy Web Applications](#).

About `ojet build` Command Options for Web Applications

Use the `ojet build` command with optional arguments to build a development version of your web application before serving it to a browser.

The following table describes the available options and provides examples for their use.

Option	Description
--theme	Theme to use for the application. The theme defaults to redwood for web applications and hybrid mobile applications. Note: If you have migrated to JET 9.0.0 and later, and want to continue building with your Alta theme, you can specify alta:web, or for hybrid mobile themes: alta:android, alta:ios, alta:windows). Note that the Alta theme is supported through JET 10.0.0 but is expected to become deprecated beyond that. For details about migrating applications, see Oracle JET Application Migration for Release 9.2.0 . You can also enter a different <i>themename</i> for a custom theme as described in About CSS Variables and Custom Themes in Oracle JET for Redwood themes and, for a migrated custom Alta theme, in Customize Alta Themes Using the Tooling .
--themes	Themes to include in the application, separated by commas. If you don't specify the --theme flag as described above, Oracle JET will use the first element that you specify in --themes as the default theme.
--cssvars	Injects a Redwood theme CSS file that supports working with CSS custom properties when you want to override CSS variables to customize the Redwood theme, as described in About CSS Variables and Custom Themes in Oracle JET .
--sass	Manages Sass compilation. If you add Sass and specify the --theme or --themes option, Sass compilation occurs by default and you can use --sass=false or --no-sass to turn it off. If you add Sass and do not specify a theme option, Sass compilation will not occur by default, and you must specify --sass=true or --sass to turn it on. For details about theming with Sass, see Work with Sass .

Serve a Web Application

Use `ojet serve` to run your web application in a local web server for testing and debugging. By default, the Oracle JET live reload option is enabled which lets you make changes to your application code that are immediately reflected in the browser.

Before you begin:

- Familiarize yourself with the `ojet serve` command option `theme` when you want to run the application with an optional platform and a custom theme, as described in [Customize Alta Themes Using the Tooling](#).
- Optionally, use the `ojet serve` command with the `--release` option to serve a release-ready version of your application, as described in [Package and Deploy Web Applications](#).

To run your web application from a Command Prompt window:

1. In a Command Prompt window, change to the application's root directory and use the `ojet serve` command with optional arguments to launch the application.

```
ojet serve [--server-port=server-port-number --livereload-port=livereload-port-number
            --livereload
            --sass
            --build
            --cssvars=enabled|disabled
            --theme=themename --themes=theme1,theme2,...
            --server-only
        ]
```

For example, the following command will launch your application in the default web browser with live reload enabled.

```
ojet serve
```

 **Tip:**

You can enter `ojet help` at a terminal prompt to get help specific to the Oracle JET tooling commands and `ojet serve --help` to get additional help with serve options.

2. Make any desired code change in the `src` folder, and the browser will update automatically to reflect the change unless you set the `--no-livereload` flag.

While the application is running, the terminal window remains open, and the watch task waits for any changes to the application. For example, if you change the content in `src/js/views/dashboard.html`, the watch task will reflect the change in the terminal as shown below on a Windows desktop.

```
Starting watcher...
Listening on port 35729...
Server ready: http://localhost:8000
Watching files....
Watcher: sass is ready...
Watcher: sourceFiles is ready...
Watcher: themes is ready...
Changed: c:\web-app-navbar\src\js\views\dashboard.html
Page reloaded resume watching...
```

3. To terminate the process, close the application and press `Ctrl+C` in the command window. You may need to enter `Ctrl+C` a few times before the process terminates.

About `ojet serve` Command Options for Web Applications

Use `ojet serve` to run your web application in a local web server for testing and debugging.

The following table describes the available `jet serve` options and provides examples for their use.

Option	Description
<code>server-port</code>	Server port number. If not specified, defaults to 8000.
<code>livereload-port</code>	Live reload port number. If not specified, defaults to 35729.
<code>livereload</code>	<p>Enable the live reload feature. Live reload is enabled by default (<code>--livereload=true</code>).</p> <p>Use <code>--livereload=false</code> or <code>--no-livereload</code> to disable the live reload feature.</p> <p>Disabling live reload can be helpful if you're working in an IDE and want to use that IDE's mechanism for loading updated applications.</p>
<code>build</code>	<p>Build the app before you serve it. By default, an app is built before you serve it (<code>--build=true</code>).</p> <p>Use <code>--build=false</code> or <code>--no-build</code> to suppress the build if you've already built the application and just want to serve it.</p>
<code>theme</code>	<p>Theme to use for the application. The theme defaults to <code>redwood</code> for web applications and hybrid mobile applications.</p> <p>Note: If you have migrated to JET 9.0.0 and later, and want to continue building with your Alta theme, you can specify <code>alta:web</code>, or for hybrid mobile themes: <code>alta:android</code>, <code>alta:ios</code>, <code>alta:windows</code>). Note that the Alta theme is supported through JET 10.0.0 but is expected to become deprecated beyond that. For details about migrating applications, see Oracle JET Application Migration for Release 9.2.0.</p> <p>You can also enter a different <code>themename</code> for a custom theme as described in About CSS Variables and Custom Themes in Oracle JET for Redwood themes and, for a migrated custom Alta theme, in Customize Alta Themes Using the Tooling.</p>
<code>themes</code>	<p>Themes to use for the application, separated by commas.</p> <p>If you don't specify the <code>--theme</code> flag as described above, Oracle JET will use the first element that you specify in <code>--themes</code> as the default theme. Otherwise Oracle JET will serve the application with the theme specified in <code>--theme</code>.</p>
<code>--cssvars</code>	Injects a Redwood theme CSS file that supports working with CSS custom properties when you want to override CSS variables to customize the Redwood theme. For details about theming with CSS variables, see About CSS Variables and Custom Themes in Oracle JET .

Option	Description
sass	<p>Manages Sass compilation. If you add Sass and specify the <code>--theme</code> or <code>--themes</code> option, Sass compilation occurs by default and you can use <code>--sass=false</code> or <code>--no-sass</code> to turn it off.</p> <p>If you add Sass and do not specify a theme option, Sass compilation will not occur by default, and you must specify <code>--sass=true</code> or <code>--sass</code> to turn it on. For details about theming with Sass, see Work with Sass.</p>
server-only	Serves the application, as if to a browser, but does not launch a browser. Use this option in cloud-based development environments so that you can attach your browser to the app served by the development machine.

Serve a Web Application to a HTTPS Server Using a Self-signed Certificate

You can customize the JET CLI tooling to serve your web application to a HTTPS server instead of the default HTTP server that the Oracle JET `ojet serve` command uses.

Do this if, for example, you want to approximate your development environment more closely to a production environment where your web application will eventually be deployed. Requests to your web application when it is deployed to a production environment will be served from an SSL-enabled HTTP server (HTTPS).

To implement this behavior, you'll need to install a certificate in your web application directory. You'll also need to configure the `before_serve.js` hook to do the following:

- Create an instance of Express to host the served web application.
Express is the Node.js web application framework that Oracle JET tooling sets up and uses to host the web application that you serve when you run `ojet serve`.
- Set up HTTPS on the Express instance that you've created. You specify the HTTPS protocol, identify the location of the self-signed certificate that you placed in the application directory, and specify a password.
- Pass the modified Express instance and the SSL-enabled server to the JET tooling so that `ojet serve` uses your configuration rather than the ready-to-use configuration provided by the Oracle JET tooling.
- To ensure that live reloads works when your web application is served to the HTTPS server, you'll also create an instance of the live reload server and configure it to use SSL.

If you can't use a certificate issued by a certificate authority, you can create your own certificate (a self-signed certificate). Tools such as OpenSSL, Keychain Access on Mac, and the Java Development Kit's `keytool` utility can be used to perform this task for you. For example, using the Git Bash shell that comes with Git for Windows, you can run the following command to create a self-signed certificate with the OpenSSL tool:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
-nodes
```

Once you've obtained the self-signed certificate that you want to use, install it in your application's directory. For example, place the two files generated by the previous command in your application's root directory:

```
...  
.gitignore  
cert.pem  
key.pem  
node_modules  
...
```

Once you have installed the self-signed certificates in your application, you configure the script for the `before_serve` hook point. To do this, open the `AppRootDir/scripts/hooks/before_serve.js` with your editor and configure it as described by the comments in the following example configuration.

```
'use strict';

module.exports = function (configObj) {
    return new Promise((resolve, reject) => {
        console.log("Running before_serve hook.");

        // Create an instance of Express, the Node.js web application
        // framework that Oracle
        // JET tooling uses to host the web applications that you serve
        // using ojet serve
        const express = require("express");

        // Set up HTTPS
        const fs = require("fs");
        const https = require("https");

        // Specify the self-signed certificates. In our example, these
        // files exist
        // in the root directory of our project.
        const key = fs.readFileSync("./key.pem");
        const cert = fs.readFileSync("./cert.pem");
        // If the self-signed certificate that you created or use requires
        // a
        // password, specify it here:
        const passphrase = "1234";

        const app = express();

        // Pass the modified Express instance and the SSL-enabled server to
        // the Oracle JET tooling
        configObj['express'] = app;
        configObj['urlPrefix'] = 'https';
        configObj['server'] = https.createServer({
            key: key,
            cert: cert,
            passphrase: passphrase
        }, app);
```

```
// Enable the Oracle JET live reload option using its default port
number so that
    // any changes you make to application code are immediately
reflected in the browser after you
    // serve it
const tinylr = require("tiny-lr");
const lrPort = "35729";

    // Configure the live reload server to also use SSL
    configObj["liveReloadServer"] = tinylr({ lrPort, key, cert,
passphrase });

    resolve(configObj);
});
};


```

Once you have completed these configuration steps, run the series of commands (`ojet build` and `ojet serve`, for example) that you typically run to build and serve your web application. As the certificate that you are using is a self-signed certificate rather than a certificate issued by a certificate authority, the browser that you use to access the web application displays a warning the first time that you access the web application. Acknowledge the warning and click the options that allow you to access your web application. On Google Chrome, for example, you click **Advanced** and **Proceed to localhost (unsafe)** if your web application is being served to `https://localhost:8000/`.

Once your web application opens in the browser, you'll see that the HTTPS protocol is used and an indicator that the connection is not secure, because you are not using a certificate from a certificate authority. You can also view the certificate information to confirm that it is the self-signed certificate that you created. In Google Chrome, click **Not secure** and **Certificate** to view the certificate information.



The `before_serve` hook point is one of a series of script hook points that you can use to customize the tooling workflow for Oracle JET applications. See [Customize the Web Application Tooling Workflow](#).

Serve a Web Application Using Path-based Routing

Oracle JET apps use parameter-based routing by default. With a couple of changes, you can use path-based routing instead.

With parameter-based routing, the URLs that appear in a user's browser may not be descriptive or easy to remember. For example, a JET app that uses the `navbar` template displays the following URLs in the browser when served locally:

- `http://localhost:8000/?ojr=dashboard`
- `http://localhost:8000/?ojr=incidents`

By way of contrast, the same application configured to use path-based routing uses the following URLs when the app displays the Dashboard or Incidents page:

- `http://localhost:8000/dashboard`
- `http://localhost:8000/incidents`

To implement this behavior, you need to configure the JET app so that when it receives a request from the client for a page, it rewrites the URL before it serves the request. Specifically, you'll need to do the following:

- Update the `appConfig.js` file so that your app uses path-based routing by creating an instance of the `UrlPathAdapter` class that takes the base URL from which the app is served as a parameter.
- Configure the `before_serve.js` hook to do the following:
 - Create an instance of Express to host the served web application.
Express is the Node.js web application framework that Oracle JET tooling sets up and uses to host the web application that you serve when you run `ojet serve`.
 - Write a `toFilePath(url)` function to inspect the path of the requested URL. If the request is for any of the typical file extensions (`.js`, `.ts`, and so on), the Express instance handles these requests while `toFilePath(url)` passes other requests to the app's `index.html` file for JET's `CoreRouter` to manage.
 - Pass the modified Express instance to the JET tooling so that `ojet serve` uses your configuration rather than the ready-to-use configuration provided by the Oracle JET tooling.

To update the `appConfig.js` file so that your app uses path-based routing, open the `AppRootDir/src/js/appController.js` with your editor and configure it as described by the comments in the following example configuration.

```
// Replace the entries that the JET tooling generates for 'ojs/ojurllibparamadapter' and UrlParamAdapter
// with entries for 'ojs/ojurllpathadapter' and UrlPathAdapter
define(... 'ojs/ojurllpathadapter', 'ojs/ojarraydataprovider', 'ojs/ojknockouttemplateutils', 'ojs/ojmodule-element', 'ojs/ojknockout',
    function(... UrlPathAdapter, ArrayDataProvider,
KnockoutTemplateUtils) {
    ...
let baseUrl = "/";
    let router = new CoreRouter(navData, {
        urlAdapter: new UrlPathAdapter(baseUrl)
    });
    router.sync();
    ...
}
```

To configure the script for the `before_serve` hook point, open the `AppRootDir/scripts/hooks/before_serve.js` with your editor and configure it as described by the comments in the following example configuration. Note that the following example configuration also includes entries to serve the application to a HTTPS server that uses a self-signed certificate. For information about creating the self-signed certificate

and placing it in your project, see [Serve a Web Application to a HTTPS Server Using a Self-signed Certificate](#).

```
"use strict";

module.exports = function (configObj) {

    /*
        Inspects the path of the requested
        URL. If the request is for any of the extensions (js, ts, and so
        on),
        the Express instance handles these requests while other requests are
        passed to the app's index.html file for the JET CoreRouter to
        manage.
    */
    function toFilePath(url) {
        const indexPath = "/index.html";
        const matchStaticFiles = url.match(/\/(js|ts|css|ui)\.*/);
        return matchStaticFiles ? matchStaticFiles[0] : indexPath;
    }

    return new Promise((resolve, reject) => {
        console.log("Running before_serve hook.");

        const express = require("express");

        /*
        Set up HTTPS for use.
        */
        const fs = require("fs");
        const https = require("https");
        const key = fs.readFileSync("./key.pem");
        const cert = fs.readFileSync("./cert.pem");
        const passphrase = "1234";

        /*
        Set up path info and url-rewrite rules so that the
        CoreRouter's urlPathAdapter will work.
        The path.join function assigns the returned path segment to
        the pathToWeb constant.
        */
        const path = require("path");
        const pathToWeb = path.join(__dirname, "../../web");
        const app = express();

        /*
        Gets anything after the slash ("/*") and sends it to the functions
        that we have defined to rewrite the URL (pathToWeb and the
        toFilePath(url) )
        */
        app.get("/*", (req, res) => {
            res.sendFile(path.join(pathToWeb, toFilePath(req.url)));
        });
    });
}
```

```
/*
Pass our modified Express instance and the SSL-enabled server
to the tooling configuration
*/
configObj['express'] = app;
configObj['urlPrefix'] = 'https';
configObj['server'] = https.createServer({
    key: key,
    cert: cert,
    passphrase: passphrase
}, app);

resolve(configObj);
});
};
```

The `before_serve` hook point is one of a series of script hook points that you can use to customize the tooling workflow for Oracle JET applications. See [Customize the Web Application Tooling Workflow](#).

Customize the Web Application Tooling Workflow

Hook points that Oracle JET tooling defines let you customize the behavior of the JET build and serve processes when you want to define new steps to execute during the tooling workflow using script code that you write.

When you create an application, Oracle JET tooling generates script templates in the `/scripts/hooks` application subfolder. To customize the Oracle JET tooling workflow you can edit the generated templates with the script code that you want the tooling to execute for specific hook points during the build and serve processes. If you do not create a custom hook point script, Oracle JET tooling ignores the script templates and follows the default workflow for the build and serve processes.

To customize the workflow for the build or serve processes, you edit the generated script template file named for a specific hook point. For example, to trigger a script at the start of the tooling's build process, you would edit the `before_build.js` script named for the hook point triggered before the build begins. That hook point is named `before_build`.

Therefore, customization of the build and serve processes that you enforce on Oracle JET tooling workflow requires that you know the following details before you can write a customization script.

- The type of application that you created: either a web application or a hybrid mobile application.
- The Oracle JET build or serve mode that you want to customize:
 - Debug — The default mode when you build or serve your application, which produces the source code in the built application.
 - Release — The mode when you build the application with the `--release` option, which produces minified and bundled code in a release-ready application.
- The appropriate hook point to trigger the customization.

- The location of the default hook script template to customize.

About the Script Hook Points for Web Applications

The Oracle JET hooks system defines various script trigger points, also called hook points, that allow you to customize the create, build, serve and restore workflow across the various command-line interface processes. Customization relies on script files and the script code that you want to trigger for a particular hook point.

The following table identifies the hook points and the workflow customizations they support in the Oracle JET tooling create, build, serve, and restore processes. Unless noted, hook points for the build and serve processes support both debug and release mode.

Hook Point	Supported Tooling Process	Description
after_app_create	create	This hook point triggers the script with the default name <code>after_app_create.js</code> immediately after the tooling concludes the create application process. A script for this hook point can be used with a web application or a hybrid mobile application.
after_app_restore	restore	This hook point triggers the script with the default name <code>after_app_restore.js</code> immediately after the tooling concludes the restore application process. A script for this hook point can be used with a web application or a hybrid mobile application.
before_build	build	This hook point triggers the script with the default name <code>before_build.js</code> immediately before the tooling initiates the build process. A script for this hook point can be used with a web application or a hybrid mobile application.
before_release_build	build (release mode only)	This hook point triggers the script with the default name <code>before_release_build.js</code> before the minification step and the requirejs bundling step occur. A script for this hook point can be used with a web application or a hybrid mobile application.
before_hybrid_build	build	This hook point triggers the script with the default name <code>before_hybrid_build.js</code> before the <code>cordovaPrepare</code> and <code>CordovaCompile</code> build steps occur. A script for this hook point can be used only with a hybrid mobile application.
before_app_type_script	build / serve	This hook point triggers the script with the default name <code>before_app_typescript.js</code> after the build process or serve process steps occur. Use the hook to update, add or remove TypeScript compiler options defined by your application's <code>tsconfig.json</code> compiler configuration file. The hook system passes your reference to the modified <code>tsconfig</code> object to the TypeScript compiler. A script for this hook point can only be used with a TypeScript application.

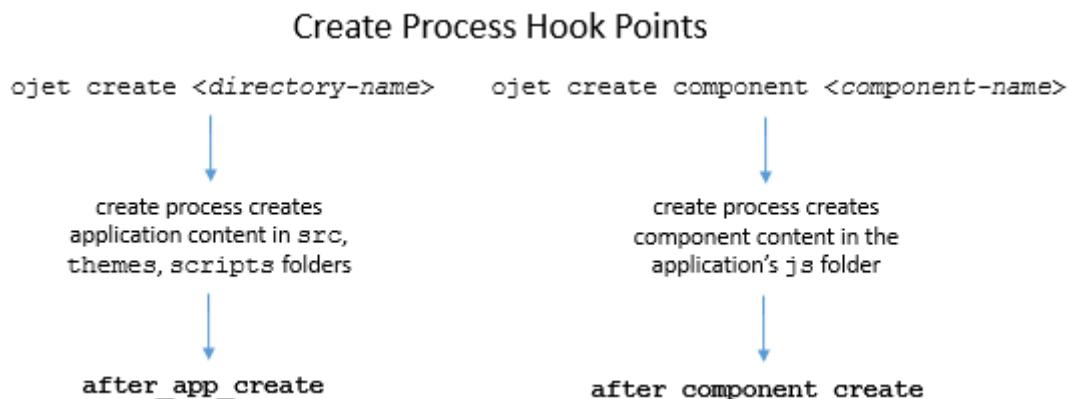
Hook Point	Supported Tooling Process	Description
after_app_types build / serve script		This hook point triggers the script with the default name <code>after_app_typescript.js</code> after the build process or serve process steps occur and immediately after the <code>before_app_typescript</code> hook point executes. This hook provides an entry point for applications that require further processing, such as compiling generated <code>.jsx</code> output using babel. A script for this hook point can only be used with a TypeScript application.
before_component build / serve _typescript		This hook point triggers the script with the default name <code>before_component_typescript.js</code> after the build process or serve process steps occur. Use the hook to update, add or remove TypeScript compiler options defined by your application's <code>tsconfig.json</code> compiler configuration file. The hook system passes your reference to the modified <code>tsconfig</code> object to the TypeScript compiler. A script for this hook point can only be used with a TypeScript application.
after_component build / serve _typescript		This hook point triggers the script with the default name <code>after_component_typescript.js</code> after the build process or serve process steps occur and immediately after the <code>before_component_typescript</code> hook point executes. This hook provides an entry point for applications that require further processing, such as compiling generated <code>.jsx</code> output using babel. A script for this hook point can only be used with a TypeScript application.
before_optimize build / serve (release mode only)		This hook point triggers the script with the default name <code>before_optimize.js</code> before the release mode build/serve process minifies the content. A script for this hook point can be used with a web application or a hybrid mobile application.
before_component build / serve _optimize		This hook point triggers the script with the default name <code>before_component_optimize.js</code> before the build/serve process minifies the content. A script for this hook point can be used to modify the build process specifically for a project that defines a Web Component.
after_build build		This hook point triggers the script with the default name <code>after_build.js</code> immediately after the tooling concludes the build process. A script for this hook point can be used with a web application or a hybrid mobile application.
after_component build _create		This hook point triggers the script with the default name <code>after_component_create.js</code> immediately after the tooling concludes the create Web Component process. A script for this hook point can be used to modify the build process specifically for a project that defines a Web Component.
after_component build (debug mode only) _build		This hook point triggers the script with the default name <code>after_component_build.js</code> immediately after the tooling concludes the Web Component build process. A script for this hook point can be used to modify the build process specifically for a project that defines a Web Component.

Hook Point	Supported Tooling Process	Description
before_serve	serve	This hook point triggers the script with the default name <code>before_serve.js</code> before the web serve process connects to and watches the application. In the case of hybrid builds, it also precedes the <code>cordovaClean</code> and <code>cordovaServe</code> step. A script for this hook point can be used with a web application or a hybrid mobile application.
after_serve	serve	This hook point triggers the script with the default name <code>after_serve.js</code> after all build process steps complete and the tooling serves the application. A script for this hook point can be used with a web application or a hybrid mobile application.
before_watch	serve	This hook point triggers the script with the default name <code>before_watch.js</code> after the tooling serves the application and before the tooling starts watching for application changes. A script for this hook point can be used with a web application or a hybrid mobile application.
after_watch	serve	This hook point triggers the script with the default name <code>after_watch.js</code> after the tooling starts the watch and after the tooling detects a change to the application. A script for this hook point can be used with a web application or a hybrid mobile application.

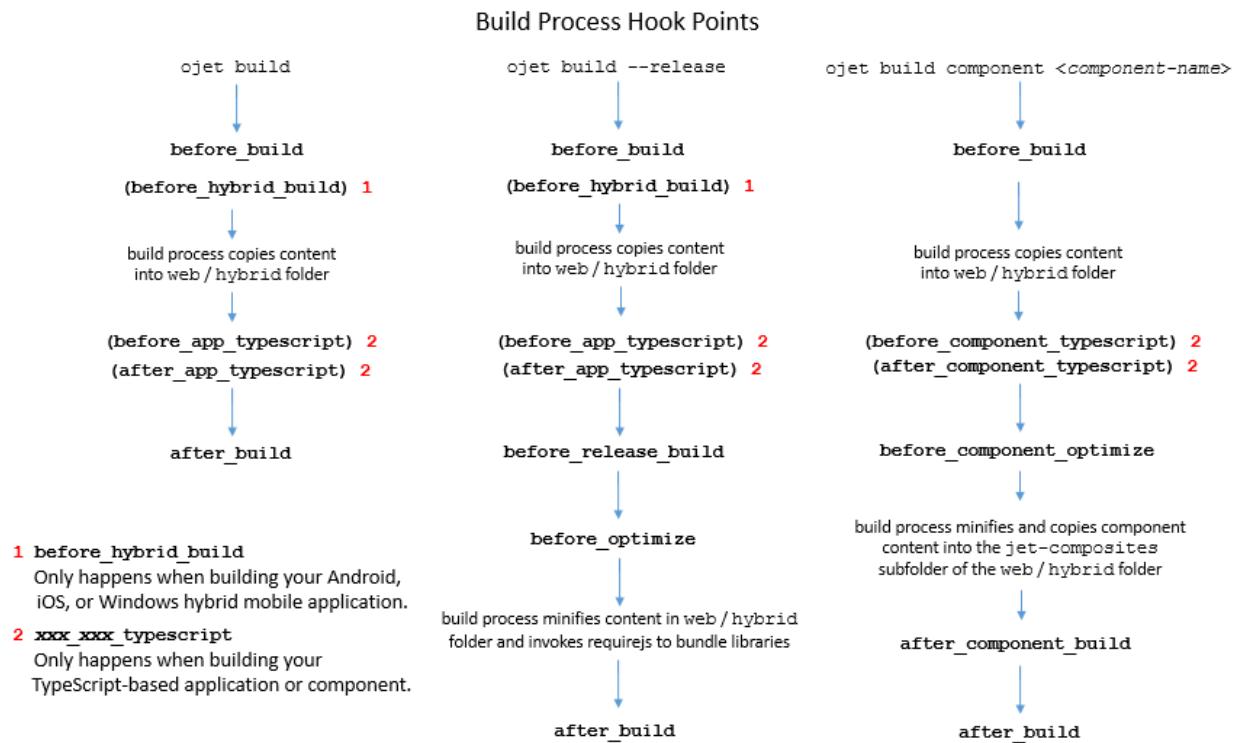
About the Process Flow of Script Hook Points

The Oracle JET hooks system defines various script trigger points, also called hook points, that allow you to customize the create, build, serve, and restore workflow across the various command-line interface processes.

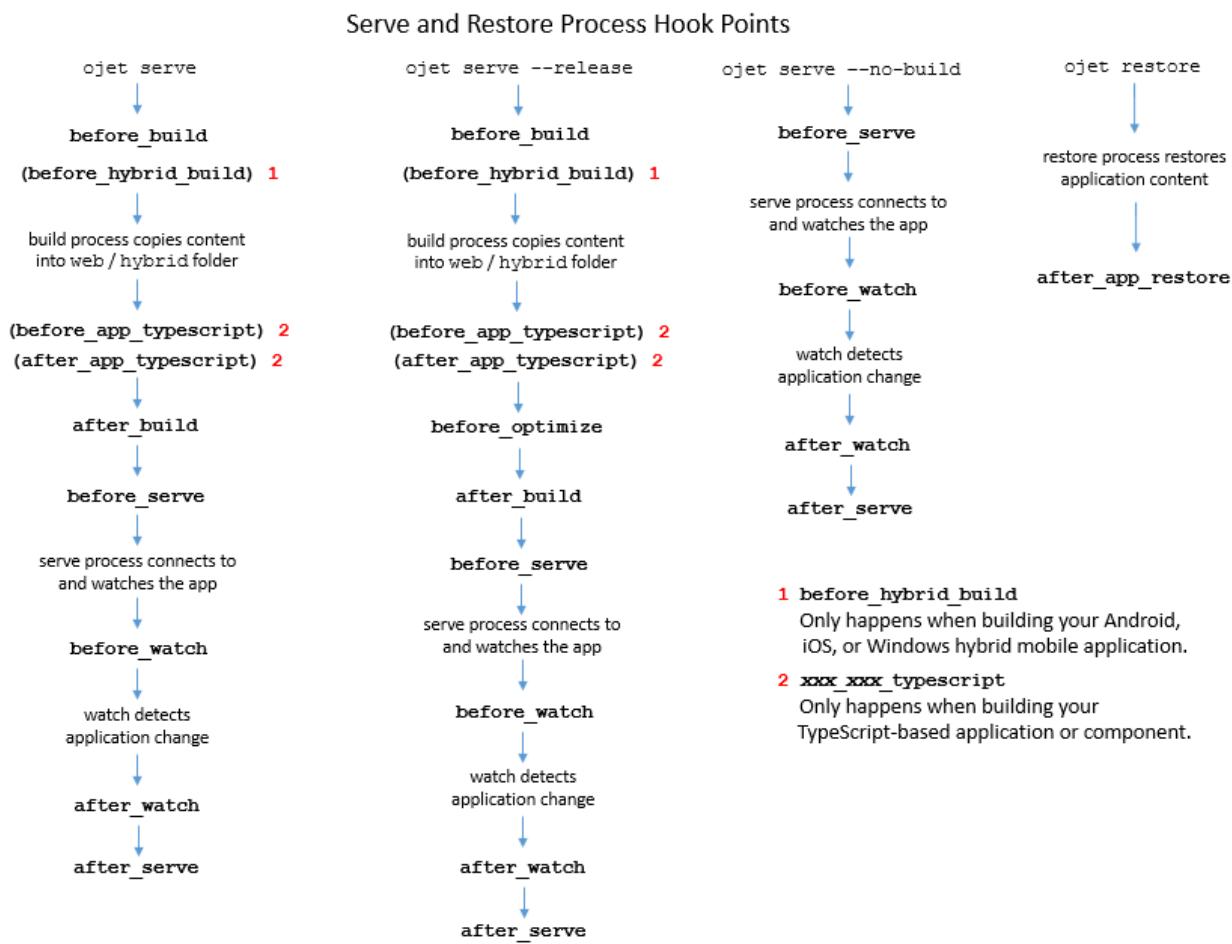
The following diagram shows the script hook point flow for the create process.



The following diagram shows the script hook point flow for the build process.



The following diagram shows the script hook point flow for the serve and restore processes.



Change the Hooks Subfolder Location

When you create an application, Oracle JET tooling generates script templates in the `/scripts/hooks` application subfolder. Your development effort may require you to relocate hook scripts to a common location, for example to support team development.

By default, the hooks system locates the scripts in the `hooks` subfolder using a generated JSON file (`hooks.json`) that specifies the script paths. When the tooling reaches the hook point, it executes the corresponding script which it locates using the `hooks.json` file. If you relocate hook script(s) to a common location, you must edit the `hooks.json` file to specify the new location for the hook scripts that you relocated, as illustrated by the following example.

```
{
  "description": "OJET-CLI hooks configuration file",
  "hooks": {
    "after_app_create": "scripts/hooks/after_app_create.js",
    ...
    "after_serve": "http://example.com/cdn/common/scripts/hooks/after_serve.js"
  }
}
```

Create a Hook Script for Web Applications

You can create a hook point script to define a new command-line interface process step for your web application. To create a hook script, you edit the hook script template associated with a specific hook point in the tooling build and serve workflow.

The Oracle JET hooks system defines various script trigger points, also called hook points, that allow you to customize the build and serve workflow across the various build and serve modes. Customization relies on script files and the script code that you want to trigger for a particular hook point. Note that the generated script templates that you modify with your script code are named for their corresponding hook point.

To customize the workflow for the build or serve processes, you edit the generated script template file named for a specific hook point. For example, to trigger a script at the start of the tooling's build process, you would edit the `before_build.js` script named for the hook point triggered before the build begins. That hook point is named `before_build`.

A basic example illustrates a simple customization using the `before_optimize` hook, which allows you to control the RequireJS properties shown in bold to modify the application's bundling configuration.

```
requirejs.config(
{
  baseUrl: "web/js",
  name: "main-temp",
  paths: {
    // injector:mainReleasePaths
    "knockout": "libs/knockout/knockout-3.x.x.debug",
    "jquery": "libs/jquery/jquery-3.x.x",
    "jqueryui-amd": "libs/jquery/jqueryui-amd-1.x.x",
    ...
  }
  // endinjector
  out: "web/js/main.js"
}
...
...
```

A script for this hook point might add one line to the `before_optimize` script template, as shown below. When your build the application with this customization script file in the default location, the tooling triggers the script before calling `requirejs.out()` and changes the `out` property setting to a custom directory path. The result is the application-generated `main.js` is output to the named directory instead of the default `web/js/main.js` location.

```
module.exports = function (configObj) {
  return new Promise((resolve, reject) => {
    console.log("Running before_optimize hook.");
    configObj.requirejs.out = 'myweb/js/main.js';
    resolve(configObj);
  });
};
```

 **Tip:**

If you want to change application path mappings, it is recommended to always edit the `path_mappings.json` file. An exception might be when you want application runtime path mappings to be different from the mappings used by the bundling process, then you might use a `before_optimize` hook script to change the `requirejs.config.paths` property.

The following example illustrates a more complex build customization using the `after_build` hook. This hook script adds a customize task after the build finishes.

```
'use strict';

const fs = require('fs');
const archiver = require('archiver');

module.exports = function (configObj) {
  return new Promise((resolve, reject) => {
    console.log("Running after_build hook.");

    //Set up the archive
    const output = fs.createWriteStream('my-archive.war');
    const archive = archiver('zip');

    //Callbacks for the archiver
    output.on('close', () => {
      console.log('Files were successfully archived.');
      resolve();
    });

    archive.on('warning', (error) => {
      console.warn(error);
    });

    archive.on('error', (error) => {
      reject(error);
    });

    //Archive the web folder and close the file
    archive.pipe(output);
    archive.directory('web', false);
    archive.finalize();
  });
};
```

In this example, assume the script templates reside in the default folder generated when you created the application. The goal is to package the application into a ZIP file. Because packaging occurs after the application build process completes, this script is triggered for the `after_build` hook point. For this hook point, the modified script template `after_build.js` will contain the script code to zip the application, and because the `.js` file resides in the default location, no hooks system configuration changes are required.

 **Tip:**

OJET tooling reports when hook points are executed in the message log for the build and serve process. You can examine the log in the console to understand the tooling workflow and determine exactly when the tooling triggers a hook point script.

Pass Arguments to a Hook Script for Web Applications

You can pass extra values to a hook script from the command-line interface when you build or serve the web application. The hook script that you create can use these values to perform some workflow action, such as creating an archive file from the contents of the web folder.

You can add the `--user-options` flag to the command-line interface for Oracle JET to define user input for the hook system when you build or serve the web application. The `--user-options` flag can be appended to the build or serve commands and takes as arguments one or more space-separated, string values:

```
ojet build --user-options="some string1" "some string2" "some stringx"
```

For example, you might write a hook script that archives a copy of the build output after the build finishes. The developer might pass the user-defined parameter `archive-file` set to the archive file name by using the `--user-options` flag on the Oracle JET command line.

```
ojet build web --user-options="archive-file=deploy.zip"
```

If the flag is appended and the appropriate input is passed, the hook script code may write a ZIP file to the `/deploy` directory in the root of the project. The following example illustrates this build customization using the `after_build` hook. The script code parses the user input for the value of the user defined `archive-file` flag with a promise to archive the application after the build finishes by calling the NodeJS function `fs.createWriteStream()`. This hook script is an example of taking user input from the command-line interface and processing it to achieve a build workflow customization.

```
'use strict';
const fs = require('fs');
const archiver = require('archiver');
const path = require('path');

module.exports = function (configObj) {
    return new Promise((resolve, reject) => {
        console.log("Running after_build hook.");

        //Check to see if the user set the flag
        //In this case we're only expecting one possible user defined
        //argument so the parsing can be simple
        const options = configObj.userOptions;
        if (options){
```

```
const userArgs = options.split('=');
if (userArgs.length > 1 && userArgs[0] === 'archive-file'){
    const deployRoot = 'deploy';
    const outputArchive = path.join(deployRoot,userArgs[1]);

    //Ensure the output folder exists
    if (!fs.existsSync(deployRoot)) {
        fs.mkdirSync(deployRoot);
    }

    //Set up the archive
    const output = fs.createWriteStream(outputArchive);
    const archive = archiver('zip');

    //callbacks for the archiver
    output.on('close', () => {
        console.log(`Archive file ${outputArchive} successfully
created.`);
        resolve();
    });

    archive.on('error', (error) => {
        console.error(`Error creating archive ${outputArchive}`);
        reject(error);
    });

    //Archive the web folder and close the file
    archive.pipe(output);
    archive.directory('web', false);
    archive.finalize();
}
else {
    //Unexpected input - fail with information message
    reject(`Unexpected flags in user-options: ${options}`);
}
}
else {
    //nothing to do
    resolve();
}
});
```

3

Understanding the Hybrid Mobile Application Workflow

Oracle JET includes support for hybrid mobile application development using Apache Cordova. The toolkit provides iOS, Android, and Windows Alta themes and UI behavior on Oracle JET components, starter applications, design patterns, and tooling support.

Before you can create your first hybrid mobile application, you should become familiar with the Oracle JET mobile features and third-party technologies. You must also install the prerequisite packages and Oracle JET mobile tools.

Topics

- [Install the Mobile Tooling](#)
- [Create a Hybrid Mobile Application Using the Oracle JET Command-Line Interface](#)
- [Use Cordova Plugins to Access Mobile Device Services](#)

Once you have created a hybrid mobile application, added Cordova plugins, and familiarized yourself with Oracle JET's mobile features, refer to the following information for details about other tasks that you need to do to complete your hybrid mobile application.

- [About Securing Hybrid Mobile Applications](#)
- [Test Hybrid Mobile Applications](#)
- [Debug Hybrid Mobile Applications](#)
- [Package and Publish Hybrid Mobile Applications](#)

Tip:

If you're strictly interested in developing web applications that run in desktop and mobile browsers, you don't need to install the mobile tooling. For information on developing web applications, see [Understanding the Web Application Workflow](#)

Install the Mobile Tooling

To create Oracle JET hybrid mobile applications, you must first install Cordova on your development machine. To develop hybrid mobile applications for Android, iOS, or Windows, you must also install the Android, iOS, or Windows development tools.

To install the mobile tooling:

If needed, install the tooling prerequisites as described in [Install Oracle JET Tooling](#).

1. [Install Apache Cordova](#)

2. (Optional) [Install Android Development Tools](#)
3. (Optional) [Install iOS Development Tools](#)
4. (Optional) [Install Windows Development Tools](#)

Install Apache Cordova

Install Apache Cordova on your development machine.

Oracle JET uses plugins developed with the Apache Cordova framework to access the capabilities of the devices on which your JET mobile application is installed such as, for example, camera and GPS. JET also uses Cordova to build and serve hybrid mobile applications by invoking the Cordova command-line interface that you install with Cordova.

- As Administrator on Windows or using sudo on Macintosh and Linux systems, enter the following command at a terminal prompt to install Cordova:

```
[sudo] npm install -g cordova
```

For additional information about Cordova, see [Overview](#) in Cordova's documentation.

Install Android Development Tools

Install the Android SDK to deploy a JET hybrid mobile application to Android devices.

The Android SDK provides the tools that build and package your application into an .APK file (the file type that installs applications on Android devices), an emulator to create Android Virtual Devices (AVD) where you can test your application if you do not have access to a physical Android device, and an OEM USB driver to connect your development machine to a physical Android device through a USB cable if you do have a device. This last option enables you to deploy an application from your development machine to the Android device.

Android Studio, Google's IDE for Android development, includes the Android SDK in its installation and provides wizard options that simplify the management of the SDK platforms and tools that you need.

Install Android Studio, and the Android SDK that it includes, by downloading the installation file from <https://developer.android.com/studio/index.html>. The Android Developer's website provides installation instructions for Windows, Mac, and Linux. See <https://developer.android.com/studio/install.html>.

Once you have installed Android Studio, perform the tasks described in the following topics:

- [Install an Emulator Accelerator](#)
- [Create an Android Virtual Device](#)
- [Set Up Your Android Device to Install an App from Your Development Machine](#)
- [Install Gradle and Configure Gradle Proxy Settings](#)
- [Configure Environment Variables to Reference JDK and Android SDK Installations](#)

Android requires that all .APK files be digitally signed with a certificate before they can be installed. For apps in development, the Android SDK automatically creates a debug keystore and certificate and sets the keystore and key passwords the first

time that you build an .APK file. On macOS and Unix, it creates these resources in `$HOME/.android/debug.keystore`. On Windows, they are in the directory referenced by the `%USERPROFILE%` variable (for example, `c:\Users\JDOE\.android`).

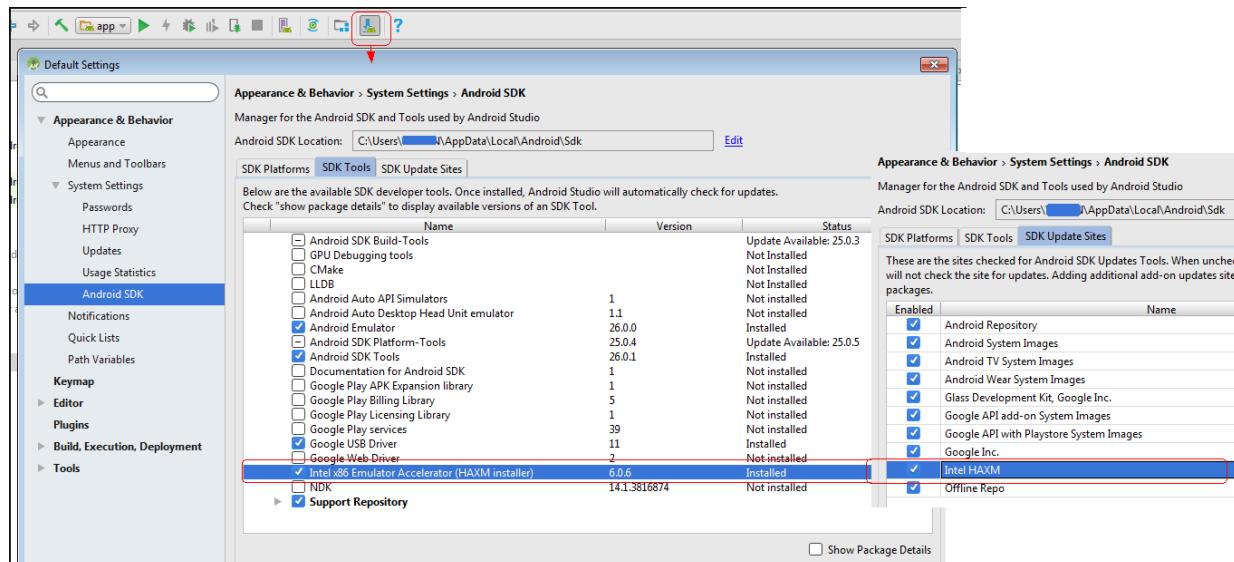
When you have completed development and want to sign and package the Android app for distribution, see [Package a Hybrid Mobile App on Android](#).

Install an Emulator Accelerator

You can accelerate the performance of the emulator on Intel x86-based development machines by installing an emulator accelerator.

The emulator accelerator speeds up the performance of the emulator and the AVDs that it emulates by allocating additional resources from your development machine. A standard installation of Android Studio includes the emulator accelerator ready-to-use. If you did not perform a standard installation of Android Studio, you can install the emulator accelerator separately, as described below. Once installed, the accelerator appears in the SDK Tools list of the SDK Manager, as shown in the image below. Use Android Studio's **Tools** and **SDK Manager** menu options to open the SDK Manager or the icon that appears on the Android Studio toolbar.

If this is the first time that you use Android Studio, click the **Start a new Android Studio project** link on the Welcome screen of Android Studio. Once you create the project, you'll have access to the aforementioned menu options.



Make sure that the update site for the emulator accelerator that you want to download is selected in the **SDK Update Sites** tab as shown. Once downloaded, execute the installer. See <https://developer.android.com/studio/run/emulator-acceleration.html#accel-vm>.

Create an Android Virtual Device

An Android Virtual Device (AVD) replicates an Android device on your development computer. It is a useful option for testing, especially if you only have access to one or a limited range of physical Android devices.

The AVD Manager that you launch from Android Studio by clicking **Tools** and **AVD Manager** has a range of ready-to-use virtual devices, mostly of those devices developed by Google itself, such as the Nexus and Pixel XL range. Other Android device vendors, such as Samsung, provide specifications on their websites that you can use to create the AVD yourself.

Google maintains documentation describing how to manage AVDs. See <https://developer.android.com/studio/run/managing-avds.html>.

If this is the first time that you use Android Studio, click the **Start a new Android Studio project link** on the Welcome screen. Once you create the project, you'll have access to the Android Studio menu options described in the following procedure.

To create an AVD:

1. In Android Studio, launch the Android Virtual Device Manager by selecting **Tools** > **AVD Manager**.
2. In the Your Virtual Devices screen, click **Create Virtual Device**.
3. In the Select Hardware screen, select a phone device, such as Pixel, and then click **Next**.
4. In the System Image screen, click **Download** for one of the recommended system images. Agree to the terms to complete the download.
5. After the download completes, select the system image from the list and click **Next**.
6. On the next screen, leave all the configuration settings unchanged and click **Finish**.
7. In the Your Virtual Devices screen, select the device you just created and click **Launch this AVD in the emulator**.

Set Up Your Android Device to Install an App from Your Development Machine

You can install your app directly from your development machine to your Android device by configuring the Android device and connecting it to your development machine using a USB cable.

If you are developing on Windows, you might need to install the appropriate USB driver for your device. For help installing drivers, see the document [OEM USB Drivers](#).

To set up your Android device:

1. Connect your device to your development machine with a USB cable.
2. On Android 4.2 and newer, go to **Settings** > **About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.
3. On earlier versions of Android (pre-4.2), enable USB debugging on your device by going to **Settings** > **Developer options**.

Install Gradle and Configure Gradle Proxy Settings

Gradle is the tool that the Android SDK invokes to build the apps that you deploy to your AVD or Android device.

Prior to the release of cordova-android@6.4.0, you could use the instance of Gradle that Android Studio installed to build the apps you deploy. With the cordova-

android@6.4.0 release (and later), you must install Gradle separately from Android Studio. For information about how to do this, see [Gradle's documentation](#).

Once you have installed Gradle, you must configure proxy settings if you work inside a corporate network. To do this, create a `gradle.properties` file in your `~/.gradle` directory that includes the following properties so that you can successfully serve your app to the AVD or physical device.

```
systemProp.http.proxyHost=www-proxy.server.url.com
systemProp.http.proxyPort=proxy-port-number
systemProp.https.proxyHost=www-proxy.server.url.com
systemProp.https.proxyPort=proxy-port-number
```

If you are a Windows user, you locate your `~/.gradle` directory by navigating to `%USERPROFILE%\.gradle`. If you are a macOS or Linux user, navigate to `~/.gradle`.

Configure Environment Variables to Reference JDK and Android SDK Installations

To function correctly, the Cordova CLI that the Oracle JET CLI communicates with requires that you configure certain environment variables.

The Cordova CLI attempts to set these environment variables for you, but in certain cases you may need to set them manually. In any event, it is a good idea to review and confirm the values of the following environment variables before you start development:

- Set `JAVA_HOME` to the location of your JDK installation

For example, on Windows, set `JAVA_HOME` to a value similar to `C:\Program Files\Java\jdk1.8.0_171`.

- Set `ANDROID_HOME` to the location of your Android SDK installation

For example, on Windows set `ANDROID_HOME` to a value similar to `C:\Users\John\AppData\Local\Android\Sdk`. You can obtain the Android SDK location from the Android SDK page of the Default Settings dialog that you access by selecting **Tools > SDK Manager** in Android Studio.

We also recommend that you add the Android SDK's `tools`, `tools/bin`, and `platform-tools` directories to your `PATH` environment variable.

The process to set the environment variables depends upon your development environment:

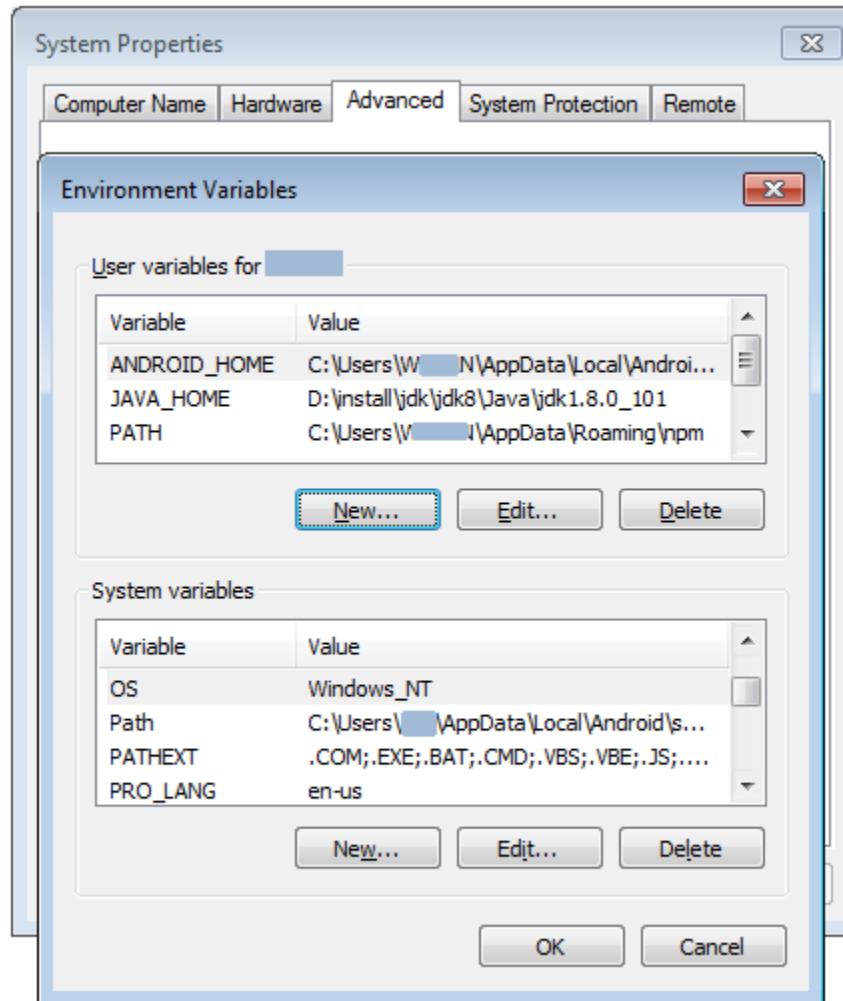
- Windows

Set the variables permanently in **Control Panel > System and Security > System > Advanced System Settings > Environment Variables**.

For example, to include the Android SDK's `tools` and `platform-tools` directories in your `PATH` environment variable, modify the entry that the Environment Variables dialog displays to include entries similar to the following:

```
Path=C:\Users\JET\AppData\Local\Android\sdk\tools;C:\Users\JET\AppData\Local\Android\sdk\platform-tools;C:\Users\JET\AppData\Roaming\npm
```

To set the `JAVA_HOME` and `ANDROID_HOME` environment variables, click **New** in the Environment Variables dialog and set the values of the variables to the location of the JDK and Android SDK locations, as shown in the following image.



If you have opened a command window, close it and open a new one for the change to take effect.

- Macintosh and Linux

On Mac and Linux systems, the settings depend upon your default shell. For example, on a Mac system using the Bash shell, add the following lines in `~/.bash_profile`:

```
export JAVA_HOME=/Development/jdkInstallDir/  
export ANDROID_HOME=/Development/android-sdk/
```

To update your PATH environment variable, add a line resembling the following where you substitute the paths with your local Android SDK installation's location:

```
export PATH=${PATH}:/Development/android-sdk/platform-tools:/Development/  
android-sdk/tools
```

Reload your terminal to see this change reflected or run the following command:

```
$ source ~/.bash_profile
```

On a Linux system using C Shell, you would use the following format:

```
setenv JAVA_HOME=/Development/jdkInstallDir/  
setenv ANDROID_HOME=/Development/android-sdk/
```

Reload your terminal to see this change reflected or run the following command:

```
~/.cshrc
```

Install iOS Development Tools

To develop applications for the iOS platform, you must install the Xcode development environment from the App Store which is only available on macOS.

After installing Xcode, execute the following command in a terminal window to install the Xcode command-line tools:

```
xcode-select --install
```

These steps are sufficient for developing iOS applications and testing on iOS simulators. If, however, you want to use an actual iOS device for testing or to publish your application to the App Store, you must join the Apple iOS Developer program and create an appropriate provisioning profile. For additional information, see [Manage profiles](#) in Apple's documentation. For details about using the Oracle JET tooling to package and deploy your hybrid mobile application, see [Package a Hybrid Mobile App on iOS](#).

In addition you must install a tool that is used by the Oracle JET CLI to deploy apps to iOS devices by executing the following command in a terminal window:

```
sudo npm install -g ios-deploy --unsafe-perm=true --allow-root
```

Once you have installed the iOS development tools, you can create a hybrid mobile application, as described in [Create a Hybrid Mobile Application Using the Oracle JET Command-Line Interface](#).

Install Windows Development Tools

JET hybrid mobile apps for Windows devices must be developed on computers that run the Windows 10 operating system.

JET supports the creation of hybrid mobile apps that are Universal Windows Platform (UWP) apps. UWP is the platform for Windows 10. For more information, see [What's a Universal Windows Platform \(UWP\) app?](#) in Microsoft's documentation.

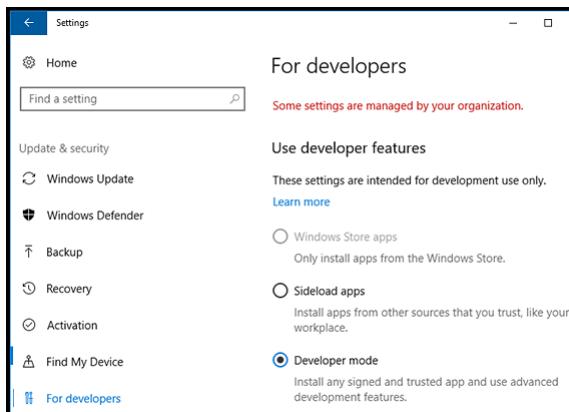
Enable Developer Mode on Windows 10

Developer mode allows you to install any signed and trusted apps to your Windows development machine, whereas the default Windows 10 setting only permits the installation of apps from the Windows Store.

1. From the **For developers** settings dialog, choose **Developer mode**.
2. Read the disclaimer in the dialog that appears and click Yes to accept the change.

 **Note:**

If your device is owned by an organization, some options might be disabled by your organization. Contact your system administrator to ensure you have the options enabled that you require.

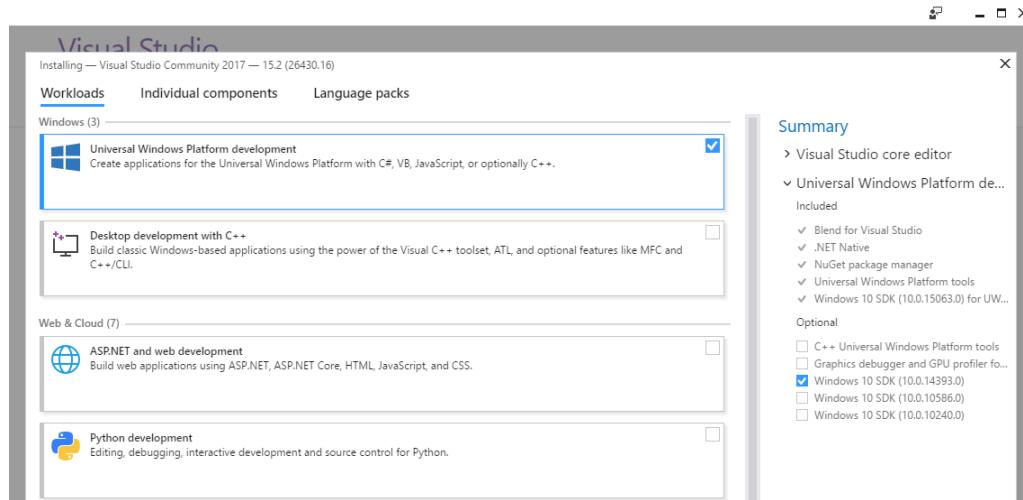


For information about Developer mode, and other settings that configure development on Windows 10 computers, see <https://docs.microsoft.com/en-us/windows/uwp/get-started/enable-your-device-for-development>.

Install Visual Studio

Install Visual Studio as it includes the Windows 10 SDK and the Universal Windows App Development Tools that you require to develop and deploy a JET hybrid mobile application on your computer.

Microsoft offers a number of editions of Visual Studio. The Visual Studio Community edition is a free, fully-featured IDE for students, open-source and individual developers. Visual Studio can be downloaded from <https://www.visualstudio.com/vs/>. During download and installation, select the **Universal Windows Platform development** workload and the latest version of the Windows 10 SDK that appears under the Optional list of items that you can install with the workload, as shown in the following image.



Run Visual Studio after you install it to execute any required setup tasks. This prompts you to enable Developer mode, if you have not already enabled this mode on your computer.

Configure Environment Variable to Reference Microsoft Build Tools

After you install Visual Studio 2017, create a system environment variable (`MSBUILDDIR`) that references the location of the MSBuild Tools within the Visual Studio 2017 installation.

1. On your Windows 10 computer, in the input field for the Windows Cortana virtual assistant, enter `Edit the system environment variables` and select it when Cortana returns it as the best match.
2. In the System Properties dialog that opens, click **Environment Variables**, then click **New** in the System variables section of the Environment Variables dialog.
3. In the Variable name field, enter `MSBUILDDIR`, and in the Variable value field, enter the directory location of the MSBuild Tools which is typically `C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\MSBuild\15.0\Bin` in a default Visual Studio 2017 installation.
4. Click **OK** to close the open dialogs.

Install a Personal Information Exchange File in Your Computer's Certificate Store

Install a personal information exchange file (.PFX) in certificate stores on your Windows 10 computer. The PFX file packages a private key file and a certificate file that you use to sign your app.

This is a one-time task that, once complete, enables you to build and serve JET hybrid mobile applications in debug mode on your Windows 10 computer. You also require a PFX file to sign your JET hybrid mobile application when you complete development and want to distribute it.

The following procedure describes how to create PVK and CER files without password protection that you package into a PVK file that is also without password protection.

For more information about the switches and arguments supported by the Microsoft tools that you use to create and package a PFX file, see the following Microsoft documentation:

- [MakeCert](#)
- [Pvk2Pfx](#)

For more information about packaging a completed JET hybrid app for Windows, see [Package a Hybrid Mobile App on Windows](#).

To create and install a PFX file:

1. At a command prompt with Administrator privileges, enter the following commands:
 - a. Change to the directory that contains the Windows SDK. For example:

```
chdir "C:\Program Files (x86)\Windows Kits\10\bin\x64"
```
 - b. Use the MakeCert tool to create certificate files with the specified parameters. For example:

```
makecert.exe -sv C:\aDirectory\doc.pvk -n "CN=Doc Example,OU=JET,O=Oracle,C=US" -r -h 0 -eku "1.3.6.1.5.5.7.3.3,1.3.6.1.4.1.311.10.3.13"
C:\aDirectory\doc.cer
```
 - c. Package the resulting certificate files into a PFX file. For example:

```
pvk2pfx.exe -pvk C:\aDirectory\doc.pvk -spc C:\aDirectory\doc.cer -pfx C:\aDirectory\doc.pfx
```

This creates a PFX file named doc.pfx in the C:\aDirectory\ directory.

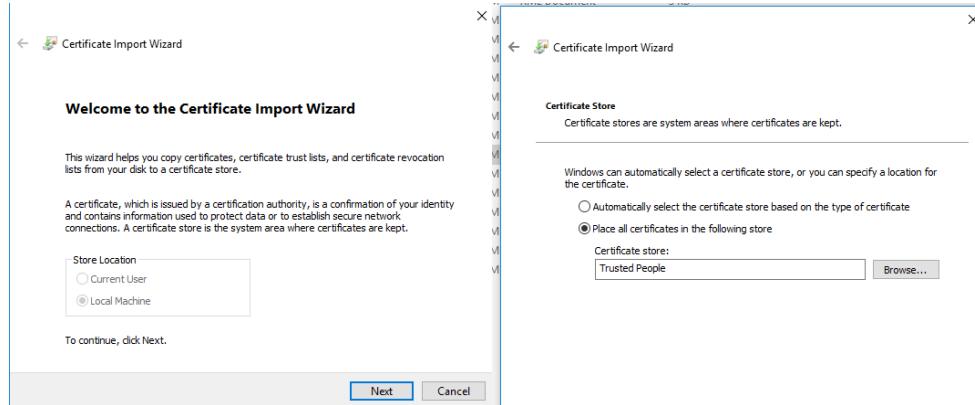
2. Install the PFX file in the following Certificate Stores on your computer

Store Location	Certificate Store
Current User	Personal
Current User	Trusted People
Local Machine	Trusted People

3. To do this, double-click the PFX file to invoke the Certificate Import Wizard, choose the appropriate store location (for example, Current User) and click **Next** to proceed to the page where you choose the Certificate Store, as shown by the following figure where the Trusted People certificate store is selected.

 **Note:**

The file created in our previous example is not password-protected, so leave the Password input field blank in the Private key protection page of the wizard.



4. Click **Finish** to complete the installation of the certificate in the certificate store and repeat the process to install the PFX file in the remaining certificate stores.

Create a Hybrid Mobile Application Using the Oracle JET Command-Line Interface

Use the Oracle JET mobile tooling commands to create, build, run, and customize hybrid mobile applications for Android, iOS, and Windows mobile devices. You can create an application that contains a blank template or one pre-configured with layouts and styling for the desired platform.

Before you use the mobile tooling, verify that you have installed all the prerequisite packages and configured your target platforms if needed. For additional information, see [Install the Mobile Tooling](#).

1. [Scaffold a Hybrid Mobile Application](#)
2. [\(Optional\) Build a Hybrid Mobile Application](#)
3. [Serve a Hybrid Mobile Application](#)
4. [Review Your Application Settings in the config.xml File](#)
5. [Change the Splash Screen and App Launcher Icon](#)
6. [\(Optional\) Customize the Hybrid Mobile Application Tooling Workflow](#)

Note:

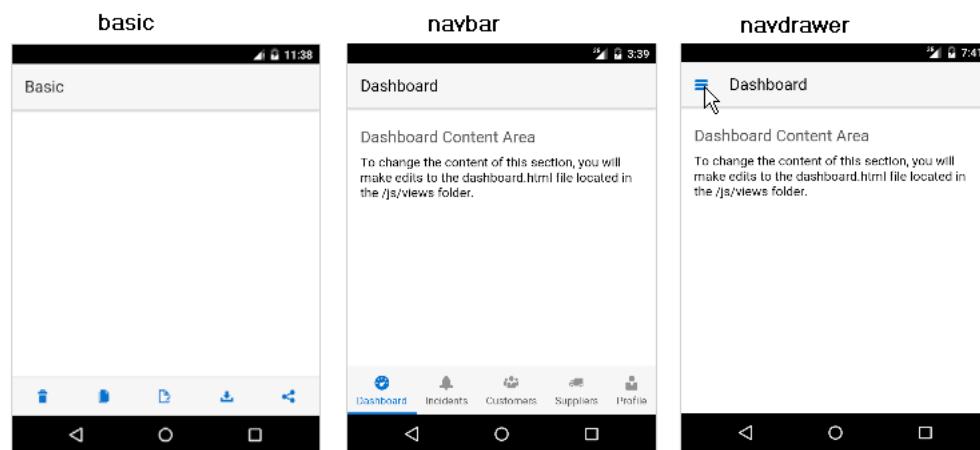
You can add hybrid platforms (Android, iOS, and Windows) to an existing web application. See [Add Hybrid Mobile Features to Web Applications](#).

Scaffold a Hybrid Mobile Application

Use the Oracle JET CLI to scaffold a hybrid mobile application for iOS, Android, and Windows mobile devices. You can scaffold the application to contain a blank template or a Starter Template pre-configured with layout styles, a navigation bar, or a navigation drawer. Additionally, each Starter Template supports TypeScript

development should you wish to create your hybrid mobile application in TypeScript. After scaffolding, you can modify the app as desired.

The following image shows the differences between the templates as shown on an Android Nexus 4 emulator using the Android Alta theme. The blank template contains `index.html` but no UI features, and is not shown in the image that follows. The basic template adds styling but no sample content, and can be used as the basis for your app if you have no navigational requirements or if you want to design the entire app yourself. The navbar and navdrawer templates contain sample content and best practices for layouts and styling that you can also modify as needed. The same app will run on any Android, iOS, or Windows emulator or device, using the appropriate Alta theme for the platform.



1. Open a terminal window and change to your development directory.
2. At a command prompt, enter `ojet create --hybrid` with optional arguments to create the Oracle JET application and scaffolding.

```
ojet create [directory] --hybrid
  [--appid=application-id] [--appname=application-name]
  [--template={template-name:[web|hybrid]|template-url|
  template-file}]
  [--platforms=android,ios,windows|--platform=android|ios|
  windows]
  [--typescript]
```

For example, to create a hybrid mobile application named `Sample NavBar` in the `MySampleApp` directory using the `navbar` template and targeted for Android devices, enter the following command:

```
ojet create MySampleApp --hybrid --appname="Sample NavBar" --
  template=navbar --platform=android
```

To scaffold the same hybrid mobile application but with support for TypeScript development, add the `--typescript` argument to the command:

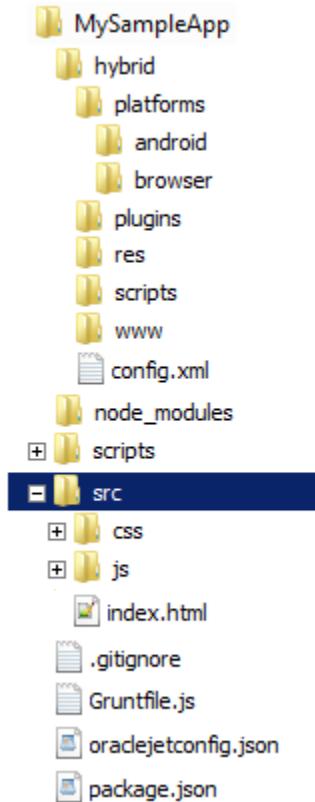
```
ojet create MySampleApp --hybrid --appname="Sample NavBar" --template=navbar --platform=android --typescript
```

 **Tip:**

You can also enter `ojet help create` at a terminal prompt to get help with the Oracle JET CLI create arguments.

The scaffolding will take some time to complete. When successful, the terminal will display: Your app is ready! Change to your new app directory `MySampleApp...`.

The new application will have a directory structure similar to the one shown in the following image.



The application folders contain the application and configuration files that you will modify as needed for your own application. See [About the Hybrid Mobile Application File Structure](#).

About ojet create Command Options for Mobile Hybrid Applications

Use `ojet create` with optional arguments to create the Oracle JET hybrid mobile application and scaffolding.

The following table describes the available options and provides examples for their use.

Option	Description
<code>directory</code>	Application location. If not specified, the application is created in the current directory.
<code>appid</code>	Application ID entered in reverse domain style: <code>com.mydomain.myappname</code> . The appid is equivalent to Package Name on Android and Bundle ID on iOS. If not specified, the appid defaults to <code>org.oraclejet.directory</code> , using the directory you specified in the scaffolding command. For example, if you specified <code>MyApp</code> for the directory, the default appid will be <code>org.oraclejet.MyApp</code> .
<code>appname</code>	Application name displayed on device. To include spaces in the title, use quotes: <code>--appname="My Sample Application"</code> . If not specified, the appid defaults to the directory you specified in the scaffolding command.
<code>template</code>	Template to use for the application. Specify one of the following: <ul style="list-style-type: none"> <code>template-name</code> Predefined template. You can specify basic, blank, navbar , or navdrawer. If you don't specify a template, your application will be configured with the blank template. <code>template-URL</code> URL to zip file containing the name of the zipped application: <code>http://path-to-app/app-name.zip</code>. <code>template-file</code> Path to zip file on your local file system containing the name of a zipped application: "<code>path-to-app/app-name.zip</code>". For example: <pre>--template="C:\Users\SomeUser\app.zip" --template="/home/users/SomeUser/app.zip" --template("~/projects/app.zip")</pre> If the <code>src</code> folder is present in the zip file, then all content will be placed under the <code>src</code> directory of the application, except for the <code>script</code> folder which remains in the root. If no <code>src</code> folder is present, the contents of the zip file will be placed at the root of the new application.
<code>platforms</code>	Comma-separated list of platform names. You can specify one or more of the following: ios, android, or windows without spaces. For example: <code>ios,android</code> . If you don't specify a platform, the command will prompt you for your choice of platform after verifying that the necessary prerequisites are available.
<code>platform</code>	Platform name. You can specify one of the following: ios, android, or windows. If you don't specify a platform, the command will prompt you for your choice of platform after verifying that the necessary prerequisites are available.

About the Hybrid Mobile Application File Structure

The Oracle JET scaffolding process creates files and folders that you modify as needed for your application.

Directory or File	Description
hybrid	Contains platform-specific files that are merged in with a copy of the application files at build time. Most of the files in this directory are staging files that you should not commit to your source control system. One exception is the <code>hybrid/config.xml</code> file.
hybrid/config.xml	<p>Contains the Cordova global configuration settings. You can edit the <code>config.xml</code> file to specify core Cordova API features, plugins, and platform-specific settings.</p> <p>For example, the following entry sets the application's orientation to landscape-only on all platforms.</p>
	<pre data-bbox="659 751 1437 910"><?xml version='1.0' encoding='utf-8'?> <widget ... <preference name="Orientation" value="landscape" /> </widget></pre>
	<p>Commit the <code>config.xml</code> file to your source control system, if you use one. For more information about the <code>config.xml</code> file, see Review Your Application Settings in the config.xml File.</p>
scripts	Contains template hook scripts that you can modify to define new build and serve steps for your application. See Customize the Hybrid Mobile Application Tooling Workflow .
node_modules	Contains the Node.js modules used by the tooling.
src	Site root for your application. Contains the application files that you can modify as needed for your own application and should be committed to source control.
	The content will vary, depending upon your choice of template. Each template, even the blank one, will contain an <code>index.html</code> file and a <code>main.js</code> RequireJS bootstrap file.
	Other templates may contain view templates and viewModel scripts pre-populated with content. For example, if you specified the navbar template during creation, the <code>js/views</code> and <code>js/viewModels</code> folders will contain the templates and scripts for a hybrid mobile application that uses a nav bar for navigation.
.gitignore	Defines rules for application folders to ignore when using a GIT repository to check in application source. Users who do not use a GIT repository can use <code>ojet strip</code> to avoid checking in content that Oracle JET always regenerates. Note this file must not be deleted since the <code>ojet strip</code> command depends on it.
oraclejetconfig.json	Contains the default source and staging file paths that you can modify if you need to change your application's file structure.
package.json	Defines npm dependencies and project metadata.

Modify the Hybrid Mobile Application's File Structure

You can modify your scaffolded application's file structure if the default structure doesn't meet your needs.

The `oraclejetconfig.json` file in your application's top level directory contains the default source and staging file paths that you can modify.

```
{
  "paths": {
    "source": {
      "common": "src",
      "web": "src-web",
      "hybrid": "src-hybrid",
      "javascript": "js",
      "styles": "css",
      "themes": "themes"
    },
    "staging": {
      "web": "web",
      "hybrid": "hybrid",
      "themes": "themes"
    }
  },
  "serveToBrowser": "chrome",
  "generatorVersion": "9.2.0"
}
```

To change the hybrid mobile application's file structure:

1. In your application's top level directory, open `oraclejetconfig.json` for editing.
2. In `oraclejetconfig.json`, change the paths as needed and save the file.

For example, if you want to change the default `styles` path from `css` to `app-css`, edit the following line in `oraclejetconfig.json`:

```
"styles": "app-css"
```

3. Rename the directories as needed for your application, making sure to change only the paths listed in `oraclejetconfig.json`. For example, if you changed `styles` to `app-css` in `oraclejetconfig.json`, change the application's `css` directory to `app-css`.
4. Update your application files as needed to reference the changed path

For example, if you modify the path to the CSS for your application, update the links appropriately in your application's `index.html`.

```
<link rel="icon" href="app-css/images/favicon.ico" type="image/x-icon" />

<!-- This is the main css file for the default Alta theme -->
<!-- injector:theme -->
<link rel="stylesheet" href="app-css/libs/oj/v9.2.0/alta/oj-
```

```

alta-min.css" type="text/css" />
<!-- endinjector -->

<!-- This contains icon fonts used by the starter template -->
<link rel="stylesheet" href="app-css/demo-alta-site-min.css"
type="text/css" />

<!-- This is where you would add any app specific styling -->
<link rel="stylesheet" href="app-css/override.css" type="text/
css" />

```

To use the new paths, in your application's top level directory, build your application using the appropriate command-line arguments.

Add Web Browser Capability to Hybrid Mobile Applications

Add pages designed to run in a web browser to your hybrid mobile application by using the `ojet add web` command. When you run this command, the tooling creates a new `src-web` directory, enabling you to create both hybrid mobile and web applications from the same source.

Since applications that you intend to run in a browser or on a mobile device are designed for very different screen sizes, this approach allows you to add view and `viewModel` files that you create specifically for the browser. Then when you build the original application by using the command `ojet build web`, the tooling will copy your added files from the `src-web` folder and merge them with the contents of the application's `src` folder. Finally, serving the application, by using the command `ojet serve web`, populates the `web` staging folder with the merged source files and the tooling runs the application in the browser from this location.

Before you begin:

- Familiarize yourself with the folder structure of your hybrid mobile application. The file locations will vary in this procedure if you modified your directory structure, as described in [Modify the Hybrid Mobile Application's File Structure](#) or [Modify the Web Application's File Structure](#).

To add web browser capability to your hybrid mobile application:

1. At a terminal prompt, in your application's top level directory, enter the following command to add hybrid mobile features to your web application:

```
ojet add web
```

When you run the command, Oracle JET tooling creates two new empty source directories that you can use for platform specific content: `src-hybrid` and `src-web`.

2. To make changes to content that apply to both web and hybrid platforms, edit content in `src`.
3. To make changes to content that apply only to the web or hybrid platform, add the new content to `src-web` or `src-hybrid` as needed.

When you build your application, the content in the platform specific file takes precedence over content in `src`. For example, if you create a new `src-web/js/viewModels/dashboard.html` file, content in that file will take precedence over

the `src/js/viewModels/dashboard.html` file when you build the application as a hybrid mobile application with the `web` option.

 **Note:**

It is important that files you add to the `src-web` or `src-hybrid` folder maintain the same folder structure as the `src` merge target folder. For example, based on the original application's default folder structure, view or viewModel files specific to the web platform that you add should appear in the `src-web/js/view` and `src-web/js/viewModels` directories.

Build a Hybrid Mobile Application

Use Oracle JET CLI to build your Android, iOS, or Windows hybrid mobile application for testing and debugging.

Change to the application's root directory and use the `ojet build` command for each platform that your hybrid mobile application will support.

```
ojet build [android|ios|windows]
            [--build-config=path/buildConfig.json --destination=device|
emulator
            --cssvars=enabled|disabled
            --theme=themename --themes=theme1,theme2,...
            --sass
            --platform-options="string"]
```

 **Tip:**

You can also enter `ojet help` at a terminal prompt to get help for specific Oracle JET CLI options.

For example, the following command will build the sample Android application shown in [Scaffold a Hybrid Mobile Application](#)

```
ojet build android
```

The command will take some time to complete. If it's successful, you'll see the following message: Done. The command will also output the name and location of the built application in `hybrid/platforms/android`, `hybrid/platforms/ios`, or `hybrid/platforms/windows`.

By default `ojet build` creates a debug version of your application. You can also use the `ojet build` command with the `--release` option to build a release-ready version of your application. For information, see [Package and Publish Hybrid Mobile Applications](#).

WARNING:

If you use a proxy server and specify the Android platform, the build command will fail the first time you issue it. To resolve this, create a `gradle.properties` file in your `HOME/.gradle` directory and rerun the build command. The file should contain the following:

```
systemProp.http.proxyHost=proxy-server-URL
systemProp.http.proxyPort=80
systemProp.https.proxyHost=proxy-server-URL
systemProp.https.proxyPort=80
```

About `ojet build` Command Options for Hybrid Mobile Applications

Use the `ojet build` command with optional arguments to build a development version of your hybrid mobile application before serving it to a browser.

The following table describes the commonly-used options and provides examples for their use.

Option	Description
<code>[android ios windows]</code>	Desired platform. Enter android, ios, or windows.
<code>build-config</code>	Path to <code>buildConfig.json</code> . The <code>buildConfig.json</code> file contains details that Cordova can use to sign the application. You do not need this file when building a debug version of your app for Android or Windows, or if you are building your app for deployment to an iOS simulator. However, you must configure one for testing on an iOS device and when you're ready to release your Android, iOS, or Windows application. For additional information, see Package and Publish Hybrid Mobile Applications .
<code>destination</code>	Required for iOS applications. Specify one of the following: <ul style="list-style-type: none"> <code>emulator</code>: builds your application for deployment to the iOS Simulator. <code>device</code>: builds your application for deployment to an iOS device. Be aware, though, if you want to deploy your application to an iOS device, you must take additional steps as described in Package a Hybrid Mobile App on iOS. Typically, you can develop and test your iOS application in the simulator or web browser until you're ready to build your release.
<code>device</code>	Equivalent to <code>destination=device</code> .
<code>emulator</code>	Equivalent to <code>destination=emulator</code> .

Option	Description
theme	<p>Theme to use for the application. The theme defaults to redwood for hybrid mobile applications.</p> <p>Note: If you have migrated to JET 9.0.0 and later, and want to continue building with your Alta theme, for hybrid mobile themes, you can specify: alta:android, alta:ios, alta:windows). Note that the Alta theme is supported through JET 10.0.0 but is expected to become deprecated beyond that. For details about migrating applications, see Oracle JET Application Migration for Release 9.2.0.</p> <p>You can also enter a different <i>themename</i> for a custom theme as described in About CSS Variables and Custom Themes in Oracle JET for Redwood themes and, for a migrated custom Alta theme, in Customize Alta Themes Using the Tooling.</p>
themes	<p>Themes to include in the application, separated by commas without spaces.</p> <p>If you don't specify the --theme flag as described above, Oracle JET will use the first element that you specify in --themes as the default theme.</p>
--cssvars	<p>Injects a Redwood theme CSS file that supports working with CSS custom properties when you want to override CSS variables for the Redwood theme. For details about theming with CSS variables, see About CSS Variables and Custom Themes in Oracle JET.</p>
sass	<p>Manages Sass compilation. If you add Sass and specify the --theme or --themes option, Sass compilation occurs by default and you can use --sass=false or --no-sass to turn it off. If you add Sass and do not specify a theme option, Sass compilation will not occur by default, and you must specify --sass=true or --sass to turn it on. For details about theming with Sass, see Work with Sass.</p>
platform-options	<p>Platform-specific options that will pass verbatim to the Cordova CLI.</p> <p>This option is typically required for Windows device deployments or if a Cordova plugin contains multiple static libraries for different CPUs. If the value passed contains a quoted string, the quotation marks must be escaped.</p> <p>For example, you can use platform-options to specify Windows architectures. By default, the architecture defaults to anycpu.</p> <p>To specify a single architecture, use the --arch option and specify arm, x86, x64, or anycpu.</p>
	<pre>--platform-options="--arch arm x86 x64 anycpu"</pre>
	<p>To specify multiple architectures, use the --archs option with a space-separated list passed in as a quoted string. Note that you must escape the quotation marks as shown below.</p>
	<pre>--platform-options="--archs=\"arm x86 x64\""</pre>

Serve a Hybrid Mobile Application

Use Oracle JET CLI to launch your hybrid mobile application in a browser, simulator, or mobile device for testing and debugging. When you serve your application to a browser or emulator, a live reload option is enabled, and changes you make to the code are immediately reflected in the running application.

Before you begin:

- Familiarize yourself with the `ojet serve` command option `theme` when you want to run the application with an optional platform and a custom theme, as described in [Customize Alta Themes Using the Tooling](#).
- Optionally, use the `ojet serve` command with the `--release` option to serve a release-ready version of your application, as described in [Package and Publish Hybrid Mobile Applications](#).
- If you want to send your application to an iOS device, you must take additional steps as described in [Package a Hybrid Mobile App on iOS](#).

At a command prompt, change to the application's top level directory and use the `ojet serve` command with options to launch the application.

```
ojet serve [ios|android|windows]
    [--build-config=path/buildConfig.json
     --server-port=server-port-number --livereload-port=live-
     reload-port-number
     --destination=emulator[:emulator-name] | browser[:browser-
     name] | device[:device-name] | server-only
     --livereload --build
     --cssvars=enabled|disabled
     --theme=themename --themes=theme1,theme2,...
     --sass
     --platform-options="string"]
```

Tip:

You can also enter `ojet help` at a terminal prompt to get help for specific Oracle JET CLI commands.

The application will launch in a local browser, emulator/simulator, or device depending upon the options you specify. The following table shows examples.

Command	Description
<code>ojet serve windows --browser=firefox</code>	Launches a Windows version of the application in the Firefox browser.
<code>ojet serve ios</code>	Launches the application in the iOS Simulator using the Alta iOS theme.
<code>ojet serve android --destination=emulator:MyEmulator</code>	Launches the application in the Android emulator using the AVD named "MyEmulator". The emulator name is case-sensitive.

Command	Description
<code>ojet serve android --device</code>	Launches the application on the attached Android mobile device.

The terminal will also output the names of the files as they are loaded. If your application contains multiple views, the output will reflect the names of the views and associated files as you navigate through the application.

WARNING:

If you specify the Android platform, use a proxy server and skipped the `ojet build` step, the `serve` command will fail the first time you issue it. To resolve this, create a `gradle.properties` file in your `HOME/.gradle` directory and rerun the `serve` command. The file should contain the following:

```
systemProp.http.proxyHost=proxy-server-URL
systemProp.http.proxyPort=80
systemProp.https.proxyHost=proxy-server-URL
systemProp.https.proxyPort=80
```

If you left live reload enabled (default, `--livereload=true`), the terminal window updates to reflect that the code has changed. For example, if you save a change to `dashboard.html` in an application scaffolded with the navbar or navdrawer template, the terminal window outputs the name of the changed file, and the browser or emulator/simulator updates with the change. Live reload is disabled when you serve an application to a device.

To terminate the batch job when using live reload, press `Ctrl+C` in the command window and then enter `y` if prompted to terminate the batch job.

About `ojet serve` Command Options for Hybrid Mobile Applications

Use `ojet serve` to run your hybrid mobile application in a local web server for testing and debugging.

The following table describes the commonly-used options and provides examples for their use.

Option	Description
<code>[ios android windows]</code>	Desired platform. Enter <code>android</code> , <code>ios</code> , or <code>windows</code> .
<code>build-config</code>	Specify the path to <code>buildConfig.json</code> . The <code>buildConfig.json</code> file contains details that Cordova can use to sign the application.
	You do not need this file when building a debug version of your application for Android or Windows, or if you are building your app for deployment to an iOS simulator. However, you must configure one for testing on an iOS device and for pre-release testing of your Android, iOS or Windows application.
<code>server-port</code>	Server port number. If not specified, defaults to 8000.

Option	Description
livereload-port	Live reload port number. If not specified, defaults to 35729.
destination	<p>Specify one of the following:</p> <ul style="list-style-type: none"> • emulator: Displays your application in the default Android AVD, iOS Simulator, or Windows Emulator. To use a different emulator, append its name to the emulator option: <code>--destination=emulator:emulator-name</code>.

Tip:

You can view the list of available emulators for each platform by invoking the following from your app's top-level folder:

```
hybrid/platforms/{platform}/
cordova/lib/list-emulator-images
```

where {platform} is one of android, ios, or windows.

- **browser:** Displays your application in the Chrome browser on your local machine.

If you don't have Chrome installed or want to use a different browser, append the name of the desired browser to the browser option:

```
--destination=firefox|edge|ie|opera|
safari|chrome
```

Tip:

To change your application's default browser from Chrome, open `oraclejetconfig.json` in the application's top level directory and change the name of the browser in `defaultBrowser`. For example, to change the default browser to Firefox, edit `oraclejetconfig.json` as shown below.

```
"defaultBrowser": "firefox"
```

- **device:** Sends the application to an attached device. Optionally, append the name of the device to device option: `--destination=device:myDevice`.
- **server-only:** Serves the application, as if to a browser, but does not launch a browser. Use this option in cloud-based development environments so that you can attach your browser to the app served by the development machine.

<code>browser[=browser-name]</code>	Equivalent to <code>destination=firefox[:browser-name]</code> .
<code>emulator[=emulator-name]</code>	Equivalent to <code>destination=emulator[:emulator-name]</code> .
<code>device[=device-name]</code>	Equivalent to <code>destination=device[:device-name]</code> .
<code>server-only</code>	Equivalent to <code>destination=server-only</code> .

Option	Description
livereload	<p>Enable the live reload feature. Live reload is enabled by default (<code>--livereload=true</code>). Use <code>--livereload=false</code> or <code>--no-livereload</code> to disable the live reload feature. Disabling live reload can be helpful if you're working in an IDE and want to use that IDE's mechanism for loading updated applications.</p>
build	<p>Build the app before you serve it. By default, an app is built before you serve it (<code>--build=true</code>). Use <code>--build=false</code> or <code>--no-build</code> to suppress the build if you've already built the application and just want to serve it.</p>
theme	<p>Theme to use for the application. The theme defaults to <code>redwood</code> for hybrid mobile applications.</p> <p>Note: If you have migrated to JET 9.0.0 and later, and want to continue building with your Alta theme, for hybrid mobile themes, you can specify: <code>alta:android</code>, <code>alta:ios</code>, <code>alta:windows</code>. Note that the Alta theme is supported through JET 10.0.0 but is expected to become deprecated beyond that. For details about migrating applications, see Oracle JET Application Migration for Release 9.2.0.</p> <p>You can also enter a different <code>themename</code> for a custom theme as described in About CSS Variables and Custom Themes in Oracle JET for Redwood themes and, for a migrated custom Alta theme, in Customize Alta Themes Using the Tooling.</p>
themes	<p>Themes to use for the application, separated by commas. If you don't specify the <code>--theme</code> flag as described above, Oracle JET will use the first element that you specify in <code>--themes</code> as the default theme. Otherwise Oracle JET will build the application with the theme specified in <code>--theme</code>.</p>
<code>--cssvars</code>	<p>Injects a Redwood theme CSS file that supports working with CSS custom properties when you want to override CSS variables to customize the Redwood theme. For details about theming with CSS variables, see About CSS Variables and Custom Themes in Oracle JET.</p>
sass	<p>Manages Sass compilation. If you add Sass and specify the <code>--theme</code> or <code>--themes</code> option, Sass compilation occurs by default and you can use <code>--sass=false</code> or <code>--no-sass</code> to turn it off. If you add Sass and do not specify a theme option, Sass compilation will not occur by default, and you must specify <code>--sass=true</code> or <code>--sass</code> to turn it on. For details about theming with Sass, see Work with Sass.</p>

 **Note:**

The option that you choose controls both Sass compilation in the build step and Sass watch in the serve step.

Option	Description
platform-options	<p>Platform-specific options that will pass verbatim to the Cordova CLI.</p> <p>This option is typically required for Windows device deployments or if a Cordova plugin contains multiple static libraries for different CPUs. If the value passed contains a quoted string, the quotation marks must be escaped.</p> <p>For example, you can use platform-options to specify Windows architectures. By default, the architecture defaults to <code>anycpu</code>.</p> <p>To specify a single architecture, use the <code>--arch</code> option and specify <code>arm</code>, <code>x86</code>, <code>x64</code>, or <code>anycpu</code>.</p> <pre data-bbox="750 623 1183 650"><code>--platform-options="--arch arm"</code></pre>

Review Your Application Settings in the config.xml File

The `AppRootDir/hybrid/config.xml` file contains a number of default settings that you may want to review and modify before you package and publish your app for distribution to end users.

You can configure entries that affect the behavior of your app across all platforms and entries that are applied only to a specific platform.

Examples of entries for all platforms that you may want to modify in your file before you publish your app include the following:

- Attribute values of the `widget` element. Specifically, the values for the `id` and `version` attributes.

The value of the `id` attribute determines the unique identifier for this app. By default, JET apps use a combination of reverse domain notation with the `oraclejet.org` domain name as input and the app's short name. Change this to use, for example, the reverse of your company's domain name.

- The value of the `version` attribute identifies the version number of your app to end users. It appears, for example, on the App Info screen on Android devices. Change it to an appropriate value.
- Value for the `name` element. This is the name of the app displayed on the springboard of your user's mobile device and in the app stores.

Apart from these generic settings that affect your app irrespective of the platform where it is deployed (Android, iOS, or Windows), you can configure a range of other entries in the `config.xml` file that set preferences for your app on specific platforms.

The following example `AppRootDir/hybrid/config.xml` file displays a number of illustrative examples.

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="org.oraclejet.docexample" version="1.0.0" xmlns="http://www.w3.org/ns/widgets"
       xmlns:cdv="http://cordova.apache.org/ns/1.0">
  <name>docexample</name>
  <description>A sample Oracle JavaScript Extension Toolkit (JET) mobile app based on Cordova</description>
```

```

<author email="undefined" href="http://www.oraclejet.org">Oracle Jet Team</
authors>
<content src="index.html"/>
<plugin name="cordova-plugin-whitelist" spec="1"/>
<access origin="*"/>
<allow-intent href="http:///*/*"/>
...
<allow-intent href="geo:/*"/>
<platform name="windows">
    <preference name="windows-target-version" value="10.0"/>
    <icon src="res/icon/windows/Square30x30Logo.scale-100.png" width="30"
height="30"/>
    ...
    <splash src="res/screen/windows/SplashScreenPhone.scale-240.png"
width="1152" height="1920"/>
</platform>
<platform name="android">
    <allow-intent href="market:/*"/>
    <preference name="DisallowOverscroll" value="true"/>
<!-- The following entry displays your app using the full screen of the Android
device and thus hides the Android
status bar and menu button.-->
<preference name="Fullscreen" value="true" />
    <icon src="res/icon/android/icon-ldpi.png" width="36" height="36"/>
    ...
    <splash src="res/screen/android/splash-port-xxxhdpi.9.png" density="port-
xxxhdpi"/>
</platform>
<platform name="ios">
    <allow-intent href="itms:/*"/>
    <allow-intent href="itms-apps:/*"/>
    <preference name="Orientation" value="all"/>
    <icon src="res/icon/ios/icon-small.png" width="29" height="29"/>
    ...
    <splash src="res/screen/ios/Default-Portrait~ipad.png" width="768"
height="1024"/>
</platform>
...
</widget>
```

For more information about the `AppRootDir/hybrid/config.xml` file, see https://cordova.apache.org/docs/en/latest/config_ref/.

Change the Splash Screen and App Launcher Icon

Replace the JET-provided images with those that you want your app to use as a splash screen or an app launcher icon.

The default splash screen behavior of your app depends on the platform where you run the app. On Android, for example, the default behavior is to display a white screen. JET provides a set of splash screen images for each platform that can be used in your app. These images are stored in the following sub-directories of your app's hybrid directory.

```

AppRootDirectory/hybrid/res/screen
+---android
|   splash-land-hdpi.9.png
|   ...
|   splash-port-xxxhdpi.9.png
```

```

|
+---ios
|   Default-568h@2x~iphone.png
|   ...
|   Default~iphone.png
|
\---windows
    SplashScreen.scale-100.png
    SplashScreenPhone.scale-240.png

```

To use these splash screens, install the `cordova-plugin-splashscreen` plugin by executing the following command:

```
ojet add plugin cordova-plugin-splashscreen
```

You can replace the JET-provided splash screen images with your own images, matching the names and sizes. You can change the behavior of the splash screen by configuring your app's `AppRootDir/hybrid/config.xml` file. The following example shows how you display the splash screen for 4000 milliseconds.

```
<preference name="SplashScreenDelay" value="4000" />
```

For more information about the plugin, including how to configure its behavior plus platform-specific information, see <https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-splashscreen/>.

JET provides a set of app launcher icons for each platform that can be used in your app. To use an alternative app launcher icon to the JET-provided icons, replace the images in the following directories.

```

AppRootDir/hybrid/res/icon
+---android
|   icon-hdpi.png
|   ...
|   icon-xxxhdpi.png
|
+---ios
|   icon-40.png
|   ...
|   icon@2x.png
|
\---windows
    Square150x150Logo.scale-100.png
    ...
    Wide310x150Logo.scale-240.png

```

Both the splash screen and app launcher icons that your app uses are referenced from your app's `AppRootDir/hybrid/config.xml` file, as shown by the following example excerpts.

```

<platform name="windows">
    ...
    <icon src="res/icon/windows/Square30x30Logo.scale-100.png" width="30"
height="30"/>
    ...
    <splash src="res/screen/windows/SplashScreen.scale-100.png" width="620"
height="300"/>

<platform name="android">

```

```

...
<icon src="res/icon/android/icon-ldpi.png" width="36" height="36"/>
...
<splash src="res/screen/android/splash-land-ldpi.9.png" density="land-ldpi"/>
...
<platform name="ios">
...
<icon src="res/icon/ios/icon-small.png" width="29" height="29"/>
...
<splash src="res/screen/ios/Default@2x~iphone.png" width="640" height="960"/>
...

```

For more information about icons and their entries in the `config.xml` file, see https://cordova.apache.org/docs/en/latest/config_ref/images.html.

Customize the Hybrid Mobile Application Tooling Workflow

Hook points that Oracle JET tooling defines let you customize the behavior of the build and serve processes when you want to define new steps to execute during the tooling workflow using script code that you write.

When you create a hybrid mobile application, Oracle JET tooling generates script templates in the `/scripts/hooks` application subfolder. To customize the Oracle JET tooling workflow you can edit the generated templates with the script code that you want the tooling to execute for specific hook points during the build and serve processes. If you do not create a custom hook point script, Oracle JET tooling ignores the script templates and follows the default workflow for the build and serve processes.

To customize the workflow for the build or serve processes, you edit the generated script template file named for a specific hook point. For example, to trigger a script at the start of the tooling's build process, you would edit the `before_hybrid_build.js` script named for the hook point triggered before the build begins. That hook point is named `before_hybrid_build`.

Therefore, customization of the build and serve processes that you enforce on Oracle JET tooling workflow requires that you know the following details before you can write a customization script.

- The type of application that you created: either a web application or a hybrid mobile application.
- The Oracle JET build or serve mode that you want to customize:
 - Debug — The default mode when you build or serve your application, which produces the source code in the built application.
 - Release — The mode when you build the application with the `--release` option, which produces minified and bundled code in a release-ready application.
- The appropriate hook point to trigger the customization.
- The location of the default hook script template to customize.

About the Script Hook Points for Hybrid Mobile Applications

The Oracle JET hooks system defines various script trigger points, also called hook points, that allow you to customize the create, build, serve, and restore workflow

across the various command-line interface processes. Customization relies on script files and the script code that you want to trigger for a particular hook point.

The following table identifies the hook points and the workflow customizations they support in the Oracle JET tooling create, build, serve, and restore processes. Unless noted, hook points for the processes support both debug and release mode.

Hook Point	Tooling Process	Description
after_app_create	create	This hook point triggers the script with the default name <code>after_app_create.js</code> immediately after the tooling concludes the create application process. A script for this hook point can be used with a web application or a hybrid mobile application.
after_app_restore	restore	This hook point triggers the script with the default name <code>after_app_restore.js</code> immediately after the tooling concludes the restore application process. A script for this hook point can be used with a web application or a hybrid mobile application.
before_build	build	This hook point triggers the script with the default name <code>before_build.js</code> immediately before the tooling initiates the build process. A script for this hook point can be used with a web application or a hybrid mobile application.
before_release_build	(release mode only)	This hook point triggers the script with the default name <code>before_release_build.js</code> before the minification step and the requirejs bundling step occur. A script for this hook point can be used with a web application or a hybrid mobile application.
before_hybrid_build	build	This hook point triggers the script with the default name <code>before_hybrid_build.js</code> before the <code>cordovaPrepare</code> and <code>CordovaCompile</code> build steps occur. A script for this hook point can be used only with a hybrid mobile application.
before_app_typescript	build / serve script	This hook point triggers the script with the default name <code>before_app_typescript.js</code> after the build process or serve process steps occur. Use the hook to update, add or remove TypeScript compiler options defined by your application's <code>tsconfig.json</code> compiler configuration file. The hook system passes your reference to the modified <code>tsconfig</code> object to the TypeScript compiler. A script for this hook point can only be used with a TypeScript application.
after_app_typescript	build / serve script	This hook point triggers the script with the default name <code>after_app_typescript.js</code> after the build process or serve process steps occur and immediately after the <code>before_app_typescript</code> hook point executes. This hook provides an entry point for applications that require further processing, such as compiling generated <code>.jsx</code> output using babel. A script for this hook point can only be used with a TypeScript application.

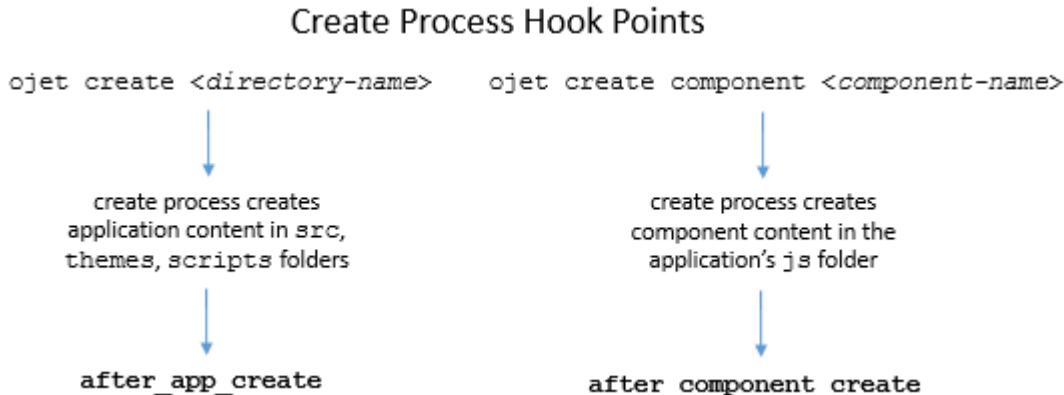
Hook Point	Tooling Process	Description
before_component_typescript	build / serve	This hook point triggers the script with the default name <code>before_component_typescript.js</code> after the build process or serve process steps occur. Use the hook to update, add or remove TypeScript compiler options defined by your application's <code>tsconfig.json</code> compiler configuration file. The hook system passes your reference to the modified <code>tsconfig</code> object to the TypeScript compiler. A script for this hook point can only be used with a TypeScript application.
after_component_typescript	build / serve	This hook point triggers the script with the default name <code>after_component_typescript.js</code> after the build process or serve process steps occur and immediately after the <code>before_component_typescript</code> hook point executes. This hook provides an entry point for applications that require further processing, such as compiling generated <code>.jsx</code> output using babel. A script for this hook point can only be used with a TypeScript application.
before_optimize	(release mode only)	This hook point triggers the script with the default name <code>before_optimize.js</code> before the release mode build/serve process minifies the content. A script for this hook point can be used with a web application or a hybrid mobile application.
before_component_optimize	build / serve	This hook point triggers the script with the default name <code>before_component_optimize.js</code> before the build/serve process minifies the content. A script for this hook point can be used to modify the build process specifically for a project that defines a Web Component.
after_build	build	This hook point triggers the script with the default name <code>after_build.js</code> immediately after the tooling concludes the build process. A script for this hook point can be used with a web application or a hybrid mobile application.
after_component_create	build	This hook point triggers the script with the default name <code>after_component_create.js</code> immediately after the tooling concludes the create Web Component process. A script for this hook point can be used to modify the build process specifically for a project that defines a Web Component.
after_component_build	(debug mode only)	This hook point triggers the script with the default name <code>after_component_build.js</code> immediately after the tooling concludes the Web Component build process. A script for this hook point can be used to modify the build process specifically for a project that defines a Web Component.
before_serve	serve	This hook point triggers the script with the default name <code>before_serve.js</code> before the web serve process connects to and watches the application. In the case of hybrid builds, it also precedes the <code>cordovaClean</code> and <code>cordovaServe</code> step. A script for this hook point can be used with a web application or a hybrid mobile application.

Hook Point	Tooling Process	Description
after_serve	serve	This hook point triggers the script with the default name <code>after_serve.js</code> after all build process steps complete and the tooling serves the application. A script for this hook point can be used with a web application or a hybrid mobile application.
before_watch	serve	This hook point triggers the script with the default name <code>before_watch.js</code> before the web serve process connects to and watches the application. A script for this hook point can be used with a web application or a hybrid mobile application.
before_watch	serve	This hook point triggers the script with the default name <code>before_watch.js</code> after the tooling serves the application and before the tooling starts watching for application changes. A script for this hook point can be used with a web application or a hybrid mobile application.
after_watch	serve	This hook point triggers the script with the default name <code>after_watch.js</code> after the tooling starts the watch and after the tooling detects a change to the application. A script for this hook point can be used with a web application or a hybrid mobile application.

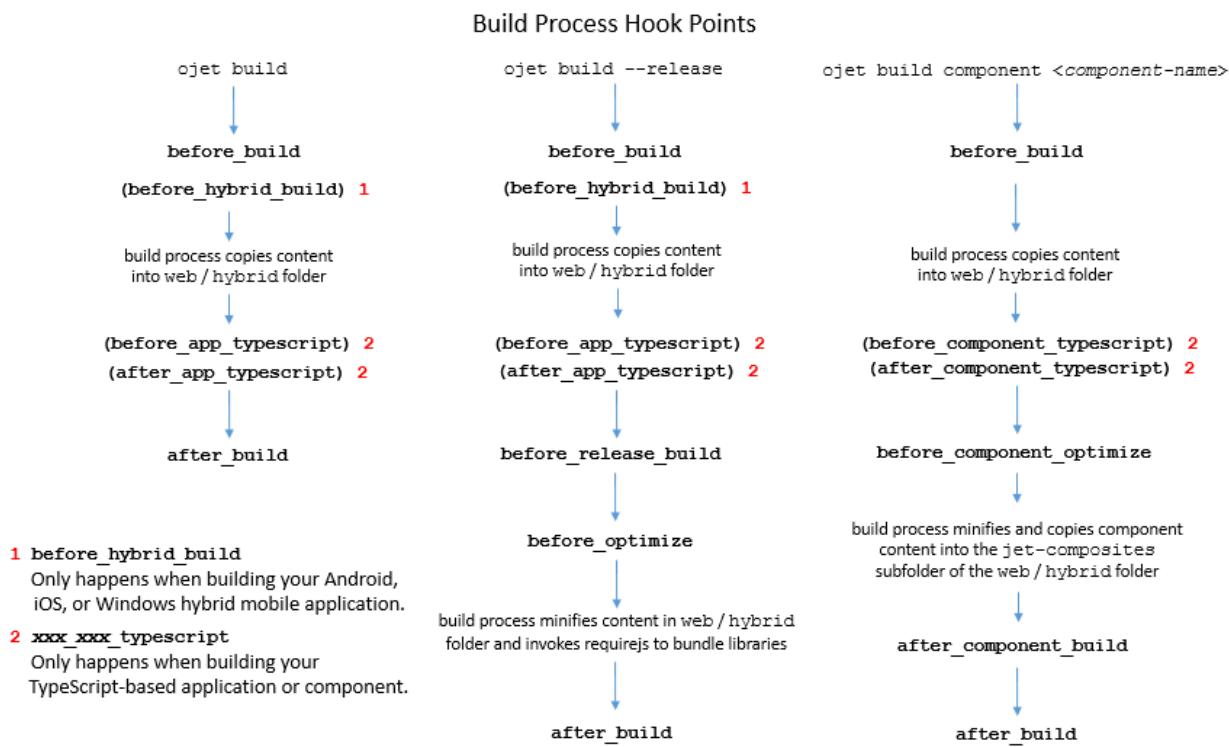
About the Process Flow of Script Hook Points

The Oracle JET hooks system defines various script trigger points, also called hook points, that allow you to customize the create, build, serve, and restore workflow across the various command-line interface processes.

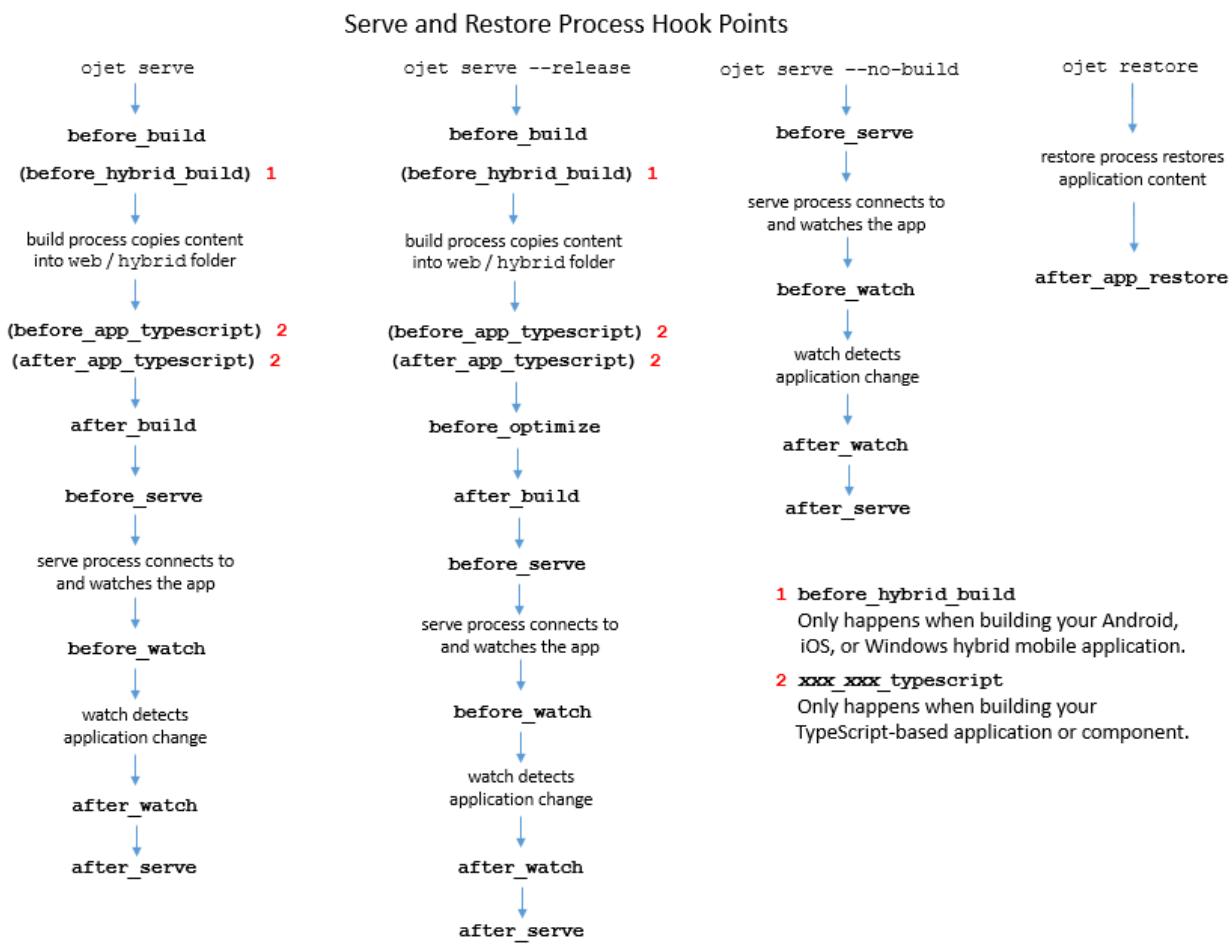
The following diagram shows the script hook point flow for the create process.



The following diagram shows the script hook point flow for the build process.



The following diagram shows the script hook point flow for the serve and restore processes.



Change the Hooks Subfolder Location

When you create an application, Oracle JET tooling generates script templates in the `/scripts/hooks` application subfolder. Your development effort may require you to relocate hook scripts to a common location, for example to support team development.

By default, the hooks system locates the scripts in the `hooks` subfolder using a generated JSON file (`hooks.json`) that specifies the script paths. When the tooling reaches the hook point, it executes the corresponding script which it locates using the `hooks.json` file. If you relocate hook script(s) to a common location, you must edit the `hooks.json` file to specify the new location for the hook scripts that you relocated, as illustrated by the following example.

```
{
    "description": "OJET-CLI hooks configuration file",
    "hooks": {
        "after_app_create": "scripts/hooks/after_app_create.js",
        ...
        "after_serve": "http://example.com/cdn/common/scripts/hooks/
after_serve.js"
    }
}
```

Create a Hook Script for Hybrid Mobile Applications

You can create a hook point script to define a new command-line interface process step for your hybrid mobile application. To create a hook script, you edit the script template associated with a specific hook point in the tooling build and serve workflow.

The Oracle JET hooks system defines various script trigger points, also called hook points, that allow you to customize the build and serve workflow across the various build and serve modes. Customization relies on script files and the script code that you want to trigger for a particular hook point. Note that the generated script templates that you modify with your script code are named for their corresponding hook point.

To customize the workflow for the build or serve processes, you edit the generated script template file named for a specific hook point. For example, to trigger a script at the start of the tooling's build process, you would edit the `before_hybrid_build.js` script named for the hook point triggered before the build begins. That hook point is named `before_hybrid_build`.

A basic example illustrates a simple customization using the `before_optimize` hook, which allows you to control the RequireJS properties shown in bold to modify the application's bundling configuration.

```
requirejs.config(
{
  baseUrl: "web/js",
  name: "main-temp",
  paths: {
    // injector:mainReleasePaths
    "knockout": "libs/knockout/knockout-3.x.x.debug",
    "jquery": "libs/jquery/jquery-3.x.x",
    "jqueryui-amd": "libs/jquery/jqueryui-amd-1.x.x",
    ...
  }
  // endinjector
  out: "web/js/main.js"
}
...
...
```

A script for this hook point might add one line to the `before_optimize` script template, as shown below. When your build the application with this customization script file in the default location, the tooling triggers the script before calling `requirejs.out()` and changes the `out` property setting to a custom directory path. The result is the application-generated `main.js` is output to the named directory instead of the default `web/js/main.js` location.

```
module.exports = function (configObj) {
  return new Promise((resolve, reject) => {
    console.log("Running before_optimize hook.");
    configObj.requirejs.out = 'myweb/js/main.js';
    resolve(configObj);
  });
};
```

 **Tip:**

If you want to change application path mappings, it is recommended to always edit the `path_mappings.json` file. An exception might be when you want application runtime path mappings to be different from the mappings used by the bundling process, then you might use a `before_optimize` hook script to change the `requirejs.config` path property.

The following example illustrates a more complex build customization using the `after_build` hook. This hook script adds a customize task after the build finishes.

```
'use strict';

const fs = require('fs');
const archiver = require('archiver');

module.exports = function (configObj) {
    return new Promise((resolve, reject) => {
        console.log("Running after_build hook.");

        //Set up the archive
        const output = fs.createWriteStream('my-archive.war');
        const archive = archiver('zip');

        //Callbacks for the archiver
        output.on('close', () => {
            console.log('Files were successfully archived.');
            resolve();
        });

        archive.on('warning', (error) => {
            console.warn(error);
        });

        archive.on('error', (error) => {
            reject(error);
        });

        //Archive the src folder and close the file
        archive.pipe(output);
        archive.directory('src', false);
        archive.finalize();
    });
};
```

In this example, assume the script templates reside in the default folder generated when you created the application. The goal is to package the application source into a ZIP file. Because packaging occurs after the application build process completes, this script is triggered for the `after_build` hook point. For this hook point, the modified script template `after_build.js` will contain the script code to zip the application, and because the `.js` file resides in the default location, no hooks system configuration changes are required.

 **Tip:**

OJET tooling reports when hook points are executed in the message log for the build and serve process. You can examine the log in the console to understand the tooling workflow and determine exactly when the tooling triggers a hook point script.

Pass Arguments to a Hook Script for Hybrid Mobile Applications

You can pass extra values to a hook script from the command-line interface when you build or serve the web application. The hook script that you create can use these values to perform some workflow action, such as creating an archive file from the contents of the `src` folder.

You can add the `--user-options` flag to the command-line interface for Oracle JET to define user input for the hook system when you build or serve the web application. The `--user-options` flag can be appended to the build or serve commands and takes as arguments one or more space-separated, string values:

```
ojet build --user-options="some string1" "some string2" "some stringx"
```

For example, you might write a hook script that archives a copy of the `src` folder after the build finishes. The developer might pass the user-defined parameter `archive-file` set to the archive file name by using the `--user-options` flag on the Oracle JET command line.

```
ojet build web --user-options="archive-file=archive.zip"
```

If the flag is appended and the appropriate input is passed, the hook script code may write a ZIP file to the `/archive` directory in the root of the project. The following example illustrates this build customization using the `after_build` hook. The script code parses the user input for the value of the user defined `archive-file` flag with a promise to archive the application source after the build finishes by calling the NodeJS function `fs.createWriteStream()`. This hook script is an example of taking user input from the command-line interface and processing it to achieve a build workflow customization.

```
'use strict';
const fs = require('fs');
const archiver = require('archiver');
const path = require('path');

module.exports = function (configObj) {
  return new Promise((resolve, reject) => {
    console.log("Running after_build hook.");

    //Check to see if the user set the flag
    //In this case we're only expecting one possible user defined
    //argument so the parsing can be simple
    const options = configObj.userOptions;
    if (options){
```

```

const userArgs = options.split('=');
if (userArgs.length > 1 && userArgs[0] === 'archive-file'){
    const archiveRoot = 'archive';
    const outputArchive = path.join(archiveRoot,userArgs[1]);

    //Ensure the output folder exists
    if (!fs.existsSync(archiveRoot)) {
        fs.mkdirSync(archiveRoot);
    }

    //Set up the archive
    const output = fs.createWriteStream(outputArchive);
    const archive = archiver('zip');

    //callbacks for the archiver
    output.on('close', () => {
        console.log(`Archive file ${outputArchive} successfully
created.`);
        resolve();
    });

    archive.on('error', (error) => {
        console.error(`Error creating archive ${outputArchive}`);
        reject(error);
    });

    //Archive the web folder and close the file
    archive.pipe(output);
    archive.directory('src', false);
    archive.finalize();
}
else {
    //Unexpected input - fail with information message
    reject(`Unexpected flags in user-options: ${options}`);
}
else {
    //nothing to do
    resolve();
}
});
};

```

Use Cordova Plugins to Access Mobile Device Services

You can enable user access to device features, such as camera, geolocation and the local file system, by including Cordova plugins into your JET app.

Topics

- [About Apache Cordova and Cordova Plugins](#)
- [Use a Plugin in Your App](#)
- [Cordova Plugins Recommended by Oracle JET](#)

- Use a Different Web View in Your JET Hybrid Mobile App

About Apache Cordova and Cordova Plugins

JET uses plugins developed with the Apache Cordova framework to access the capabilities of the devices on which your hybrid mobile application is installed.

Apache Cordova is an open-source cross-platform development framework that enables application developers to use standard web technologies (HTML, CSS, and JavaScript) to develop hybrid mobile applications. A hybrid mobile application refers to mobile applications that are native applications which are installed onto mobile devices in the usual manner, but which use a web view to render the UI rather than using the platform's native UI components. For additional information about Apache Cordova, see the Overview page at <http://cordova.apache.org/docs/en/latest/guide/overview/index.html>.

Since a hybrid mobile application is developed using standard web technologies, the same code can be reused across the supported platforms (such as Android, iOS and Windows). Apache Cordova provides tools that enable you to add support for the platforms that the application will run on independent of this code.

JET uses Apache Cordova to develop web technology-based applications for some of the mobile platforms supported by Apache Cordova. So, one could say that the JET hybrid mobile application that you develop is an Apache Cordova application where JET provides features such as components and themes that determine the look and feel of your application.

A plugin is a package of code that allows the Cordova web view within which your application renders to communicate with the native platform on which it runs. The plugin does this by providing a JavaScript interface to native components that allows your application to use native device capabilities, such as camera, geolocation, and so on.

So, assume for example, that you want your application to use a camera. In that case, you look for an existing plugin to provide access to the camera on the platform(s) where your application will be installed as there is a good possibility that someone has already developed a plugin to address this requirement.

If you cannot find a plugin that meets your requirements, you can develop your own plugin. Although this [blog post](#) makes reference to another mobile development framework (MAF), it provides a suitable introduction to creating a Cordova plugin that could be used in a JET application or any Cordova-based application.

To find a plugin to use in your application, go to the Plugins page at <https://cordova.apache.org/plugins/> that provides a registry of core and third-party Cordova plugins. Core plugins are plugins provided by Apache Cordova while third-party plugins are those developed by independent developers. Core plugins display a blue vertical strip to the left of the card that describes the plugin, as shown in the following figure, where two core plugins (`cordova-plugin-media` and `cordova-plugin-contacts`) appear after a third-party plugin named `cordova-plugin-ble-peripheral`.

The screenshot shows the Apache Cordova Plugins page with three listed plugins:

- cordova-plugin-ble-peripheral**: Version 0.0.1 by **don**. Last updated 385 days ago. Supports iOS.
- cordova-plugin-media**: Version 3.0.1 by **filmaj**. Last updated 25 days ago. Supports Android, iOS, Windows, BlackBerry, Linux, and FireOS.
- cordova-plugin-contacts**: Version 2.3.1 by **filmaj**. Last updated 25 days ago. Supports Android, iOS, Windows, BlackBerry, Linux, and FireOS.

The Plugins page categorizes plugins according to the platform on which the plugins will run. Some plugins run on a wide variety of mobile platforms (Android, iOS, Windows, and so on) while other plugins may only support one platform. Ensure that the plugins you install in your application support the platforms where your application will run.

Each entry generally provides information such as a brief description of the plugin's functionality, the platforms it supports, and the number of days since it was updated. It then links to a web page where you can access more detail, such as how to install the plugin in your application and use it once installed.

Use a Plugin in Your App

You add the plugin and then write code that invokes the plugin's JavaScript interface to access the device APIs.

To add a plugin to your app, change to the application's root directory and use the `ojet add plugin` command. The following example illustrates how you install the `phonegap-plugin-barcodescanner` plugin.

```
ojet add plugin phonegap-plugin-barcodescanner
```

Once you have added the plugin to your app, you need to write code in your app to use the plugin. The following code excerpts illustrate how you might use the `phonegap-plugin-barcodescanner` plugin by exposing a button in the `appRootDir/src/js/views/incidents.html` page of a JET app built using the `navbar` template that invokes the barcode scanner on the device.

```
...
<ojo-button id='b1Scan' on-oj-action='{{buttonClick}}' label='Scan'></ojo-button>
...
```

The following code excerpt shows the entries to add to the `appRootDir/src/js/viewModels/incidents.js` file which invokes the barcode scanner when an end user clicks the button that the `incidents.html` page renders.

```
self.buttonClick = function(data, event){  
    cordova.plugins.barcodeScanner.scan(  
        function (result) {  
            alert("We got a barcode\n" +  
                "Result: " + result.text + "\n" +  
                "Format: " + result.format + "\n" +  
                "Cancelled: " + result.cancelled);  
        },  
        function (error) {  
            alert("Scanning failed: " + error);  
        }  
    );  
}
```

Cordova Plugins Recommended by Oracle JET

JET recommends a number of Cordova plugins that you can use in your app to provide native device functionality.

The [Cordova Plugins](#) page on the Oracle JET web site lists a number of plugins that have been successfully used in the verification testing of JET sample or demo apps. Although Oracle JET recommends these plugins, it does not support them.

Use a Different Web View in Your JET Hybrid Mobile App

JET hybrid mobile apps use the default web view supplied by each mobile operating system.

On the Android platform, this is the Android platform's `WebView`. A number of Cordova plugins exist that enable you to use a different web view in your hybrid mobile app. One example for the Android platform is the `cordova-plugin-crosswalk-webview` plugin that configures your app to use the Crosswalk web view to bring performance improvements and compatibility improvements across older Android versions. Install this plugin in your app using the following command:

```
ojet add plugin cordova-plugin-crosswalk-webview
```

For more information about the `cordova-plugin-crosswalk-webview` plugin, read its documentation.

For apps that run on the iOS platform, consider adding the `cordova-plugin-wkwebview-file-xhr` plugin to your app so that your app uses the more performant WKWebView instead of the default UIWebView used on iOS devices.

```
ojet add plugin cordova-plugin-wkwebview-file-xhr
```

For more information about the `cordova-plugin-wkwebview-file-xhr` plugin, read its documentation.

Designing Responsive Applications

Oracle JET includes classes that support a flexbox-based layout, 12-column responsive grid system, design patterns, responsive form layout, and responsive JavaScript that you can use to design responsive web and hybrid mobile applications.

Topics:

- [Typical Workflow for Designing Responsive Applications in Oracle JET](#)
- [Oracle JET and Responsive Design](#)
- [Media Queries](#)
- [Oracle JET Flex, Grid, Form, and Responsive Helper Class Naming Convention](#)
- [Oracle JET Flex Layouts](#)
- [Oracle JET Grids](#)
- [Responsive Layout and Content Design Patterns](#)
- [Responsive Form Layouts](#)
- [Add Responsive Design to Your Application](#)
- [Use Responsive JavaScript](#)
- [Use the Responsive Helper Classes](#)
- [Create Responsive CSS Images](#)
- [Change Default Font Size](#)
- [Control the Size and Generation of the CSS](#)

Typical Workflow for Designing Responsive Applications in Oracle JET

Oracle JET includes classes for creating responsive applications. After you create your application, you can fine tune your design using Oracle JET responsive JavaScript and helper classes.

To design responsive applications in Oracle JET, refer to the typical workflow described in the following table:

Task	Description	More Information
Understand Oracle JET's support for responsive design	Understand the Oracle JET flex layout, 12-column mobile-first grid, and form layout framework classes, and the responsive class naming conventions.	Oracle JET and Responsive Design Media Queries Oracle JET Flex, Grid, Form, and Responsive Helper Class Naming Convention Oracle JET Flex Layouts Oracle JET Grids Responsive Layout and Content Design Patterns Responsive Form Layouts
Create a responsive application using the Oracle JET responsive classes.	Add application, flex, grid, and form layout classes to your application.	Add Responsive Design to Your Application
Fine tune your responsive design	Use responsive JavaScript, helper classes, and responsive CSS images to complete the responsive design. Change default font size across your application.	Use Responsive JavaScript Use the Responsive Helper Classes Create Responsive Images Change Default Font Size

Oracle JET and Responsive Design

Responsive design describes a design concept that uses fluid grids, scalable images, and media queries to present alternative layouts based on the media type. With responsive design, you can configure Oracle JET applications to be visually appealing on a wide range of devices, ranging from small phones to wide-screen desktops.

Oracle JET includes classes that support a [flexible box layout](#). In a flex layout, you can lay out the children of a flex container in any direction, and the children will grow to fill unused space or shrink to avoid overflowing the parent. You can also nest boxes (for example, horizontal inside vertical or vertical inside horizontal) to build layouts in two dimensions.

Oracle JET also provides a 12-column grid system and form layout classes that include styles for small, medium, large, and extra large screens or devices that you can use in conjunction with the flex layout classes to achieve finer control of your application's layout. The grid system and form classes use media queries to set the style based on the width of the screen or device, and you can use them to customize your page layout based on your users' needs.

In addition, media queries form the basis for responsive helper classes that show or hide content, align text, or float content based on screen width. They are also the basis for responsive JavaScript that loads content conditionally or sets a component's option based on screen width.

Media Queries

CSS3 media queries use the `@media` at-rule, media type, and expressions that evaluate to true or false to define the cases for which the corresponding style rules will be applied. Media queries form the basis for Oracle JET's responsive classes.

```
<style>
@media media_types (expression){
    /* media-specific rules */
}
</style>
```

The CSS3 specification defines several media types, but specific browsers may not implement all media types. The media type is optional and applies to all types if not specified. The following media query will display a sidebar only when the screen is wider than 767 pixels.

```
@media (max-width: 767px){
    .facet_sidebar {
        display: none;
    }
}
```

Oracle JET defines CSS3 media queries and style class custom properties to define screen widths for all themes included with Oracle JET.

Width and Custom Property Name	Redwood Theme: Default Range in Pixels	Alta Theme: Default Range in Pixels	Device Examples
small \$screenSmallRange	0-599	0-767	phones
medium \$screenMediumRange	600-1023	768-1023	tablet portrait
large \$screenLargeRange	1024-1439	1024-1280	tablet landscape, desktop
extra large \$screenXlargeRange	1440 and up	1281 and up	large desktop

For printing, Oracle JET uses the large screen layout for printing in landscape mode and the medium screen layout for printing in portrait mode.

Oracle JET's size defaults and media queries are defined in the Sass variables contained in `site_root/scss/oj/9.2.0/common/_oj.common.variables.scss` and are used in the grid, form, and responsive helper style classes. The following code sample shows the responsive screen width variables and a subset of the responsive media queries. In most cases the defaults are sufficient, but be sure to check the file for additional comments that show how you might modify the variables for your application if needed.

```
// responsive screen widths
$screenSmallRange: 0, 767px !default;
$screenMediumRange: 768px, 1023px !default;
$screenLargeRange: 1024px, 1280px !default;
$screenXlargeRange: 1281px, null !default;

// responsive media queries
$responsiveQuerySmallUp: "print, screen" !default;
$responsiveQuerySmallOnly: "screen and (max-width: #{upper-bound($screenSmallRange)})" !default;
```

```

$responsiveQueryMediumUp:   "print, screen and (min-width: #{lower-
bound($screenMediumRange)})" !default;
$responsiveQueryMediumOnly: "print and (orientation: portrait), screen
and (min-width: #{lower-bound($screenMediumRange)}) and (max-width:
#{upper-bound($screenMediumRange)})" !default;
$responsiveQueryMediumDown: "print and (orientation: portrait), screen
and (max-width: #{upper-bound($screenMediumRange)})" !default;

$responsiveQueryLargeUp:    "print and (orientation: landscape), screen
and (min-width: #{lower-bound($screenLargeRange)})" !default;
$responsiveQueryLargeOnly:  "print and (orientation: landscape), screen
and (min-width: #{lower-bound($screenLargeRange)}) and (max-width:
#{upper-bound($screenLargeRange)})" !default;
$responsiveQueryLargeDown:  "print and (orientation: landscape), screen
and (max-width: #{upper-bound($screenLargeRange)})" !default;

$responsiveQueryXlargeUp:   "screen and (min-width: #{lower-
bound($screenXlargeRange)})" !default;
$responsiveQueryXlargeOnly: null !default;
$responsiveQueryXlargeDown: null !default;

$responsiveQueryXXlargeUp:  null !default;

$responsiveQueryPrint:      null !default;

```

Responsive media queries are based on the screen widths defined in the `$screen{size}Range` variables and a range qualifier. For example:

- `$responsiveQuerySmallUp` applies to all screens in the `$screenSmallRange` or wider.
- `$responsiveQuerySmallOnly` applies only to screens in the `$screenSmallRange`.
- `$responsiveQueryXlargeDown` applies to all screens in the `$screenXlargeRange` and narrower.

For additional information about Oracle JET's use of Sass and theming, see [Using CSS and Themes in Applications](#).

For additional information about CSS3 media queries, see https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries and <http://www.w3.org/TR/css3-mediaqueries>.

Oracle JET Flex, Grid, Form, and Responsive Helper Class Naming Convention

The Oracle JET flex, grid, form, and responsive style classes use the same naming convention which can help you identify the style size, function, and number of columns the class represents.

Each class follows the same format as shown below:

`oj-size-function-[1-12]columns`

Size can be one of `sm`, `md`, `lg`, `xl`, and `print` and are based on the media queries described in [Media Queries](#). Oracle JET will apply the style to the size specified and any larger sizes unless *function* is defined as `only`. For example:

- `oj-lg-hide` hides content on large and extra-large screens.
- `oj-md-only-hide` hides content on medium screens. The style has no effect on other screen sizes.

You can find a summary of the classes available to you for responsive design in Oracle JET in the [Oracle® JavaScript Extension Toolkit \(JET\) Styling Reference](#).

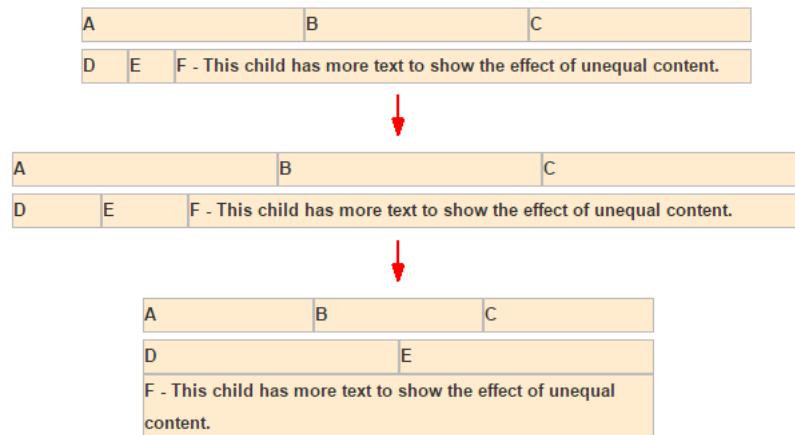
Oracle JET Flex Layouts

Use the Oracle JET `oj-flex` and `oj-flex-item` classes to create flexible box layouts that are based on the CSS flexible box layout model.

In the flex layout model, you create flex containers with children that you can lay out in any direction or order. As the available unused space grows or shrinks, the children grow to fill the unused space or shrink to avoid overflowing the parent.

To create a basic flex layout, add the `oj-flex` class to a container element (HTML `div`, for example) and then add the `oj-flex-item` class to each of the container's children.

The following image shows an example of a default flex layout using the Oracle JET flex box styles. The sample contains two flex containers, each with three children. As the screen size widens, the flex container allocates unused space to each of the children. As the screen size shrinks below the width of one of the flex items, the flex container will wrap the content in that item as needed to no wider than the maximum display width. In this example, this has the effect of causing the F child to wrap to the next row.



The markup for this flex layout is shown below, with the flex layout classes highlighted in bold. The `demo-flex-display` class sets the color, font weight, height, and border around each flex item in the layout.

```
<div id="container">
  <div class="demo-flex-display">
    <div class="oj-flex">
```

```
<div class="oj-flex-item">A</div>
<div class="oj-flex-item">B</div>
<div class="oj-flex-item">C</div>
</div>

<div class="oj-flex">
  <div class="oj-flex-item">D</div>
  <div class="oj-flex-item">E</div>
  <div class="oj-flex-item">F - This child has more text to show
the effect of unequal content.</div>
</div>
</div>
</div>
```

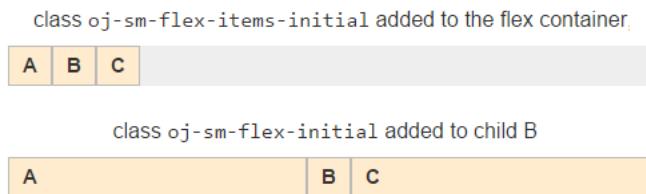
You can customize the flex layout using styles detailed in [Flex Layout Styling](#) and described below.

About Modifying the `flex` Property

Oracle JET provides layout classes that you use to modify the properties of your responsive layout.

The Oracle JET layout classes are based on the [CSS Flexible Box Layout Module](#) which defines CSS flex common values for flex item sizing. By default, Oracle JET's flex layout defaults to the auto CSS flex property which allows a flex item to shrink or grow as needed for responsive layouts. However, the CSS model sets the default `flex` property to `initial`, which allows a flex item to shrink but will not allow it to grow.

You can achieve the same effect by adding the `oj-sm-flex-items-initial` class to the flex container to set the `flex` property to `initial` for all child flex items, or add the `oj-sm-flex-initial` class to an individual flex item to set its property to `initial`. The following image shows the effect.



The code sample below shows the markup. In this example, padding is also added to the content using the `oj-flex-items-pad` class on the parent container.

```
<div id="container">
  <div class="demo-flex-display oj-flex-items-pad">
    <div class="oj-flex oj-sm-flex-items-initial">
      <div class="oj-flex-item">A</div>
      <div class="oj-flex-item">B</div>
      <div class="oj-flex-item">C</div>
    </div>

    <div class="oj-flex">
```

```
<div class="oj-flex-item">A</div>
<div class="oj-sm-flex-initial oj-flex-item">B</div>
<div class="oj-flex-item">C</div>
</div>
</div>
</div>
```

You can also override the default `auto` flex property by using the `oj-size-flex-items-1` class on the flex container. This class sets the `flex` property to 1, and all flex items in the flex container with a screen size of `size` or higher will have the same width, regardless of the items' content.

class `oj-sm-flex-items-1` added to the flex container, which sets '`flex: 1`' on all children

A	B	C
A	B	C - This child has more text to show the effect of unequal content.

To set the `flex` property to 1 on an individual flex item, add `oj-sm-flex-1` to the flex item. The [Flex Layouts](#) section in the Oracle JET Cookbook includes the examples used in this section that you can use and modify to observe the flex layout's responsive behavior.

About Wrapping Content with Flex Layouts

You can set the `flex-wrap` property to `nowrap` by adding `oj-sm-flex-wrap-nowrap` to the `oj-flex` container.

By default, Oracle JET sets the CSS `flex-wrap` property to `wrap`, which sets the flex container to multi-line. Child flex items will wrap content to additional lines when the screen width shrinks to less than the width of the flex item's content. However, the CSS model sets the `flex-wrap` property to `nowrap`, which sets the flex container to single-line. When a child item's content is too wide to fit on the screen, the content will wrap within the child.

The following image shows the effect of changing the `flex-wrap` property to `nowrap`.

A	B
C - This child has more text to show the effect of unequal content.	

add class '`oj-sm-flex-wrap-nowrap`'



A	B	C - This child has more text to show the effect of unequal content.
---	---	---

About Customizing Flex Layouts

You can customize an Oracle JET flex layout by adding the appropriate style to the flex container or child. The flex layout classes support some commonly-used values.

- `flex-direction`
- `align-items`
- `align-self`
- `justify-content`
- `order`

The Oracle JET Cookbook includes examples for customizing your flex layout at: [Flex Layouts](#).

Oracle JET Grids

Use the Oracle JET grid classes with flex layouts to create grids that vary the number and width of columns based on the width of the user's screen.

Topics:

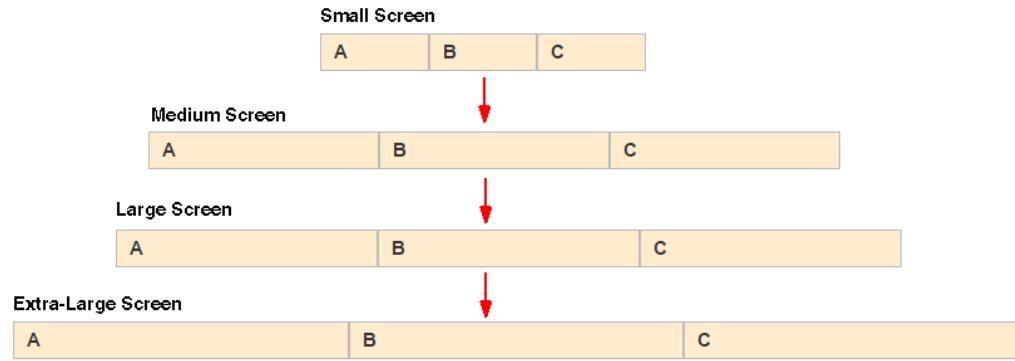
- [About the Grid System](#)
- [The Grid System and Printing](#)
- [Grid Convenience Classes](#)

The [Responsive Grids](#) section in the Oracle JET Cookbook provides several examples and recipes for using the Oracle JET grid system, and you should review them to get accustomed to the grid system.

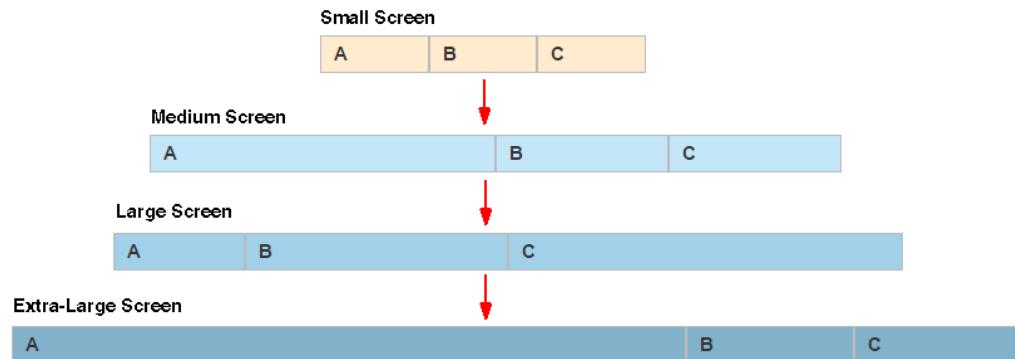
About the Grid System

Oracle JET provides a 12-column responsive mobile-first grid system that you can use for responsive design. The grid builds upon the Oracle JET flex layout and allows you to specify the widths of each flex item using sizing classes for small, medium, large, and extra-large screens.

For example, you can use the grid classes to change the default display in the [Flex Layouts Auto](#) example to use different widths for the flex items when the screen size changes. As shown in the image below, the flex layout by default will allocate the unused space evenly to the three flex items regardless of the screen size.



When the defaults are not sufficient, you can specify relative widths for the flex items when the screen size changes. In the following image, the flex layout is using grid classes to define different widths when the screen size changes to medium, large, and extra large.



The grid classes follow the [Oracle JET Flex, Grid, Form, and Responsive Helper Class Naming Convention](#). Use `oj-size-numberofcolumns` to set the width to the specified `numberofcolumns` when the screen is the specified size or larger. For example:

- `oj-sm-6` works on all screen sizes and sets the width to 6 columns.
- `oj-lg-3` sets the width to 3 columns on large and extra-large screens.
- `oj-sm-6` and `oj-lg-3` on the same flex item sets the width to 6 columns wide on small and medium screens and 3 columns wide on large and extra-large screens.

Design for the smallest screen size first and then customize for larger screens as needed. You can further customize the grid by adding one of the [Grid Convenience Classes](#) or by using one of the responsive helper classes described in [Use the Responsive Helper Classes](#).

The following code sample shows the markup for the modified Flex Auto Layout display, with grid classes defined for medium, large, and extra-large screens.

```
<div class="oj-flex">
  <div class="oj-md-6 oj-lg-2 oj-xl-8 oj-flex-item">A</div>
  <div class="oj-md-3 oj-lg-4 oj-xl-2 oj-flex-item">B</div>
```

```
<div class="oj-md-3 oj-lg-6 oj-xl-2 oj-flex-item">C</div>
</div>
```

When the screen size is small, the flex layout default styles are used, and each item uses the same amount of space. When the screen size is medium, the A flex item will use 6 columns, and the B and C flex items will each use 3 columns. When the screen size is large, The A flex item will use 2 columns, the B flex item will use 4 columns, and the C flex item will use 6 columns. Finally, when the screen size is extra large, the A flex item will use 8 columns, and the B and C flex items will each use 2 columns.

For a complete example that illustrates working with the grid system, see [Responsive Grids](#).

The Grid System and Printing

The Oracle JET grid system applies the large styles for printing in landscape mode and the medium style for printing in portrait mode if they are defined. You can use the defaults or customize printing using the print style classes.

In the grid example below, Row 2 and Row 4 include the `oj-md-*` style classes. Row 3 and Row 4 include the `oj-lg-4` style for all columns in the row.

```
<div class="demo-grid-sizes demo-flex-display">
  <div class="oj-flex oj-flex-items-pad">
    <div class="oj-sm-9 oj-flex-item"></div>
    <div class="oj-sm-3 oj-flex-item"></div>
  </div>
  <div class="oj-flex oj-flex-items-pad">
    <div class="oj-sm-6 oj-md-9 oj-flex-item"></div>
    <div class="oj-sm-6 oj-md-3 oj-flex-item"></div>
  </div>
  <div class="oj-flex oj-flex-items-pad">
    <div class="oj-sm-6 oj-lg-4 oj-flex-item"></div>
    <div class="oj-sm-4 oj-lg-4 oj-flex-item"></div>
    <div class="oj-sm-2 oj-lg-4 oj-flex-item"></div>
  </div>
  <div class="oj-flex oj-flex-items-pad ">
    <div class="oj-sm-8 oj-md-6 oj-lg-4 oj-xl-2 oj-flex-item"></div>
    <div class="oj-sm-2 oj-md-3 oj-lg-4 oj-xl-8 oj-flex-item"></div>
    <div class="oj-sm-2 oj-md-3 oj-lg-4 oj-xl-2 oj-flex-item"></div>
  </div>
</div>
```

As shown in the following print preview, when you print this grid in landscape mode, the `oj-lg-4` style classes will be applied on Row 3 and Row 4. When you print the grid in portrait mode, the `oj-md-*` style classes apply on Row 2 and Row 4.

Print Preview - Landscape

Row 1	S-9	S-3
Row 2	M-9	M-3
Row 3	L-4	L-4
Row 4	L-4	L-4

Print Preview - Portrait

Row 1	S-9	S-3
Row 2	M-9	M-3
Row 3	S-6	S-4
Row 4	M-6	M-3

If you want to change the printing default, you can set the Sass `$responsiveQueryPrint` variable to print in a custom settings file. After you enable the print classes, you can add the `oj-print-numberofcolumns` style class to the column definition. This has the effect of changing the column sizes for printing purposes only. In the following example, Row 1 includes the `oj-print-6` class for each column in the row.

```
<div class="oj-flex oj-flex-items-pad">
  <div class="oj-sm-9 oj-print-6 oj-flex-item"></div>
  <div class="oj-sm-3 oj-print-6 oj-flex-item"></div>
</div>
```

In normal mode, Row 1 contains two columns, one column with a size of 9 and one column with a size of 3, regardless of screen size. If you do a print preview, however, you'll see that Row 1 will print with two columns in portrait and landscape mode, both with a size of 6.

Row 1 - Normal Display

Row 1	S-9	S-3
-------	-----	-----

Print Preview - Landscape

Row 1	p-6	p-6
-------	-----	-----

Print Preview - Portrait

Row 1	p-6	p-6
-------	-----	-----

For information about setting Sass variables in a custom settings file, see [Customize Alta Themes Using the Tooling](#).

Grid Convenience Classes

Oracle JET's grid system includes convenience classes that make it easier to create two- and four- column layouts with specified widths.

- `oj-size-odd-cols-numberofcolumns`: Use this in a 2-column layout. Instead of putting sizing classes on every column, you can put a single class on the flex parent. The number of columns specifies how many of the 12 columns the odd-numbered columns can use. In a 2-column layout, the even-numbered columns will take up the remainder of the columns.

For example, setting `oj-md-odd-cols-4` on the flex parent will have the effect of setting the odd column (`col1`) width to 4 and the even column (`col2`) width to 8 for all rows in the grid on medium-size screens and higher.

col 1	col 2
col 1	col 2
col 1	col 2

The code sample below shows the grid configuration used to render the figure. The example also sets `oj-sm-odd-cols-12` which will set the odd column width to 12 on small screens, displaying `col2` on a new row.

```
<div class="oj-md-odd-cols-4 oj-flex-items-pad">
  <div class="oj-flex">
    <div class="oj-flex-item">col 1</div>
    <div class="oj-flex-item">col 2</div>
  </div>
  <div class="oj-flex">
    <div class="oj-flex-item">col 1</div>
    <div class="oj-flex-item">col 2</div>
  </div>
  <div class="oj-flex">
    <div class="oj-flex-item">col 1</div>
    <div class="oj-flex-item">col 2</div>
  </div>
</div>
```

You could achieve the same effect by defining `oj-md-4` for the first column's width and `oj-md-8` for the second column's width on each flex item.

```
<div class="oj-flex-items-pad">
  <div class="oj-flex">
    <div class="oj-sm-12 oj-md-4 oj-flex-item">col 1</div>
    <div class="oj-sm-12 oj-md-8 oj-flex-item">col 2</div>
  </div>
  <div class="oj-flex">
    <div class="oj-sm-12 oj-md-4 oj-flex-item">col 1</div>
    <div class="oj-sm-12 oj-md-8 oj-flex-item">col 2</div>
  </div>
  <div class="oj-flex">
    <div class="oj-sm-12 oj-md-4 oj-flex-item">col 1</div>
    <div class="oj-sm-12 oj-md-8 oj-flex-item">col 2</div>
  </div>
</div>
```

`oj-size-even-cols-numberofcolumns`: Use in a 4-column layout. In this layout, you must use both the `odd-cols` class to control the width of odd-numbered columns and the `even-cols` class to control the width of the even columns.

For example, setting `oj-md-odd-cols-2` and `oj-md-even-cols-4` on the flex parent has the effect of setting the first and third column widths to 2, and the second and fourth column widths to 4.

col 1	col 2	col 3	col 4
col 1	col 2	col 3	col 4

The code sample below shows the grid configuration used to render the figure.

```
<div class="oj-sm-odd-cols-12 oj-md-odd-cols-2 oj-md-even-cols-4 oj-flex-items-pad">
  <div class="oj-flex">
    <div class="oj-flex-item">col 1</div>
    <div class="oj-flex-item">col 2</div>
    <div class="oj-flex-item">col 3</div>
    <div class="oj-flex-item">col 4</div>
  </div>
  <div class="oj-flex">
    <div class="oj-flex-item">col 1</div>
    <div class="oj-flex-item">col 2</div>
    <div class="oj-flex-item">col 3</div>
    <div class="oj-flex-item">col 4</div>
  </div>
</div>
```

If you don't use the convenience classes, you must define the size classes on every column in every row as shown below.

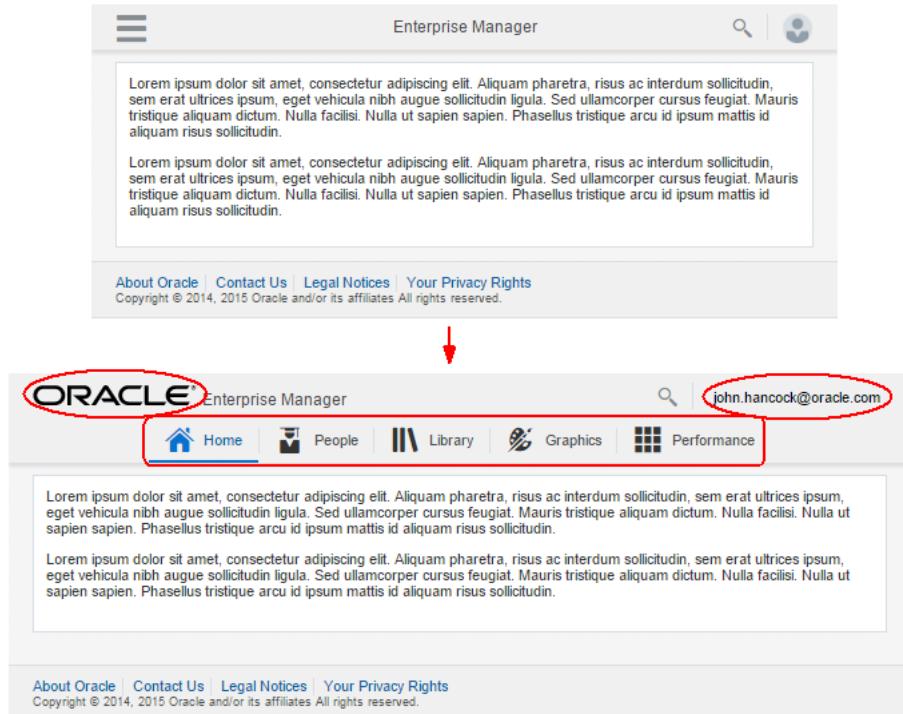
```
<div class="oj-flex-items-pad">
  <div class="oj-flex">
    <div class="oj-sm-odd-cols-12 oj-md-2 oj-flex-item">col 1</div>
    <div class="oj-sm-odd-cols-12 oj-md-4 oj-flex-item">col 2</div>
    <div class="oj-sm-odd-cols-12 oj-md-2 oj-flex-item">col 3</div>
    <div class="oj-sm-odd-cols-12 oj-md-4 oj-flex-item">col 4</div>
  </div>
  <div class="oj-flex">
    <div class="oj-sm-odd-cols-12 oj-md-2 oj-flex-item">col 1</div>
    <div class="oj-sm-odd-cols-12 oj-md-4 oj-flex-item">col 2</div>
    <div class="oj-sm-odd-cols-12 oj-md-2 oj-flex-item">col 3</div>
    <div class="oj-sm-odd-cols-12 oj-md-4 oj-flex-item">col 4</div>
  </div>
</div>
```

Responsive Layout and Content Design Patterns

Oracle JET provides application layout classes to use on your page to create responsive header, content, and footer sections. In addition, Oracle JET includes support for the column drop, layout shift, view switcher, and off-canvas responsive page content design patterns.

- [Web Application Patterns](#)

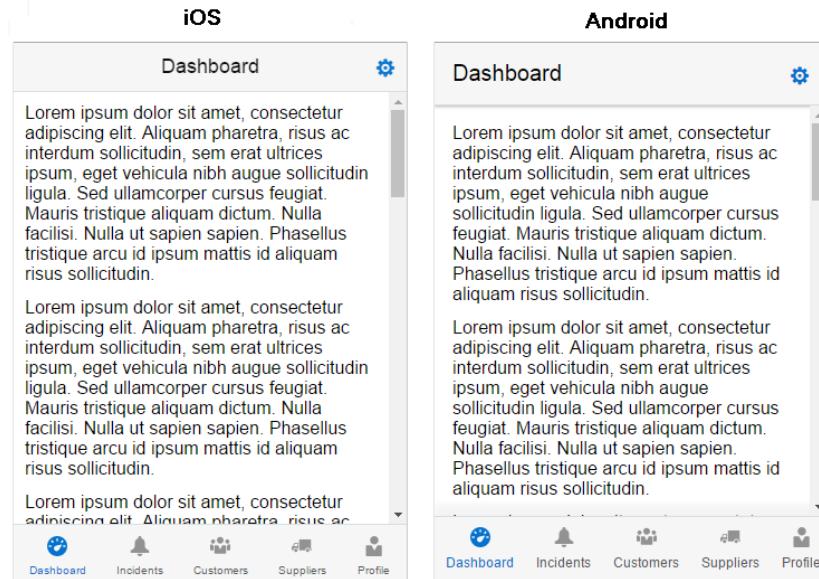
Oracle JET provides the `oj-web-applayout-*` responsive classes that you can use in conjunction with a flex layout to configure your web page for responsive behavior. The following image shows the same page displayed on a small and large screen. When the screen size increases, the page displays the Oracle logo, user's email address, and navigation bar.



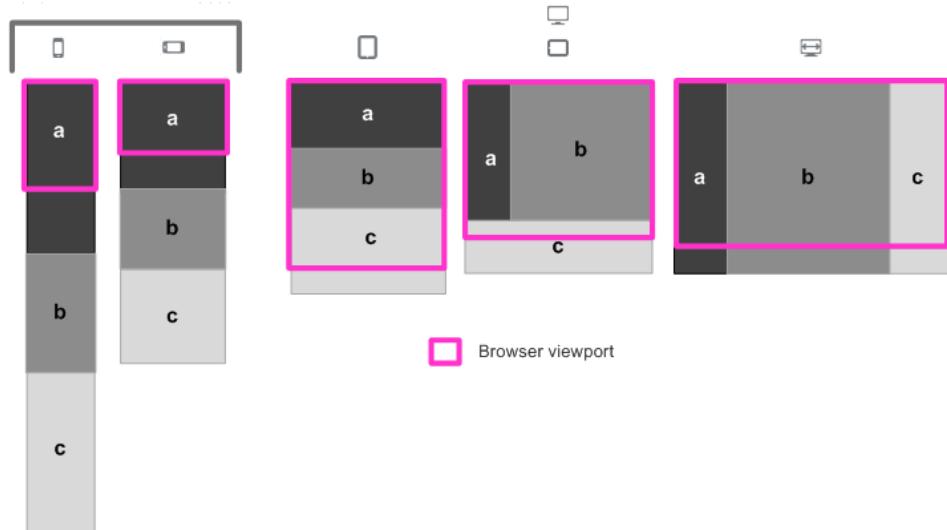
- [Hybrid Mobile Application Patterns](#)

Oracle JET provides the `oj-hybrid-applayout-*` responsive classes that you can use in conjunction with a flex layout and native theming to create hybrid mobile applications for iOS and Android mobile devices.

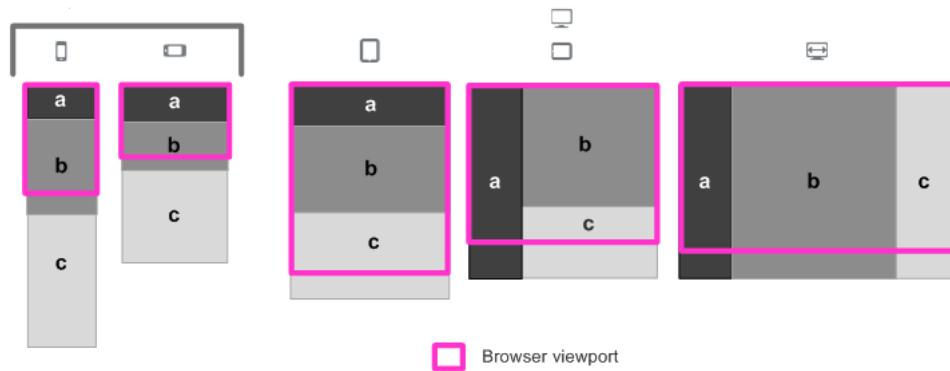
The image below shows the hybrid mobile header navBar sample pattern displayed in a desktop browser using the Alta iOS and Android themes.



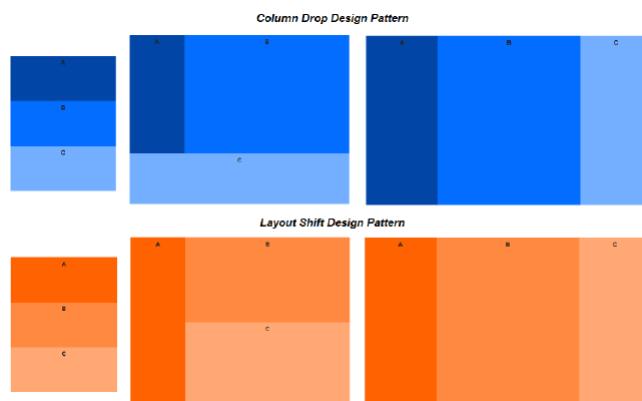
- **Column Drop:** Columns drop below each other for smaller screens. At the widest breakpoint, content displays in columns. As the display size is reduced, columns drop from the rightmost side of the display and add as rows below the remaining column(s).



- **Layout Shift:** Column layout changes for smaller screens. Layout shape may differ at each breakpoint as content is repositioned. Rather than simply adding or removing columns, for example, the leftmost column could transition to a row at the top of the display.

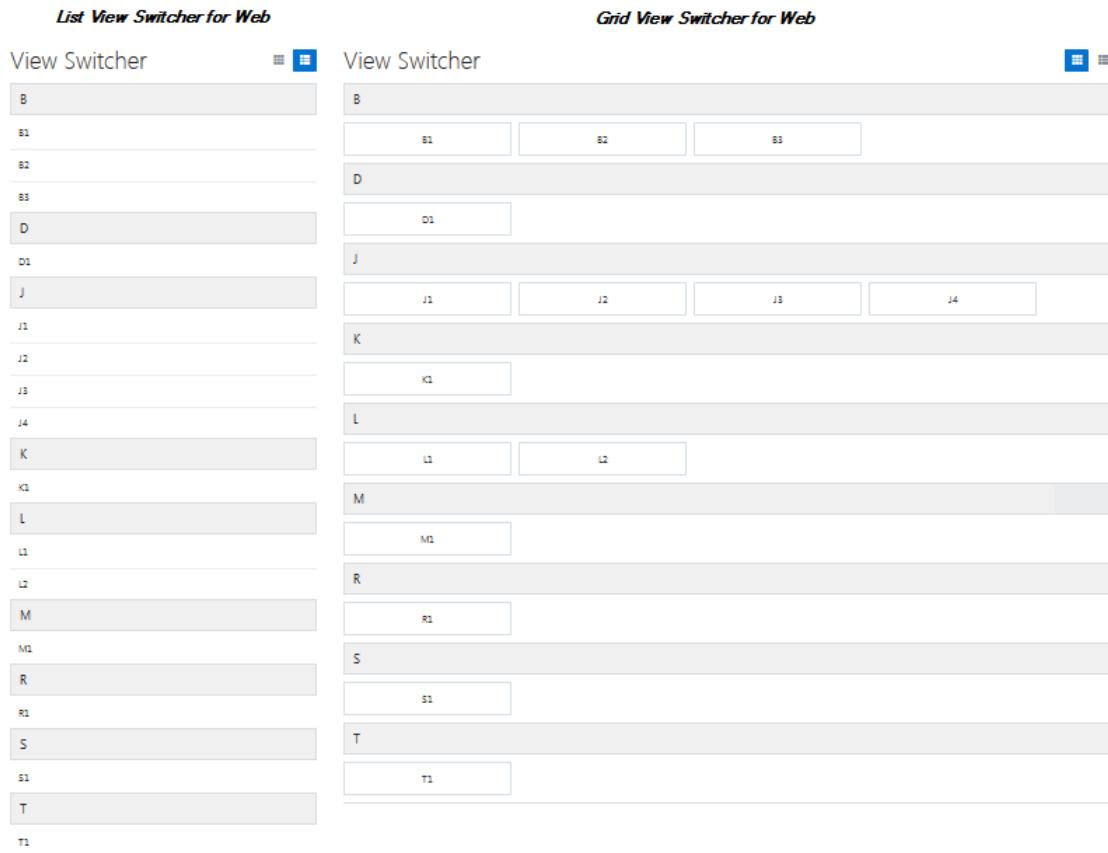


The difference between the two design patterns is subtle, and you can get a better feel for the difference by looking at the Oracle JET Cookbook examples at differing widths or on different devices. In the figure below, the column drop and layout shift patterns are shown at three different screen widths on a desktop display. In this example, the difference is noticeable in the middle figure for each pattern.

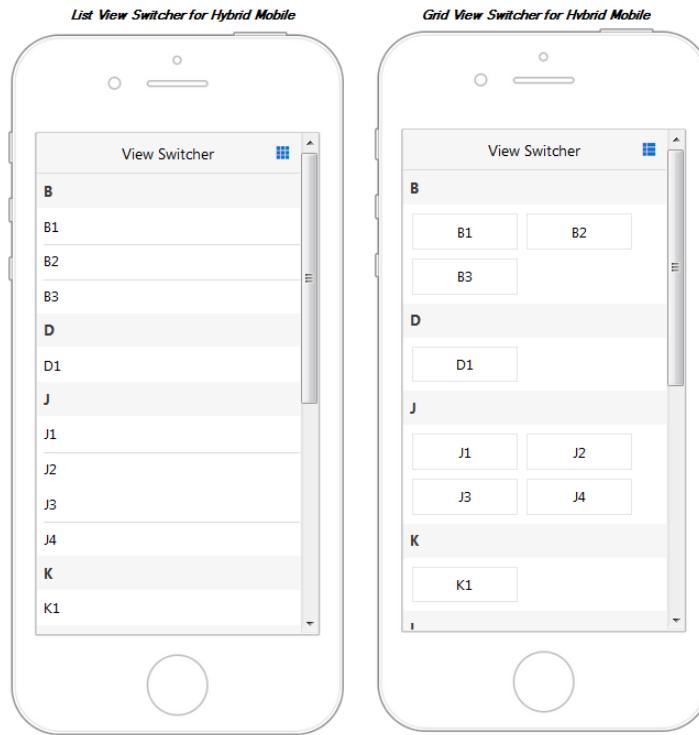


- **List - View Switcher** The View Switcher allows you to switch between list view and grid view.

The image below shows the View Switcher web layout displayed in a desktop browser. The View Switcher switches between list view and grid view as follows.

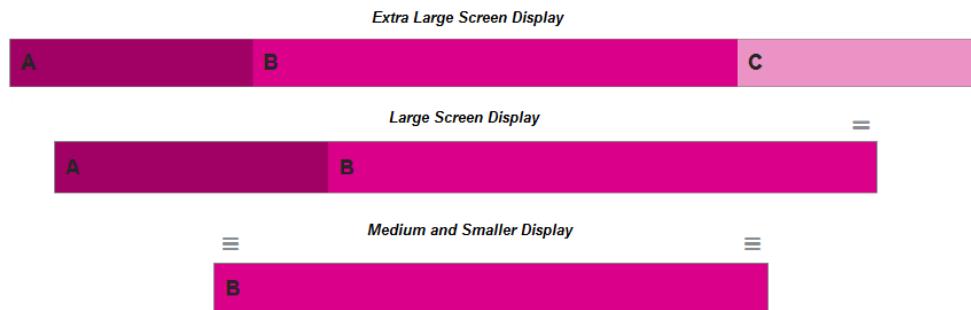


The image below shows the list view and grid view patterns of the `View_Switcher` hybrid layout on a hybrid mobile application with a fixed header.



- **Off-Canvas Push** and **Off-Canvas Overlay**: You can use Oracle JET's [OffcanvasUtils](#) to make an off-canvas partition that responsively change its position from off-screen to fixed inside the viewport when the browser width is changed.

In the image below, when the screen size is extra large, the off-canvas design shows three columns: the off-canvas partition on the start edge labeled A, the main content labeled B, and the off-canvas partition on the end edge labeled C. When the screen size is large, the off-canvas position on the end edge (C) is hidden off-screen, and the launch button indicates its availability. For medium screens and smaller, both the start edge and end edge partitions are hidden off-screen, and two launch buttons indicate their availability.



For additional information about using `offCanvasUtils` in your application layout, see [Work with offCanvasUtils](#).

Responsive Form Layouts

Oracle JET provides the `oj-form-layout` component that you can use to create form layouts that adjust to the size of the user's screen. Use the `oj-input-text` and `oj-text-area` custom elements within the `oj-form-layout` component to create an organized layout.

For more information on `oj-form-layout` component, see [Work with Form Layouts](#).

Add Responsive Design to Your Application

To create your responsive application using Oracle JET, design for the smallest device first and then customize as needed for larger devices. Add the applicable application, flex, grid, form, and responsive classes to implement the design.

To design a responsive application using Oracle JET classes:

1. Determine whether you want to use the Oracle JET [Responsive Layout and Content Design Patterns](#) or provide your own to lay out your page.
2. Design the application content's flex layout.

For help, see [Oracle JET Flex Layouts](#)

3. If the flex layout defaults are not sufficient and you need to specify column widths when the screen size increases, add the appropriate responsive grid classes to your flex items.

For help, see [Oracle JET Grids](#).

4. If you're adding a form to your page, add the appropriate form style classes.

For help, see [Responsive Form Layouts](#)

5. Customize your design as needed.

For additional information, see:

- [Use Responsive JavaScript](#)
- [Use the Responsive Helper Classes](#)
- [Create Responsive Images](#)
- [Change Default Font Size](#)

For the list of responsive design classes and their behavior, see the Responsive* classes listed in the [Oracle® JavaScript Extension Toolkit \(JET\) Styling Reference](#).

For Oracle JET Cookbook examples that implement responsive design, see:

- [Application Patterns](#)
- [Flex Layouts](#)
- [Responsive Grids](#)
- [Form Layouts](#)
- [Responsive Helpers](#)
- [Responsive JavaScript Framework Queries](#)

Use Responsive JavaScript

Oracle JET includes the `ResponsiveUtils` and `ResponsiveKnockoutUtils` utility classes that leverage media queries to change a component's `value` option or load content and images based on the user's screen size or device type.

Topics:

- [The Responsive JavaScript Classes](#)
- [Change a Custom Element's Attribute Based on Screen Size](#)
- [Conditionally Load Content Based on Screen Size](#)
- [Create Responsive Images](#)

The Responsive JavaScript Classes

The `ResponsiveUtils` and `ResponsiveKnockoutUtils` responsive JavaScript classes provide methods that you can use in your application's JavaScript to obtain the current screen size and use the results to perform actions based on that screen size. In addition, the `ResponsiveUtils` provides a method that you can use to compare two screen sizes, useful for performing actions when the screen size changes.

JavaScript Class	Methods	Description
<code>responsiveUtils</code>	<code>compare(size1, size2)</code>	<p>Compares two screen size constants. Returns a negative integer if the first argument is less than the second, a zero if the two are equal, and a positive integer if the first argument is more than the second.</p> <p>The screen size constants identify the screen size range media queries. For example, the <code>ResponsiveUtils.SCREEN_RANGE.SM</code> constant corresponds to the Sass <code>\$screenSmallRange</code> variable and applies to screen sizes smaller than 768 pixels in width.</p>
<code>responsiveUtils</code>	<code>getFrameworkQuery(frameworkQueryKey)</code>	<p>Returns the media query to use for the framework query key parameter.</p> <p>The framework query key constant corresponds to a Sass responsive query variable. For example, the <code>ResponsiveUtils.FRAMEWORK_QUERY_KEY.SM_UP</code> constant corresponds to the <code>\$responsiveQuerySmallUp</code> responsive query which returns a match when the screen size is small and up.</p>
<code>responsiveKnockoutUtils</code>	<code>createMediaQueryObservable(queryString)</code>	<p>Creates a Knockout observable that returns true or false based on a media query string. For example, the following code will return true if the screen size is 400 pixels wide or larger.</p> <pre>var customQuery = ResponsiveKnockoutUtils.createMediaQueryObservable(' (min-width: 400px) !);</pre>

JavaScript Class	Methods	Description
responsiveKnockoutUtils	createScreenRangeObservable()	<p>Creates a computed Knockout observable, the value of which is one of the <code>ResponsiveUtils.SCREEN_RANGE</code> constants.</p> <p>For example, on a small screen (0 - 767 pixels), the following code will create a Knockout observable that returns <code>ResponsiveUtils.SCREEN_RANGE.SM</code>.</p> <pre>self.screenRange = ResponsiveKnockoutUtils.createScreenRangeObservable();</pre>

For additional detail about `responsiveUtils`, see the [ResponsiveUtils API documentation](#). For more information about `responsiveKnockoutUtils`, see [ResponsiveKnockoutUtils](#).

Change a Custom Element's Attribute Based on Screen Size

You can set the value for a custom element's attribute based on screen size using the responsive JavaScript classes. For example, you may want to add text to a button label when the screen size increases using the `oj-button` element's `display` attribute.



In this example, the `oj-button` element's `display` attribute is defined for icons. The code sample below shows the markup for the button.

```
<div id="optioncontainer">
  <oj-button display="[[large() ? 'all' : 'icons']]>
    <span slot='startIcon' class="oj-fwk-icon oj-fwk-icon-calendar"></span>
    calendar
  </oj-button>
</div>
```

The code sample also sets the `oj-button` `display` attribute to `all`, which displays both the label and icon when the `large()` method returns `true`, and icons only when the `large()` method returns `false`.

The code sample below shows the code that sets the value for `large()` and completes the knockout binding. In this example, `lgQuery` is set to the `LG_UP` framework query key which applies when the screen size is large or up. `this.large` is initially set to `true` as the result of the call to `ResponsiveKnockoutUtils.createMediaQueryObservable(lgQuery)`. When the screen changes to a smaller size, the `self.large` value changes to `false`, and the `display` attribute value becomes `icons`.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojresponsiveutils', 'ojs/ojresponsiveknockoututils', 'ojs/ojknockout', 'ojs/ojbutton'],
  function(ko, Bootstrap, ResponsiveUtils, ResponsiveKnockoutUtils)
{
  function MyModel(){
    // observable for large screens
```

```
var lgQuery = ResponsiveUtils.getFrameworkQuery(  
    ResponsiveUtils.FRAMEWORK_QUERY_KEY.LG_UP);  
  
this.large =  
ResponsiveKnockoutUtils.createMediaQueryObservable(lgQuery);  
}  
  
Bootstrap.whenDocumentReady().then(  
    function ()  
    {  
        ko.applyBindings(new MyModel(),  
document.getElementById('optioncontainer'));  
    }  
);  
});
```

The Oracle JET Cookbook contains the complete code for this example which includes a demo that shows a computed observable used to change the button label's text depending on the screen size. You can also find examples that show how to use custom media queries and Knockout computed observables. For details, see [Responsive JavaScript Framework Queries](#).

For additional information about working with oj-button, see [Work with Buttons](#).

Conditionally Load Content Based on Screen Size

You can change the HTML content based on screen size using the responsive JavaScript classes. For example, you might want to use a larger font or a different background color when the screen size is large.

Small and Medium Screens

This is the content in the sm/md template.

Large and Extra Large Screens

This is the content in the lg/xl template.

In this example, the HTML content is defined in Knockout templates. The markup uses Knockout's data-bind utility to display a template whose name depends on the value returned by the `large()` call. If the screen is small or medium, the application will use the `sm_md_template`. If the screen is large or larger, the application will use the `lg_xl_template`.

```
<div id="sample_container">  
  
    <!-- large template -->  
    <script type="text/html" id="lg_xl_template">  
        <div id="lg_xl"  
            style="background-color:lavenderblush;  
            padding: 10px; font-size: 22px" >  
            This is the content in the <strong>lg/xl</strong> template.  
        </div>  
    </script>
```

```
<!-- small template -->
<script type="text/html" id="sm_md_template">
<div id="sm_md"
    style="background-color:lightcyan;
           padding: 10px; font-size: 10px" >
    This is the content in the <strong>sm/md</strong> template.
</div>
</script>

<!-- display template -->
<div data-bind="template: {name: large() ? 'lg_xl_template' :
                           'sm_md_template'}"></div>
</div>
```

The code that sets the value for `large()` is identical to the code used for setting component option changes. For details, see [Change a Custom Element's Attribute Based on Screen Size](#).

For the complete code used in this example, see the [Responsive Loading with JavaScript](#) demo in the Oracle JET Cookbook.

Create Responsive Images

You can use the responsive JavaScript classes to load a different image when the screen size changes. For example, you may want to load a larger image when the screen size changes from small and medium to large and up.



In this example, the image is defined in a HTML `img` element. The markup uses Oracle JET's attribute binding to display a larger image when the `large()` call returns `true`.

```
<div id="container">
    <p>current width:<br/>
        <strong>
            <span>
                <oj-bind-text value="[[large() ? 'large' : 'not large']]></oj-bind-text>
            </span>
        </strong>
    </p>

    
</div>
```

The code that set the value for `large()` is identical to the code used for setting component option changes. For details, see [Change a Custom Element's Attribute Based on Screen Size](#).

 **Note:**

The image will not begin to load until the JavaScript is loaded. This could be an issue on devices with slower connections. If performance is an issue, you can use responsive CSS as described in [Create Responsive CSS Images](#). You could also use the HTML picture element which supports responsive images without CSS or JavaScript. However, browser support is limited and may not be an option for your environment.

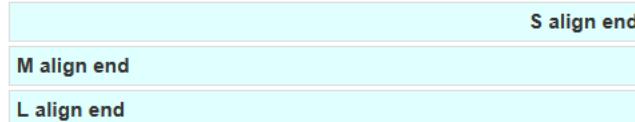
For the complete code used in this example, see the Oracle JET Cookbook [Responsive Images](#) demos.

Use the Responsive Helper Classes

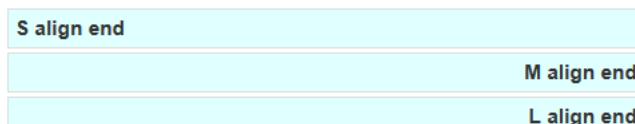
Use the Oracle JET generic responsive utility classes to hide content, end align text, and set float in your grid.

- `oj-size-hide`: Hide content at the specified `size`.
- `oj-size-text-align-end`: In left-to-right languages, set `text-align` to right. In right-to-left languages, set `text-align` to left.

Text End Alignment - Left to Right



Text End Alignment - Right to Left



- `oj-size-float-end`: In left-to-right languages, set `float` to right. In right-to-left languages, set `float` to left.

Float End - Left to Right



Float End - Right to Left



- `oj-size-float-start`: In left-to-right languages, set `float` to `left`. In right-to-left languages, set `float` to `right`.

To see examples that implement the responsive helper classes, consult the Oracle JET Cookbook at [Responsive Helpers](#).

Create Responsive CSS Images

Use CSS generated from Sass media query variables and responsive queries to use a different, larger image when the screen width changes from small to large.



The code below shows the markup that defines the image. In this example, `bulletlist` is a CSS class generated from the Sass responsive variables and media queries.

```
<div role="img" class="oj-icon bulletlist" title="bulleted list image"></div>
```

The following image shows the `bulletlist` CSS class. When the screen is small or medium size, the `icon_small.png` image loads. When the screen increases to large or larger, or to print, the `icon.png` loads instead.

```
.bulletlist {  
    width: 24px;  
    height: 24px; }  
.bulletlist:before {  
    content: url("images/hiResContrast/icon_small.png"); }  
@media print and (orientation: landscape), screen and (min-width: 1024px) {  
    .bulletlist {  
        width: 48px;  
        height: 48px; }  
    .bulletlist:before {  
        content: url("images/hiResContrast/icon.png"); } }
```

The Oracle JET Cookbook includes the Sass variables and queries used to generate the `bulletlist` CSS class. You can also find a Sass mixin that makes generating the CSS easier, but you are not required to use SCSS to create responsive CSS images.

In addition, the Oracle JET Cookbook includes examples that show high resolution images, sprites, button images, and more. For details, see [Responsive CSS Images](#).

Note:

You can also use responsive JavaScript to change the images based on screen size. For details, see [Create Responsive Images](#).

Change Default Font Size

By default, Oracle JET includes the Redwood theme, starting in JET release 9.0.0, that set a default font size of 1em (16px) on the root (`html`) element. This font size is

optimized for visibility and touchability on mobile devices, but you can customize the size as needed for your application.

Topics

- [Change Default Font Size Across the Application](#)
- [Change Default Font Size Based on Device Type](#)

Change Default Font Size Across the Application

The browser's font size is defined in the Sass `$rootFontSize` variable and included in the generated CSS `html` class. You can use Sass to change the variable or override the generated CSS.

To change the browser default font size across your application, do *one* of the following:

- In a custom Sass settings file, modify the Sass `$rootFontSize` variable, and regenerate the CSS.
For details about customizing Oracle JET themes, see [Customize Alta Themes Using the Tooling](#).
- In your application-specific CSS, override the font-size setting for the `html` class.
For example, to set the browser's default font size to 12 pixels, add the following to your application-specific CSS:

```
html {  
    font-size: 12px;  
}
```

Change Default Font Size Based on Device Type

You can change the default font size based on device type by detecting the device type used to access the browser and then setting the appropriate style on the `html` element.

To change the browser default font size based the user's device type:

1. Use whatever means you like to detect that the browser is running on the specified device.
For example, you may want to change the browser's default font size on a desktop device. Use your own code or a third party tool to detect the device type.
2. When your application detects that the user is accessing the browser with the specified device, on the `html` element in your markup, set `style="font-size: xxpx"`. Substitute the desired pixel size for `xx`.
For example, to set the font size to 12 pixels when the application detects the specified device, add logic to your application to add the highlighted code to your markup.

```
<html style="font-size: 12px">  
    ... contents omitted  
</html>
```

Perform this step before initializing components to avoid issues with some Oracle JET components that measure themselves.

Control the Size and Generation of the CSS

You can change the size of the CSS content automatically generated by Oracle JET so that unused classes or particular types of classes are compressed, removed, excluded, or not generated.

When you use the responsive framework classes, Oracle JET generates a large number of classes that you may not need. Here are some steps you can take to control the size and generation of the CSS.

- Use compression.

The responsive classes are often repetitive and compress well. For details about compressing your CSS, see [Optimizing Performance](#).

- Remove unused classes.

By default, Oracle JET generates responsive classes small, medium, large, and xlarge screens. If you know that your application will not use some of these classes, you can set the associated \$responsiveQuery* variables to none.

```
// If you don't want xlarge classes, you could set:  
$screenXlargeRange: none;  
$responsiveQueryLargeOnly: none;  
$responsiveQueryXlargeUp: none;
```

- Exclude unused classes from the application layout, flex, grid, form layout, and responsive helper groups.

You can use the following variables to exclude classes from these groups altogether:

- \$includeAppLayoutClasses
- \$includeAppLayoutWebClasses
- \$includeFlexClasses
- \$includeGridClasses
- \$includeFormLayoutClasses
- \$includeResponsiveHelperClasses

For additional information about using the \$include* variables, see [Use Variables to Control CSS Content](#).

- Stop generation of a particular responsive helper class.

For finer-grained control, there are additional variables that you can set to false to generation of a particular type of class.

Variable	Description
\$responsiveGenerateHide	Generate hide classes like .oj-md-hide.
\$responsiveGenerateTextAlignEnd	Generate text-align end classes like .oj-md-text-align-end.

Variable	Description
\$responsiveGenerat eFloatStart	Generate float start classes like .oj-md-float-start.
\$responsiveGenerat eFloatEnd	Generate float end classes like .oj-md-float-end.

5

Using RequireJS for Modular Development

Oracle JET includes RequireJS, a third party JavaScript library that you can use in your application to load only the Oracle JET libraries you need. Using RequireJS, you can also implement lazy loading of modules or create JavaScript partitions that contain more than one module.

Topics:

- [Typical Workflow for Using RequireJS](#)
- [About Oracle JET and RequireJS](#)
- [Use RequireJS in an Oracle JET Application](#)
- [Add Third-Party Tools or Libraries to Your Oracle JET Application](#)
- [Troubleshoot RequireJS in an Oracle JET Application](#)
- [About JavaScript Partitions and RequireJS in an Oracle JET Application](#)

Typical Workflow for Using RequireJS

Understand Oracle JET's module organization before using RequireJS in an Oracle JET application. You should also understand why you might use RequireJS in your application. For some features, you must use RequireJS. In other cases, its use is optional.

To start using RequireJS for modular development in Oracle JET, refer to the typical workflow described in the following table:

Task	Description	More Information
Understand Oracle JET's use of RequireJS	Understand Oracle JET's module organization and why you might use RequireJS in your Oracle JET application.	About Oracle JET and RequireJS
Configure RequireJS	Configure RequireJS in your Oracle JET application to load Oracle JET modules as needed.	Use RequireJS in an Oracle JET Application
(Optional) Add third-party tools or libraries to your Oracle JET application	Understand the process to add third-party tools or libraries to your Oracle JET application.	Add Third-Party Tools or Libraries to Your Oracle JET Application
Troubleshoot RequireJS	Troubleshoot issues with RequireJS.	Troubleshoot RequireJS in an Oracle JET Application
(Optional) Use JavaScript partitions and RequireJS in an Oracle JET application.	Configure RequireJS bundles.	About JavaScript Partitions and RequireJS in an Oracle JET Application

About Oracle JET and RequireJS

RequireJS is a JavaScript file and module loader that simplifies managing library references and is designed to improve the speed and quality of your code.

RequireJS implements the Asynchronous Module Definition (AMD) API which provides a mechanism for asynchronously loading a module and its dependencies.

Oracle JET's modular organization enables application developers to load a subset of needed features without having to execute `require()` calls for each referenced object. Each Oracle JET module represents one functional area of the toolkit, and it typically defines more than one JavaScript object.

You do not have to use RequireJS to reference Oracle JET libraries, but it is required if you plan to use Oracle JET's internationalization or data visualization components in your application. The Oracle JET download includes the RequireJS library, and it is used by default in the Oracle JET Starter Templates and Cookbook examples.

For more information about RequireJS, see <http://requirejs.org>.

About Oracle JET Module Organization

The Oracle JET modules are listed in the following table with description and usage tips. Use this table to determine which modules you must load in your application. Where your application can directly interact with a module API, the available objects that the module returns also appear in the table. Your application would typically call functions on the returned object or instantiate new objects via the constructor function. For more information about module loading in Oracle JET applications, see [API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\) - JET Module Loading Overview](#).

 **Note:**

Certain functionality that had been previously available from the now deprecated `ojcore`, `ojvalidation-base`, `ojvalidation-datetime`, and `ojvalidation-number` modules is provided by refactored modules that return their own object. The table indicates which modules have been refactored. You should explicitly import needed modules in the dependency list of your `require` function to use its classes.

Oracle JET Module	Refactored	Available Objects	Description	When to Use
<code>ojs/ojmodel</code>	No	<code>Collection</code> <code>Events</code> <code>Model</code> <code>OAuth</code> <code>URLError</code>	Classes of the JET Common Model	Use if your application uses the Oracle JET Common Model.

Oracle JET Module	Refactored	Available Objects	Description	When to Use
ojs/ojknockout-model	No	KnockoutUtils	Utilities for integrating Oracle JET's Common Model into Knockout.js	Use if your application uses the Oracle JET Common Model, and you want to integrate with Knockout.js.
ojs/ojcomponent	No	Varies by component	<p>Oracle JET component modules. Examples include</p> <ul style="list-style-type: none"> • ojs/ojbutton • ojs/ojtoolbar • ojs/ojtabs <p>Most Oracle JET components have their own module with the same name in lowercase and without hyphens as shown above, except for the following components:</p> <ul style="list-style-type: none"> • oj-buttonset-*: ojs/ojbutton • oj-input-password: ojs/ojinputtext • oj-text-area: ojs/ojinputtext • oj-combobox-*: ojs/ojselectcombobox • oj-select-*: ojs/ojselectcombobox • oj-spark-chart: ojs/ojchart • oj-*gauge: ojs/ojgauge 	Use component modules that correspond to any Oracle JET component in your application.
ojs/ojknockout	No	ComponentBinding	Oracle JET data binding for global attributes of any HTML element in the user interface	Use when your application includes HTML elements (JET custom elements included) with databound global attributes (ones that use the : (colon) prefix) or that use the [[.]]/{{.}} syntax (for global attributes of custom elements).
ojs/ojcorerouter	No	urlParamAdapter, urlPathAdapter, CoreRouter, CoreRouterState	Class for managing routing in single page applications	Use if your single page application needs to manage routing.

Oracle JET Module	Refactored	Available Objects	Description	When to Use
ojs/ojmodule	No	ModuleBinding	Binding that implements navigation within a region of a single page application	Use if your single page application needs to manage navigation within a page region.
ojs/ojmodule-element	No	ModuleElementAnimation	Component that implements navigation within a region of a single page application	Use if your single page application needs to manage navigation within a page region.
ojs/ojmoduleanimations	No	ModuleAnimations	Used in conjunction with ojs/ojmodule-element. Adds animation support via CSS animation effects.	Use if your application adds animation effects.
ojs/ojcontext	Yes, from ojcore	BusyContext, Context	Class that exposes the BusyContext that keeps track of components that are currently animating or fetching data.	Use if your application needs to query the busy state of components on the page.
ojs/ojconfig	Yes, from ojcore	Config	Class for setting and retrieving configuration options.	Use if your application needs to set or retrieve application configuration details.
ojs/ojlogger	Yes, from ojcore	Logger	Utilities for setting up a logger and collecting logging information	Use if your application needs to define logger callback functions.
ojs/ojresponsiveutils	Yes, from ojcore	ResponsiveUtils	Utilities for working with responsive screen widths and ranges. Often used in conjunction with ojs/ojresponsiveknockoututils to create knockout observables that can be used to drive responsive page behavior.	Use if your application needs to work with responsive page design.
ojs/ojthemeutils	Yes, from ojcore	ThemeUtils	Utilities for getting theme information	Use if your application needs to know about the theme's fonts or the target platform.
ojs/ojtimeutils	Yes, from ojcore	TimeUtils	Utilities for time information	Use if your application needs to calculate the position of a given time point within a range.
ojs/ojtranslation	Yes, from ojcore	Translations	Service for retrieving translated resources	Use if your application needs to work with translated resources.

Oracle JET Module	Refactored	Available Objects	Description	When to Use
ojs/ojattributegroupHandler	No	AttributeGroupHandle, ColorAttributeGroupHandler, ShapeAttributeGroupHandler	Classes for managing attribute groups	Use if your application needs to generate attribute values from data set values (key value pairs).
ojs/ojknockouttemplateutils	No	KnockoutTemplateUtils	Utilities for converting Knockout templates to a renderer function that can be used in JET component renderer APIs	Use if your application needs to work with Knockout templates.
ojs/ojresponsiveknockoututils	No	ResponsiveKnockoutUtils	Utilities for creating Knockout observables to implement responsive page design	Use if your application needs to create observables for responsive page design.
ojs/ojswipetoreveal	No	SwipeToRevealUtils	Utilities for setting up and handling swipe to reveal on an offcanvas element	Use if your application needs to support the swipe gesture.
ojs/ojkeyset	No	KeySet, KeySetImpl, AllKeySetImpl	Class for working with selection items in ojTable, ojListView, and ojDataGrid components	Use if your application needs to work with selections as a set.
ojs/ojdiagram-utils	No	DiagramUtils	Utilities for working with a JSON object to support the ojDiagram component	Use if your application creates an ojDiagram component from a JSON object.
ojs/ojoffcanvas	No	OffcanvasUtils	Class for controlling off-canvas regions	Use if your application needs to manage off-canvas regions.
ojs/ojcube	No	Cube, CubeAggType, CubeAxis, CubeAxisValue, CubeCellSet, CubeCellValue, CubeHeaderSet, CubeLevel, DataColumnCube, DataValueAttributeCube	Classes for aggregating data values in ojDataGrid	Use if your application renders aggregated cubic data in an ojDataGrid component.

Oracle JET Module	Refactored	Available Objects	Description	When to Use
ojs/ ojtypedataprovider	No	ArrayDataProvider, ArrayTreeDataProvider, CollectionDataProvider, DeferredDataProvider, , FlattenedTreeDataProvider, IndexerModelTreeDataProvider, ListDataProviderView, , PagingDataProviderView, TreeDataProviderView FilterFactory	Data provider modules. Examples include: <ul style="list-style-type: none"> • ojs/ ojarraydataprovider • ojs/ ojcollectiondataProvider • ojs/ ojtreedataprovider 	Use if your application includes an Oracle JET component, and its data source is defined in one of the *DataProvider classes.
ojs/ojtimezonedata	No	no public class available	Time zone data	Use if you want to add time zone support to oj-input-date-time, oj-input-time, or converters.
ojconverter- color, ojconverter- datetime, ojconverter-number	Yes, from ojvalidation -base, ojvalidation -datetime, or ojvalidation -number	ColorConverter, converterDateTime, converterColor	Color, date, and time conversion services.	Use if your application needs to support conversion services.
ojvalidator- daterestriction, ojvalidator- datetimerange, ojvalidator-length, ojvalidator- numberrange, ojvalidator-regexp, ojvalidator- required	Yes, from ojvalidation -base, ojvalidation -datetime, or ojvalidation -number	DateRestrictionValidator, DateTimeRangeValidator, LengthValidator, NumberRangeValidator, RegExpValidator, RequiredValidator	Date and number validation services.	Use if your application needs to support validation services.
ojasyncvalidator- daterestriction, ojasyncvalidator- datetimerange, ojasyncvalidator-length, ojasyncvalidator- numberrange, ojasyncvalidator- regexp, ojasyncvalidator- required	Yes, from ojvalidation -base, ojvalidation -datetime, or ojvalidation -number	AsyncDateRestrictionValidator, AsyncDateTimeRangeValidator, AsyncLengthValidator, AsyncNumberRangeValidator, AsyncRegExpValidator,	Async date and number validation services.	Use if your application needs to support async validation services.

About RequireJS in an Oracle JET Application

Oracle JET includes the RequireJS library and sample bootstrap file in the Oracle JET download.

The code below shows the `main-template.js` bootstrap file distributed with the Oracle JET base distribution. Typically, you place the bootstrap file in your application's `js` directory and rename it to `main.js`. The comments in the code describe the purpose of each section. The sections that you normally edit are highlighted in bold.

```
(function () {

    // At runtime, detects IE11 browser to provide polyfill support.
    function _ojIsIE11() {
        var nAgt = navigator.userAgent;
        return nAgt.indexOf('MSIE') !== -1 || !nAgt.match(/Trident.*rv:11./);
    };
    var _ojNeedsES5 = _ojIsIE11();

    requirejs.config(
    {
        baseUrl: 'js',

        // Path mappings for the logical module names
        // Do not remove injector start and end comments. Update the main-
        release-paths.json
        // for release mode when updating the mappings.
        paths:
        // injector:mainReleasePaths
        {

        }
        // endinjector
    );
}());

// This section configures the i18n plugin. It is merging the Oracle JET built-
in translation
// resources with a custom translation file.
// Any resource file added, must be placed under a directory named "nls". You
can use a path mapping or you can define
// a path that is relative to the location of this main.js file.
config: {
    ojL10n: {
        merge: {
            //'ojtranslations/nls/ojtranslations': 'resources/nls/
myTranslations'
        }
    }
});
/***
 * A top-level require call executed by the Application.
 * Although 'knockout' would be loaded in any case (it is specified as
dependencies
 * by the modules themselves), we are listing it explicitly to get the
references to the 'ko'
 * object in the callback.
*/
```

```

/*
 * For a listing of which JET component modules are required for each component,
 * see the specific component demo pages in the JET cookbook.
 */
require(['ojs/ojbootstrap', 'knockout', 'ojs/ojknockout'],
    // Add additional JET component libraries as needed
    function(Bootstrap) { // this callback gets executed when all required modules
are loaded
        // add any startup code that you want here
    }
);
}
);

```

You can use RequireJS in a regular application as shown below, or you can use RequireJS with `oj-module` element to define view templates and `viewModels` for page sections in a single-page application. For example, the Oracle JET Starter Templates use the `oj-module` element with RequireJS to use a different view and `viewModel` when the user clicks one of the navigation buttons.

For additional information about the Oracle JET Starter Templates, see [About the Starter Templates](#). For more information about using `ojModule` and templates, see [Creating Single-Page Applications](#).

Use RequireJS in an Oracle JET Application

To use RequireJS in your application, edit the bootstrap file to add the Oracle JET modules you need. You can also add your own modules as needed for your application code.

If needed, install Oracle JET and install RequireJS at <http://requirejs.org>.

To use RequireJS in an Oracle JET Application:

1. In the bootstrap file or your application scripts, in the `require()` definition, add additional Oracle JET modules as needed.
2. Add any scripts that your application uses to the `require()` definition and update the `function(ko)` definition to include the script.
3. Add any application startup code to the callback function.
4. If your application includes resource bundles, enter the path to the bundle in the `merge` section.

Here's an example of the steps in order.

`oj-dialog`

```

require(['knockout', 'ojs/ojmodel', 'ojs/ojknockout-model', 'ojs/ojdialog'],
    function(ko) // obtaining a reference to the oj namespace
{
}
);

```

Then, to use a script named `myapp.js`, add the highlighted code to your `require()` definition.

```

require(['myapp', 'knockout', 'ojs/ojmodel', 'ojs/ojknockout-model', 'ojs/
ojdialog'],

```

```

        function(myapp, ko) // obtaining a reference to the oj namespace
    {
    }
);

```

Next, you have a Knockout binding call for an element named `dialogWrapper`. Add that to the callback function.

```

require(['myapp', 'knockout', 'ojs/ojmodel', 'ojs/ojknockout-model', 'ojs/
ojdialog'],
function(myapp, ko) // obtaining a reference to the oj namespace
{
    ko.applyBindings(new app()/*View Model instance*/,
                    document.getElementById('dialogWrapper'));
}
);

```

Finally, you have a translations bundle, which you add to the `merge` section.

```

config: {
    ojL10n: {
        merge: {
            'ojtranslations/nls/ojtranslations': 'resources/nls/myTranslations'
        }
    }
}

```

For more information about module loading in Oracle JET applications, see [API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\) - JET Module Loading Overview](#).

Add Third-Party Tools or Libraries to Your Oracle JET Application

You can add third-party tools or libraries to your Oracle JET application. The steps to take will vary, depending on the method you used to create your application.

If you used command-line tooling to scaffold your application, you will install the library and make modifications to `src/js/path_mapping.json`. If you created your application using any other method and are using RequireJS, you will add the library to your application and update the RequireJS bootstrap file, typically `main.js`.

Note:

This process is provided as a convenience for Oracle JET developers. Oracle JET will not support the additional tools or libraries and cannot guarantee that they will work correctly with other Oracle JET components or toolkit features.

To add a third-party tool or library to your Oracle JET application, do one of the following:

- If you created your application with command-line tooling, perform the following steps.

1. In your main project directory, enter the following command in a terminal window to install the library using npm:

```
npm install library-name --save
```

For example, enter the following command to install a library named `my-library`:

```
npm install my-library --save
```

2. Add the new library to the path mapping configuration file.

- a. Open `src/js/path_mapping.json` for editing.

A portion of the file is shown below.

```
{
  "baseUrl": "js",
  "use": "local",

  "cdns": {
    "jet": "https://static.oracle.com/cdn/jet/9.2.0/default/
js",
    "css": "https://static.oracle.com/cdn/jet/9.2.0/default/
css",
    "config": "bundles-config.js"
  },
  "3rdparty": "https://static.oracle.com/cdn/jet/
9.2.0/3rdparty"
  },

  "libs": {

    "knockout": {
      "cdn": "3rdparty",
      "cwd": "node_modules/knockout/build/output",
      "debug": {
        "src": "knockout-latest.debug.js",
        "path": "libs/knockout/knockout-#{version}.debug.js",
        "cdnPath": "knockout/knockout-3.x.x"
      },
      "release": {
        "src": "knockout-latest.js",
        "path": "libs/knockout/knockout-#{version}.js",
        "cdnPath": "knockout/knockout-3.x.x"
      }
    },
    ...
  }
}
```

... contents omitted

- b. Copy one of the existing entries in "libs" and modify as needed for your library.

The sample below shows modifications for `my-library`, a library that contains both minified and debug versions.

```
...
"libs": {
  "my-library": {
    "cdn": "3rdparty",
    "cwd": "node_modules/my-library/dist",
    "debug": {
      "src": "my-library.debug.js",
      "path": "libs/my-library/my-library.debug.js",
      "cdnPath": ""
    },
    "release": {
      "src": "my-library.js",
      "path": "libs/my-library/my-library.js",
      "cdnPath": ""
    }
  },
  ...
}
```

In this example, `cwd` points to the location where npm installed the library, `src` points to a path or array of paths containing the files that will be copied during a build, and `path` points to the destination that will contain the built version.

Note:

If you use a CDN, add the URL to the CDN in the entry for `cdnPath`.

- If you didn't use the tooling to create your application, perform the following steps to add the tool or library.

1. In the application's `js/libs` directory, create a new directory and add the new library and any accompanying files to it.

For example, for a library named `my-library`, create the `my-library` directory and add the `my-library.js` file and any needed files to it. Be sure to add the minified version if available.

2. In your RequireJS bootstrap file, typically `main.js`, add a link to the new file in the path mapping section and include the new library in the `require()` definition.

For example, add the highlighted code below to your bootstrap file to use a library named `my-library`.

```
requirejs.config({
  // Path mappings for the logical module names
  paths:
  {
    'knockout': 'libs/knockout/knockout-3.x.x',
    'jquery': 'libs/jquery/jquery-3.x.x.min',
    ... contents omitted
```

```

    'text': 'libs/require/text',
    'my-library': 'libs/my-library/my-library'
},
require(['knockout', 'my-library'],
function(ko) // this callback gets executed when all required modules
are loaded
{
    // add any startup code that you want here
}
);

```

For additional information about using RequireJS to manage your application's modules, see [Using RequireJS for Modular Development](#).

Troubleshoot RequireJS in an Oracle JET Application

RequireJS issues are often related to modules used but not defined.

Use the following tips when troubleshooting issues with your Oracle JET application that you suspect may be due to RequireJS:

- Check the JavaScript console for errors and warnings. If a certain object in the oj namespace is undefined, locate the module that contains it based on the information in [About Oracle JET Module Organization](#) or the [Oracle JET Cookbook](#) and add it to your application.
- If the components you specified using Knockout.js binding are not displayed and you are not seeing any errors or warnings, verify that you have added the ojs/ojknockout module to your application.

About JavaScript Partitions and RequireJS in an Oracle JET Application

RequireJS supports JavaScript partitions that contain more than one module.

You must name all modules using the RequireJS `bundles` option and supply a path mapping with the configuration options.

```

requirejs.config(
{
    bundles:
    {
        'commonComponents': ['ojL10n', 'ojtranslations/nls/ojtranslations',
                            'ojs/ojknockout', 'ojs/ojcomponentcore',
                            'ojs/ojbutton', 'ojs/ojpopup'],
        'tabs': ['ojs/ojtabs', 'ojs/ojconveyorbelt']
    }
});

```

In this example, two partition bundles are defined: `commonComponents` and `tabs`.

RequireJS ships with its own Optimizer tool for creating partitions and minifying JavaScript code. The tool is designed to be used at build time with a complete project that is already configured to use RequireJS. It analyzes all static dependencies and creates partitions out of modules that are always loaded together. The Oracle JET team recommends that you use an optimizer to minimize the number of HTTP requests needed to download the modules.

For additional information about the RequireJS Optimizer tool, see <http://requirejs.org/docs/optimization.html>.

For additional information about optimizing performance of Oracle JET applications, see [Optimizing Performance](#).

6

Creating Single-Page Applications

Oracle JET includes the `oj-module` component and `CoreRouter` framework class that you can use to create single-page applications that simulate the look and feel of desktop applications.

Topics:

- [Typical Workflow for Creating Single-Page Applications in Oracle JET](#)
- [Design Single-Page Applications Using Oracle JET](#)
- [Use the oj-module Element](#)

Typical Workflow for Creating Single-Page Applications in Oracle JET

Understand Oracle JET support for designing single-page applications. Learn how to use `oj-module` component and `oj-module` lifecycle listeners in your application.

To create a single-page application (SPA) in Oracle JET, refer to the typical workflow described in the following table:

Task	Description	More Information
Create a single-page Oracle JET application	Identify Oracle JET support for single-page applications and how to use it to design your Oracle JET application.	Design Single-Page Applications Using Oracle JET
Create view templates and <code>viewModels</code>	Identify the features and benefits of <code>oj-module</code> and how to use it in your Oracle JET application.	Use the oj-module Element

Design Single-Page Applications Using Oracle JET

Oracle JET includes Knockout for separating the model layer from the view layer and managing the interaction between them. Using Knockout, the Oracle JET `oj-module` component, and the Oracle JET `CoreRouter` framework class, you can create single-page applications that look and feel like a standalone desktop application.

Topics:

- [Understand Oracle JET Support for Single-Page Applications](#)
- [Create a Single-Page Application in Oracle JET](#)

Understand Oracle JET Support for Single-Page Applications

Single-page applications (SPAs) are typically used to simulate the look and feel of a standalone desktop application. Rather than using multiple web pages with links between them for navigation, the application uses a single web page that is loaded only once. If the page changes because of the user's interaction, only the portion of the page that changed is redrawn.

Oracle JET includes support for single page applications using the `CoreRouter` class for virtual navigation in the page, the `oj-module` component for managing view templates and `viewModel` scripts, and Knockout for separating the model layer from the view layer and managing the binding between them. In the Oracle JET Cookbook, you can view a number of implementations that use the `CoreRouter` class. These implementations range from the simple, that switch tabs, to more complex examples that use parameters and child routers. See the [CoreRouter](#) demo in the Oracle JET Cookbook.

When routing a single-page application, the page doesn't reload from scratch but the content of the page changes dynamically. In order to be part of the browser history and provide bookmarkable content, the Oracle JET `CoreRouter` class emulates the act of navigating using the HTML5 history push state feature. The `CoreRouter` also controls the URL to look like traditional page URLs. However, there are no resources at those URLs, and you must set up the HTML server. This is done using a simple rule for a rewrite engine, like [mod rewrite module for Apache HTTP server](#) or a rewrite filter like [UrlRewriteFilter](#) for servlets.

In general, use query parameters when your application contains only a few views that the user will bookmark and that are not associated with a complex state. Use path segments to display simpler URLs, especially for nested paths such as `customers/cust/orders`.

The Oracle JET Cookbook uses the Oracle JET `oj-module` feature to manage the Knockout binding. With `oj-module`, you can store your HTML content for a page section in an HTML fragment or template file and the JavaScript functions that contain your `viewModel` in a `viewModel` file.

When `oj-module` and `CoreRouter` are used in conjunction, you can configure an `oj-module` object where the module name is the router state. When the router changes state, `oj-module` will automatically load and render the content of the module name specified in the value of the current `RouterState` object.

Create a Single-Page Application in Oracle JET

The Oracle JET Cookbook includes complete examples and recipes for creating a single-page application using path segments and query parameters for routing and examples that use routing with the `oj-module` component. Regardless of the routing method you use, the process to create the application is similar.

To create a single-page application in Oracle JET:

If needed, create the application that will house your main HTML5 page and supporting JavaScript. For additional information, see [Understanding the Web Application Workflow](#) or [Understanding the Hybrid Mobile Application Workflow](#).

1. Design the application's structure and identify the templates and ViewModels that your application will require.
2. Add code to your application's main script that defines the states that the router can take, and add the `ojs/ojcorerouter` module to your `require()` list.
3. Add code to the markup that triggers the state transition and displays the content of the current state.

When the user clicks one of the buttons in the header, the content is loaded according to the router's current state.

For additional information about creating templates and ViewModels, see [Use the `oj-module` Element](#).

4. To manage routing within a module, add a child router using `CoreRouter.createChildRouter()`.
5. Add any remaining code needed to complete the content or display.

See the [CoreRouter](#) demo in the Oracle JET Cookbook that implements `CoreRouter` and provides a link to the API documentation.

Use the `oj-module` Element

With the `oj-module` element, you can store your HTML content for a page section in an HTML fragment or template file and the JavaScript functions that contain your `viewModel` in a `viewModel` file.

Many of the Oracle JET Cookbook and sample applications use `oj-module` to manage the Knockout binding.

To use `oj-module` in your Oracle JET application:

If needed, create the application that will house your main HTML5 page and supporting JavaScript. See [Understanding the Web Application Workflow](#). Oracle JET applications are built with default views and `viewModel`s folders under `application_folder/src/js`.

1. In your RequireJS bootstrap file (typically `main.js`) add `ojs/ojmodule-element` to the list of RequireJS modules, along with any other modules that your application uses.

```
require(['knockout', 'ojs/ojmodule-element-utils', 'ojs/ojcorerouter', 'ojs/ojlogger', 'ojs/ojresponsiveknockoututils', 'ojs/ojarraydatasource', 'ojs/ojoffcanvas', 'ojs/ojknockouttemplateutils', 'ojs/ojmodule-element', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/ojmenu', 'ojs/ojmodule', 'text', 'ojs/ojcheckboxset', 'ojs/ojswitch'])
```

2. Create your view templates and add them to the `views` folder as the default location.
3. Create your `viewModel` scripts and add them to the `viewModels` folder as the default location.
4. Add code to the application's HTML page to reference the view template or `viewModel` in the `oj-module` element. To obtain the router configuration, set the `config` attribute of the `oj-module` element to the `koObservableConfig` observable created by the `ModuleRouterAdapter`.

```
<oj-module role="main" class="oj-panel" style="padding-bottom:30px"  
config="[[moduleAdapter.koObservableConfig]]"></oj-module>
```

For more information about CoreRouter and oj-module, see the Oracle JET CoreRouter and oj-module API documentation.

 **Tip:**

oj-module is not specific to single-page applications, and you can also use it to reuse content in multi-page applications. However, if you plan to reuse or share your content across multiple applications, consider creating Oracle JET Web Components instead. Web Components are reusable components that follow the HTML5 Web Component specification. They have the following benefits:

- Web Components have a contract. The API for a Web Component is well defined by its `component.json`, which describes its supported properties, methods, and events in a standard, universal, and self-documenting way. Providing a standardized contract makes it easier for external tools or other applications to consume these components.
- Web Components include version and dependency metadata, making it clear which versions of Oracle JET they support and what other components they may require for operation.
- Web Components are self-contained. A Web Component definition can contain all the libraries, styles, images, and translations that it needs to work.

To learn more about Web Component features, see [Working with Oracle JET Web Components](#).

Work with oj-module's ViewModel Lifecycle

The oj-module element provides lifecycle listeners that allow you to specify actions to take place at defined places in the ViewModel's lifecycle.

For example, you can specify actions to take place when the ViewModel is about to be used for the View transition, after its associated View is inserted into the document DOM, and after its View and ViewModel are inactive.

The following table lists the available methods with a description of their usage.

Method Name	Description
<code>connected()</code>	<p>The optional method will be invoked after the View is inserted into the DOM.</p> <p>This method might be called multiple times:</p> <ul style="list-style-type: none">• after the View is created and inserted into the DOM• after the View is reconnected after being disconnected• after a parent element (oj-module) with attached View is reconnected to the DOM

Method Name	Description
transitionCompleted()	This optional method will be invoked after transition to the new View is complete. This includes any possible animation between the old and the new View.
disconnected()	<p>This optional method will be invoked when the View is disconnected from the DOM.</p> <p>This method might be called multiple times:</p> <ul style="list-style-type: none"> • after the View is disconnected from the DOM • after a parent element (oj-module) with attached View is disconnected from the DOM

You can also find stub methods for using the oj-module lifecycle methods in some of the Oracle JET templates. For example, the navbar template, available as a template when you [Scaffold a Web Application](#), defines stub methods for connected(), disconnected(), and transitionCompleted(). Comments describe the expected parameters and use cases.

```
function DashboardViewModel() {
    var self = this;
    // Below are a set of the ViewModel methods invoked by the oj-module component.
    // Please reference the oj-module jsDoc for additional information.

    /**
     * Optional ViewModel method invoked after the View is inserted into the document DOM. The application can put logic that requires the DOM being attached here.
     * This method might be called multiple times - after the View is created and inserted into the DOM and after the View is reconnected after being disconnected.
     */
    self.connected = function() {
        // Implement if needed
    };

    /**
     * Optional ViewModel method invoked after the View is disconnected from the DOM.
     */
    self.disconnected = function() {
        // Implement if needed
    };

    /**
     * Optional ViewModel method invoked after transition to the new View is complete.
     * That includes any possible animation between the old and the new View.
     */
    self.transitionCompleted = function() {
        // Implement if needed
    };
}

/*
```

```
    * Returns an instance of the ViewModel providing one instance of the
ViewModel. If needed,
    * return a constructor for the ViewModel so that the ViewModel is
constructed
    * each time the view is displayed.
*/
return DashboardViewModel;
}
```

For more, see the [oj-module](#) API documentation.

Understanding Oracle JET User Interface Basics

Oracle JET User Interface (UI) components extend the `htmlElement` prototype to implement the World Wide Web Consortium (W3C) web component specification for custom elements. Custom elements provide a more declarative way of working with Oracle JET components and allow you to access properties and methods directly on the DOM layer.

Topics:

- [Typical Workflow for Working with the Oracle JET User Interface](#)
- [About the Oracle JET User Interface](#)
- [About Binding and Control Flow](#)
- [Add an Oracle JET Component to Your Page](#)

Typical Workflow for Working with the Oracle JET User Interface

Identify Oracle JET user interface components, understand their common functionality, and identify reserved words. You should also understand the steps required to add a component to your HTML5 page.

To work with the Oracle JET user interface, refer to the typical workflow described in the following table.

Task	Description	More Information
Understand the UI framework	Identify Oracle JET UI components, understand their common functionality, and identify Oracle JET reserved words.	About the Oracle JET User Interface
Understand Oracle JET's binding and control flow	Identify Oracle JET binding elements and how to use them in your application.	About Binding and Control Flow
Add an Oracle JET component to your page	Identify the steps you must take to add an Oracle JET component to your HTML5 page. Optionally, add animation effects to your application.	Add an Oracle JET Component to Your Page

About the Oracle JET User Interface

Oracle JET includes components, patterns, and utilities to use in your application. The toolkit also includes an API specification (if applicable) and one or more code examples in the Oracle JET Cookbook.

Topics:

- [Identify Oracle JET UI Components, Patterns, and Utilities](#)
- [About Common Functionality in Oracle JET Components](#)
- [About Oracle JET Reserved Namespaces and Prefixes](#)

Identify Oracle JET UI Components, Patterns, and Utilities

The [Oracle JET Cookbook](#) lists all the components, design patterns, and utilities available for your use. By default, the cookbook is organized into functional sections, but you can also click **Sort** to arrange the contents alphabetically.

The Cookbook contains samples that you can edit online and see the effects of your changes immediately. You'll also find links to the API documentation, if applicable.

About Common Functionality in Oracle JET Components

All Oracle JET components are implemented as custom HTML elements, and programmatic access to these components is similar to interacting with any HTML element.

Custom Element Structure

Oracle JET custom element names start with `oj-`, and you can add them to your page the same way you would add any other HTML element. In the following example, the `oj-label` and `oj-input-date-time` custom elements are added as child elements to a standard HTML `div` element.

```
<div id="div1">
  <oj-label for="dateTime">Default</oj-label>
  <oj-input-date-time id="dateTime" value='{{value}}'>
  </oj-input-date-time>
</div>
```

Each custom element can contain one or more of the following:

- Attributes: Modifiers that affect the functionality of the element.

String literals will be parsed and coerced to the property type. Oracle JET supports the following string literal type coercions: boolean, number, string, Object, Array, and any. The any type, if used by an element, is marked with an asterisk (*) in the element's API documentation and coerced to Objects, Arrays, or strings.

In the `oj-input-date-time` element defined above, `value` is an attribute that contains a Date object. It is defined using a binding expression that indicates

whether the element's ViewModel should be updated if the attribute's value is updated.

```
<ojs-input-date-time id="dateTime" value='{{value}}'>
```

The `{{...}}` syntax indicates that the element's `value` property will be updated in the element's ViewModel if it's changed. To prevent the attribute's value from updating the corresponding ViewModel, use the `[[]]` syntax.

- **Methods:** Supported methods

Each custom element's supported method is documented in its API.

- **Events:** Supported events

Events specific to the custom element are documented in its API. Define the listener's method in the element's ViewModel.

```
var listener = function( event )  
{  
    // Check if this is the end of "inline-open" animation for inline  
    // message  
    if (event.detail.action == "inline-open") {  
        // Add any processing here  
    }  
};
```

Reference the listener using the custom element's DOM attribute, JavaScript property, or the `addEventListener()`.

- **Use the DOM attribute.**

Declare a listener on the custom element using `on-event-name` syntax.

```
<ojs-input-date-time on-oj-animate-start='[[listener]]'></ojs-input-  
date-time>
```

Note that in this example the listener is declared using the `[[]]` syntax since its value is not expected to change.

- **Use the JavaScript property.**

Specify a listener in your ViewModel for the `.onEventName` property.

```
myInputDateTime.onOjAnimateEnd = listener
```

Note that the JavaScript property uses camelCase for the `onOjAnimateEnd` property. The camelCased properties are mapped to attribute names by inserting a dash before the uppercase letter and converting that letter to lower case, for example, `on-oj-animate-end`.

- **Use the `addEventListener()` API.**

```
myInputDateTime.addEventListener('ojAnimateEnd', listener);
```

By default, JET components will also fire `propertyChanged` custom events whenever a property is updated, for example, `valueChanged`. You can define and add a listener using any of the three methods above. When referencing a `propertyChanged` event declaratively, use `on-property-changed` syntax.

```
<oj-input-date-time value="{{currentValue}}" on-value-changed="{{valueChangedListener}}></oj-input-date-time>
```

- Slots

Oracle JET elements can have two types of child content that determine the content's placement within the element.

- Any child element with a supported slot attribute will be moved into that named slot. All supported named slots are documented in the element's API. Child elements with unsupported named slots will be removed from the DOM.

```
<oj-table>
  <div slot='bottom'><oj-paging-control></oj-paging-control></div>
</oj-table>
```

- Any child element lacking a slot attribute will be moved to the default slot, also known as a regular child.

A custom element will be recognized only after its module is loaded by the application. Once the element is recognized, Oracle JET will register a busy state for the element and will begin the process of upgrading the element from a normal element to a custom element. The element will not be ready for interaction until the upgrade process is complete. The application should listen to either the page-level or element-scoped `BusyContext` before attempting to interact with any JET custom elements. However, property setting (but not property getting) is allowed before the `BusyContext` is initialized. See the [BusyContext](#) API documentation on how `BusyContexts` can be scoped.

The upgrade of custom elements relies on a binding provider which manages the data binding. The binding provider is responsible for setting and updating attribute expressions. Any custom elements within its managed subtree will not finish upgrading until the provider applies bindings on that subtree. By default, there is a single binding provider for a page, but subtree specific binding providers can be added by using the `data-oj-binding-provider` attribute with values of `none` and `knockout`. The default binding provider is `knockout`, but if a page or DOM subtree does not use any expression syntax or `knockout`, the application can set `data-oj-binding-provider="none"` on that element so that its dependent JET custom elements do not wait for bindings to be applied to finish upgrading.

Other Common Functionality

Oracle JET custom elements also have the following functionality in common:

- Context menus

Custom elements support the `slot` attribute to add context menus to Oracle JET custom elements, described in each element's API documentation.

```
<oj-some-element>
  <!-- use the contextMenu slot to designate this as the context menu for this component -->
```

```
<oj-menu slot="contextMenu" style="display:none" aria-label="Some
element's context menu"
  ...
</oj-menu>
</oj-some-element>
```

- Keyboard navigation and other accessibility features

Oracle JET components that support keyboard navigation list the end user information in their API documentation. For additional information about Oracle JET components and accessibility, see [Developing Accessible Applications](#).

- Drag and drop

Oracle JET includes support for standard HTML5 drag and drop and provides the `dnd-polyfill` library to extend HTML5 drag and drop behavior to supported mobile and desktop browsers. In addition, some Oracle JET custom elements such as `oj-table` support drag and drop behavior through the `dnd` attribute. For specific details, see the component's API documentation and cookbook examples. To learn more about HTML5 drag and drop, see http://www.w3schools.com/html/html5_draganddrop.asp.

- Deferred rendering

Many Oracle JET custom elements support the ability to defer rendering until the content shown using `oj-defer`. To use `oj-defer`, wrap it around the custom element.

```
<oj-collapsible id="defer">
  <h4 id="hd" slot="header">Deferred Content</h4>
  <oj-defer>
    <oj-module config='[[deferredRendering/content]]'>
      </oj-module>
    </oj-defer>
  </oj-collapsible>
```

Add the deferred content to the application's view and `ViewModel`, `content.html` and `content.js`, as specified in the `oj-module` definition. For the complete code example, see [Collapsibles - Deferred Rendering](#).

For a list of custom elements that support `oj-defer`, see [oj-defer](#).

Custom Element Examples and References

The [Oracle JET Cookbook](#) and [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) provide examples that illustrate how to work with custom elements. In addition, the Cookbook provides demos with editing capability that allow you to modify the sample code directly and view the results without having to download the sample.

To learn more about the World Wide Web Consortium (W3C) web component specification for custom elements, see [Custom Elements](#).

About Oracle JET Reserved Namespaces and Prefixes

Oracle JET reserves the `oj` namespace and prefixes. This includes, but is not limited to component names, namespaces, pseudo-selectors, public event prefixes, CSS styles, Knockout binding keys, and so on.

About Binding and Control Flow

Oracle JET includes components and expressions to easily bind dynamic elements to a page in your application using Knockout.

Topics

- [Use `oj-bind-text` to Bind Text Nodes](#)
- [Bind HTML attributes](#)
- [Use `oj-bind-if` to Process Conditionals](#)
- [Use `oj-bind-for-each` to Process Loop Instructions](#)
- [Bind Style Properties](#)
- [Bind Event Listeners to JET and HTML Elements](#)

Use `oj-bind-text` to Bind Text Nodes

Oracle JET supports binding text nodes to variables using the `oj-bind-text` element and by importing the `ojknockout` module.

The `oj-bind-text` element is removed from the DOM after binding is applied. For example, the following code sample shows an `oj-input-text` and an `oj-button` with a text node that are both bound to the `buttonLabel` variable. When the input text is updated, the button text is automatically updated as well.

```
<div id='button-container'>
    <oj-button id='button1'>
        <span><oj-bind-text value="[[buttonLabel]]"></oj-bind-text></span>
    </oj-button>
    <br><br>
    <oj-label for="text-input">Update Button Label:</oj-label>
    <oj-input-text id="text-input" value="{{buttonLabel}}"></oj-input-text>
</div>
```

The script to create the view model for this example is shown below.

```
require(['ajs/ajbootstrap', 'knockout', 'ajs/ajknockout', 'ajs/ajbutton', 'ajs/ajinputtext', 'ajs/ajlabel'],
    function(Bootstrap, ko)
{
    function ButtonModel() {
        this.buttonLabel = ko.observable("My Button");
    }

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new ButtonModel(), document.getElementById('button-'))
```

```
container'));  
    }  
);  
});
```

The figure below shows the output for the code sample.



The Oracle JET Cookbook contains the complete example used in this section. See [Text Binding](#).

Bind HTML attributes

Oracle JET supports one-way attribute data binding for attributes on any HTML element by prefixing ':' to the attribute name and by importing the `ojknockout` module.

To use an HTML attribute in an HTML or JET element, prefix a colon to it. JET component-specific attributes do not need the prefix.

The following code sample shows two JET elements and two HTML elements that use both the prefixed and non-prefixed syntax. Since the label and input elements are native HTML elements, all of their data bound attributes should use the colon prefixing. The `oj-label` and `oj-input-text` use the prefix only for native HTML element attributes and the non-prefixed syntax for component-specific attributes.

```
<div id="demo-container">  
  <oj-label for="[[inputId1]]">oj-input-text element</oj-label>  
  <oj-input-text :id="[[inputId1]]" value="{{value}}"></oj-input-text>  
  <br><br>  
  <label :for="[[inputId2]]">HTML input element</label>  
  <br>  
  <input :id="[[inputId2]]" :value="{{value}}" style="width:100%;max-  
  width:18em"/>  
</div>
```

The script to create the view model for this example is shown below.

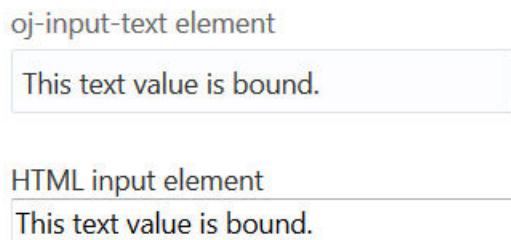
```

require(['ojs/ojbootstrap', 'knockout', 'ojs/ojinputtext', 'ojs/ojlabel', 'ojs/ojknockout'],
    function(Bootstrap, ko)
{
    function ViewModel()
    {
        this.inputId1 = 'text-input1';
        this.inputId2 = 'text-input2';
        this.value = "This text value is bound.";
    }

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new ViewModel(), document.getElementById('demo-
container'));
        }
    );
});

```

The figure below shows the output for the code sample.



The Oracle JET Cookbook contains the complete example used in this section. See [Attribute Binding](#).

Use oj-bind-if to Process Conditionals

Oracle JET supports conditional rendering of elements by using the `oj-bind-if` element and importing the `ojs/ojknockout` module.

The `oj-bind-if` element is removed from the DOM after binding is applied, and must be wrapped in another element such as a `div` if it is used for slotting. The `slot` attribute has to be applied to the wrapper since `oj-bind-if` does not support it. For example, the following code sample shows an image that is conditionally rendered based on the option chosen in an `oj-buttonset`.

```

<div id="demo-container">
    <oj-buttonset-one class="oj-buttonset-width-auto" value="{{buttonValue}}>
        <oj-option id="onOption" value="on">On</oj-option>
        <oj-option id="offOption" value="off">Off</oj-option>
    </oj-buttonset-one>
    <br><br>
    <div>Image will be rendered if the button is on:</div>
    <oj-bind-if test="[[buttonValue() === 'on']]>
        <oj-avatar role="img" aria-label="Avatar of Amy Bartlet" size="md"
initials="AB"

```

```
        src="images/composites/avatar-image.jpg" class="oj-avatar-image">
    </oj-avatar>
</oj-bind-if>
</div>
```

In the above example, the `oj-avatar` element is an icon which can display a custom or placeholder image. See [oj-avatar](#).

The script to create the view model for this example is shown below.

```
require(['ojs/ojbootstrap', 'knockout', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/ojavatar'],
  function(Bootstrap, ko)
  {
    function ViewModel()
    {
      this.buttonValue = ko.observable("off");
    }

    Bootstrap.whenDocumentReady().then(
      function ()
      {
        ko.applyBindings(new ViewModel(), document.getElementById('demo-
container'));
      }
    );
  });
});
```

The figure below shows the output for the code sample. When the `oj-buttonset` is set to 'on', the `oj-avatar` element is rendered and displayed.



Image will be rendered if the button is on:

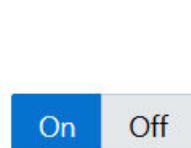


Image will be rendered if the button is on:



The Oracle JET Cookbook contains the complete example used in this section. See [If Binding](#).

Use `oj-bind-for-each` to Process Loop Instructions

Oracle JET supports processing loop instructions, such as binding items from an array by using the `oj-bind-for-each` element and by importing the `ojknockout` module.

The `oj-bind-for-each` element only accepts a single template element as its direct child. Any markup to be duplicated, such as `li` tags, must be placed inside the template tag. For example, the following code sample shows an unordered list nested inside another unordered list. The list items are created using an `oj-bind-text` tag inside nested `oj-bind-for-each` elements.

```
<div id="form-container">
  <ul>
    <oj-bind-for-each data="[[categories]]" data-oj-as="category">
      <template>
        <li>
          <ul>
            <oj-bind-for-each data="[[category.data.items]]" data-oj-as="item">
              <template data-oj-as="item">
                <li>
                  <oj-bind-text value="[[category.data.name + ' : ' + item.data]]"></oj-bind-text>
                </li>
              </template>
            </oj-bind-for-each>
          </ul>
        </li>
      </template>
    </oj-bind-for-each>
  </ul>
</div>
```

In the above example, the `data-oj-as` attribute provides an alias for the bound data. This alias is referenced in the nested `oj-bind-for-each` and `oj-bind-text` elements.

The script to create the view model for this example is shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout'],
  function (ko, Bootstrap)
{
  function SimpleModel()
  {
    this.categories =
    [ { name:"Fruit",items:["Apple","Orange","Banana"] },
      {name:"Vegetables",items:["Celery","Corn","Spinach"] }
    ];
  };

  Bootstrap.whenDocumentReady().then(
    function ()
    {
      ko.applyBindings(new SimpleModel(), document.getElementById('form-
container'));
    }
  );
});
```

The figure below shows the output for the code sample.

- ○ Fruit : Apple
- Fruit : Orange
- Fruit : Banana
- ○ Vegetables : Celery
- Vegetables : Corn
- Vegetables : Spinach

The Oracle JET Cookbook contains the complete example used in this section. See [Foreach Binding](#).

Bind Style Properties

The Oracle JET attribute binding syntax also supports Style attributes, which can be passed as an object or set using dot notation. The `ojknockout` module must be imported.

The style attribute binding syntax accepts an Object in which style properties should be referenced by their JavaScript names. Applications can also set style sub-properties using dot notation, which uses the CSS names for the properties. The code sample below shows two block elements with style attributes. The first element binds a style Object, while the second binds properties directly to the defined style attributes.

```
<div id="demo-container">
  <div :style="[[style]]">Data bound style attribute</div>
  <br>
  <div :style.color="[[fontColor]]" :style.font-style="[[fontStyle]]">Data bound
  style using dot notation</div>
</div>
```

The script to create the view model for this example is shown below. The style object referenced above is highlighted below.

```
require(['ajs/ajbootstrap', 'knockout', 'ajs/ajinputtext', 'ajs/ajlabel', 'ajs/
ojknockout'],
  function(Bootstrap, ko)
{
  function ViewModel()
  {
    this.fontColor = "blue";
    this.fontStyle = "italic";
    this.style = {"fontWeight": "bold", "color": "red"};
  }

  Bootstrap.whenDocumentReady().then(
    function ()
    {
      ko.applyBindings(new ViewModel(), document.getElementById('demo-
container'));
    }
  );
});
```

The figure below shows the output for the code sample.

Data bound style attribute

Data bound style using dot notation

The Oracle JET Cookbook contains the complete example used in this section. See [Style Binding](#).

Bind Event Listeners to JET and HTML Elements

Oracle JET provides one-way attribute data binding for event listeners on JET and HTML elements using the `on-[eventname]` syntax and by importing the `ojknockout` module.

Oracle JET event attributes provide two key advantages over native HTML event listeners. First, they provide three parameters to the listener:

- `event`: The DOM event, such as click or mouse over.
- `data`: equal to `bindingContext['$data']`. When used in iterations, such as in an `oj-bind-for-each`, this parameter is the same as `bindingContext['$current']`.
- `bindingContext`: The entire data binding context (or scope) that is applied to the element.

Second, they have access to the model state and can access functions defined in the ViewModel using the `data` and `bindingContext` parameters.

Note:

The `this` context is not directly available in the event listeners. This is the same behavior as native HTML event listeners.

For example, the following code sample shows an `oj-button` that uses the `on-obj-action` event attribute and an HTML button that uses the `on-click` event attribute to access custom functions defined in the ViewModel shown below.

```
<div id="demo-container">
  <oj-label for="button1">oj-button element</oj-label>
  <oj-button id="button1" on-obj-action="[[clickListener1]]">Click me!</oj-button>
  <br><br>
  <label for="button2">HTML button element</label>
  <br>
  <button id="button2" on-click="[[clickListener2]]">Click me!</button>
  <br><br>
  <div style="font-weight:bold;color:#ea5b3f;">
    <oj-bind-text value="[[message]]"></oj-bind-text>
  </div>
</div>
```

 **Note:**

HTML events use the prefix “on”, such as `onclick` and `onload`. JET events use the prefix “on-”, such as `on-click` and `on-load`.

The script to create the view model for this example is shown below. Note the usage of the `data` attribute to access the `message` parameter.

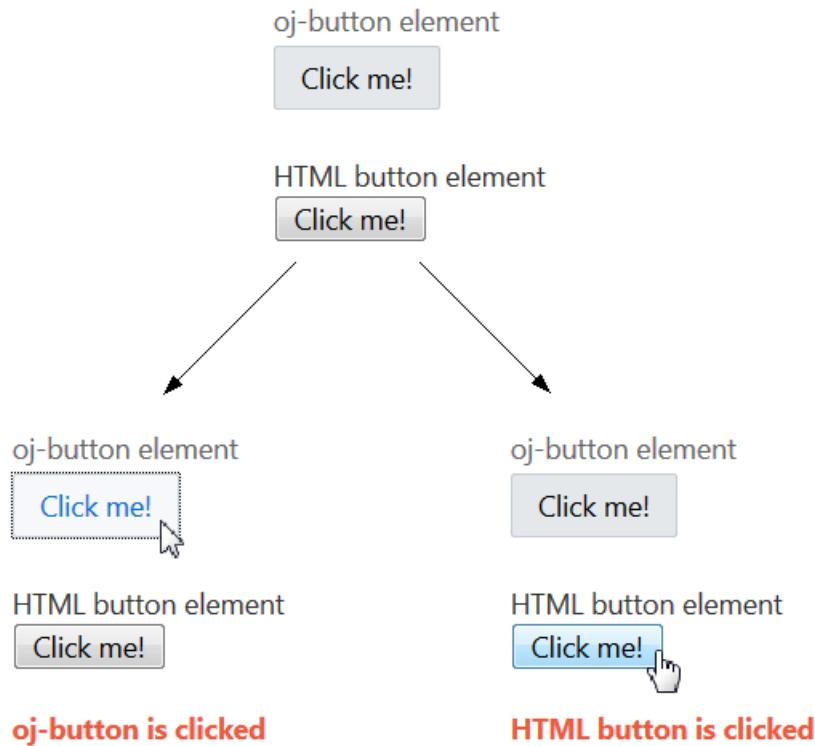
```
require(['ojs/objbootstrap', 'knockout', 'ojs/ojbutton', 'ojs/ojlabel', 'ojs/ojknockout'],
    function(Bootstrap, ko)
{
    function ViewModel()
    {
        this.message = ko.observable();

        this.clickListener1 = function(event, data, bindingContext)
        {
            data.message('oj-button is clicked');
        };

        this.clickListener2 = function(event, data, bindingContext)
        {
            data.message('HTML button is clicked');
        };
    }

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new ViewModel(), document.getElementById('demo-
container'));
        }
    );
});
```

The figure below shows the output for the code sample.



The Oracle JET Cookbook contains the complete example used in this section. See [Event Binding](#).

Bind Classes

The Oracle JET attribute binding syntax has enhanced support for the class attribute, and can accept in addition to a string, an object or an array. This can be used to set classes on components. The `ojknockout` module must be imported.

The `:class` attribute binding can support expressions that resolve to a space delimited string, an Array of class names, or an Object of class to a boolean value or expression for toggling the class in the DOM. Object values can be used to toggle classes on and off. Array and string values can be used only to set classes.

For example, the following code sample shows an `oj-input-text` and an `HTML input` that both use `:class`.

```
<oj-input-text id="input1"
    :class="[[{'oj-form-control-text-align-right': alignRight}]]"
    value="Text Content"></oj-input-text>
<oj-button id="button2" on-oj-action="[[clickListener2]]">Toggle Alignment</oj-
button>

<input id="input2" :class="[[classArrayObs]]"
    value="Text Content" style="width:100%;max-width:18em"/>
<oj-button id="button1" on-oj-action="[[clickListener1]]">Add Class</oj-button>
```

The script to create the view model for this example is shown below.

```
require(['ojs/ojbootstrap', 'knockout', 'ojs/ojlabel', 'ojs/ojinputtext', 'ojs/ojbutton', 'ojs/ojknockout'],
    function(Bootstrap, ko)
{
    function ViewModel()
    {
        this.alignRight = ko.observable(false);
        this.classList = ko.observableArray();
        var classList = ['pink', 'bold', 'italic'];
        var self = this;

        this.clickListener1 = function(event, data, bindingContext) {
            var newClass = classList.pop();
            this.classList.push(newClass);

            // Disable the add button once we're out of classes
            if (classList.length === 0)
                document.getElementById('button1').disabled = true;
        }.bind(this);

        this.clickListener2 = function(event, data, bindingContext) {
            self.alignRight(!this.alignRight());
        }.bind(this);
    }

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new ViewModel(), document.getElementById('demo-
container'));
        }
    );
});
```

The CSS class styles used by the `classList` variable above are shown below.

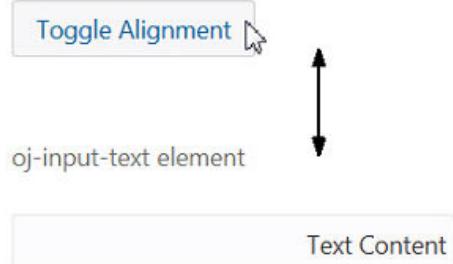
```
.bold {
    font-weight: bold;
}
.italic {
    font-style: italic;
}
.pink {
    color: #ff69b4;
}
```

The figure below shows the first of the two outputs for the code sample. The button acts as a toggle to switch on and off the `oj-form-control-text-align-right` class property and hence change the alignment of the text.

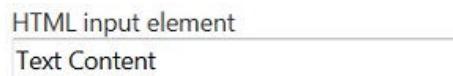
oj-input-text element



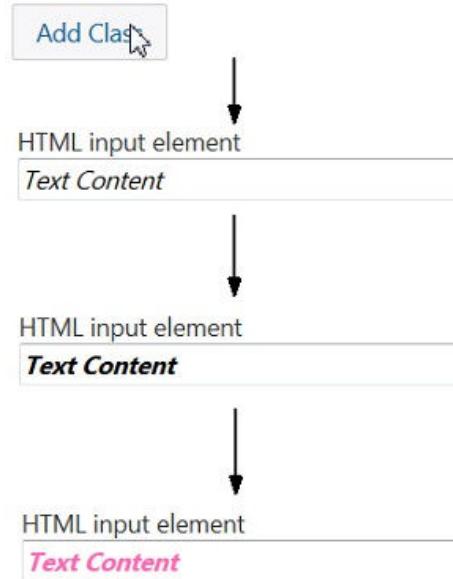
The oj-input-text element uses :class attribute binding with an Observable where an observable is used to toggle the addition/removal of a right alignment CSS class from the DOM. Click the button to toggle the alignment class on the oj-input-text.



The figure below shows the second of the two outputs for the code sample. The button calls a function to take a pre-defined array of classes and add them to the input element. Each class has CSS modifications that come into effect when the class is added.



The HTML input element uses :class attribute binding with an Observable Array that gets updated with additional CSS classes. Click the button to add a new class to the class array on the HTML input element.



The Oracle JET Cookbook contains the complete example used in this section. See [Class Binding](#).

Add an Oracle JET Component to Your Page

Use the Oracle JET Cookbook recipes and API documentation to locate examples that illustrate the specific element and functionality you want to add to your page.

If you haven't already, create the application that you will use for this exercise.

To add an Oracle JET custom element to your page:

1. Using the Oracle JET Cookbook, select the Oracle JET element that you want to add.
2. If you've set up your app using a Starter Template, or are using a page fragment, add the element to the `define` block.
3. Follow the example's recipe and add the markup to your HTML page. Modify the attributes to your need.

```
<div id="div1">
  <oj-label for="dateTime">Default</oj-label>
  <oj-input-date-time id="dateTime" value='{{value}}'>
    </oj-input-date-time>

    <br/><br/>

    <span class="oj-label">Current component value is:</span>
    <span><oj-bind-text value="[[value]]"></oj-bind-text></span>

</div>
```

In this example, the `oj-input-date-time` element is declared with its `value` attribute using `{{...}}` expression syntax, which indicates that changes to the value will also update the corresponding value in the ViewModel. Each Oracle JET custom element includes additional attributes that are defined in the custom element's API documentation.

4. Use the Oracle JET Cookbook for example scripts and the syntax to use for adding the custom element's Require module and ViewModel to your RequireJS bootstrap file or module.

For example, the basic demo for `oj-input-date-time` includes the following script that you can use in your application.

```
require(['ajs/ajbootstrap', 'knockout', 'ajs/ajvalidation-base', 'ajs/ajknockout', 'ajs/ajdatetimepicker', 'ajs/ajtimezonedata', 'ajs/ajlabel'],
  require(['knockout', 'ajs/ajbootstrap',
    'ajs/ajconverterutils-i18n', 'ajs/ajknockout',
    'ajs/ajdatetimepicker', 'ajs/ajlabel'],
```

```
function (Bootstrap, ko, ConverterUtilsI18n)
{
  function SimpleModel()
  {
    this.value =
    ko.observable(ConverterUtilsI18n.IntlConverterUtils.dateToLocalIso(new
```

```
        Date(2020, 0, 1)));
    }

    Bootstrap.whenDocumentReady().then(
        function ()
    {
        ko.applyBindings(new SimpleModel(), document.getElementById('div1'));
    }
);
});
```

If you already have a RequireJS bootstrap file or module, compare your file with the Cookbook sample and merge in the differences. For details about working with RequireJS, see [Using RequireJS for Modular Development](#).

The Cookbook sample used in this section is the [Date and Time Pickers](#) demo.

Add Animation Effects

You can use the `oj-module` component's `animation` property in conjunction with the `ModuleAnimations` namespace to configure animation effects when the user transitions between or drills into views. If you're not using `oj-module`, you can use the `AnimationUtils` namespace instead to add animation to Oracle JET components or HTML elements.

Adding Animation Effects Using the `oj-module` Component

The `ModuleAnimations` namespace includes pre-configured implementations that you can use to configure the following animation effects:

- `coverStart`: The new view slides in to cover the old view.
- `coverUp`: The new view slides up to cover the old view.
- `drillIn`: Animation effect is platform-dependent.
 - Web and iOS: `coverStart`
 - Android: `coverUp`
 - Windows: `zoomIn`
- `drillOut`: Animation effect is platform-dependent.
 - Web and iOS: `revealEnd`
 - Android: `revealDown`
 - Windows: `zoomOut`
- `fade`: The new view fades in and the old view fades out.
- `goLeft`: Navigate to sibling view on the left. Default effect is platform-dependent.
 - Web and iOS: `none`
 - Android and Windows: `pushRight`
- `goRight`: Navigate to sibling view on the right. Default effect is platform-dependent.
 - Web and iOS: `none`
 - Android and Windows: `pushLeft`
- `pushLeft`: The new view pushes the old view out to the left.

- `pushRight`: The new view pushes the old view out to the right.
- `revealDown`: The old view slides down to reveal the new view.
- `revealEnd`: The new view slides left or right to reveal the new view, depending on the locale.
- `zoomIn`: The new view zooms in.
- `zoomOut`: The old view zooms out.

For examples that illustrate how to add animation with the `oj-module` component, see [Animation Effects with Module Component](#).

Adding Animation Effects Using AnimationUtils

The `AnimationUtils` namespace includes methods that you can use to configure the following animation effects on HTML elements and Oracle JET components:

- `collapse`: Use for collapsing the element
- `expand`: Use for expanding the element
- `fadeIn` and `fadeOut`: Use for fading the element into and out of view.
- `flipIn` and `flipOut`: Use for rotating the element in and out of view.
- `ripple`: Use for rippling the element.
- `slideIn` and `slideOut`: Use for sliding the element into and out of view.
- `zoomIn` and `zoomOut`: Using for zooming the element into and out of view.

Depending on the method's options, you can configure properties like `delay`, `duration`, and `direction`. For examples that illustrate how to configure animation using the `AnimationUtils` namespace, see [Animation Effects](#).

Manage the Visibility of Added Component

Follow the recommended best practice when you programmatically manage the display of Oracle JET components in the DOM.

If you manage the display of collection components and other complex component by using the CSS `display` property values of `none` or `block`, you may also need to use `Components.subtreeHidden(node)` when you hide a component and `Components.subtreeShown(node)` when you display the component. The `node` parameter refers to the root of the subtree in the DOM for the component that you are hiding or displaying. You need to notify Oracle JET if you change the display status of these types of component to ensure that the component instance continues to work correctly in the application.

Use of these methods is demonstrated in [Masonry Layout Deferred Rendering](#) in the Oracle JET Cookbook where they hide and show additional content in a tile that the `oj-masonry-layout` component resizes.

Not all components where you programmatically manage the display require you to notify Oracle JET when you change their display state. Components such as `oj-collapse`, `oj-dialog`, and `oj-popup` are examples of components where you do not need to call `Components.subtreeHidden(node)` or `Components.subtreeShown(node)`. These components manage rendering when visibility is changed and also manage any components that they contain. Similarly,

you do not need to use these methods in conjunction with the `oj-bind-if` component because Oracle JET rewrites the appropriate part of the DOM in response to the evaluation of the `oj-bind-if` component's test condition.

Failure to use the `subtreeHidden` and `subtreeShown` methods to notify Oracle JET when you hide or show a component can result in unexpected behavior for the component, such as failure to render data or failure to honor other component attribute settings. For more information about the `subtreeHidden` and `subtreeShown` methods, see [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#).

8

Working with Oracle JET User Interface Components

Oracle JET provides a variety of user interface (UI) components that you can configure for use in your application. The Oracle JET Cookbook includes examples for working with collections, controls, forms, visualizations, and other features. You may also find this collection of tips and tricks helpful.

Topics

- [Work with Oracle JET UI Components - A Typical Workflow](#)
- [Work with Collections](#)
- [Work with Controls](#)
- [Work with Forms](#)
- [Work with Layout and Navigation](#)
- [Work with Visualizations](#)

Work with Oracle JET UI Components - A Typical Workflow

Identify tips and tricks for working with Oracle JET user interface components.

To work with the Oracle JET user interface components, refer to the typical workflow described in the following table. Use the task descriptions to locate the component that you want to add to your application from the Description column.

Task	Description	More Information
Work with collections	Identify tips and tricks for working with data grids, list views, pagination, row expanders, tables, and trees.	Work with Collections
Work with controls	Identify tips and tricks for working with buttons, button sets, conveyor belts, film strips, menus, progress indicators, HTML tags, toolbars, and trains.	Work with Controls
Work with forms	Identify tips and tricks for working with form elements, including checkbox and radio sets, comboboxes, form controls, form layout, input, labels, selection, validation and user assistance components.	Work with Forms
Work with layout and navigation components, elements, and patterns	Identify tips and tricks for working with accordions, collapsibles, dialogs, drawer utilities, panels, popups, and tabs.	Work with Layout and Navigation

Task	Description	More Information
Work with visualizations	Identify tips and tricks for working with data visualizations, including charts, diagrams, gauges, NBoxes, sunbursts, tag clouds, thematic maps, timelines, and treemaps.	Work with Visualizations

Work with Collections

Use Oracle JET data collection components to display data in tables, data grids, list views, or trees.

The Oracle JET data collection components include `oj-table`, `oj-data-grid`, `oj-tree`, and `oj-list-view`, and you can use them to display records of data. `oj-table` and `oj-data-grid` both display data in rows and columns, and your choice of component depends upon your use case. Use `oj-tree-view` to display hierarchical data such as a directory structure and `oj-list-view` to display a list of data. The toolkit also includes pagination and row expanders that display hierarchical data in a data grid or table row.

The [Oracle JET Cookbook](#) and [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) include complete demos and examples for using the collection components, and you may also find the following tips and tricks helpful.

Note:

If you programmatically control the display of a collection component, such as `oj-table`, review [Manage the Visibility of Added Component](#).

Topics:

- [Choose a Table, Data Grid, or List View](#)
- [Work with Data Grids](#)
- [Work with List Views](#)
- [Work with Pagination](#)
- [Work with Row Expanders](#)
- [Work with Tables](#)
- [Work with Tree Views](#)

Choose a Table, Data Grid, or List View

Oracle JET provides the `oj-table`, `oj-data-grid`, and `oj-list-view` components for displaying records of data in rows and columns, and this section can help you decide which component to use in your application.

The `oj-table` component displays records of data on a row basis. It's best used when you have simple data that can be presented as rows of fields, and it should be your first choice as it provides a simpler layout to represent the data and also supports most

of the common features, unless you require advanced features. A selection in the table provides you with the row of data and all of the fields in that row or record. The sizing of the table is based on the content itself. The height and width of the cells is adjusted for the content included. You can write templates using `oj-table` elements such as `tr`, `td`, `th`, and so on.

The `oj-data-grid` is designed to provide grid functionality. It provides the ability to select individual or ranges of cells of data. It's best used when you need to display totals or tallies of data across columns or rows of data. The `oj-data-grid` is designed to be much more flexible in its layout and functionality than the `oj-table` component. It's a low-level component that you can shape in your application to how you want the data to be displayed. The overall size of the data grid is not determined by its content, and the developer specifies the exact height and width of the container. The data grid acts as a viewport to the contents, and unlike a table its size is not determined by the size of the columns and rows. With this custom HTML `oj-data-grid` element, you can host the template content inside it.

The `oj-list-view` element displays a list of data or a list of grouped data. It is best used when you need to display a list using an arbitrary layout or content. You can also use `oj-list-view` to display hierarchical data that contains nested lists within the root element.

The table below provides a feature comparison of the `oj-table`, `oj-data-grid`, and `oj-list-view` components.

Feature	oj-table	oj-data-grid	oj-list-view
Column/Row sizing	Controlled by content or CSS styles. Percent values supported for width and height.	Controlled by cell dimensions. Does not support percent values for width and height.	No
User-resizable column	Yes	Yes	No
User-resizable row	No	Yes	No
Row reordering	No	Yes	No
Column sorting	Yes	Yes	No
Column selection	Yes	Yes	No
Row selection	Yes	Yes	Yes
Cell selection	No	Yes	No
Marquee selection	No	Yes	No
Row header support	No	Yes	No
Pagination	Page, high water mark	Page, high water mark, virtual scrolling (see note)	Page, high water mark
Custom cell templates	Yes	Yes	No
Custom row templates	Yes	No	Yes

Feature	oj-table	oj-data-grid	oj-list-view
Custom cell renderers	Yes	Yes	No
Custom row renderers	Yes	No	Yes
Row expander support	Yes	Yes	No
Cell stamping	Yes	Yes	No
Cell merging	No	Yes	No
Render aggregated cubic data	No	Yes	No
Base HTML element	oj-table	oj-data-grid	oj-list-view
Custom footer template	Yes (provides access to column data for passing to a JavaScript function)	No (cell level renderers used for column and row data manipulations)	No
Cell content editing	Yes	Yes	No
Content filtering	Yes	Yes	No
KeySet API support	Yes	No	Yes

 **Note:**

True virtual scrolling is available as a feature of oj-data-grid. Modern design principles should be considered and implemented before implementing virtual scrolling. It is much more desirable to present the end user with a filtered list of data that will be more useful to them, than to display thousands of rows of data and expect them to scroll through it all. True virtual scrolling means that you can perform scrolling in both horizontal and vertical directions with data being added and removed from the underlying DOM. High water mark scrolling (with lazy loading) is the preferred method of scrolling, and you should use it as a first approach.

KeySet objects are used to represent the selected items in a component. The collection components generally uses an array for the selected items or an object that defines the range of selected items. You can modify these components to use a KeySet object that handles the representation of selected items. Note that the Data Grid component does not currently support KeySet object referencing for the selected items. The oj-list-view component can use KeySet to determine the selected items. The oj-table component can use KeySet to determine the selected items for the rows or columns. If both values are specified, then row will take precedence and column will be reset to an empty KeySet.

About DataProvider Filter Operators

The DataProvider interface is used to get runtime data for JET components that display list of items. DataProvider implementations use filter operators for filtering.

You can specify two types of filters:

- Attribute Filter: Provides filters with the functionality of attribute operator filtering.

```
interface AttributeFilter<D> extends AttributeFilterDef<D> {
    filter(item: D, index?: number, array?: Array<D>): boolean;
}
type Filter<D> = AttributeFilter<D> | CompoundFilter<D>;

type RecursivePartial<T> = {
    [P in keyof T]?: 
        T[P] extends (infer U)[] ? RecursivePartial<U>[] : 
        T[P] extends object ? RecursivePartial<T[P]> : 
        T[P];
};

interface AttributeFilterDef<D> {
    readonly op: AttributeFilterOperator.AttributeOperator;
    readonly attribute?: keyof D;
    readonly value: RecursivePartial<D>;
}

type AttributeFilterAttributeExpression = '*';
```

- Compound Filter: Provides filter operators for compound operations.

```
interface CompoundFilter<D> extends CompoundFilterDef<D>, 
BaseFilter<D> {
}

interface CompoundFilterDef<D> {
    readonly op: CompoundFilterOperator.CompoundOperator;
    readonly criteria: Array<AttributeFilterDef<D> | 
CompoundFilterDef<D>>;
}
```

Filter definition which filters on DepartmentId value 10:

```
{op: '$eq', value: {DepartmentId: 10}}
```

Filter definition which filters on DepartmentId value 10 and DepartmentName is Hello:

```
{op: '$eq', value: {DepartmentId: 10, DepartmentName: 'Hello'}}
```

Filter definition which filters on subobject Location where State is California and DepartmentName is Hello:

```
{op: '$eq', value: {DepartmentName: 'Hello', Location: {State: 
'California'}}}
```

For additional detail and the complete list of operators that you can use with Attribute filters, see the [AttributeFilterDef](#) API documentation.

For additional detail and the complete list of operators that you can use with Compound filters, see the [CompoundFilterDef](#) API documentation.

Work with Data Grids

The `oj-data-grid` element displays data in cells inside a grid consisting of rows and columns. `oj-data-grid` is themable and supports WAI-ARIA authoring practices for accessibility. You can configure the `oj-data-grid` element for cell selection with row and column headers or for row selection with column headers.

	First Name	Last Name	Email Address	Phone #	Date Hired	Salary
100	Steven	King	SKING	515.123.4567	Jun 17, 1987	\$24,000.00
101	Neena	Kochhar	NKOCHHAR	515.123.4568	Sep 21, 1989	\$17,000.00
102	Lex	De Haan	LDEHAAN	515.123.4569	Jan 13, 1993	\$17,000.00
103	Alexander	Hunold	AHUNOLD	590.423.4567	Jan 3, 1990	\$9,000.00
104	Bruce	Ernst	BERNST	590.423.4568	May 21, 1991	\$6,000.00
105	David	Austin	DAUSTIN	590.423.4569	Jun 25, 1997	\$4,800.00
106	Valli	Pataballa	VPATABAL	590.423.4560	Feb 5, 1998	\$4,800.00
107	Diana	Lorentz	DLORENTZ	590.423.5567	Feb 7, 1999	\$4,200.00
108	Nancy	Greenberg	NGREENBE	515.124.4569	Aug 17, 1994	\$12,000.00

Cell based data grids can be configured with single or multiple cell selection.

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

 **Note:**

When you configure the data grid for row selection, the grid has a look and feel that is similar to the `oj-table` element. However, there are distinct differences in functionality between the `oj-data-grid` and `oj-table` elements. For additional information, see [Choose a Table, Data Grid, or List View](#).

To create the data grid, define an `oj-data-grid` element in your HTML file and assign a meaningful id, width, and height. Also, specify the data property.

```
<oj-data-grid id="datagrid"
    style="width:100%;max-width:234px;height:130px"
    aria-label="Data Grid Cell Based Grid Demo"
    data="[[dataSource]]">
</oj-data-grid>
```

In this example, the `aria-label` is added to the `oj-data-grid` element for accessibility. `oj-data-grid` implicitly defines an Aria role as application, and you must add the `aria-label` to distinguish it from other elements defined with the Aria application role. For additional information about Oracle JET and accessibility, see [Developing Accessible Applications](#).

The data is defined in the `dataSource` object and can be one of the following:

- `ArrayDataSource`: Use to define data in a static array.

The array can be a single array where each item in the array represents a row in the data grid, or a two dimensional array where each item represents a cell in the grid. For the data grid shown in this section, the data is defined as a two dimensional array.

```
require(['knockout', 'ojs/objbootstrap', 'ojs/ojarraydatagriddatasource',
        'ojs/ojknockout', 'ojs/odatagrid'],
       function(ko, arrayModule)
{
    function viewModel()
    {
        var self = this;
        var dataArray = [
            ['1', '2', '3'],
            ['4', '5', '6'],
            ['7', '8', '9']
        ];
        self.dataSource = new arrayModule.ArrayDataSource(dataArray);
    }
    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new viewModel(),
            document.getElementById('datagrid'));
        }
    );
});
```

ArrayDataSource also supports custom sort behavior through its comparator property. For details, consult the [ArrayDataSource API](#) documentation (in the browser page, select **Javascript Only APIs** to view the class documentation).

- CollectionDataSource: Use to define data using the Oracle JET Common Model. The data grid will respond to events generated by the underlying Collection.

For more details about CollectionDataSource, see the [CollectionDataSource API](#) documentation (in the browser page, select **Javascript Only APIs** to view the class documentation).

- PagingDataSource: Use to include pagination. For additional information, see [Work with Pagination](#).
- FlattenedTreeDataSource: Use to display hierarchical data in the data grid. The user can expand and collapse rows in the data grid. For additional information, see [Work with Row Expanders](#).
- CubeDataSource: Use to include aggregate values on column headers, row headers, or both. For additional information, see [Work with CubeDataSource](#).
- Custom data source: Use to provide your own data source for the data grid. The Oracle JET Cookbook contains examples for creating custom data sources, including an example with nested headers.

 **Note:**

Only data grid components that use a custom data source support the use of nested row or column headers.

The Oracle JET Cookbook includes the complete example for the data grid used in this section at [Data Grids](#). The cookbook also includes examples that show row-based data grids, editable data grids, and data grids with visualizations, end headers, custom cell renderers, merged cells, and custom data sources.

Work with CubeDataSource

Use CubeDataSource to render aggregated cubic data in your data grid. You can aggregate values on column headers, row headers, or both column and row headers. You can also define a page axis to filter or otherwise restrict the display.

For example, you may have a collection that contains data for sales, number of units sold, and sales tax data for vehicles sold by car dealerships, and you'd like to aggregate the data to show sales, unit, and tax data by year and city. Your data also contains the type of vehicle sold, its color, and type of drive train, and you'd also like to aggregate the data to show the sales, unit, and tax data grouped by product, color, and drive train.

You can configure `oj-data-grid` with the CubeDataSource to achieve the desired grouping. The following image shows the runtime display.

			New York						Redwood Shores						Boston						
			2014			2015			2014			2015			2014			2015			
			units	sales	tax	units	sales	tax	units	sales	tax	units	sales	tax	units	sales	tax	units	sales		
Coupe	White	FWD	107	889	6.11%	181	1453	5.56%	121	1326	6.39%	137	177	5.01%	81	349	7.03%	131	1726		
		AWD	23	1116	2.38%	54	1099	7.37%	31	897	4.87%	120	149	4.35%	69	1712	5.86%	49	640		
	Black	FWD	93	647	5.36%	62	933	5.58%	55	833	6.54%	107	862	7.37%	169	869	5.58%	112	1062		
		AWD	59	1735	5.76%	124	1337	3.40%	83	1266	2.36%	29	1103	3.45%	148	1499	3.97%	59	808		
	Red	FWD	65	363	6.58%	120	1157	1.73%	61	1269	4.34%	74	1326	1.38%	127	826	4.26%	84	1010		
		AWD	160	596	4.88%	74	544	4.58%	108	660	7.07%	38	577	4.01%	187	1144	6.17%	127	547		
	Sedan	White	FWD	99	456	5.97%	3	955	4.87%	135	1775	4.89%	94	1094	3.99%	55	734	4.27%	103	1490	
			AWD	45	754	8.55%	81	675	2.02%	83	796	7.18%	184	1012	6.00%	166	779	4.61%	31	821	
	Black	FWD	74	676	8.26%	12	64	6.39%	135	1010	0.97%	115	104	6.46%	105	1797	6.19%	25	930		
		AWD	98	980	8.44%	54	1258	2.15%	165	471	3.93%	134	497	8.37%	87	677	8.30%	95	179		
	Red	FWD	69	884	6.19%	62	1599	7.74%	61	1828	1.15%	45	1527	8.03%	67	376	5.01%	190	1505		
		AWD	45	1749	5.28%	121	268	6.34%	111	914	4.73%	67	540	2.71%	105	578	6.55%	123	255		

The data grid uses JSON data for its data source. The code sample below shows a portion of the JSON array.

```
[
  {
    "index": 0,
    "units": 80,
    "sales": 535,
    "tax": 0.0234,
    "year": "2014",
    "gender": "Male",
    "product": "Coupe",
    "city": "New York",
    "drivetrain": "FWD",
    "color": "White"
  },
  {
    "index": 1,
    "units": 95,
    "sales": 610,
    "tax": 0.0721,
    "year": "2015",
    "gender": "Male",
    "product": "Coupe",
    "city": "New York",
    "drivetrain": "FWD",
    "color": "White"
  },
  {
    "index": 2,
    "units": 27,
    "sales": 354,
    "tax": 0.0988,
    "year": "2014",
    "gender": "Female",
    "product": "Coupe",
    "city": "New York",
    "drivetrain": "FWD",
    "color": "White"
  }
]
```

```

        "color": "White"
    },
]

]
```

The data also contains a column for the gender of the buyer which isn't included in the display. The totals displayed in the grid come from applying the aggregation across any JSON rows that match up on type, color, drivetrain, year, and city. For the data in this example, this has the effect of grouping the Male and Female values and applying the aggregation. For example, the units shown in the grid for the New York sales in 2014 of white FWD coupes comes from totaling the highlighted values:

80 + 27 = 107 units

The following code sample shows the markup for the data grid.

```
<ojs-data-grid id="datagrid"
    style="width:100%;height:400px;max-width:851px;"
    aria-label="Cubic Data Source Grid Demo"
    data="[[dataSource]]"
    cell.renderer="[[cellRenderer]]"
></ojs-data-grid>
```

In this example, the datasource is defined in the application's main script.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojmodel', 'ojs/ojcube',
        'ojs/ojconverter-number', 'ojs/ojknockout',
        'ojs/ojdatagrid', 'ojs/ojselectcombobox'],
       function(ko, Bootstrap, Model, Cube, NumberConverter)
{
    function dataGridModel() {
        var cube = null;
        var topLevelItems = [];

        var collection = new Model.Collection(null, {
            url: 'cookbook/dataCollections/dataGrid/cubeGrid/cubedata.json'
        });
        var dataArr = null;

        function generateCube(dataArr, axes) {
            return new DataValueAttributeCube(dataArr, axes,
                [{attribute:'units',aggregation:Cube.CubeAggType['SUM']},
                 {attribute:'sales'},
                 {attribute:'tax',aggregation:Cube.CubeAggType['AVERAGE']}]);
        }

        collection.fetch({success:function() {
            dataArr = collection.map(function(model) {
                return model.attributes;
            });
            cube = generateCube(dataArr, axes);

            var topLevelItems = getItemsForLevel(2, 0);
            this.colors(topLevelItems);

            this.currentColor(topLevelItems[0].value);

            this.dataSource(new Cube.CubeDataGridDataSource(cube));
        }.bind(this)});
```

```

var axes = [
    {axis: 0, levels: [
        {attribute: 'city'},
        {attribute: 'year'},
        {dataValue: true}]},
    {axis: 1, levels:[
        {attribute: 'product'}
        {attribute: 'color'},
        {attribute: 'drivetrain'}]}];

this.dataSource = ko.observable();

...

Bootstrap.whenDocumentReady().then(
    function()
    {
        ko.applyBindings(new dataGridModel(),
document.getElementById('wrapper'));
    }
);
});
```

The `CubeDataGridDataSource` parameter is instantiated with the return value of the `generateCube()` function, which is an `Cube` object. The `Cube` class provides functions for the `DataProviderAttributeCube` class which creates the object to convert the row set data into grouped, cubic data. The `DataProviderAttributeCube` constructor takes the following parameters:

- `rowset`: An array of objects containing name-value pairs

In this example, the JSON data is mapped to the `dataArr` object which is defined as an `Collection` object.

```
var collection = new Collection(null, {
    url: 'cookbook/dataCollections/dataGrid/cubeGrid/cubedata.json'
});
```

- `layout`: An array of objects that contains the axis number and levels to use for the aggregation

The axis number indicates where you want the aggregated data displayed: 0 for column headers, and 1 for row headers. The levels tell the cube which values to aggregate and the order to display them. The code sample below shows the layout used in this example. The `dataValue` property indicates which level to display for the units, sales, and tax aggregated values.

```

var axes = [
    {axis: 0, levels: [
        {attribute: 'city'},
        {attribute: 'year'},
        {dataValue: true}]},
    {axis: 1, levels: [
        {attribute: 'product'},
        {attribute: 'color'},
        {attribute: 'drivetrain'}]}];
```

You can configure an additional axis that you can use to filter or otherwise restrict the display of data. For example, you can add a third axis which includes the color and drivetrain attributes, as shown in the following code sample:

```
var axes = [
    {axis: 0, levels: [
        {attribute: 'city'},
        {attribute: 'year'},
        {dataValue: true} ]},
    {axis: 1, levels: [
        {attribute: 'product'} ]},
    {axis: 2, levels:[
        {attribute: 'color'},
        {attribute: 'drivetrain'} ]}];
```

When you add a third axis to the page, the axis will not be visible. However, you can use the setPage() method to use the axis as a page axis to filter the display. For example, setting color and drivetrain on the third axis has the effect of restricting the display to aggregate only those values that match both the color and drivetrain attributes.

In this example, you have six page combinations: White FWD, White AWD, Black FWD, Black AWD, Red FWD, and Red AWD products. If you set the color attribute to White and the drivetrain attribute to 4WD as shown in the following image, when you render the page only the values for White, 4WD products (Coupe, Sedan, Wagon, SUV, Van, and Truck) display on the screen.

The screenshot shows a user interface for filtering data. At the top left is a dropdown menu with two items: 'color' and 'White'. Below it is another dropdown menu with two items: 'drivetrain' and 'FWD'. Both of these dropdowns are highlighted with a red border. To the right is a table of sales data for various vehicle types (Coupe, Sedan, Wagon, SUV, Van, Truck) across three cities (New York, Redwood Shores, Boston) for the years 2014 and 2015. The table has columns for units sold, sales, and tax percentages. The 'Coupe' row is also highlighted with a red border. The table is scrollable, indicated by a scrollbar at the bottom.

	New York						Redwood Shores						Boston					
	2014			2015			2014			2015			2014			2015		
	units	sales	tax	units	sales	tax	units	sales	tax	units	sales	tax	units	sales	tax	units	sales	tax
Coupe	107	889	6.11%	181	1453	5.56%	121	1326	6.39%	137	177	5.01%	81	349	7.03%	131		
Sedan	99	456	5.97%	3	955	4.87%	135	1775	4.89%	94	1094	3.99%	55	734	4.27%	103		
Wagon	82	593	6.82%	49	534	6.76%	77	1334	5.85%	155	1060	0.43%	33	791	7.27%	106		
SUV	54	548	4.59%	69	1182	6.86%	107	1457	6.70%	102	1080	9.07%	117	622	5.38%	74		
Van	110	614	5.35%	80	885	8.38%	13	1578	5.35%	148	713	2.12%	99	1513	8.47%	158		
Truck	83	1420	6.74%	135	918	0.88%	99	1435	5.24%	136	782	3.72%	146	768	5.20%	78		

- **dataValues:** An array of objects that contains the name of the attribute in the row set that represents the data, an optional label, and the aggregation type. The aggregation type is also optional and defaults to `SUM`. You can also set it to one of the aggregation types shown in the following table.

Aggregation Type	Description
AVERAGE	Average the values.
COUNT	Count the number of values.
CUSTOM	Specify a custom callback function to do the aggregation.
FIRST	Substitute the first value encountered in the collection.
MAX	Calculate the maximum of the values.
MIN	Calculate the minimum of the values.
NONE	Substitute a null for the value.
STDDEV	Calculate the standard deviation of the values.
SUM	Total the values.
VARIANCE	Calculate the variance of the values.

The code sample below shows the function that defines the `dataValues` for the cube used in this section. The cube will sum the units and sales data and will average the sales tax data.

```
function generateCube(dataArr, axes) {
    return new DataValueAttributeCube(dataArr, axes,
        [{attribute:'units',aggregation:CubeAggType['SUM']},
         {attribute:'sales'},
         {attribute:'tax',aggregation:CubeAggType['AVERAGE']}]);
}
```

The Oracle JET Cookbook demos at [Data Grids \(Cubic Data Source\)](#) contain the complete code for the example used in this section, including the complete code for setting the page axis and rendering the cell content. For additional detail and methods that you can use for working with the cube, see the [CubeDataSource API](#) documentation (in the browser page, select **Javascript Only APIs** to view the class documentation).

Work with List Views

The `oj-list-view` element enhances the HTML list (`ul`) element to provide a themable, WAI-ARIA compliant component that displays a list of data.

The `oj-list-view` element supports single and multiple selection, high water mark scrolling when working with table data, and hierarchical content. By default, each list item's children items are hidden upon initial display. To specify that one or more items are expanded when `oj-list-view` first loads, use the `expanded` attribute and specify the key set containing the keys of the items that should be expanded.

```
<oj-list-view id="listview"
    aria-label="list with hierarchical data"
    expanded="[[expanded]]"
    item.focusable="[[itemOnly]]">


- A
  - </span>

```

```

        <span class="name">Amy Bartlet</span>
    </div>
    <div>
        <span class="oj-text-xs oj-text-secondary-color">Vice
President</span>
    </div>
    </div>
</li>
...contents omitted
</ul>
</li>
<li id="b">
    ...contents omitted
</li>
<li id="c">
    ...contents omitted
</li>
</ul>
</oj-list-view>
```

In your application's viewModel, define the key set. The following example sets the `oj-list-view` element to expand the cookbook item on initial display. Note that you must also add the `ojkeyset` module and `keySet` function to the require definition.

```

require(['knockout', 'ojs/ojkeyset', 'ojs/ojknockout', 'ojs/ojlistview'],
function(ko, keySet)
{
    function viewModel()
    {
        this.itemOnly = function(context)
        {
            return context['leaf'];
        };
        this.expanded = new keySet.KeySetImpl(['a', 'b', 'c']);
    } ...contents omitted );
```

You can use `AllKeySetImpl` class to set all the list items in a `oj-list-view` element to be expanded on initial display as follows:

```
this.expanded = new keySet.AllKeySetImpl();
```

For the complete example, see [List View Using Expanded Option](#) using `expanded` attribute.

Understand the Data Requirements for List Views

The data provider for the `oj-list-view` component can be one of the following:

- Flat or hierarchical static HTML

The following image shows examples of list views that display static content, one with flat content and one using a nested list for hierarchical content.

The left screenshot displays a flat list view with 7 items, each showing a small profile picture and the employee's name and title. The right screenshot shows a hierarchical list view where items are grouped into sections labeled A and B. Section A contains 4 items, and section B contains 4 items. Each section has a collapse/expand icon.

Section	Name	Title
A	Chris Black	Oracle Cloud Infrastructure GTM Channel Director EMEA
	Christine Cooper	Senior Principal Escalation Manager
	Chris Benalamore	Area Business Operations Director EMEA & JAPAC
	Christopher Johnson	Vice-President HCM Application Development
	Samire Christian	Consulting Project Technical Manager
	Kurt Marchris	Customer Service Analyst
	Zelda Christian Cooperman	Senior Principal Escalation Manager
B	Alfred Marchris	Principal Developer
	Andrew Chrismon	Consulting Project Technical Manager
	Annett Christy	Area Business Operations Director EMEA & JAPAC
	Bart Christian	Consulting Project Technical Manager
Ben Marchris	Customer Service Analyst	
Brie Christian Cooperman	Senior Principal Escalation Manager	

The following code sample shows a portion of the markup used to create the list view with hierarchical static content using the `oj-list-view` element. The element allows multiple selections and expands both list item groups A and B upon initial display.

```
<oj-list-view id="listview" aria-label="list with hierarchical data"
selection-mode="multiple" item.selectable="[[itemOnly]]">


- A
  - </span>


Amy Bartlet



Vice President

... contents omitted
- B

... contents omitted


</oj-list-view>
```

The code to apply the binding is shown below.

```

require(['knockout', 'ojs/ojbootstrap', 'ojs/ojlistview'],
    function(ko, Bootstrap)
{
    function viewModel()
    {
        this.itemOnly = function(context)
        {
            return context['leaf'];
        };
    }

    Bootstrap.whenDocumentReady().then(
        function()
        {
            ko.applyBindings(new viewModel(),
document.getElementById('listview'));
        }
    );
});

```

See [List View Using Static Data](#) for the complete example to create `oj-list-view` components using static data and static hierarchical data.

- Oracle JET ArrayTreeDataProvider.

To use a `ArrayTreeDataProvider`, you specify the method that returns the tree data in the `data` attribute for the component.

```

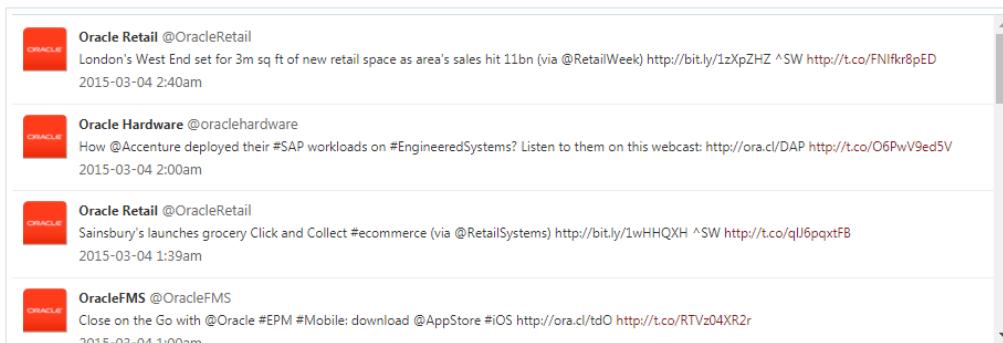
<oj-list-view id="listview" aria-label="list using json data"
    data="[[dataProvider]]" selection-mode="single"
item.renderer="[[renderer]]" item.focusable="[[itemOnly]]"
    item.selectable="[[itemOnly]]" drill-mode="none">
</oj-list-view>

```

The Oracle JET Cookbook contains the complete example for creating a list view with JSON data using an `ArrayTreeDataProvider` at [List View Using Hierarchical JSON Data](#). You can also find an example for creating a list view with JSON data using an `IndexerModelTreeDataProvider` that also contains an index at [Indexer](#).

- Oracle JET CollectionDataProvider and PagingDataProviderView.

Use the `CollectionDataProvider` data attribute when `Collection` is the model for the underlying data, or you want to add scrolling or pagination to your `oj-list-view`. The following image shows a list view using a `CollectionDataProvider` object for its data.



In this example, high water mark scrolling is enabled through the component attribute's scroll-policy. High water mark scrolling is a scroll policy (loadMoreOnScroll) that fetches the next set of rows when the user scrolls towards the end of the scrollable list.

```
<ojs-list-view id="listview" aria-label="list using collection"
    style="width:100%;height:300px;overflow-x:hidden"
    data="[[dataProvider]]"

    item.renderer="[[KnockoutTemplateUtils.getRenderer('tweet_template')]]"
    selection-mode="single"
    scroll-policy="loadMoreOnScroll" scroll-policy-options.fetch-
    size="15">
</ojs-list-view>
```

For the complete example, including the script that creates the CollectionDataProvider object, see [List View Using Collection](#).

You can also find cookbook examples that add [Pull to Refresh](#) and [Swipe to Reveal](#) touch capability to an oj-list-view created with the CollectionDataProvider object.

- Oracle JET ArrayDataProvider

Use the ArrayDataProvider when you want to use an observable array or array as data for List View.

The following example shows the HTML markup using the data attribute in the oj-list-view element:

```
<ojs-list-view id="listview" aria-label="list using observable array"
    data="[[dataProvider]]" selection-mode="multiple"
    selected="{{selectedItems}}" item.renderer="[[renderer]]">
</ojs-list-view>
```

In the following JavaScript sample code, data in an array is passed to a variable, dataProvider, using the ArrayDataProvider object:

```
this.dataProvider = new ArrayDataProvider(this.allItems, {'keyAttributes': 'id'});
```

For the complete example, see [List View Using and ArrayDataProvider](#).

Note:

If you do not specify a data provider in the list view component's data attribute, Oracle JET will examine the child elements inside the root element and use it for static content. If the root element has no children, Oracle JET will render an empty list.

Work with List Views and Inline Templates

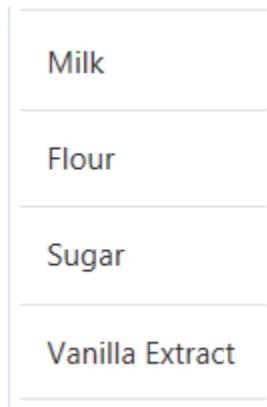
Use inline templates to specify what gets rendered inside the list items.

You can use an inline template to contain the markup for your list item content and reference the name of the template in the <template> element's slot attribute. Unlike specifying the elements to display the content in the list format, you must specify

only the content to be rendered within the list and not the `` elements. When the inline template is executed for each value passed to it, the inline template will have access to the binding context containing the following properties:

- `$current` - an object that contains information for the current item being rendered
 - `componentElement` - the `<ojs-list-view>` custom element
 - `data` - the data for the current item being rendered
 - `index` - the zero-based index of the current item being rendered
 - `key` - the key of the current item being rendered
 - `depth` (available when hierarchical data is provided) - the depth of the current item being rendered. The depth of the first level children under the invisible root is 1
 - `leaf` (available when hierarchical data is provided) - whether the current item is a leaf node or not
 - `parentKey` (available when hierarchical data is provided) - the key of the parent item. The parent key is null for root nodes
- `alias` - if `data-oj-as` attribute is specified, the value will be used to provide an application-named alias for `$current` value.

In the following image, an inline template specifies the content to be rendered within the list created using `oj-list-view`.



The HTML code sample below shows a portion of the markup for a list view using an inline template. The template uses the template-level alias set on the `template` element by using the `data-oj-as` attribute. The alias provides the alias for a specific template instance and has the same subproperties as the `$current` variable. See [template slots](#) in the API documentation.

```
<oj-list-view id="listview" aria-label="list using observable array"
    data="[[dataProvider]]" selection-mode="multiple"
    selected="{{selectedItems}}>
    <template slot="itemTemplate" data-oj-as="item">
        <span>
            <oj-bind-text value='[[item.data.item]]'></oj-bind-text>
        </span>
    </template>
</oj-list-view>
```

```
</template>
</oj-list-view>
```

The `selected` attribute monitors the current selected value in the array.

In the following JavaScript sample code, array data in a variable, `allItems`, is passed to a variable, `dataProvider`, using the `ArrayDataProvider` object:

```
this.itemToAdd = ko.observable("");
this.allItems = ko.observableArray([
    {"id": 1, "item": "Milk"},
    {"id": 2, "item": "Flour"},
    {"id": 3, "item": "Sugar"},
    {"id": 4, "item": "Vanilla Extract"}
]);
this.selectedItems = new ObservableKeySet();
var lastItemId = this.allItems().length;

this.dataProvider = new ArrayDataProvider(this.allItems,
    {'keyAttributes': 'id'});
```

See [List View Using Inline Template](#) for the complete example to create a list view using an inline template.

Work with Pagination

Use the `oj-paging-control` element to add pagination to the `oj-table` and `oj-data-grid` elements or the HTML list element. Pagination displays the number of pages and rows in the table or grid, and the user can use pagination to move between pages, jump to a specific page, or navigate to the first or last page of data.

In the following image, `oj-paging-control` element is added to the `oj-table` element and initialized with the default display.

Department Id ↑	Department Name ↑	Location Id ↑	Manager Id ↑
10015	ADFPM 1001 neverending	200	300
556	BB	200	300
10	Administration	200	300
20	Marketing	200	300
30	Purchasing	200	300
40	Human Resources1	200	300
50	Administration2	200	300
60	Marketing3	200	300
70	Purchasing4	200	300
80	Human Resources5	200	300

Page of 5 (1-10 of 45 items) | < < 2 3 4 5 > > | Last

To add pagination to a table, define the table's data as a `PagingDataProviderView` object, and add the `oj-paging-control` using the same `PagingDataProviderView` object for its data attribute. Specify the number of rows to display in the `oj-paging-control` element's `page-size` attribute. The code sample below shows the markup defining the table and pagination components. In this example, `page-size` is set to 10.

```
<div id="pagingControlDemo">
    <oj-table id="table" summary="Department List" aria-label="Departments Table"
        data='[[pagingDataProvider]]'
        columns='[{"headerText": "Department Id", "field": "DepartmentId"}, {"headerText": "Department Name", "field": "DepartmentName"}, {"headerText": "Location Id", "field": "LocationId"}, {"headerText": "Manager Id", "field": "ManagerId"}]'
        style='width: 100%;'
    <oj-paging-control id="paging" data='[[pagingDataProvider]]' page-size='10' slot='bottom'>
    </oj-paging-control>
</oj-table>
</div>
```

The script that populates the `pagingDataProvider` with data and completes the Knockout binding is shown below. In this example, the table's data is defined in a `PagingDataProviderView` object, and the `pagingDataProvider` defines the `PagingDataProviderView` as a new `ArrayDataProvider` object.

```
require(['knockout', 'ojs/objbootstrap','ojs/ojpagingdataproviderview',
        'ojs/ojarraydataprovider', 'ojs/ojknockout', 'ojs/ojtable', 'ojs/ojpagingcontrol'],
       function(ko, Bootstrap, PagingDataProviderView, ArrayDataProvider)
{
    function viewModel()
    {
        var deptArray = [{DepartmentId: 10015, DepartmentName: 'ADFPM 1001 neverending', LocationId: 200, ManagerId: 300}, {DepartmentId: 556, DepartmentName: 'BB', LocationId: 200, ManagerId: 300}, {DepartmentId: 10, DepartmentName: 'Administration', LocationId: 200, ManagerId: 300}, {DepartmentId: 20, DepartmentName: 'Marketing', LocationId: 200, ManagerId: 300}, {DepartmentId: 30, DepartmentName: 'Purchasing', LocationId: 200, ManagerId: 300}, {DepartmentId: 40, DepartmentName: 'Human Resources1', LocationId: 200, ManagerId: 300}, {DepartmentId: 50, DepartmentName: 'Administration2', LocationId: 200, ManagerId: 300}, {DepartmentId: 60, DepartmentName: 'Marketing3', LocationId: 200, ManagerId: 300}, {DepartmentId: 70, DepartmentName: 'Purchasing4', LocationId: 200, ManagerId: 300}, {DepartmentId: 80, DepartmentName: 'Human Resources5', LocationId: 200, ManagerId: 300}];
```

```
neverending', LocationId: 200, ManagerId: 300},
        {DepartmentId: 556, DepartmentName: 'BB', LocationId: 200, ManagerId:
300},
        {DepartmentId: 10, DepartmentName: 'Administration', LocationId: 200,
ManagerId: 300},
        {DepartmentId: 20, DepartmentName: 'Marketing', LocationId: 200,
ManagerId: 300},
        ...contents omitted
        {DepartmentId: 13022, DepartmentName: 'Human Resources15', LocationId:
200, ManagerId: 300}];
    this.pagingDataProvider = new PagingDataProviderView(new
ArrayDataProvider(deptArray, {keyAttributes: 'DepartmentId'}));
}

var vm = new viewModel;

Bootstrap.whenDocumentReady().then(
    function()
    {
        ko.applyBindings(vm, document.getElementById('pagingControlDemo'));
    }
);
});
```

To add a paging control to `oj-data-grid`, define the data grid's data as a `PagingDataProviderView` object, and add the `oj-paging-control` element using the same `PagingDataProviderView` object for its data attribute. Set the `page-size` attribute equal to the fetch size for the data collection, if creating the data grid from an `CollectionDataProvider` object.

The Oracle JET Cookbook contains complete examples for adding pagination to `oj-table`, `oj-data-grid`, and HTML lists at [Pagination](#). You can also find the link to the [oj-paging-control](#) API documentation as well as examples that show different options for customizing the paging display.

For additional information about working with the `oj-table` element, see [Work with Tables](#). For more information about working with the `oj-data-grid` element, see [Work with Data Grids](#).

Work with Row Expanders

Use the Oracle JET `oj-row-expander` element to expand or collapse rows in a data grid or table to display hierarchical data. The `oj-row-expander` element renders the expand/collapse icon with the appropriate indentation and works directly with the flattened data source. In the following image, the row expander is used with an `oj-data-grid` element, and the user can expand the tasks to display subtasks and dates.

	Resource	Start Date	End Date
▼ Task 1	Chadwick	1/1/2014	10/1/2014
▶ Task 1-1	Chris	1/1/2014	3/1/2014
▼ Task 1-2	Jim	3/1/2014	6/1/2014
Task 1-2-1	Jay	3/1/2014	5/1/2014
Task 1-2-2	Karin	5/1/2014	6/1/2014
Task 1-3	Chadwick	6/1/2014	8/1/2014
Task 1-4	Chris	8/1/2014	10/1/2014
▶ Task 2	Henry	4/1/2014	12/1/2014
▶ Task 3	Jay	5/1/2014	11/1/2014

To use `oj-row-expander` with `oj-data-grid`, create an `oj-data-grid` element and assign a meaningful ID to it and specify properties on the `oj-data-grid`. In the HTML file, specify a row header template that adds the `oj-row-expander` element to the row header. Specify the grid's data in an `FlattenTreeDataSource` object. In the code sample below, the `oj-row-expander` element is defined in the data grid's row template. The element's `context` option references the object obtained from the data grid's column renderer.

```
<oj-data-grid
    id="datagrid"
    style="width:100%;max-width:502px;height:400px"
    aria-label="Data Grid with Row Expander"
    data="[[dataSource]]"
    selection-mode.cell="single"

    header.column.renderer="[[KnockoutTemplateUtils.getRenderer('column_header_template')]]"
        header.column.style="width:100px;"
        header.column.resizable.width="enable"

    header.row.renderer="[[KnockoutTemplateUtils.getRenderer('row_header_template')]]"
    "
        header.row.style="width:200px;"
        cell.class-name="oj-helper-justify-content-flex-start"
    ></oj-data-grid>

<script type="text/html" id="column_header_template">
    <oj-bind-if test="$context.key=='resource'">
        <span> <oj-bind-text value="Resource"></oj-bind-text></span>
```

```
</oj-bind-if>
<oj-bind-if test="$context.key=='start'">
    <span> <oj-bind-text value="Start Date"></oj-bind-text></span>
</oj-bind-if>
<oj-bind-if test="$context.key=='end'">
    <span> <oj-bind-text value="End Date"></oj-bind-text></span>
</oj-bind-if>
</script>

<script type="text/html" id="row_header_template">
    <oj-row-expander context="[$context]"></oj-row-expander>
    <span><oj-bind-text value="[$context.data]"></oj-bind-text></span>
</script>
```

The data for the `FlattenTreeDataSource` object can come from local or fetched JSON, or an `Collection` object. In the example in this section, the data is read from a JSON file. The code sample below shows a portion of the JSON.

```
        }  
    ]
```

The script that reads the JSON file and defines the `dataSource` object as an `FlattenTreeDataGridDataSource` is shown below.

```
require(['knockout', 'ojs/objbootstrap', 'ojs/ojflattentreddatagriddatasource',
        'ojs/ojjsontreedatasource', 'ojs/ojknockouttemplateutils',
        'text!../cookbook/dataCollections/rowExpander/dataGridRowExpander/
projectData.json',
        'ojs/ojknockout', 'ojs/ojdatagrid', 'ojs/ojrowexpander'],
       function(ko, Bootstrap, flattenedModule, JsonTreeDataSource,
KnockoutTemplateUtils, jsonDataStr)
{
    function viewModel()
    {
        var self = this;
        self.KnockoutTemplateUtils = KnockoutTemplateUtils;
        var options = {
            'rowHeader': 'name',
            'columns': ['resource', 'start', 'end']
        };
        self.dataSource = ko.observable();

        self.dataSource(new flattenedModule.FlattenTreeDataGridDataSource(
            new JsonTreeDataSource(JSON.parse(jsonDataStr)), options));
    }

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new viewModel(),
document.getElementById('datagrid'));
        }
    );
});
```

To use the row expander with `oj-data-grid`, add the `oj-row-expander` element to the HTML markup, and specify the grid data in an `FlattenTreeDataGridDataSource` object. The Oracle JET Cookbook at [Row Expanders](#) contains an example that uses the row expander with `oj-data-grid`. The cookbook also contains the complete code for the example in this section and a link to the API documentation for `oj-row-expander`. In addition, you can find examples that use a `Collection` object for the data of an `oj-table` and that initialize the row expander with one or more rows expanded.

For additional information about working with the `oj-data-grid` element, see [Work with Data Grids](#). For more information about working with the `oj-table` element, see [Work with Tables](#).

Work with Tables

The Oracle JET `oj-table` component enhances the HTML `table` element to provide support for accessibility, custom cell and row templates and renderers, theming, row expansion, pagination, editable table, and editable array and collection tables.

<input type="checkbox"/>	Department Id	↑	Department Name	↑↓	Location Id	↑↓	Manager Id	↑↓
<input type="checkbox"/>	3		ADPFM 1001 neverending		200		300	
<input type="checkbox"/>	5		BB		200		300	
<input type="checkbox"/>	10		Administration		200		300	

The editing behavior in an `oj-table` element is enabled when the `edit-mode` attribute is set to `rowEdit`. Once the `edit-mode` attribute is set, the table initially renders as read-only before you can proceed to edit a single row at a time. To edit a row within a table, you can either double-click or press F2 to edit the cells within the row. You can also programmatically set the rows to single-row editable mode by using the `edit-row` attribute with a row key.

The [Tables](#) demos in the Oracle JET Cookbook include the complete example for this table and a link to the [oj-table](#) API documentation. The cookbook also includes an example that creates the `oj-table` component using data defined in an `ArrayDataProvider` object and examples that show tables with custom row and cell templates, selection, sorting, reordering, scrolling, custom cell renderers, editable table, and drag and drop support.

Understand the Data Requirements for Table

You can define the table's data in an array using the `ArrayDataProvider` object, in a row expander using `FlattenedTreeDataProvider`, in a paging functionality using `PagingDataProviderView`, or in a collection, such as from an external data source, using `CollectionDataProvider`.

The data source for the `oj-table` element can be one of the following:

- Oracle JET `ArrayDataProvider`

Use the `ArrayDataProvider` when you want to use an observable array or array as data for the table.

The following example shows the HTML markup using the `data` attribute in the `oj-table` element:

```
<oj-table id='table' aria-label='Departments Table'
          data='[[dataProvider]]'
          ...
          columns='[{"headerText": "Department Id",
                     "field": "DepartmentId",
                     "headerClassName": "oj-sm-only-hide",
                     "className": "oj-sm-only-hide"}, {"resizable": "enabled"
                     {"headerText": "Department Name",
                     "field": "DepartmentName"}, {"resizable": "enabled"
                     {"headerText": "Location Id",
                     "field": "LocationId",
                     "headerClassName": "oj-sm-only-hide",
                     "className": "oj-sm-only-hide"}, {"resizable": "enabled"}]
```

```

        {"headerText": "Manager Id",
         "field": "ManagerId",
         "resizable": "enabled"}]]'
      style='width: 100%; height:100%;'
    </oj-table>

```

In the following JavaScript sample code, array data in a variable, deptArray, is passed to a variable, dataProvider, using the `ArrayDataProvider` object:

```

var deptArray = [{DepartmentId: 1001, DepartmentName: 'ADDFPM 1001
neverending', LocationId: 200, ManagerId: 300},
    {DepartmentId: 556, DepartmentName: 'BB', LocationId: 200, ManagerId:
300},
    {DepartmentId: 110, DepartmentName: 'Marketing13', LocationId: 200,
ManagerId: 300},
    {DepartmentId: 120, DepartmentName: 'Purchasing14', LocationId: 200,
ManagerId: 300},
    {DepartmentId: 130, DepartmentName: 'Human Resources15', LocationId:
200, ManagerId: 300}];
self.dataProvider = new ArrayDataProvider(deptArray, {keyAttributes:
'DepartmentId'});

```

For the complete example, see [Table using ArrayDataProvider](#).

- Oracle JET `FlattenedTreeDataProvider`

Use the `FlattenedTreeDataProvider` for a table when you want to pass data to an `oj-row-expander` component to expand or collapse rows to display hierarchical data.

The following example shows the HTML markup using the `data` attribute in the `oj-table` element and `oj-row-expander` in the table definition element:

```

<oj-table
  id="table" aria-label="Tasks Table"
  data="[[dataProvider]]"
  row-renderer='[[KnockoutTemplateUtils.getRenderer("row_template",
true)]]'>
  columns='[{"headerText": "Task Name", "sortProperty": "name"},
            {"headerText": "Resource", "sortProperty": "resource"},
            {"headerText": "Start Date", "sortProperty": "start"},
            {"headerText": "End Date", "sortProperty": "end"}]'>
</oj-table>
<script type="text/html" id="row_template">
  <tr>
    <td>
      <oj-row-expander context="[$context.rowContext]"></oj-row-expander>
      <span><oj-bind-text value="[$context.data.name]"></oj-bind-text></
span>
    </td>
    <td>
      <span><oj-bind-text value="[$context.data.resource]"></oj-bind-
text></span>
    </td>
    <td>
      <span><oj-bind-text value="[$context.data.start]"></oj-bind-text></
span>
    </td>
    <td>
      <span><oj-bind-text value="[$context.data.end]"></oj-bind-text></
span>
    </td>
  </tr>
</script>

```

```
</tr>
</script>
```

In the following JavaScript sample code, the `dataProvider` variable is instantiated as the `FlattenedTreeDataProvider`:

```
self.dataProvider = ko.observable();
var arrayTreeDataProvider = new
ArrayTreeDataProvider(JSON.parse(jsonDataStr), {keyAttributes: 'id'});
self.dataProvider(new FlattenedTreeDataProvider(arrayTreeDataProvider,
{expanded: self.expanded}));
```

For the complete example, see [Table using FlattenedTreeDataProvider](#).

- Oracle JET PagingDataProviderView

Use the `PagingDataProviderView` to add paging functionality to a table.

The following example shows the HTML markup using the `data` attribute in the `oj-paging-control` element within an `oj-table` element:

```
<oj-table id='table' aria-label='Departments Table'
    data='[[pagingDataProvider]]'
    columns='[{"headerText": "Department Id", "field": "DepartmentId"}, 
        {"headerText": "Department Name", "field": 
        "DepartmentName"}, 
        {"headerText": "Location Id", "field": "LocationId"}, 
        {"headerText": "Manager Id", "field": "ManagerId"}]' 
    style='width: 100%;'
    <oj-paging-control id="paging" data='[[pagingDataProvider]]' page-
size='10' slot='bottom'>
    </oj-paging-control>
</oj-table>
```

In the following JavaScript sample code, the table data is defined in a JavaScript array, `deptArray`, which is passed into the definition for the `ArrayDataProvider`. The `self.pagingDataProvider` is then defined as a `PagingDataProviderView` which gets its data from the `ArrayDataProvider`.

```
var deptArray = [{DepartmentId: 10015, DepartmentName: 'ADPMP 1001
neverending', LocationId: 200, ManagerId: 300},
    {DepartmentId: 556, DepartmentName: 'BB', LocationId: 200, ManagerId:
300},
    {DepartmentId: 10, DepartmentName: 'Administration', LocationId: 200,
ManagerId: 300},
    {DepartmentId: 20, DepartmentName: 'Marketing', LocationId: 200,
ManagerId: 300},
    {DepartmentId: 30, DepartmentName: 'Purchasing', LocationId: 200,
ManagerId: 300},
    {DepartmentId: 13022, DepartmentName: 'Human Resources15', LocationId:
200, ManagerId: 300}];
self.pagingDataProvider = new PagingDataProviderView(new
ArrayDataProvider(deptArray, {keyAttributes: 'DepartmentId'}));
```

For the complete example, see [Table using PagingDataProviderView](#).

- `CollectionDataProvider`: Use when your data is coming from an `Collection` object, typically representing an external data source.

Understand `oj-table` and Sorting

The `oj-table` element enables single column sorting by default if the underlying data supports sorting. Using the `sortable` property of the element's `columnDefaults` attribute, you can control sorting for the entire table, or you can use the `sortable` property of the `columns` attribute to control sorting on specific columns.

When you configure a column for sorting, the column header displays arrow indicators to indicate that the column is sortable when the user hovers over the column header. In the following image, the Department ID is sortable, and the sort indicator is showing a down arrow to indicate that the sort is currently descending. The user can select the down arrow to change the sort back to ascending.

<input type="checkbox"/>	Department Id	Department Name	Location Id	Manager Id
<input type="checkbox"/>	313022	Human Resources	15	200
<input type="checkbox"/>	312022	Purchasing	14	200
<input type="checkbox"/>	311022	Marketing	13	200

The `sortable` attribute supports the following options:

- `auto`: Sort the table or indicated column if the underlying data supports sorting (default).
- `disabled`: Disable sorting on the table or indicated column.
- `enabled`: Enable sorting on the entire table or indicated column.

To enable sorting on specific columns:

- Set the `sortable` property to `none` on the table's `columnDefaults` attribute to disable the `auto` default behavior.
- Set the `sortable` property to `enabled` on the columns that you want to sort.

The following code sample shows the markup to create the department table shown in this section. In this example, the table is configured to enable sorting on only the Department Id column.

```
<oj-table id="table"
    aria-label="Departments Table"
    data='[[dataProvider]]'
    column-defaults='{"sortable": "disabled"}'
    columns='[{"headerText": "Department Id",
        "field": "DepartmentId",
        "sortable": "enabled"},
      {"headerText": "Department Name",
        "field": "DepartmentName"},
      {"headerText": "Location Id",
        "field": "LocationId"},
      {"headerText": "Manager Id",
        "field": "ManagerId"}]'>
```

`oj-table` element sorting uses standard JavaScript array sorting. If your application requires custom sorting, you can use [ArrayDataProvider](#) and its `sortComparators` property. For an example, see [Tables - Custom Sort](#).

For additional information about `oj-table` and sorting options, see the [oj-table API](#) documentation.

For examples in the Oracle JET Cookbook that implement tables and table sorting, see [Tables](#).

Work with Tables and Inline Templates

Use inline templates to specify what gets rendered inside the tables.

You can use an inline template to contain the markup for your table content and reference the name of the template in the `<template>` element's slot attribute. Unlike specifying the table elements, such as column header and column footer to display the content in the table format, you must specify only the content to be rendered within the table and not the `<td>` elements. When the inline template is executed for each value passed to it, the inline template will have access to the binding context containing the following properties:

- `$current` - an object that contains information for the current item being rendered
 - `componentElement` - the `<oj-table>` element
 - `data` - the data for the current item being rendered
 - `index` - the zero-based index of the current item being rendered
 - `key` - the key of the current item being rendered
 - `row` - the data for the current item
- `alias` - if `data-oj-as` attribute is specified, the value will be used to provide an application-named alias for `$current` value.

In the following image, an inline template specifies the content to be rendered within the table created using `oj-table`.

Department Id	Department Name	Location Id	Manager Id	Employee Count	Rating
1001	ADFPM 1001 neverending	200	300	20	★★★★★
556	BB	200	300	10	★★★★★
10	Administration	200	300	30	★★★★★
20	Marketing	200	300	20	★★★★★
30	Purchasing	200	300	50	★★★★★
40	Human Resources1	200	300	200	★★★★★
50	Administration2	200	300	20	★★★★★
60	Marketing3	200	300	30	★★★★★

The HTML code sample below shows a portion of the markup for a table using various inline templates for column header, column footer, data for table cells and others.

```
<ojs-table id='table' aria-label='Departments Table'
    data='[[dataProvider]]'
    columns='{{columnArray}}'
    columns-default='{"sortable": "disabled"}'
    style='width: 100%;'>
    <template slot="cellTemplate" data-oj-ass="cell">
        <ojs-bind-text value="[[cell.data]]"></ojs-bind-text>
    </template>
    <template slot="ratingCellTemplate" data-oj-as="cell">
        <ojs-rating-gauge value='[[cell.data]]' readonly
style="width:60px;height:15px;">
        </ojs-rating-gauge>
    </template>
    <template slot="headerTemplate" data-oj-as="header">
        <ojs-bind-text value="[[header.data]]"></ojs-bind-text>
    </template>
    <template slot="totalFooterTemplate" data-oj-as="cell">
        <div id="table:emp_total"><ojs-bind-text
value='{{emp_total_func(footer)}}'></ojs-bind-text></div>
    </template>
</ojs-table>
```

The selected attribute monitors the current selected value in the array.

In the following JavaScript sample code, array data in a variable, deptArray, is passed to a variable, dataProvider, using the `ArrayDataProvider` object:

```
var deptArray = [{DepartmentId: 1001, DepartmentName: 'ADFPM 1001 neverending',
LocationId: 200, ManagerId: 300, EmployeeCount: 20, Rating: 1},
{DepartmentId: 556, DepartmentName: 'BB', LocationId: 200, ManagerId: 300,
EmployeeCount: 10, Rating: 1},
{DepartmentId: 10, DepartmentName: 'Administration', LocationId: 200,
```

```
        ManagerId: 300, EmployeeCount: 30, Rating: 2},
        {DepartmentId: 20, DepartmentName: 'Marketing', LocationId: 200, ManagerId:
300, EmployeeCount: 20, Rating: 3},
        {DepartmentId: 30, DepartmentName: 'Purchasing', LocationId: 200, ManagerId:
300, EmployeeCount: 50, Rating: 4},
        {DepartmentId: 40, DepartmentName: 'Human Resources1', LocationId: 200,
ManagerId: 300, EmployeeCount: 200, Rating: 5},
        {DepartmentId: 50, DepartmentName: 'Administration2', LocationId: 200,
ManagerId: 300, EmployeeCount: 20, Rating: 1.5},
        {DepartmentId: 60, DepartmentName: 'Marketing3', LocationId: 200, ManagerId:
300, EmployeeCount: 30, Rating: 2.5}]];
self.dataProvider = new ArrayDataProvider(deptArray, {keyAttributes:
'DepartmentId'});
```



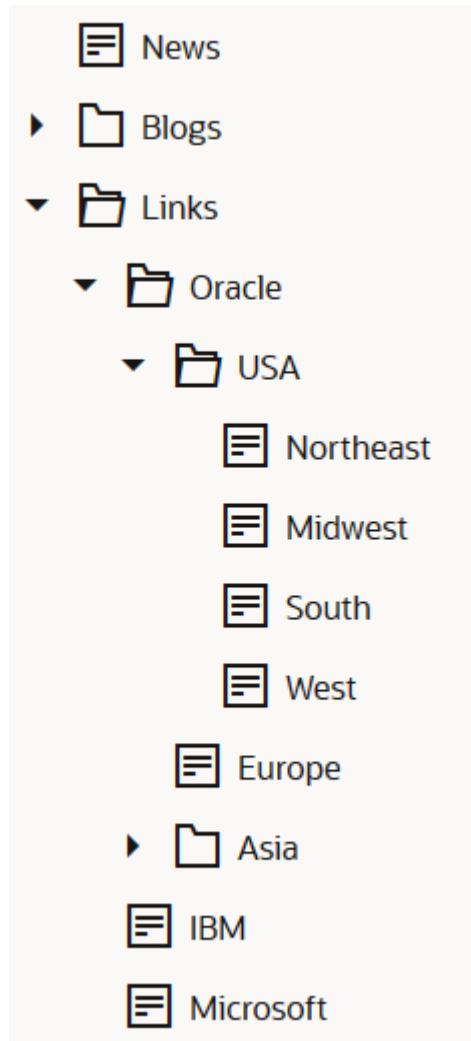
```
self.columnArray = [{ "headerText": "Department Id",
"field": "DepartmentId"}, {"headerText": "Department Name",
"field": "DepartmentName"}, {"headerText": "Location Id",
"field": "LocationId"}, {"headerText": "Manager Id",
"field": "ManagerId"}, {"headerText": "Employee Count",
"field": "EmployeeCount", "footerTemplate": "totalFooterTemplate"}, {"headerText": "Rating",
"field": "Rating", "template": "ratingCellTemplate"}];
```

See [Table Using Inline Template](#) for the complete example to create a table using an inline template.

Work with Tree Views

The `oj-tree-view` component displays the hierarchical relationship between the items of a tree.

Each element in the tree is called a node, and the top levels of the hierarchy are referred to as the root nodes. The descendants of the root nodes are its children, and each child node can also contain children. Users select a node to display its children.



In this example, the expanded Links node is a root node, and the Oracle, IBM, and Microsoft nodes are its children. The Oracle child node contains the USA, Europe, and Asia nodes which also contain child nodes. The USA node is expanded to show its Northeast, Midwest, South, and West child nodes.

To create the tree view component, add the `oj-tree-view` element to the HTML markup. Assign it a meaningful ID and specify attributes on the `oj-tree-view` element. When you have static content you can construct the tree using a predefined HTML unordered list (`ul`) element in the `oj-tree-view` element. Alternatively, you can use a tree view item template to stamp out the parent and child nodes.

The tree view component supports `keySet` based selections. For example, you can use a `KeySet` observable that you bind to the `selected` attribute to handle the representation of the selected items.

Understand the Data Requirements for Tree Views

You can supply data to the tree view using static HTML content or a `ArrayTreeDataProvider` object. The method you choose will depend upon the type of data you provide.

The data source for the `oj-tree-view` component can be one of the following:

- static HTML content: Use when you want to define your data within HTML views. The code sample below shows the static HTML content used to render the tree view above.

```
<oj-tree-view id="treeview" data-oj-binding-provider="none" aria-label="Tree View with Static HTML">
    <ul>
        <li id="news">
            <span class="oj-treeview-item-icon"></span>
            ><span class="oj-treeview-item-text">News</span>
        </li>
        <li id="blogs">
            <span class="oj-treeview-item-icon"></span>
            ><span class="oj-treeview-item-text">Blogs</span>
            <ul>
                ... contents omitted
            </ul>
        </li>
        <li id="links">
            <span class="oj-treeview-item-icon"></span>
            ><span class="oj-treeview-item-text">Links</span>
            <ul>
                <li id="oracle">
                    <span class="oj-treeview-item-icon"></span>
                    ><span class="oj-treeview-item-text">Oracle</span>
                    <ul>
                        <li id="usa">
                            <span class="oj-treeview-item-icon"></span>
                            ><span class="oj-treeview-item-text">USA</span>
                            <ul>
                                <li id="northeast">
                                    <span class="oj-treeview-item-icon"></span>
                                    ><span class="oj-treeview-item-text">Northeast</span>
                                </li>
                                <li id="midwest">
                                    <span class="oj-treeview-item-icon"></span>
                                    ><span class="oj-treeview-item-text">Midwest</span>
                                </li>
                                <li id="south">
                                    <span class="oj-treeview-item-icon"></span>
                                    ><span class="oj-treeview-item-text">South</span>
                                </li>
                                <li id="west">
                                    <span class="oj-treeview-item-icon"></span>
                                    ><span class="oj-treeview-item-text">West</span>
                                </li>
                            </ul>
                        </li>
                    </ul>
                </li>
                <li id="europe">
                    <span class="oj-treeview-item-icon"></span>
                    ><span class="oj-treeview-item-text">Europe</span>
                </li>
                <li id="asia">
                    <span class="oj-treeview-item-icon"></span>
                    ><span class="oj-treeview-item-text">Asia</span>
                    <ul>
                        ... contents omitted
                    </ul>
                </li>
            </ul>
        </li>
    </ul>
</oj-tree-view>
```

```
</ul>
</li>
<li id="ibm">
    <span class="oj-treeview-item-icon"></span>
    <><span class="oj-treeview-item-text">IBM</span>
</li>
<li id="microsoft">
    <span class="oj-treeview-item-icon"></span>
    <><span class="oj-treeview-item-text">Microsoft</span>
</li>
</ul>
</li>
</ul>
</oj-tree-view>
```

Tip:

To ensure that there is no extra white space between the icon and the text in the display, place the icon `span` and the text `span` elements on a single line or use the broken closing bracket notation.

```
<li id="news">
    <span class="oj-treeview-item-icon"></span>
    <><span class="oj-treeview-item-text">News</span>
</li>
```

- `ArrayTreeDataProvider`: Use when the underlying data is a JSON object.

In this example below, the tree view uses JSON data that it obtains from the `treeViewData.json` file referenced and loaded by the `define` block. The sample code also shows the Knockout `applyBindings()` call to complete the component binding.

```
define(['knockout', 'ojs/ojbootstrap', 'ojs/ojarraytreedataprovider',
        'text!../demo/json/treeViewData.json',
        'ojs/oknockout', 'ojs/ojtreeview'],
       function(ko, Bootstrap, ArrayTreeDataProvider, jsonData) {
           function TreeViewModel() {
               this.data = new ArrayTreeDataProvider(JSON.parse(jsonData),
{ keyAttributes: 'id' });
           }

           Bootstrap.whenDocumentReady().then(
               function() {
                   ko.applyBindings(new TreeViewModel(),
document.getElementById('treeview'));
               });
           }
       );
```

- `CollectionTreeDataSource` : Use when your data is coming from an Collection object, typically representing an external data source.

Specifying Initial Expansion in Tree Views

You can specify the initial expansion of the tree nodes when it first loads by using the `expanded` attribute.

By default, each node's children are hidden upon initial display. To specify that one or more nodes are expanded when the tree view first loads, use the `expanded` attribute and specify the key set containing the keys of the items that should be expanded.

```
<oj-tree-view id="treeview"
    data="[[data]]"
    item.renderer="[[renderer]]"
    expanded="{{expanded}}"
    aria-label="Tree View Expansion Demo">
</oj-tree-view>
```

In your application's `viewModel`, define the key set. The example below sets the tree view to expand the `Links` node on initial display. Note that you must also add the `ojknockout-keyset` module and `keySet` function to the require definition.

```
require(['ajs/ojknockout-keyset', 'knockout', 'ajs/objootstrap', 'ajs/ojarraytreedataprovider',
        'ajs/ojknockout', 'ajs/objutton', 'ajs/ojtreetreeview'],
        function(keySet, ko, Bootstrap, ArrayTreeDataProvider, jsonData) {
            function TreeViewModel() {
                var self = this;

                self.data = new ArrayTreeDataProvider(JSON.parse(jsonData),
                {keyAttributes: 'id'});
                self.expanded = new
                keySet.ObservableExpandedKeySet().add(['links']);
                ... contents omitted
            };
        }
    );
```

Examples

The Oracle JET Cookbook contains the complete recipe and code for the samples used in this section at [Tree Views](#). The cookbook also includes a link to the [oj-treeview](#) API documentation which includes additional examples for creating tree views.

Work with Controls

Oracle JET includes buttons, menus, and container elements to control user actions or display progress against a task. For HTML elements such as simple lists, you can use the standard HTML tags directly on your page, and Oracle JET will apply styling based on the application's chosen theme.

For example, you can use the `oj-button` element as a standalone element or include in `oj-buttonset`, `oj-menu`, and `oj-toolbar` container elements.

Navigation components such as `oj-conveyor-belt`, `oj-film-strip`, and `oj-train` use visual arrows or dots that the user can select to move backward or forward through data.

To show progress against a task in a horizontal meter, you can use the `oj-progress-bar` element. To show progress against a task in a circle, you can use the `oj-progress-circle` element.

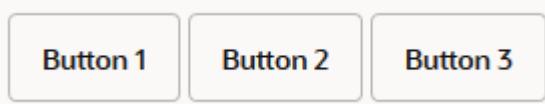
The [Oracle JET Cookbook](#) and [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) include complete demos and examples for using Oracle JET controls, and you may also find the following tips and tricks helpful.

Topics:

- [Work with Buttons](#)
- [Work with Button Sets](#)
- [Work with Conveyor Belts](#)
- [Work with File Picker](#)
- [Work with Film Strips](#)
- [Work with Menus](#)
- [Work with Progress Indicators](#)
- [Work with Tags](#)
- [Work with Toolbars](#)
- [Work with Trains](#)

Work with Buttons

Use the Oracle JET `oj-button` element to display a push button. A push button is an ordinary button that does not stay pressed when clicked. Oracle JET buttons are WAI-ARIA compliant and themable, with appropriate styles for `hover`, `active`, `checked`, and `disabled`.



Create a push button by adding the `oj-button` element to the HTML markup and adding a listener in your view model to respond to click events. The code sample below shows the markup for **Button 1**, **Button 2**, and **Button 3**.

```
<div id='buttons-container'>
    <oj-button id='button1' on-oj-action='[[buttonClick]]'>Button 1</oj-button>
    <oj-button id='button2' on-oj-action='[[buttonClick]]'>
        <span><oj-bind-text value="[[button2Text]]"></oj-bind-text></span>
    </oj-button>
    <oj-button id='button3' on-oj-action='[[buttonClick]]' display='icons'>
        Button 3
    </oj-button>
    <p>
        <p id="last" class="bold">ID of last button to be clicked:</p>
        <span id="results"><oj-bind-text value="[[clickedButton]]"></oj-bind-text></span>
    </p>
</div>
```

```
</p>
</div>
```

Each button's `on-obj-action` attribute specifies the `buttonClick` listener. In this example, the listener simply sets the value of `clickedButton` to the id of the button that the user clicked: `button1`, `button2`, or `button3`.

```
function buttonModel(){
    var self = this;

    self.button2Text = "Button 2";
    self.button3Text = "Button 3";

    self.clickedButton = ko.observable("(None clicked yet)");
    self.buttonClick = function(event){
        self.clickedButton(event.currentTarget.id);
        return true;
    }
}
```

The Oracle JET Cookbook at [Buttons](#) includes complete examples for all buttons, including a section on button styling that shows you how to add icons, control chroming, and configure colors, width, and responsive behavior. [oj-button](#) API documentation describes support for keyboard interaction, accessibility, and event handling.

Work with Button Sets

Use `oj-buttonset-many` or `oj-buttonset-one` to group related toggle buttons, such as a group of checkboxes or radio buttons.

The `oj-buttonset-many` and `oj-buttonset-one` components are themable, visual, and semantic grouping containers for Oracle JET buttons that are also compliant with WAI-ARIA. The `oj-buttonset-many` element allows the user to toggle multiple buttons, as shown in the figure below, and `oj-buttonset-one` allows the user to toggle only one.



Create the button set by adding the `oj-buttonset-many` element or `oj-buttonset-one` element to the application's markup. Set a `value` attribute on the `oj-buttonset-many` or `oj-buttonset-one` element. Inside the button set, create each button from an `oj-option` element.

```
<div id='buttons-container'>
    <oj-label id="mainlabelid" help.definition="Select one or more
    styles.">Styles</oj-label>
    <!-- Set the labelled-by attr on the buttonset for an accessible
```

```
label and help tip -->
<oj-buttonset-many id="formatset" labelled-by="mainlabelid"
value="{{formats}}">
    <oj-option value="Bold">Bold</oj-option>
    <oj-option value="Italic">Italic</oj-option>
    <oj-option value="Underline">Underline</oj-option>
</oj-buttonset-many>
</div>
```

In your view model, set an initial value for the button set and apply the bindings.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/
ojbutton'],
function(ko, Bootstrap)
{
    function ButtonModel() {
        var self = this;
        // observable array bound to the Buttonset:
        self.formats = ko.observableArray(["bold", "underline"]);
    }

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new ButtonModel(),
document.getElementById('buttons-container'));
    });
});
});
```

The Oracle JET cookbook at [Button Sets](#) contains the complete recipe to create the button set shown above.

Here are some tips for working with button sets:

- A button set that contains radios should contain all radios in the radio group.
- Checkboxes and radio buttons created by `oj-option` in the button set should specify the `value` attribute, since the `value` attribute on the `oj-buttonset` refers to that attribute.
- The application should not do anything to interfere with the focus management. For example, it should not set the `tabindex` of the buttons.
- Enabled buttons should remain user visible. Otherwise, the arrow-key navigation to the button will cause the focus to seemingly disappear.
- The button set's `focusManagement` attribute should be set to `none` when placing the button set in an `oj-toolbar` element.
- Use the `labelled-by` attribute to establish a relationship between the button set and another element. A common use is to tie the `oj-label` and the `oj-buttonset` elements together for accessibility. The `oj-label` element has an `id`, and you use the `labelled-by` attribute to tie the two components together to facilitate correct screen reader behavior.
- The application is responsible for applying WAI-ARIA `aria-label` and `aria-controls` attributes to the button set, as appropriate.

`aria-label="Choose only one. Use left and right arrow keys to navigate."`
`aria-controls="myTextEditor"`

- If the value attribute and DOM get out of sync (for example, if a set of buttons contained in a button set changes, possibly due to a Knockout binding), then the application is responsible for updating the `value` attribute.
- The application doesn't need to listen for this event, since the `value` binding will update the bound observable whenever the value state changes.

For additional information about the `oj-buttonset-many` and `oj-buttonset-one` components' attributes, events, and methods, see the [oj-buttonset API documentation](#).

Work with Conveyor Belts

The Oracle JET `oj-conveyor-belt` custom element manages overflow for a group of sibling child elements to control the number of child elements displayed and provides horizontal or vertical scrolling to cycle through the other child elements.



Define the `oj-conveyor-belt` element in the HTML file. Specify a group of child sibling elements to be managed by the `oj-conveyor-belt`. You can add the sibling child elements as either direct children of the conveyor belt or as the nested children of a container element that is itself a direct or nested child of the conveyor belt. In this example, the `oj-conveyor-belt` is configured for horizontal scrolling with a maximum width that varies on the screen width, and the sibling child elements are defined as Oracle JET `oj-button` elements.

```
<div id="conveyorbelt-horizontal-example">
<div class="oj-flex">
    <oj-conveyor-belt class="oj-lg-6 oj-md-9 oj-sm-12">
        <oj-button id="hydrogen" class="oj-sm-margin-1x">Hydrogen</oj-button>
        <oj-button id="helium" class="oj-sm-margin-1x">Helium</oj-button>
        <oj-button id="lithium" class="oj-sm-margin-1x">Lithium</oj-button>
        <oj-button id="beryllium" class="oj-sm-margin-1x">Beryllium</oj-button>
        <oj-button id="boron" class="oj-sm-margin-1x">Boron</oj-button>
        <oj-button id="carbon" class="oj-sm-margin-1x">Carbon</oj-button>
        <oj-button id="nitrogen" class="oj-sm-margin-1x">Nitrogen</oj-button>
        <oj-button id="oxygen" class="oj-sm-margin-1x">Oxygen</oj-button>
        <oj-button id="fluorine" class="oj-sm-margin-1x">Fluorine</oj-button>
        <oj-button id="neon" class="oj-sm-margin-1x">Neon</oj-button>
        <oj-button id="sodium" class="oj-sm-margin-1x">Sodium</oj-button>
        <oj-button id="magnesium" class="oj-sm-margin-1x">Magnesium</oj-button>
    </oj-conveyor-belt>
</div>
</div>
```

 **Note:**

The `oj-conveyor-belt` component does not provide accessibility features such as keyboard navigation. It is the responsibility of the application developer to make the items in the conveyor belt accessible. For tips and additional detail, see the [oj-conveyor-belt API documentation](#).

When you configure the child elements as direct children of the `oj-conveyor-belt`, the element will ensure that they are laid out according to the specified orientation. If, however, you configure the child elements as the children of a container element, you must take additional steps to ensure the correct display. For details, see the [oj-conveyor-belt API documentation](#).

The Oracle JET Cookbook [Conveyor Belts](#) demos contain the complete code for this example. In addition, you will find examples for a vertical conveyor belt and conveyor belts with nested content, tab-based scrolling, and programmatic scrolling.

 **Note:**

The Oracle JET `oj-film-strip` component also manages a group of sibling child elements to provide horizontal or vertical scrolling to cycle through the other child elements. However, it also provides the ability to:

- lay out a set of items across discrete logical pages.
- control which and how many items are shown.
- hide items outside the current viewport from tab order and screen readers.

For additional information, see [Work with Film Strips](#).

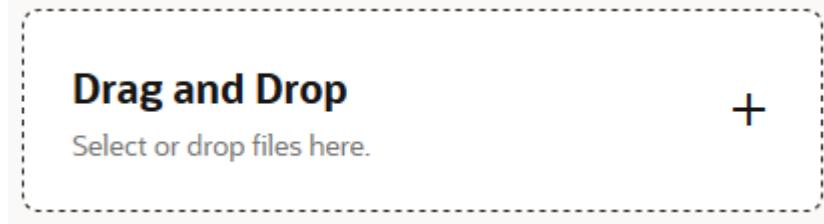
Work with File Picker

Use `oj-file-picker` to select files from the file browser or drop files from the file system to a drop zone.

By default, the Oracle JET file picker displays a clickable drop zone to select file(s) or drag and drop file(s) from the file system to the drop zone. You can customize the default drop zone text or the entire drop zone by using the named slot set to `trigger`.

The following example provides the default clickable drop zone.

```
<oj-file-picker on-oj-select='[[selectListener]]'></oj-file-picker>
```



The following example provides the clickable drop zone with customized text.

```
<oj-file-picker on-oj-select='[[selectListener]]'>
  <div tabindex='0' slot='trigger' class='oj-filepicker-dropzone'>
    <p class='oj-filepicker-text'>I'm a clickable dropzone</p>
  </div>
</oj-file-picker>
```



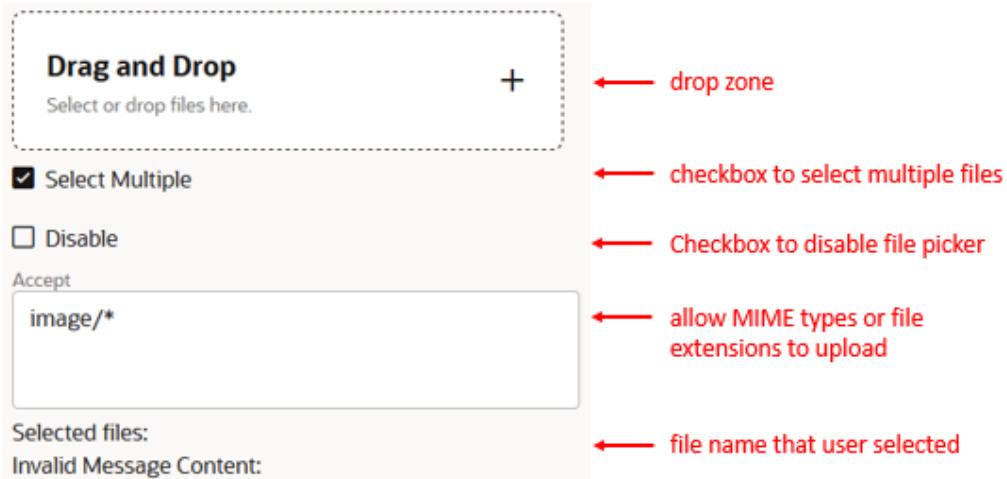
By specifying the `accept` attribute, you can allow the file picker to accept an array of strings of allowed MIME types. Default value is `undefined`. If `accept` attribute is not specified, all file types are accepted.

 **Note:**

If the `accept` attribute is specified, files with empty string type will be rejected if no match is found in the `accept` attribute value.

To allow multiple selection of files, set the `selection-mode` attribute to `multiple`. Also, to get the selected files, add an `on-select` listener.

The following image illustrates the default file picker with `selection mode` set to `multiple` file selection and `accept` attribute set to any image type.



```

<div id="parentContainer" style="padding:10px">
    <oj-file-picker accept="[[acceptArr]]" selection-mode="[[multipleStr]]" on-oj-select='[[selectListener]]'
        disabled='[[isDisabled]]' on-oj-invalid-select='[[invalidListener]]'>
        </oj-file-picker>

        <div style="padding-top:8px"></div>
        <div class="oj-choice-item">
            <oj-checkboxset id="selection" aria-label="Multiple selection"
            value="{{multiple}}>
                <oj-option id="multipleSelect" value="multiple">Select Multiple</oj-option>
            </oj-checkboxset>
        </div>

        <div style="padding-top:8px"></div>
        <div class="oj-choice-item">
            <oj-checkboxset id="disabled" aria-label="Disabled" value="{{disabled}}>
                <oj-option id="disable" value="disable">Disable</oj-option>
            </oj-checkboxset>
        </div>

        <oj-label for="acceptFld">Accept</oj-label>
        <oj-text-area id="acceptFld" rows="3" value="{{acceptStr}}></oj-text-area>

        <div style="padding-top:8px">
            <oj-label for="selected">Selected files: </oj-label>
            <span id="selected">
                <jb-bind-text value="[[fileNames().join(', ')]]"></jb-bind-text>
            </span>
        </div>
    </div>

```

In this example, the file picker uses a Knockout array to accept the selected files, identify the file name of the selected file, and to display the file name.

```

require(['knockout', 'ojs/objbootstrap', 'ojs/ojknockout', 'ojs/objfilepicker',
'ojs/ojinputtext', 'ojs/ojlabel', 'ojs/ojcheckboxset'],
function(ko, Bootstrap) {
    var self = this;

```

```
function BasicModel() {
    self.multiple = ko.observable(true);
    self.multipleStr = ko.pureComputed(function() {
        return self.multiple() ? "multiple" : "single";
    }, self);

    self.disabled = ko.observableArray();
    self.isDisabled = ko.pureComputed(function () {
        return this.disabled()[0] === 'disable' ? true : false;
    }.bind(this));

    self.acceptStr = ko.observable("image/*");
    self.acceptArr = ko.pureComputed(function() {
        var accept = self.acceptStr();
        return accept ? accept.split(",") : [];
    }, self);

    self.fileNames = ko.observableArray([]);

    self.selectListener = function(event) {
        var files = event.detail.files;
        for (var i = 0; i < files.length; i++) {
            self.fileNames.push(files[i].name);
        }
    }.bind(this);
}

Bootstrap.whenDocumentReady().then(function()
{
    ko.applyBindings(new basicModel(),
document.getElementById('parentContainer'));
});
```

The Oracle JET Cookbook [File Picker](#) demos contain the complete code for this example.

Work with Film Strips

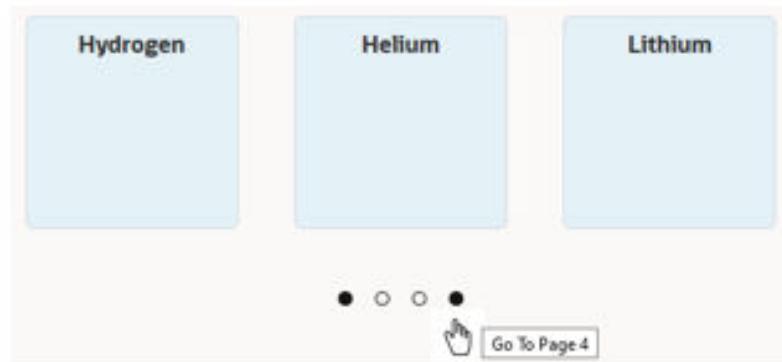
The Oracle JET `oj-film-strip` component manages a group of sibling child elements to lay out its children in a single row or column across logical pages and provides horizontal or vertical scrolling to cycle through the other child elements. You can configure the film strip to use arrows or add the `oj-paging-control` element that uses dots for scrolling through the child elements.

The following image shows two `oj-film-strip` elements configured for horizontal scrolling. The sibling child elements are panels, using the Oracle JET panel design pattern. In the top film strip, the user selects arrows for navigating through the content, and in the bottom film strip, the user selects dots for navigating through the content.

Navigation Arrows



Paging Control



Configure Film Strips

Create an `oj-film-strip` element. In the HTML file, specify a group of sibling child elements to be laid out by the `oj-film-strip`. The code sample below shows the markup for the film strip example using arrows for navigation.

```
<div id="filmStripDiv" class="oj-panel" style="margin: 20px; ">
  <oj-film-strip id="filmStrip"
    aria-label="Set of chemicals"
    arrow-placement="[[currentNavArrowPlacement]]"
    arrow-visibility="[[currentNavArrowVisibility]]">
    <oj-bind-for-each data="[[chemicals]]">
      <template data-oj-as="chemical">
        <div class="oj-panel oj-panel-alt2 demo-filmstrip-item"
          :style.display="[[getItemInitialDisplay(chemical.index)]]">
          <span><oj-bind-text value="[[chemical.data.name]]"></oj-bind-text></span>
        </div>
      </template>
    </oj-bind-for-each>
  </oj-film-strip>
</div> <!-- end filmStripDiv -->
```

You can use the `arrow-placement` attribute to control the location of the arrows. By default, it is set to `adjacent` which displays arrows outside the content, but you can set it to `overlay` to overlay the arrows on the content.

In this example, the film strip uses the `oj-bind-for-each` element to iterate through the list of chemicals defined in the application's main script, shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojfilmstrip'],
function(ko, Bootstrap)
{
    function MyModel() {
        var self = this;

        self.chemicals = [
            { name: 'Hydrogen' },
            { name: 'Helium' },
            { name: 'Lithium' },
            { name: 'Beryllium' },
            { name: 'Boron' },
            { name: 'Carbon' },
            { name: 'Nitrogen' },
            { name: 'Oxygen' },
            { name: 'Fluorine' },
            { name: 'Neon' },
            { name: 'Sodium' },
            { name: 'Magnesium' }
        ];

        self.currentNavArrowPlacement = ko.observable("adjacent");
        self.currentNavArrowVisibility = ko.observable("auto");

        getItemInitialDisplay = function(index)
        {
            return index < 3 ? '' : 'none';
        };
    };

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new ViewModel(), document.getElementById('filmstrip-
navarrows-example'));
    });
});
});
```

To set the `oj-film-strip` to loop around from first page to last page to first page, set the attribute `looping` to `page`. If `looping` is set to `off`, the navigation is bounded between the first page and last page and upon reaching the last page, the navigation cannot go back to first page again. The code sample below shows the markup for the film strip example using `looping` attribute set to `page`.

```
<div id="filmstrip-looping-example">
<div id="filmStripDiv" class="oj-panel" style="margin: 20px;">
<oj-label id="navLoopingLabel">Looping</oj-label>
<oj-film-strip id="filmStrip" aria-label="Set of chemicals" looping="page">
<oj-bind-for-each data="[[chemicals]]">
<template data-oj-as="chemical">
<div class="oj-panel oj-panel-alt2 demo-filmstrip-
item" :style.display="[[getItemInitialDisplay(chemical.index)]]">
<span><oj-bind-text value="[[chemical.data.name]]"></oj-bind-text></
span>
</div>
</template>
```

```
</oj-bind-for-each>
</oj-film-strip>
</div>
</div>
```

The `oj-film-strip` element will lay out the child items across multiple logical pages and allow for changing between logical pages. When the element is resized, the layout will adjust automatically, and the number of pages and items shown per page may change.

The Oracle JET Cookbook at [Film Strips](#) includes the complete code for the example used in this section. You can also find examples for vertical film strips, lazy loading a film strip, film strips with pagination, film strips that contain parent-child data, filmstrip to display paging information, and filmstrip with page looping.



Note:

`oj-film-strip` is a layout element, and it is the responsibility of the application developer to make the items in the film strip accessible. For tips and additional detail, see the Accessibility section in the [oj-film-strip API](#) documentation.

Work with Menus

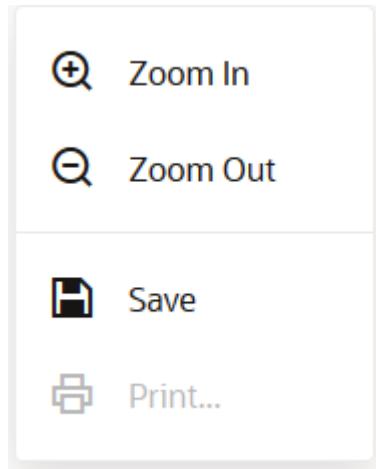
The `oj-menu` component is a themable, WAI-ARIA compliant menu with mouse and keyboard interactions for navigation. Use the `oj-menu` element on the `oj-button` element to create a menu button, or add it as a context menu to an Oracle JET element.

Topics:

- [Work with oj-menu](#)
- [Work with Menu Buttons](#)
- [Work with Context Menus](#)
- [Work with Menu Select Many](#)

Work with oj-menu

To create a menu, use the `oj-menu` element with an `oj-option` element representing each menu item. Ideally, menus should have a manageable number of items. When your menu is displayed on a small screen, such as on a mobile device, the menu will behave like a sheet menu, and its contents will slide up from the bottom of the screen.



The following code sample shows the markup used to create the basic menu. To handle menu item selection, add an action listener.

```
<div id='menubutton-container'>
    <oj-menu-button id="menuButton">
        Actions
        <!-- To handle menu item selection, use an action listener as shown, not a
        click listener. -->
        <oj-menu id="myMenu" slot="menu" style="display:none" on-obj-
action="[[menuItemAction]]">
            <oj-option id="zoomin" value="Zoom In">
                <span class="oj-menu-item-icon oj-fwk-icon oj-fwk-icon-arrow-n"
slot="startIcon"></span>Zoom In
            </oj-option>
            <oj-option id="zoomout" value="Zoom Out">
                <span class="oj-menu-item-icon demo-icon-font demo-bookmark-icon-16"
slot="startIcon"></span>Zoom Out
            </oj-option>
            <oj-option id="divider"></oj-option>
            <oj-option id="save" value="Save">
                <span class="oj-menu-item-icon demo-icon-font demo-palette-icon-24"
slot="startIcon"></span>Save
            </oj-option>
            <oj-option id="print" value="Print..." disabled="true">
                <span class="oj-menu-item-icon demo-icon-font demo-chat-icon-24"
slot="startIcon"></span>Print...
            </oj-option>
        </oj-menu>
    </oj-menu-button>
</div>
```

The following script defines the action listener.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/
ojmenu', 'ojs/ojoption'],
function(ko, Bootstrap)
{
    function MenuModel() {
        var self = this;
        self.selectedMenuItem = ko.observable("(None selected yet)");
    }
    return {
        MenuModel: MenuModel
    };
});
```

```
        self.menuItemAction = function( event ) {
            self.selectedMenuItem(event.detail.item.value);
        };
    }

Bootstrap.whenDocumentReady().then(function()
{
    ko.applyBindings(new MenuModel(), document.getElementById('menubutton-
container'));
});
});
```

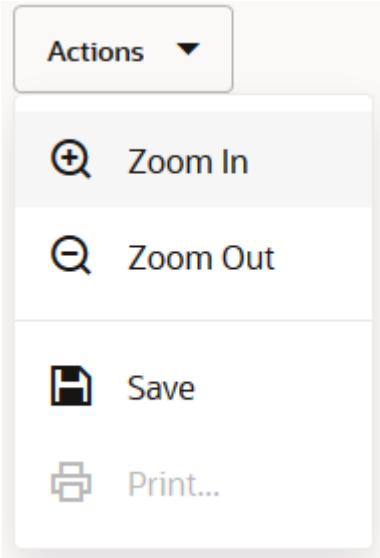
For additional information about the `oj-menu` component's attributes, events, and methods, see the [oj-menu API documentation](#).

The Oracle JET cookbook includes advanced examples for working with `oj-menu`, including demos for working with menus programmatically and using templates. For details, see [Menu \(Advanced\)](#).

Work with Menu Buttons

Menu buttons are `oj-button` components that display an `oj-menu` component when the user does one of the following:

- Clicks on the button.
- Sets focus on the button and presses the Enter, Spacebar, or Arrow Down key.



To create the menu button, add the basic `oj-menu` element as a child of an `oj-menu-button` element.

The following code sample shows the markup for the menu button, with the details for the basic menu omitted.

```
<div id='menubutton-container'>
    <oj-menu-button id="menuButton">
```

```

Actions
<!-- To handle menu item selection, use an action listener as shown, not a
click listener. -->
<ojs-menu id="myMenu" slot="menu" style="display:none" on-oj-
action="[[menuItemAction]]">
    <ojs-option id="zoomin" value="Zoom In">
        <span class="oj-fwk-icon oj-fwk-icon-arrow-n" slot="startIcon"></
span>Zoom In
    </ojs-option>
    ...contents omitted
</ojs-menu>
</ojs-menu-button>
<p>
<p>
<p>Last selected menu item:
    <span id="results"><ojs-bind-text value="[[selectedMenuItem]]"></ojs-bind-
text></span>
</p>
</div>

```

The following code sample shows the code that initializes the menu button.

```

require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/
ojmenu', 'ojs/ojoption'],
function(ko, Bootstrap)
{
    function MenuModel() {
        var self = this;
        self.selectedMenuItem = ko.observable("(None selected yet)");

        self.menuItemAction = function( event ) {
            self.selectedMenuItem(event.detail.item.value);
        };
    }

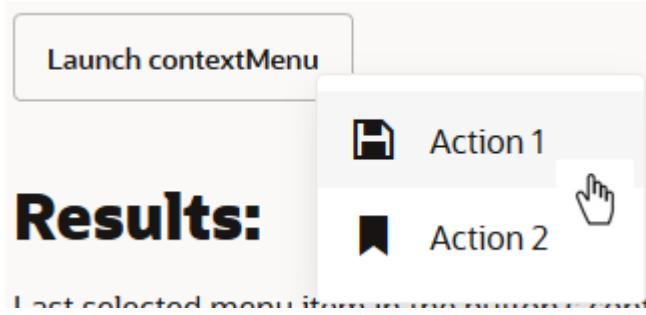
    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new MenuModel(), document.getElementById('menubutton-
container'));
    });
});

```

The Oracle JET Cookbook includes the complete code sample for the menu button shown in this section. For details, see [Menu Buttons](#).

Work with Context Menus

Context menus are `oj-menu` elements placed on the Oracle JET component using the `slot` attribute or an HTML5 element using the `contextMenu` attribute. The element on which the context menu binding is placed should be user-focusable, for keyboard accessibility through the Shift+F10 key, since focus is returned to the element when the menu is dismissed.



In this example, the `oj-menu` element is placed on an `oj-button` element. The code sample below shows the markup.

```
<div id="button-container">
    <h3>A JET component with a context menu</h3>

    The button below associates a context menu with an oj-button using the
    contextMenu slot.

    A contextMenu can be activated on the Launch contextMenu button using the
    following mouse, keyboard and touch gestures:
    <ul>
        <li>On desktop platforms, right mouse click on the button.</li>
        <li>On desktop platforms, press shift+F10 when the button has focus.</li>
        <li>For mobile or touch platforms, press and hold (long touch) on the
            button.</li>
    </ul>

    <oj-button id="myButton">
        Launch contextMenu
        <oj-menu slot="contextMenu" style="display:none" aria-label="Order Actions"
            on-oj-action="[[menuItemAction]]" >
            <oj-option id="action1" value="Action 1">
                <span class="oj-fwk-icon oj-fwk-icon-arrow-n" slot="startIcon"></
                span>Action 1
            </oj-option>
            <oj-option id="action2" value="Action 2">
                <span class="demo-icon-font demo-bookmark-icon-16" slot="startIcon"></
                span>Action 2
            </oj-option>
        </oj-menu>
    </oj-button>

    <br><br>

    <h3>Results:</h3>
    <p>Last selected menu item in the button's context menu:
        <span id="buttonResults" class="italic">
            <oj-bind-text value="[[selectedItem]]"></oj-bind-text>
        </span>
    </p>
</div>
```

The following code sample shows the code to initialize the context menu.

```
require(['knockout', 'ojs/objbootstrap', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/
ojmenu', 'ojs/ojoption'],
    function(ko, Bootstrap)
{
```

```
function MenuModel() {
    var self = this;
    self.selectedItem = {
        myButton: ko.observable("(None selected yet)") ,
        myDiv: ko.observable("(None selected yet)") 
    };

    self.menuItemAction = function( event ) {
        var launcherId = event.target.id == "myMenu" ? "myDiv" : "myButton";
        self.selectedItem[launcherId](event.detail.item.textContent);
    };
}

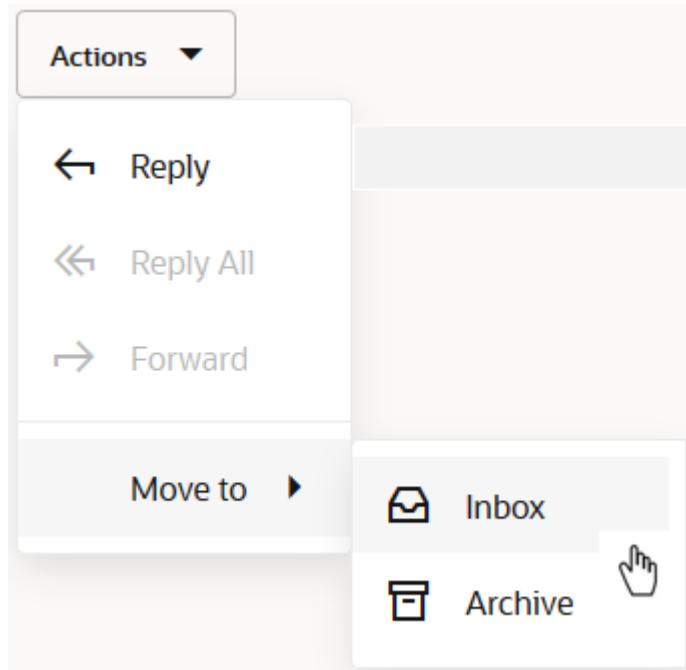
Bootstrap.whenDocumentReady().then(
    function () {
        ko.applyBindings(new MenuModel(), document.getElementById('button-
container'));
    }
);
});
```

The Oracle JET cookbook contains the complete example, including the CSS for the menu icons, at [Context Menus](#).

Work with Submenus

Use submenus when you want to organize menu items into smaller sets.

You can create submenus by nesting `oj-menu` elements under the desired `oj-option` elements. Submenus and their parent menus are always displayed as dropdown menus.



The following code sample shows the markup used to create a menu with submenus. In this example, the Move to option has a submenu with Inbox and Archive options.

```
<div id='menubutton-container'>
    <oj-menu-button id="menuButton">
        Actions
        <!-- To handle menu item selection, use an action listener as shown, not a
        click listener. -->
        <oj-menu id="myMenu" slot="menu" on-oj-action="[[menuItemAction]]">
            <oj-option id="cut" value="Reply">
                <span class="oj-ux-ico-email-reply" slot="startIcon"></span>Reply
            </oj-option>
            <oj-option id="copy" value="ReplyAll" disabled="true">
                <span class="oj-ux-ico-email-reply-all" slot="startIcon"></span>Reply All
            </oj-option>
            <oj-option id="paste" value="Forward" disabled="true">
                <span class="oj-ux-ico-email-forward" slot="startIcon"></span>Forward
            </oj-option>
            <oj-option>-----</oj-option>
            <oj-option id="zoom">
                <span>Move to</span>
                <oj-menu id="zoom_menu">
                    <oj-option id="inbox" value="Inbox">
                        <span class="oj-ux-ico-inbox" slot="startIcon"></span>Inbox
                    </oj-option>
                    <oj-option id="archive" value="Archive">
                        <span class="oj-ux-ico-archive" slot="startIcon"></span>Archive
                    </oj-option>
                </oj-menu>
            </oj-option>
        </oj-menu>
    </oj-menu-button>
</div>
```

To handle the menu selection, add an action listener. The following Java Script defines the action listener.

```
require(['knockout', 'ojs/objbootstrap', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/
ojmenu', 'ojs/ojoption'],
function(ko, Bootstrap)
{
    function MenuModel() {
        var self = this;
        self.selectedMenuItem = ko.observable("(None selected yet)");

        self.menuItemAction = function( event ) {
            self.selectedMenuItem(event.target.value);
        };
    }

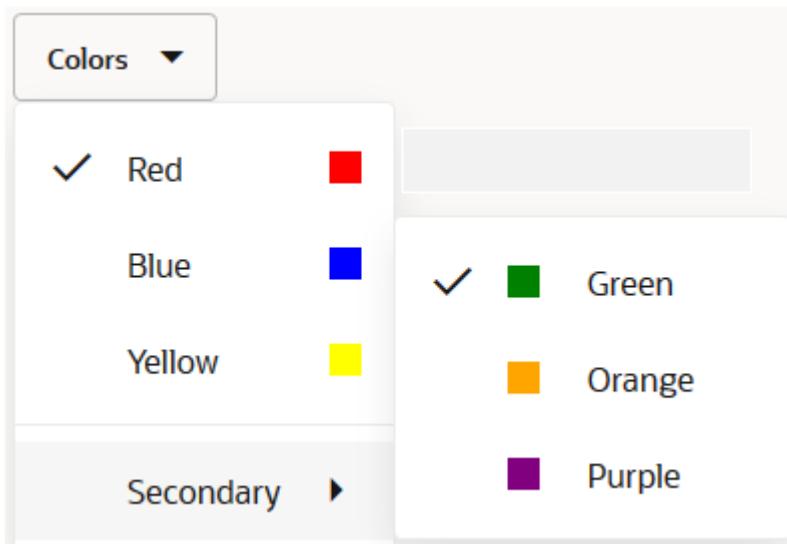
    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new MenuModel(), document.getElementById('menubutton-
container'));
    })
});
});
```

For the complete example, see [Submenus](#).

Work with Menu Select Many

You can create multiple checkbox items in the menus and submenus using `oj-menu-select-many` element.

To create a submenu that supports multiple selection of items, use the `oj-menu-select-many` element with an `oj-option` element representing each child menu item. The `oj-menu-select-many` element cannot contain submenus, but can be a child of a top-level `oj-menu` or a submenu.



The following code sample shows the markup used to create the top-level menu and submenu with checkable menu items. To handle menu selection, add an action listener.

```
<div id='menubutton-container'>
    <oj-menu-button id="menuButton">
        Colors
        <oj-menu id="myMenu" slot="menu" style="display:none" on-oj-action="[[menuItemAction]]">
            <oj-menu-select-many value="{{primaryColorsMSMValue}}">
                <oj-option value="red">
                    <span slot="endIcon" class="red icon"></span>Red
                </oj-option>
                <oj-option value="blue">
                    <span slot="endIcon" class="blue icon"></span>Blue
                </oj-option>
                <oj-option value="yellow">
                    <span slot="endIcon" class="yellow icon"></span>Yellow
                </oj-option>
            </oj-menu-select-many>
            <oj-option>---</oj-option>
            <oj-option value="secondary">
                Secondary
                <oj-menu>
                    <oj-menu-select-many value="{{secondaryColorsMSMValue}}">
                        <oj-option value="green">
                            <span slot="startIcon" class="green icon"></span>Green
                        </oj-option>
                    </oj-menu-select-many>
                </oj-menu>
            </oj-option>
        </oj-menu>
    </oj-menu-button>
</div>
```

```

        </oj-option>
        <oj-option value="orange">
            <span slot="startIcon" class="orange icon"></span>Orange
        </oj-option>
        <oj-option value="purple">
            <span slot="startIcon" class="purple icon"></span>Purple
        </oj-option>
    </oj-menu-select-many>
</oj-menu>
</oj-option>
</oj-menu>
</oj-menu-button>
<p>
<p class="bold">Last selected menu item:
<span>
    <oj-bind-text value="[[selectedMenuItem]]"></oj-bind-text>
</span>
</p>
<p class="bold">Primary color values:
<span>
    <oj-bind-text value="[[primaryColorsMSMValue]]"></oj-bind-text>
</span>
</p>
<p class="bold">Secondary color values:
<span>
    <oj-bind-text value="[[secondaryColorsMSMValue]]"></oj-bind-text>
</span>
</p>
</div>

```

The following script defines the action listener.

```

require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojbutton',
        'ojs/ojmenu', 'ojs/ojoption', 'ojs/ojmenuselectmany'],
function(ko, Bootstrap)
{
    function MenuModel() {
        this.selectedMenuItem = ko.observable("(None selected yet)");
        this.primaryColorsMSMValue = ko.observableArray(['red']);
        this.secondaryColorsMSMValue = ko.observableArray(['green']);
        this.menuItemAction = function( event ) {
            this.selectedMenuItem(event.target.value);
        }.bind(this);
    }

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new MenuModel(), document.getElementById('menubutton-
container'));
    });
});

```

For additional information about the `oj-menu` component's attributes, events, and methods, see the [oj-menu-select-many](#) API documentation.

The Oracle JET cookbook includes examples for working with `oj-menu-select-many`, including demos for working with Menu Select Many using DataProvider options and Submenus. For details, see [Menu Select Many](#).

Work with Progress Indicators

Oracle JET provides a number of components that you can use to indicate progress. Use the `oj-progress-bar` component to indicate progress against a task in a horizontal meter. Use the `oj-progress-circle` component to indicate progress against a task in a circle.

Set the value for the progress indicator in the component's `value` attribute. In the image below, the `value` attribute is set to 95 for the `oj-progress-bar` component and to 75 for the `oj-progress-circle` component. In each case, the value indicates the percentage of the task that is complete (95% or 75%).



To indicate that the value is indeterminate, set the `value` option to -1, and the progress indicator changes to reflect the indeterminate status. That is, it loops indefinitely.

To create a progress bar, use the `oj-progress-bar` element with a defined `value` attribute.

```
<div id="progressBarWrapper">
    <oj-progress-bar id="progressBar" value="{{indeterminate().length >
0 ? -1 : progressValue}}">
        </oj-progress-bar>
    . .

```

To create a progress circle, use the `oj-progress-circle` element with a defined `value` attribute.

```
<div id="progressCircleWrapper">
    <oj-progress-circle id="progressCircle" value="{{indeterminate().length >
0 ? -1 : progressValue}}">
        </oj-progress-circle>
    . .

```

The script that sets the `value` defines a Knockout observable and sets the initial value. In this example, the progress bar's initial value is set to -1, to indicate that the value is indeterminate. You can set the progress circle's initial value the same way.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojprogress-bar',
'ojs/ojinputnumber',
'ojcheckboxset', 'ojs/ojformlayout'],
function (ko, Bootstrap) {
    function ViewModel() {
        this.progressValue = ko.observable(-1);
        this.indeterminate = ko.observableArray();
    }
    Bootstrap.whenDocumentReady().then(function () {
        ko.applyBindings(new ViewModel(),
document.getElementById('progressBarWrapper'));
    });
});
```

The Oracle JET Cookbook at [Progress](#) contains example implementations that use the `oj-progress-bar` and `oj-progress-circle` components. Each implementation shows the effect of adjusting the progress indicator's value. The Oracle JET Cookbook also shows an implementation of an older progress indicator component (`oj-progress`) that you can use in applications that use the Alta theme. Use the newer components (`oj-progress-bar` and `oj-progress-circle`) in applications that use the Redwood theme.

Work with Tags

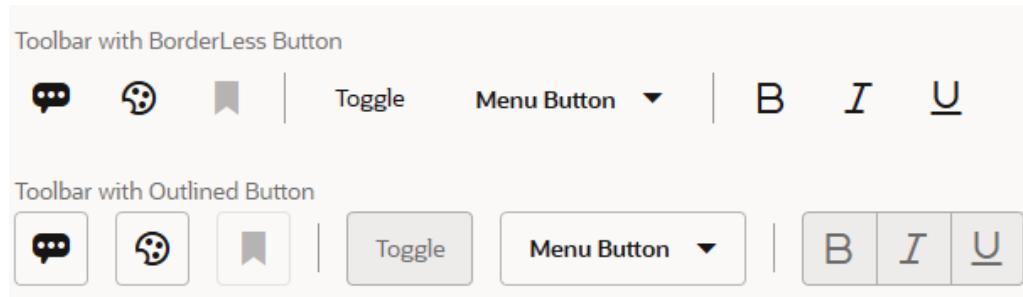
You can add HTML tags to your Oracle JET page as you would for any other HTML application. However, you should be aware that Oracle JET provides styling directly on the following tags:

- Header (`h1, h2, h3, h4`)
- Horizontal rule (`hr`)
- Link (`a`)
- List (`ul, ol, li`)
- Paragraph (`p`)

There may be use cases where you do not want to use aspects of the default theming because it causes compatibility issues. For example, you may be embedding JET components or regions in a page controlled by another technology, such as Oracle ADF. In these cases, Oracle JET provides options for theming for compatibility. For additional information, see [Work with Custom Themes](#).

Work with Toolbars

The `oj-toolbar` component is a WAI-ARIA compliant toolbar, with arrow key navigation and a single tab stop. The tab stop updates on navigation, so that tabbing back into the toolbar returns to the most recently focused button.



The `oj-toolbar` can contain buttons, menu buttons, button sets, and non-focusable content such as separator icons.

Here are some tips for working with toolbars:

- A toolbar that contains radio buttons should contain all radio buttons in the radio group.
- The application should not do anything to interfere with the focus management.
- Enabled buttons should remain user visible. Otherwise, the arrow-key navigation to the button would cause the focus to seemingly disappear.
- The button set's `focusManagement` attribute should be set to `none` when placed in a toolbar.

The Oracle JET Cookbook includes toolbar examples at [Toolbars](#). For additional information about the `oj-toolbar` component's attributes, events, and methods, see the [oj-toolbar](#) API documentation.

Work with Trains

The `oj-train` component displays a navigation visual that enables the user to move back and forth between different points. Typically, the train displays steps in a task or process.



Each step can display information about its visited state (visited, unvisited, or disabled) and a message icon of type confirmation, error, warning, or info.



To create a train, you can directly add the `oj-train` element in the HTML file. Then you can define the `selected-step` and `steps` attributes. The code sample below shows a portion of the markup for the first `oj-train` shown in this section.

```
<div id="train-container">
  <oj-train id="train"
    class="oj-train-stretch"
    style="max-width:700px;margin-left:auto;margin-right:auto;"
    on-selected-step-changed="[[updateLabelText]]"
    selected-step="{{selectedStepValue}}"
    steps="[[stepArray]]">
  </oj-train>
  ... contents omitted
</div>
```

The code sample below shows the code that applies the binding, defines the steps and captures the step selection.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojtrain', 'ojs/ojbutton'],
function(ko, Bootstrap)
{
  function TrainData() {
    var self = this;
    this.selectedStepValue = ko.observable('stp1');
    this.selectedStepLabel = ko.observable('Step One');
    this.stepArray =
      ko.observableArray(
        [{label:'Step One', id:'stp1'},
         {label:'Step Two', id:'stp2'},
         {label:'Step Three', id:'stp3'},
         {label:'Step Four', id:'stp4'},
         {label:'Step Five', id:'stp5'}]);
    this.updateLabelText = function(event) {
      var train = document.getElementById("train");
      self.selectedStepLabel(train.getStep(event.detail.value).label);
    };
  };

  var trainModel = new TrainData();

  Bootstrap.whenDocumentReady().then(function()
  {
    ko.applyBindings(trainModel, document.getElementById('train-container'));
  });
});
```

The Oracle JET Cookbook includes the complete code for this example at [Trains](#). You can also find additional examples that show a stretched train, a train with messages, and a train with button navigation.

Work with Forms

Oracle JET includes classes to create responsive form layouts and components that you can add to your form to manage labels, form validation and messaging, input, and selection. The input components also include attributes to mark an input as disabled or read-only when appropriate.

The [Oracle JET Cookbook](#) and [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) includes complete demos and examples for using forms, and you may also find the following tips and tricks helpful.

Topics:

- [Work with Checkbox and Radio Sets](#)
- [Work with Color Pickers](#)
- [Work with Comboboxes](#)
- [Work with Form Controls](#)
- [Work with Form Layouts](#)
- [Work with Input Components](#)
- [Work with Labels](#)
- [Work with Select](#)
- [Work with Sliders](#)
- [Work with Switches](#)
- [Work with Validation and User Assistance](#)

! Important:

When working with forms, use the HTML `div` element to surround any Oracle JET input components. Do not use the HTML `form` element because its postback behavior can cause unwanted page refreshes when the user submits or saves the form.

Work with Checkbox and Radio Sets

The Oracle JET `oj-checkboxset` and `oj-radioset` components enhance a group of HTML `input` elements.

Checkbox Set <code>oj-checkboxset</code>	Radioset <code>oj-radioset</code>
Colors	Colors
<input type="checkbox"/> Blue	<input type="radio"/> Blue
<input type="checkbox"/> Green	<input type="radio"/> Green
<input checked="" type="checkbox"/> Red	<input checked="" type="radio"/> Red
<input type="checkbox"/> Lime	<input type="radio"/> Lime
<input type="checkbox"/> Aqua	<input type="radio"/> Aqua

The `oj-checkboxset` and `oj-radioiset` components manage the selected value of their group and add required validation. In addition, the components manage the styles of the input elements, adding and removing the Oracle JET styles depending upon state.

To create the `oj-checkboxset` or `oj-radioiset` component, you can add an `oj-checkboxset` or `oj-radioiset` node that wraps a set of `oj-option` elements to statically define the list of options. Alternatively, to dynamically define the list of options, you can omit the `oj-option` elements and bind the component's `options` attribute to an `ArrayDataProvider`. The data provider you define in the view model provides the required value and optional label fields. The initial value of the `oj-checkboxset` or `oj-radioiset` component is defined in the element's `value` attribute.

You can also set the `oj-checkboxset` or `oj-radioiset` components to `disabled` or `readonly` in the HTML markup and specify the method that will be called when the component is set to `disabled` or `readonly`.

The following code example shows the markup, using `oj-option` elements to defines the `oj-checkboxset` shown in the image above.

```
<div id="formId">
    <oj-label id="mainlabelid">Colors</oj-label>
    <!-- You need to set the aria-labelledby attribute
        to make this accessible.
        role="group" is set for you by oj-checkboxset. -->
    <oj-checkboxset id="checkboxSetId" labelled-by="mainlabelid"
        value="{{currentColor}}>
        <oj-option id="blueopt" name="color" value="blue">Blue</oj-option>
        <oj-option id="greenopt" name="color" value="green">Green</oj-option>
        <oj-option id="redopt" name="color" value="red">Red</oj-option>
        <oj-option id="limeopt" name="color" value="lime">Lime</oj-option>
        <oj-option id="aquaopt" name="color" value="aqua">Aqua</oj-option>
    </oj-checkboxset>
</div>
```

 **Note:**

For accessibility, the `oj-checkboxset` and `oj-radioiset` components require that you set the `labelled-by` attribute on the component element. For additional information about creating accessible Oracle JET components, see [About the Accessibility Features of Oracle JET Components](#).

The code that defines the `currentColor` value is defined in the `checkboxsetModel()` function, shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojcheckboxset',
'ojs/ojlabel'],
function(ko, Bootstrap)
{
    function checkboxsetModel()
    {
        var self = this;
        self.currentColor = ko.observable(["red"]);
    }

    var vm = new checkboxsetModel();
```

```
Bootstrap.whenDocumentReady().then(function()
{
    ko.applyBindings(vm, document.getElementById('formId'));
})
});
```

The Oracle JET Cookbook contains complete examples for configuring the `oj-checkboxset` and `oj-radioset` components, including the use of an `ArrayDataProvider` to populate the component. You can also find examples that show how to disable the component or one of its input elements, how to set the component to `readonly`, display the component inline, and test validation. For details, see [Checkbox Sets](#) and [Radio Sets](#).

Work with Color Pickers

Use Oracle JET color picker components to select specific color values.

You can use Oracle JET `oj-color-palette` and `oj-color-spectrum` components to display a color palette with a predefined set of colors or to define a custom color value from a display that contains a saturation spectrum.

The `value` option of the color pickers is an object of the `Color` type. You can create an `Color` object instance from a CSS-like color string and then pass that instance.

The Oracle JET Cookbook contains the complete examples that you can use to create color pickers and define their behavior at [Color Palette](#) and [Color Spectrum](#).

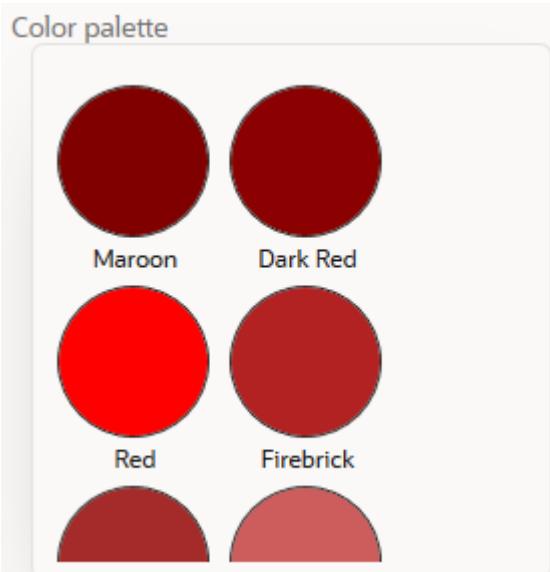
Topics:

- [Working with oj-color-palette](#)
- [Working with oj-color-spectrum](#)

Work with oj-color-palette

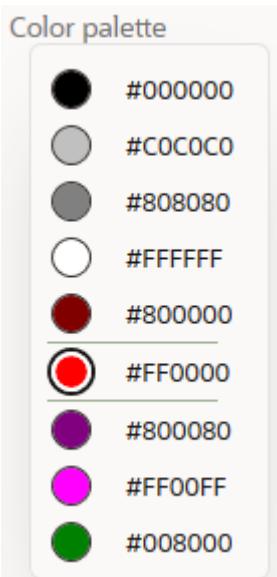
Use `oj-color-palette` to display a predefined set of colors from which a specific color can be selected. The `oj-color-palette` component supports a grid layout with or without labels and a list layout with or without labels.

The following image shows the `oj-color-palette` component using a grid layout and large color swatches with labels.



The swatch size of the `oj-color-palette` component using a grid layout can be large, small or extra small.

The following image shows the `oj-color-palette` component using a list layout with small color swatches with labels.



The swatch size of the `oj-color-palette` component using a list layout can be large or small.

To create the color palette, add the `oj-color-palette` element directly in the HTML file. The code sample below shows the markup for the color palette component that uses large color swatches with labels.

```
<div id="colorPaletteDemo">  
    ... contents omitted  
  
    <oj-label id="mainlabelid">Color palette</oj-label>  
    <div class="demo-palette-panel oj-panel oj-panel-shadow-lg">  
        <oj-color-palette class="demo-palette-picker" labelled-by="mainlabelid"
```

```

        palette="[[mypalette]]"
        swatch-size="[[swatchSize]]"
        label-display="[[labelDisplay]]"
        layout="grid"
        value="{{colorValue}}">
    </oj-color-palette>
</div>
</div>

```

The `colorValue` is defined in the view model as follows:

```

require(['knockout', 'ojs/ojbootstrap', 'ojs/ojcolor', 'ojs/ojknockout', 'ojs/
ojcolorpalette', 'ojs/ojbutton', 'ojs/ojlabel'],
function(ko, Bootstrap, Color)
{
    function Model()
    {
        var self = this;
        self.colorValue = ko.observable(Color.BISQUE);
        self.swatchSizes = ["lg", "sm", "xs"];
        self.labelDisplays = ["auto", "off"];
        self.swatchSize = ko.observable(self.swatchSizes[0]);
        self.labelXDisplay = ko.observable(self.labelXDisplay);", "off"]);

        ... remaining contents omitted
    }
})

```

To set a component to disabled, you set the `disabled` option in the markup.

```

<div id="colorPaletteDemo">

    ... contents omitted

    <oj-label id="mainlabelid">Color palette</oj-label>
    <div class="demo-palette-panel oj-panel oj-panel-shadow-lg">
        <oj-color-palette class="demo-palette-picker" labelled-by="mainlabelid"
            palette="[[mypalette]]"
            swatch-size="lg"
            label-display="auto"
            layout="grid"
            value="{{colorValue}}"
            disabled="[[paletteDisabled]]">
        </oj-color-palette>
    </div>
</div>

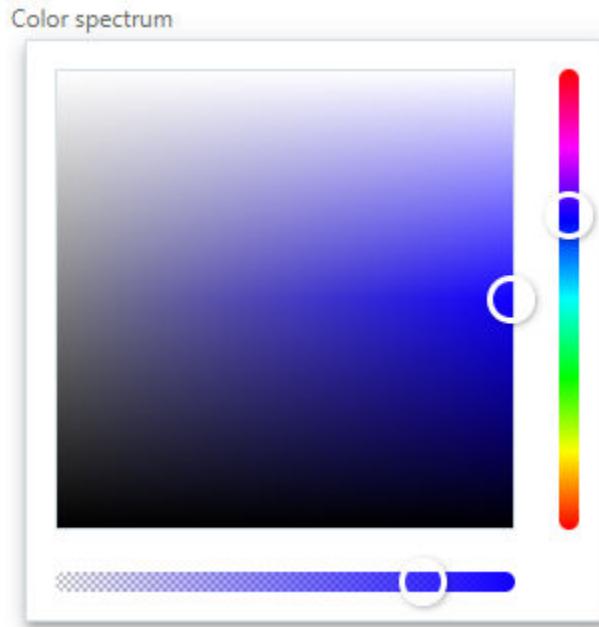
```

For additional information about adding an Oracle JET component to your page, see [Color Palette](#).

Work with `oj-color-spectrum`

Use `oj-color-spectrum` to display a saturation spectrum with hue and opacity sliders from which you can retrieve a custom color value.

The following image shows the `oj-color-spectrum` component.



To create the color spectrum, you can add the `oj-color-spectrum` element directly in the HTML file.

```
<div id="colorSpectrumDemo">  
    ... contents omitted  
  
    <oj-label id="mainlabelid">Color spectrum</oj-label>  
    <div class="demo-color-panel oj-panel oj-panel-shadow-lg">  
        <oj-color-spectrum class="demo-color-spectrum" labelled-by="mainlabelid"  
            value="{{colorValue}}"  
            on-value-changed="[[updatePreviewColor]]"  
        </oj-color-spectrum>  
    </div>  
</div>
```

You can disable the component. To set a component to disabled, you set the `disabled` option in the markup.

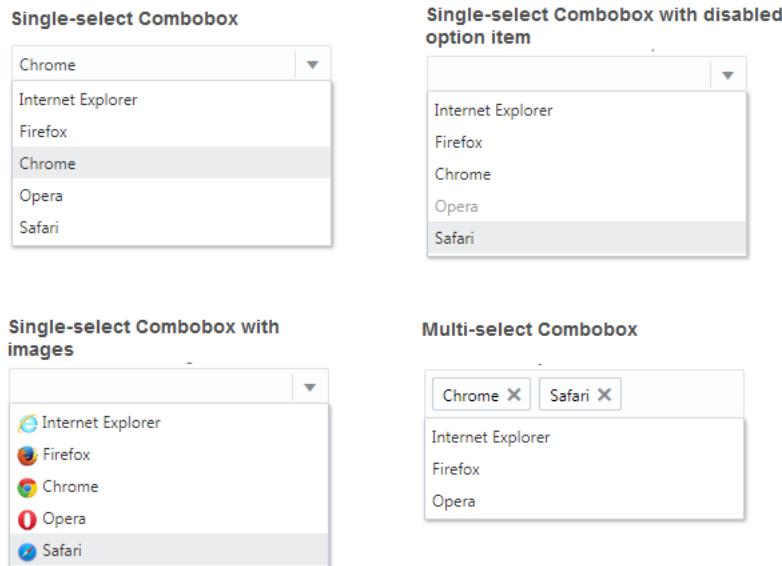
```
<div id="colorSpectrumDemo">  
    ... contents omitted  
  
    <oj-label id="mainlabelid">Color spectrum</oj-label>  
    <div class="demo-color-panel oj-panel oj-panel-shadow-lg">  
        <oj-color-spectrum class="demo-color-spectrum" labelled-by="mainlabelid"  
            value="{{colorValue}}"  
            disabled="[[spectrumDisabled]]">  
    </oj-color-spectrum>  
</div>  
</div>
```

For additional information about adding an Oracle JET component to your page, see [Color Spectrum](#).

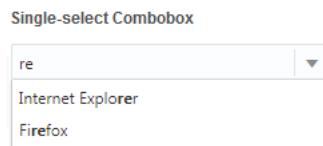
Work with Comboboxes

Use `oj-combobox-one` and `oj-combobox-many` to display read-only and editable dropdown lists.

The image below shows three single-select comboboxes and one multi-select combobox. In this example, the first combobox displays the default dropdown list. The other single-select combo boxes illustrate a disabled item option and items that include images with the item text.



The user can select one of the items from the dropdown list or erase the current value and enter text in the input field to search the list of available options. The user can also enter non-matching text to add a new item. In the example below, the user typed `re` which matches the Internet Explorer and Firefox list items.



To create the combobox, you can use the `oj-combobox-one` element or the `oj-combobox-many` element. Use the `oj-option` component to display the options in the form of a data list in the combobox. The code sample below shows the markup for the single-select combobox with all option items enabled.

```
<div id="form1">
    <oj-label for="combobox">Single-select Combobox</oj-label>
    <oj-combobox-one id="combobox" value="{{val}}"
        style="max-width:20em">
```

```

<oj-option value="Internet Explorer">Internet Explorer</oj-option>
<oj-option value="Firefox">Firefox</oj-option>
<oj-option value="Chrome">Chrome</oj-option>
<oj-option value="Opera">Opera</oj-option>
<oj-option value="Safari">Safari</oj-option>
</oj-combobox-one>
<div>
<br/>
<oj-label for="curr-value" class="oj-label">Curr value is: </oj-label>
<span id="curr-value"><oj-bind-text value="[[val]]"></oj-bind-text></span>
</div>
</div>

```

The code that defines the `val` value is defined in the `ValueModel()` constructor function, shown below.

```

require(['knockout', 'ojs/objbootstrap', 'ojs/ojknockout', 'ojs/
ojselectcombobox'],
function(ko, Bootstrap)
{
    Bootstrap.whenDocumentReady().then(
        function ()
        {
            function ValueModel() {
                this.val = ko.observableArray(["Chrome"]);
            }
            ko.applyBindings(new ValueModel(), document.getElementById('form1'));
        }
    );
});

```

You can initialize the combobox with the `options` array. Set the element's `options` attribute to a knockout `observableArray`. The array contains objects with the `value` and `label` fields in string format. Group data is also supported by specifying the `label` and `children`, which is an array of options inside the group. You can also redefine keys used in the array by specifying them in the `options-keys`. The code below defines the options binding for a multi-select combobox.

```

require(['knockout', 'ojs/objbootstrap', 'ojs/ojknockout', 'ojs/
ojselectcombobox'],
function(ko, Bootstrap)
{
    function comboboxModel () {
        this.optionsKeys1 = {label: 'regions', children: 'states', childKeys:
{value: 'state_abbr', label: 'state_name'}};
        this.optionsKeys2 = {label: 'regions', children: 'states', childKeys:
{value: 'state_abbr', label: 'state_name',
children: 'cities', childKeys: {value: 'city_abbr', label:
'city_name'}}};
        this.browsers = ko.observableArray([
            {value: 'Internet Explorer', label: 'Internet Explorer'},
            {value: 'Firefox', label: 'Firefox'},
            {value: 'Chrome', label: 'Chrome'},
            {value: 'Opera', label: 'Opera', disabled: true},
            {value: 'Safari', label: 'Safari'}
        ]);

        this.groupData = ko.observableArray([
            {label: "Alaskan/Hawaiian Time Zone",
            children: [
                {value: "AK", label: "Alaska"},


```

```

        {value: "HI", label: "Hawaii"}
    ],
    {label: "Pacific Time Zone",
     children: [
        {value: "CA", label: "California"},
        {value: "NV", label: "Nevada"},
        {value: "OR", label: "Oregon"},
        {value: "WA", label: "Washington"}
    ]
});
];

this.groupDataWithKeys = ko.observableArray([
    {regions: "Alaskan/Hawaiian Time Zone",
     states: [
        {state_abbr: "AK", state_name: "Alaska"},
        {state_abbr: "HI", state_name: "Hawaii"}
    ]},
    {regions: "Pacific Time Zone",
     states: [
        {state_abbr: "CA", state_name: "California"},
        {state_abbr: "NV", state_name: "Nevada"},
        {state_abbr: "OR", state_name: "Oregon"},
        {state_abbr: "WA", state_name: "Washington"}
    ]}
]);
;

this.triLevelGroupData = ko.observableArray([
    {regions: "Alaskan/Hawaiian Time Zone",
     states: [
        {state_abbr: "AK", state_name: "Alaska",
         cities: [
            {city_abbr: "AN", city_name: "Anchorage"}
        ]},
        {state_abbr: "HI", state_name: "Hawaii",
         cities: [
            {city_abbr: "HO", city_name: "Honolulu"},
            {city_abbr: "HL", city_name: "Hilo"}
        ]}
    ]},
    {regions: "Pacific Time Zone",
     states: [
        {state_abbr: "CA", state_name: "California",
         cities: [
            {city_abbr: "SF", city_name: "San Francisco"},
            {city_abbr: "LA", city_name: "Los Angeles"}
        ]},
        {state_abbr: "NV", state_name: "Nevada",
         cities: [
            {city_abbr: "LV", city_name: "Las Vegas"}
        ]},
        {state_abbr: "OR", state_name: "Oregon",
         cities: [
            {city_abbr: "PL", city_name: "Portland"},
            {city_abbr: "BD", city_name: "Bend"}
        ]},
        {state_abbr: "WA", state_name: "Washington",
         cities: [
            {city_abbr: "ST", city_name: "Seattle"},
            {city_abbr: "SK", city_name: "Spokane"}
        ]}
    ]}
]);
;
```

```
    ]);
}

Bootstrap.whenDocumentReady().then(
  function ()
  {
    ko.applyBindings(new comboboxModel(),
document.getElementById('form1'));
  }
);
});
```

You can also populate the combobox with an `ArrayDataProvider`. Bind the element's `options` attribute to an `ArrayDataProvider` that is created from an array containing `objects` with `value` and `label` fields in string format. The code below defines the `ArrayDataProvider` binding for the multi-select component.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojarraydataProvider', 'ojs/ojknockout', 'ojs/ojselectcombobox'],
  function(ko, Bootstrap, ArrayDataProvider)
{
  function selectModel () {
    this.selectVal = ko.observableArray(['CH', 'SA']);

    this.browsers = [
      {value: 'IE', label: 'Internet Explorer'},
      {value: 'FF', label: 'Firefox'},
      {value: 'CH', label: 'Chrome'},
      {value: 'OP', label: 'Opera'},
      {value: 'SA', label: 'Safari'}
    ];

    this.browsersDP = new ArrayDataProvider(this.browsers, {keyAttributes:
      'value'});
  }

  Bootstrap.whenDocumentReady().then(
    function ()
    {
      ko.applyBindings(new selectModel(),
document.getElementById("containerDiv"));
    }
  );
});
```

The Oracle JET Cookbook contains complete examples for configuring the single-select and multi-select comboboxes at [Comboboxes](#). You can also find examples for setting the width, handling events, adding new entries, adding read-only dropdown list, and including images with the list items.

Understand oj-combobox-one Search

The `oj-combobox-one` element can be configured to include support for search use cases.

The image below shows an example of a combobox configured for search. You can enter the new search value in the search field or click on the search field that displays a dropdown list to choose a value from.



The `oj-combobox-one` search supports the following features:

- Shows a custom search icon instead of the default dropdown arrow.
- Filters the dropdown list based on the current display value.
- Triggers events when the search value is updated by the user.

For example, when you submit a search value, the `oj-combobox-one` element triggers the `on-obj-value-updated` event that contains the search value.

The following code sample shows the markup to create a basic combobox search component shown in this section.

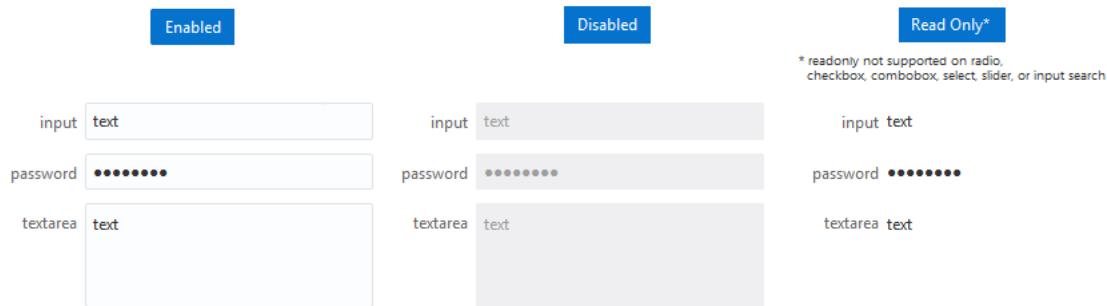
```
<oj-label for="search">Basic Search</oj-label>
<oj-combobox-one id="search"
    options="[[tagsDataProvider]]"
    filter-on-open="rawValue"
    on-obj-value-updated="[[search]]"
    placeholder="Search..."
    style="max-width:20em">
    <a slot="end" id="search-button" class="demo-search-button oj-fwk-icon-
magnifier oj-fwk-icon
oj-clickable-icon-nocontext" style="width: 32px;" role="button" aria-
label="search" on-click="[[search]]"></a>
</oj-combobox-one>
```

The Oracle JET Cookbook includes the complete code for this example at [Combobox Search](#).

For additional information about the `oj-combobox-one` component's attributes, events, and methods, see the [oj-combobox-one API documentation](#).

Work with Form Controls

Oracle JET components that support input, such as the `oj-input-text` component, provide form controls that you can use to indicate that a component is disabled or, in some cases, read only. When the component is disabled, the input fields appear grayed out, and keyboard navigation is also disabled. When the component is set to read only, the input field is not displayed, and keyboard navigation is disabled as well.



To set a component to disabled or read only, you set the `disabled` or `readonly` option in the markup and specify the method that will be called when the component is marked disabled or `readOnly`. The following code sample shows the markup for the `oj-input-text` component.

```
<div class="oj-flex">
  <div class="oj-flex-item">
    <oj-label for="inputcontrol1">input</oj-label>
  </div>
  <div class="oj-flex-item">
    <oj-input-text id="inputcontrol1" placeholder="placeholder text"
      value='{{placeholder() ? null : "text"}}'
      disabled="[[disableFormControls()]]"
      readonly="[[readonlyFormControls()]]"
      messages-custom="{{messages}}"></oj-input-text>
  </div>
</div>
```

The `disableFormControls()` and `readonlyFormControls()` methods set the Knockout observable to `false`.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojinputtext'],
  function(ko, Bootstrap)
{
  function StateModel() {
    this.disableFormControls = ko.observable(false);
    this.readonlyFormControls = ko.observable(false);
  }

  Bootstrap.whenDocumentReady().then(function()
  {
    ko.applyBindings(new StateModel(),
      document.getElementById('form-container'));
  });
});
```

 **Note:**

You can also set `disabled` as an attribute on the input element. If you use this method, then `disabled` will only be picked up at component creation. Changing the native input element's disabled state after creation will have no effect.

The Oracle JET Cookbook at [Form Controls](#) includes the complete example for the `oj-input-text` component as well as the other Oracle JET components that support `disabled` and `readOnly` options. In addition, you can find examples for using placeholder text and controlling the form's width and height.

Work with Form Layouts

Use the Oracle JET `oj-form-layout` component to create a layout that responds dynamically when used on devices with different display sizes.

Use the Oracle JET `oj-form-layout` component to create a responsive layout. Use custom elements within `oj-form-layout` component as children elements and group them to create an organized layout that can be optimized for multiple display sizes. The `oj-form-layout` supports custom elements, such as `oj-input-text` and `oj-text-area` that supports `label-hint` and `help-hints` attributes.

For each custom element, used as a child element, with `label-hint` attribute, the `oj-form-layout` generates an `oj-label` element and pairs them together in the layout as a label/value pair. When an `oj-label` element is followed by any element, the `oj-label` element will be in the label area and the following element will be in the value area. An `oj-label-value` child component allows you to place the elements in the label or value area as label and value slot children. All other elements spans the entire width of a single label/value pair.

An `oj-label-value` child component can also be used to span more than one column in a form layout. You may use the `colspan` attribute in the `oj-label-value` element to specify the number of columns to span. Note that this must be accompanied by the form layout `direction` attribute set to `row` (the default value for Redwood theme applications), or else the `colspan` attribute will be ignored. In applications with an Alta theme enabled, the default value for `direction` is `column`.

The image below shows a responsive layout using the `oj-form-layout` with various editable child components. Utilizing the `ResponsiveKnockoutUtils`, `oj-form-layout` can be made to respond to different screen sizes differently. For a display area that has sufficient space, such as a desktop monitor, the form layout can be set to two columns and have the labels inline. For a smaller display area, such as a mobile device, the `ResponsiveKnockoutUtils` can dynamically set the form layout to one column with the labels on top of their respective fields. Also, using the `oj-label-value` child component we have placed the Save and Cancel buttons.

The image displays two versions of a form layout component. The left version, labeled 'Image in a Large Screen Area', contains several input fields: a required text input ('* input 1'), a text input ('input 2'), a text area ('textarea'), a longer label ('input 3 longer label') with a corresponding text input, and a checkbox set ('Colors') with three options: Blue, Green, and Red. Below these are 'Save' and 'Cancel' buttons. The right version, labeled 'Image in a Small Screen Area', is a responsive version of the same form. It includes a required text input ('input 1 *'), a text input ('input 2'), a longer label ('input 3 longer label') with a text input, and a checkbox set ('Colors') with three options: Blue, Green, and Red. It also features 'Save' and 'Cancel' buttons.

The following code sample shows the markup for the `oj-form-layout` component with editable value child components.

```
<oj-form-layout id="ofl1" label-edge="{{labelEdge}}" max-columns="{{columns}}">
    <oj-input-text id="inputcontrol" required value="text" label-hint="input 1"></oj-input-text>
    <oj-text-area id="textareicontrol" value='text' rows="6" label-hint="textarea"></oj-text-area>
    <oj-input-text id="inputcontrol2" value="text" label-hint="input 2"></oj-input-text>
    <oj-input-text id="inputcontrol3" value="text" label-hint="input 3 longer label"></oj-input-text>
    <oj-checkboxset id="checkboxSetId" label-hint="Colors">
        <oj-option id="blueopt" value="blue">Blue</oj-option>
        <oj-option id="greenopt" value="green">Green</oj-option>
        <oj-option id="redopt" value="red">Red</oj-option>
    </oj-checkboxset>
    <oj-label-value>
        <oj-button slot="value">Save</oj-button>
        <oj-button slot="value">Cancel</oj-button>
    </oj-label-value>
</oj-form-layout>
```

In the JavaScript file, the `ResponsiveKnockoutUtils` is used to define a desired `oj-form-layout`.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojresponsiveutils', 'ojs/ojresponsiveknockoututils', 'ojs/ojlabel',
        'ojs/ojknockout', 'ojs/ojinputtext', 'ojs/ojcheckboxset', 'ojs/ojformlayout', 'ojs/ojlabelvalue', 'ojs/ojbutton'],
       function(ko, Bootstrap, ResponsiveKnockoutUtils, ResponsiveUtils)
{
    function FormLayoutModel()
    {
        this.isSmall = ResponsiveKnockoutUtils.createMediaQueryObservable(
            ResponsiveUtils.getFrameworkQuery(ResponsiveUtils.FRAMEWORK_QUERY_KEY.SM_ONLY));
        this.isLargeOrUp = ResponsiveKnockoutUtils.createMediaQueryObservable(
            ResponsiveUtils.getFrameworkQuery(ResponsiveUtils.FRAMEWORK_QUERY_KEY.LG_UP));
    }
}
```

```
ResponsiveUtils.getFrameworkQuery(ResponsiveUtils.FRAMEWORK_QUERY_KEY.LG_UP));

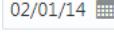
    // For small screens: 1 column and labels on top
    // For medium screens: 2 columns and labels on top
    // For large screens or bigger: 2 columns and labels inline
    this.columns = ko.pureComputed(function() {
        return this.isSmall() ? 1 : 2;
    }, this);
    this.labelEdge = ko.pureComputed(function() {
        return this.isLargeOrUp() ? "start" : "top";
    }, this);
}

Bootstrap.whenDocumentReady().then(
    function ()
    {
        ko.applyBindings(new FormLayoutModel(), document.getElementById('form-
container'));
    }
);
});
```

The Oracle JET Cookbook contains complete examples for configuring the form layout at [Form Responsive Layout](#).

Work with Input Components

The Oracle JET input components enhance browser input elements. Enhancements include support for custom validation and conversion, accessibility, internationalization, and more. The [Forms](#) page in the Oracle JET Cookbook includes examples for working with the following Oracle JET input components.

Oracle JET Component	Image	HTML5 Element
oj-input-date	 02/01/14	<oj-input-date id="date" value="{{value}}> </oj-input-date>
oj-input-date-time		<oj-input-date-time id="dateTime" value="{{value}}> </oj-input-date-time>
oj-input-number		<oj-input-number id="inputNumber-id" max="[[max]]" min="[[min]]" step="[[step]]" value="{{currentValue}}> </oj-input-number>

Oracle JET Component	Image	HTML5 Element
oj-input-password		<oj-input-password id="password" value="{{value}}> </oj-input-password>
oj-input-search		<oj-input-search id="inputSearch" value="{{value}}></oj-input-search>
oj-input-text		<oj-input-text id="inputText" value="{{value}}></oj-input-text>
oj-input-time		<oj-input-time id="time" value="{{value}}></oj-input-time>
oj-text-area		<oj-text-area id="textArea" value="{{value}}></oj-text-area>

The editable input components include converters and validators that you can customize as needed. For additional information, see [Validating and Converting Input](#).

The components also support help messages that you can customize to provide user assistance in your application. For additional information, see [Working with User Assistance](#).

Maximum length for input entered into `oj-input-text` and `oj-text-area` components is not supported as a passthrough attribute value and must be configured using the component's `length.max` attribute. The [Input Text max length](#) demo in the Oracle JET Cookbook provides an example.

Work with Labels

The `oj-label` component decorates the label text with a required icon and help icon. The user can interact with the help icon (on hover, on focus, etc) to display help description text or to navigate to an URL for more information.

To create a label, add the `oj-label` element directly in the HTML file. Use `for` on the `oj-label` element to point to the `id` of the JET Form component.

```
<oj-label for='input-text' show-required="[[isRequired]]"  
help.definition='[[helpDef]]' help.source='[[helpSource]]'>input</oj-label>  
<oj-input-text id="input-text" required="[[isRequired]]" value="text"></oj-input-  
text>
```

For accessibility, you must associate the `oj-label` component to its JET form component. For most JET form components you can do this using the `oj-label`'s `for` attribute and the JET form component's `id` attribute.

For a few JET form components (`oj-radioset`, `oj-checkboxset`, `oj-color-palette`, and `oj-color-spectrum`), you must associate the `oj-label` component to its JET form component using the `oj-label`'s `id` attribute and the JET form component's `labelled-by` attribute. For information, see the example below.

```
<oj-label id="radiosetlabel" show-required="[[isRequired]]"  
help.definition='[[helpDef]]' help.source='[[helpSource]]'>radioset</oj-label>  
<oj-radioset id="radioSetId" required="[[isRequired]]" labelled-  
by="radiosetlabel">  
    <oj-option name="color" value="red">Red</oj-option>  
    <oj-option name="color" value="blue">Blue</oj-option>  
</oj-radioset>
```

Labels are top aligned by default, following best practices for mobile devices.

The screenshot shows a JET Form with the following components:

- An `input` field labeled "input *". It contains the value "text".
- A `textarea` field labeled "textarea *". It contains the value "text".
- A `datepicker` field labeled "datepicker *". It displays the date "08/29/17" and has a calendar icon.
- A `radioset` field labeled "radioset *". It contains two radio buttons: one for "Red" and one for "Blue".

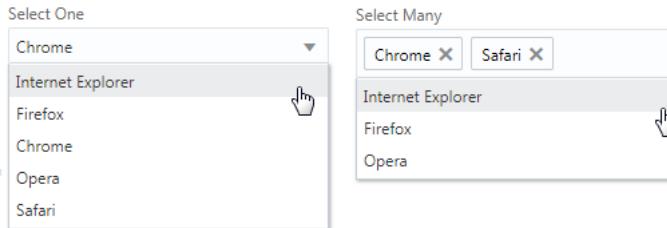
The `<oj-form-layout>` wraps the label and input components separately. So, if label component is an immediate child component of `<oj-form-layout>`, then the input component will not generate label. In this case, input component ignores label and other label related attributes.

The Oracle JET Cookbook includes additional examples for using help and required modifiers. In addition, the cookbook contains examples for making multiple labels on

one field accessible, and making multiple fields on one label accessible. For details, see [Labels](#).

Work with Select

Use the Oracle JET `oj-select-single` custom element to support single-selection dropdown lists and use the `oj-select-many` custom element to support multi-selection lists. Both select components provide builtin search filtering.



Note:

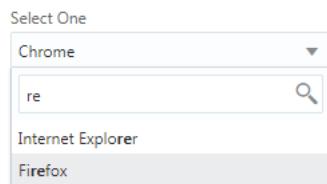
The `oj-select-single` custom element replaces the `oj-select-one` element. The new `oj-select-single` custom element has better DataProvider integration and has more powerful dropdown customization support than the old `oj-select-one` element.

The dropdown list displays when the user does one of the following:

- Clicks on the select box.
- Sets focus on the select box and starts typing or presses the Enter, Arrow Up, or Arrow Down key.

In the case of `oj-select-one` and the `oj-select-many` custom elements, if the number of options is less than the `minimumResultsForSearch` value, then by default the search box is not displayed when the dropdown is open. However, if the user starts typing when the select box is in focus, the dropdown will be open along with the search box displayed.

For example, there are five items in the dropdown list shown above so a search box is not displayed by default.



 **Tip:**

If you want to provide the user with the ability to add items to the dropdown list, use the Oracle JET `oj-combobox-one` and the `oj-combobox-many` custom elements instead to create dropdown lists that support single and multiple selection respectively. For information, see [Work with Comboboxes](#).

The code sample below shows the markup used to create the single-selection component using the `oj-select-single` custom element with an observable array for any amount of data.

```
<div id="form1">
  <oj-label for="selectSingle">Select Single</oj-label>
  <oj-select-single id="selectSingle"
    style="max-width:20em"
    data="[[browsersDP]]"
    value="{{selectVal}}">
  </oj-select-single>
  <div>
    <br/>
    <oj-label for="curr-value">Current selected value is </oj-label>
    <span id="curr-value"><oj-bind-text value="[[selectVal]]"></oj-bind-text></span>
  </div>
</div>
```

The code sample below shows the markup used to create the multi-select component using the `oj-select-many` custom element and child `oj-option` elements for small static data.

```
<div id="form1">
  <oj-label for="multiSelect">Select Many</oj-label>
  <oj-select-many id="multiSelect" value="{{selectVal}}>{{selectVal}}</oj-select-many>
  <oj-option value="IE">Internet Explorer</oj-option>
  <oj-option value="FF">Firefox</oj-option>
  <oj-option value="CH">Chrome</oj-option>
  <oj-option value="OP">Opera</oj-option>
  <oj-option value="SA">Safari</oj-option>
  </oj-select-many>
  <div>
    <br/>
    <oj-label for="curr-value">Current selected values are </oj-label>
    <span id="curr-value"><oj-bind-text value="[[selectVal]]"></oj-bind-text></span>
  </div>
</div>
```

 **Note:**

For accessibility, the select component requires that you set the `for` attribute on the label element to point to the id of the `oj-select-one`, `oj-select-many`, or `oj-select-single` element. For information about creating accessible Oracle JET components, see [About the Accessibility Features of Oracle JET Components](#).

The code samples for both custom elements define `max-width` style attribute. Use `max-width` instead of `width` to ensure that the component adjusts its width automatically when the display width changes.

The code that defines the `selectVal` value is defined in the `ValueModel()` function, shown below. `val` is defined as a Knockout observable, with its initial values set to Chrome.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojselectsingle'],
    function(ko, Bootstrap)
    {
        function ValueModel()
        {
            this.selectVal = ko.observable(['CH']);
        }

        Bootstrap.whenDocumentReady().then(
            function ()
            {
                ko.applyBindings(new ValueModel(), document.getElementById('form1'));
            }
        );
    });
});
```

You can populate an `oj-select-single` and `oj-select-many` element with an `ArrayDataProvider`. Bind the `oj-select-single` element's `data` attribute to the `ArrayDataProvider` that is created from an array containing objects with `value` and `label` fields in string format. In the case of the `oj-select-many` element, bind the `oj-options` child element to the `ArrayDataProvider`. The code below defines the `ArrayDataProvider` binding for the select-single component.



Note:

A maximum of 15 rows will be displayed in the dropdown. If you have more than 15 rows, the message, More results available, please filter further., will be displayed.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojarraydataprovider', 'ojs/ojknockout', 'ojs/ojselectsingle'],
    function(ko, Bootstrap, ArrayDataProvider)
    {
        function selectModel () {
            this.selectVal = ko.observable(['CH']);

            var browsers = [
                {value: 'IE', label: 'Internet Explorer'},
                {value: 'FF', label: 'Firefox'},
                {value: 'CH', label: 'Chrome'},
                {value: 'OP', label: 'Opera'},
                {value: 'SA', label: 'Safari'}
            ];

            this.browsersDP = new ArrayDataProvider(browsers, {keyAttributes: 'value'});
        }
    });
});
```

```
Bootstrap.whenDocumentReady().then(  
    function ()  
    {  
        ko.applyBindings(new selectModel(),  
        document.getElementById("containerDiv"));  
    }  
);  
});
```

The Oracle JET Cookbook contains complete examples for configuring `oj-select-single` and `oj-select-many` at [Select](#).

Work with Sliders

A slider component displays a horizontal or vertical bar representing a numeric value. The Oracle JET `oj-slider` component enhances the HTML `input` element to provide a slider component that is themable and WAI-ARIA compliant.

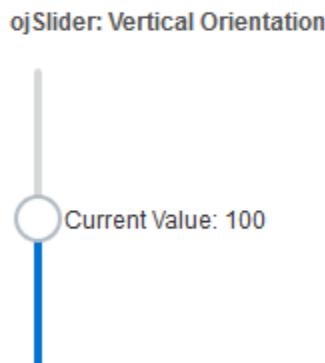
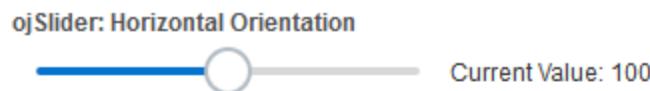
Topics

- [About the oj-slider Component](#)
- [Create Sliders](#)
- [Tips on Formatting for oj-slider](#)

About the oj-slider Component

The `oj-slider` component supports horizontal or vertical sliders with one thumb. The user can use gestures, mouse, or keyboard on the thumb to adjust the value within the slider's range.

The following image shows two `oj-slider` components. The two components illustrate sliders with horizontal and vertical orientation and their current value set to 100.



Create Sliders

To create the `oj-slider` component, create an `oj-slider` element and assign an `id` to it. Create a HTML `oj-label` element to add the `for` attribute points to the `id` of the `oj-slider` component. Use the component's `min` and `max` attributes to set the slider range and the component's `value` attribute to set the thumb's initial value.

To add the `oj-slider` component to your page:

1. Create an `oj-slider` component using the `oj-slider` element. Set values for the slider's minimum, maximum, and step values.

```
<div id="slider-container">
    <oj-label for="slider"> slider component </oj-label>
    <oj-slider id="slider" value="{{value}}" min="[[min]]" max="[[max]]"
    step="[[step]]"> </oj-slider>
</div>
```

 **Note:**

For accessibility, the `div` container includes a `label` element that associates the `oj-slider` component with the label. See the `oj-slider` API documentation for accessibility details and associated keyboard and touch end user support.

2. Add code to your application script that sets the values for the attributes you specified in the previous step. The view model for the basic slider is shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojslider'],
function(ko, Bootstrap) {
    function SliderModel() {
        var self = this;
        self.max = ko.observable(200);
        self.min = ko.observable(0);
        self.value = ko.observable(100);
        self.step = ko.observable(10);
    }
    var sliderModel = new SliderModel();

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(sliderModel, document.getElementById('slider-
    container'));
    });
});
```

The `step` attribute indicates the size of the interval the slider takes between the `min` and `max` values.

 **Note:**

The full specified value of the range (`max - min`) should be evenly divisible by `step`.

Tips on Formatting for oj-slider

The `oj-slider` component provides options that allow you to customize a horizontal slider's width, change the slider's orientation to vertical, adjust a vertical slider's height, or disable it.

You may find the following tips helpful when working with sliders.

- To change the horizontal slider's width, enter a `style` value directly on the `oj-slider` element. The example below shows how you could specify an absolute width of 25 em on the `oj-slider` used in this section.

```
<oj-label for="slider-id"> ojSlider component </oj-label>
<br>
<oj-slider id="slider-id" value="{{value}}" min="[[min]]"
max="[[max]]" step="[[step]]" style="max-width: 25em"> </oj-slider>
```

To specify a width as a percentage of the maximum width available, set the `max-width` style to a percentage.

```
style:'max-width:100%'
```

- To create a vertical slider, set the `oj-slider` component's orientation option to `vertical`.

```
<oj-label for="slider-id"> ojSlider component </oj-label>
<br>
<oj-slider id="slider-id" orientation="vertical"
value="{{value}}" min="[[min]]" max="[[max]]" step="[[step]]"
style="height: 150px"> </oj-slider>
```

- To change the vertical slider's height, set the `style` attribute directly on the `oj-slider` element.

```
<oj-label for="slider-id"> ojSlider component </oj-label>
<br>
<oj-slider id="slider-id" orientation="vertical"
value="{{value}}" min="[[min]]" max="[[max]]" step="[[step]]"
style="height: 150px"> </oj-slider>
```

- To display a slider that displays a value but does not allow interaction, set the component's `disabled` option to `true`.

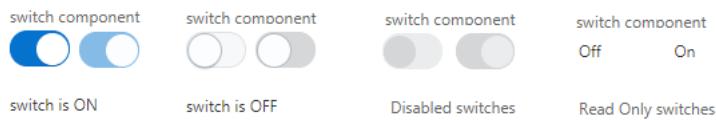
Cookbook Examples

The Oracle JET cookbook includes the complete examples shown in this section at [Sliders](#). You can also find examples that show disabled sliders and sliders with icons on the bar to manipulate the thumb.

Work with Switches

A switch component displays two mutually exclusive choices to a user, typically ON or OFF. You can also disable a switch or make it read only using component attributes and display a switch inline using built-in style classes.

The following image shows an `oj-switch` component in on, off, disabled, and read only states. The display changes when the switch has focus to provide a visual clue to the user that the switch is selectable.



To create the `oj-switch` component, add the `oj-switch` element directly in the HTML file and assign it an id. Use the component's `value` attribute to set the initial state to `true` or `false`.

The following code sample shows the markup for the `oj-switch` shown in this section. For accessibility, the form container includes a `label` element where the `for` attribute points to the id of the `oj-switch` component. See the [oj-switch API documentation](#) for accessibility details and associated keyboard and touch end user support.

```
<div id="componentDemoContent" style="width: 1px; min-width: 100%;">
    <oj-label class="oj-label" for="switch">switch component</oj-label>
    <oj-switch id="switch" value="{{isChecked}}></oj-switch><br/><br/>
    <span> switch is <oj-bind-text value="[[ isChecked() ? 'ON' : 'OFF']]></oj-
bind-text></span>
</div>
```

The `isChecked` variable specified for the `oj-switch` component's `value` attribute is defined in the application's main script, shown below. In this example, the `isChecked` variable is a boolean set to `true` by a call to the Knockout `observable()` function.

```
define(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojswitch'],
    function(ko, Bootstrap) {
        function SwitchModel() {
            var self = this;
            self.isChecked = ko.observable(false);
        }

        var switchModel = new SwitchModel();

        Bootstrap.whenDocumentReady().then(
            function()
            {
                ko.applyBindings(switchModel, document.getElementById('formId'));
            }
        );
    });
});
```

Tip:

You can configure `oj-switch` to display inline with its label.

`switch component` 

To configure an inline switch, add the `oj-label-inline` class to the switch's label.

```
<oj-label for="switch" class="oj-label-inline">switch component</oj-label>
```

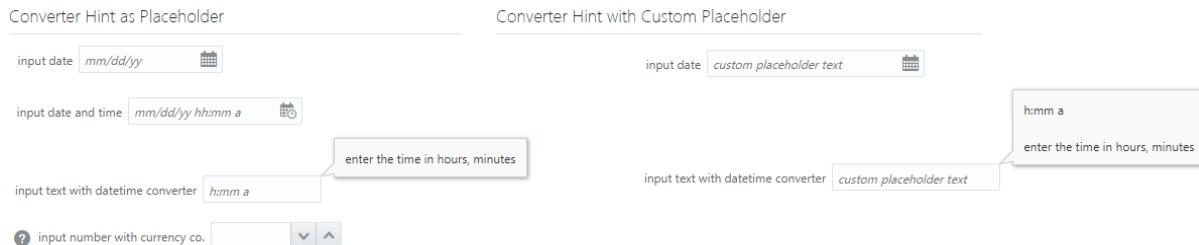
To disable the switch, set the component's `disabled` attribute to `true`. To make the switch read only, set the component's `readOnly` attribute to `true`.

The Oracle JET cookbook includes the complete example shown in this section at [Switches](#). You can also find examples that implement disabled, read only, and inline switches.

Work with Validation and User Assistance

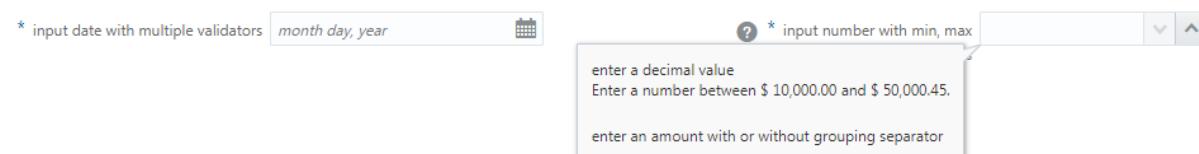
The Oracle JET editable components display help, converter and validator hints, title, and messaging content by default. If needed, you can customize the defaults for displaying content.

The image below shows examples of converter hints and title. The example on the left uses default converter hints, and the example on the right uses custom placeholder text. The title text by default appears in the note window.



You can create multiple validators on a component. The image below shows an example that uses multiple validators on an `oj-input-number` component.

Validator Hints in Notewindow



The Oracle JET Cookbook contains complete examples that you can use to customize help, converter and validator hints, and messaging content. See [User Assistance](#).

For information about validating and converting input on the Oracle JET input components, see [Validating and Converting Input](#). For information about using and customizing Oracle JET user assistance, see [Working with User Assistance](#).

Work with Layout and Navigation

Use the Oracle JET `oj-accordion`, `oj-collapsible`, `oj-dialog`, `oj-flex*`, `oj-masonry-layout`, `oj-navigation-list`, `offCanvasUtils`, `oj-panel`, `oj-popup`, `oj-size*`, and `oj-tab-bar` components and patterns to control the initial data display and allow the user to access additional content by expanding sections, selecting tabs, or displaying dialogs and popups.

The [Oracle JET Cookbook](#) and [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) include complete demos and examples for using the layout and navigation components, and you may also find the following tips and tricks helpful.

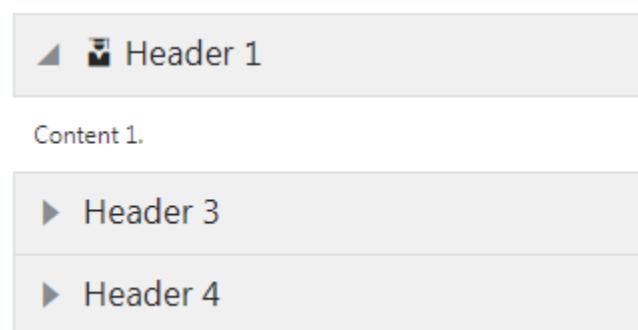
Topics:

- [Work with Accordions](#)
- [Work with Collapsibles](#)
- [Work with Dialogs](#)
- [Work with Masonry Layouts](#)
- [Work with Nav Lists](#)
- [Work with offCanvasUtils](#)
- [Work with Panels](#)
- [Work with Popups](#)
- [Work with Tab Bars](#)

For information about the flex layout (`oj-flex*`) and responsive grid (`oj-size`) classes, see [Designing Responsive Applications](#)

Work with Accordions

The Oracle JET `oj-accordion` component contains a list of `oj-collapsible` components.



You can define the JET Accordion by adding the `oj-accordion` element in the HTML file. Each child of the accordion must be an `oj-collapsible` element.

The following code shows a portion of the markup used to create the accordion shown in this section. In this example, the first collapsible (Header 1) is expanded.

```
<oj-accordion id="accordionPage">
  <oj-collapsible id="c1">
    <span><span><span class="demo-icon-font demo-education-icon-24"></span>
Header 1</span></span>
    <p class="oj-p">Content 1.</p>
  </oj-collapsible>
  <oj-collapsible id="c3">
    <span>Header 3</span>
    <p class="oj-p">Content 2.</p>
  </oj-collapsible>
  <oj-collapsible id="c4">
    <span>Header 4</span>
    <p class="oj-p">Content 3.</p>
  </oj-collapsible>
</oj-accordion>
```

The Oracle JET Cookbook at [Accordions](#) contains the sample code to create the accordion pictured in this section. You can also find examples for expanding multiple child elements and responding to events.

For additional information about working with the `oj-collapsible` component, see [Work with Collapsibles](#).

Work with Collapsibles

The Oracle JET `oj-collapsible` component contains a header and a block of content that expands or collapses when clicked.



You can define the collapsible by directly adding the `oj-collapsible` element in the HTML file. Add the header and content elements as children of the `oj-collapsible` element. You can use any valid markup to represent the header and block of content. In this example, the collapsible is using the `h3` element to contain the header and a `p` element to contain the content.

```
<oj-collapsible id="collapsiblePage">
  <h3 id="h">Header 3</h3>
  <p id="c">I'm a Collapsible.</p>
</oj-collapsible>
```

To apply the binding, you can use the Knockout `applyBindings()` method and reference the id of the element that contains the `oj-collapsible` element.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojcollapsible'],
  function(ko, Bootstrap) {
```

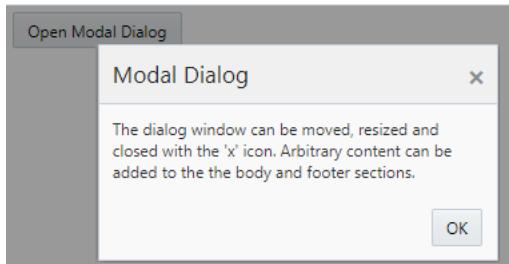
```
Bootstrap.whenDocumentReady().then(
  function ()
  {
    ko.applyBindings(null, document.getElementById('collapsiblePage'));
  }
);
});
```

The Oracle JET Cookbook [Collapsibles](#) demos contain the complete example for creating this collapsible as well as examples that show nested collapsibles, collapsibles with different header elements or borders, and event handling.

Work with Dialogs

You can use the Oracle JET `oj-dialog` component to display dialogs that are themable and follow WAI-ARIA authoring practices for accessibility.

By default, the `oj-dialog` component renders modal dialogs which require user interaction before control returns to the calling window.



Typically, you create the dialog by directly adding the `oj-dialog` element in the HTML file. Define the dialog title in the `oj-dialog` element's `title` attribute. Add content to the dialog using the `slot="body"` and `slot="footer"`.

The `oj-dialog` element includes support for the header close icon and close handler, but you must add your own markup for creating the OK button in the `slot="footer"` section, as shown in the code sample below.

```
<div id="dialogWrapper">
  <oj-dialog style="display:none" id="modalDialog1" title="Modal Dialog">
    <div slot="body">
      The dialog window can be moved, resized and closed with the 'x' icon.
      Arbitrary content can be added to the the body and footer sections.
    </div>
    <div slot="footer">
      <oj-button id="okButton" on-obj-action="[[close]]">OK
      </oj-button>
    </div>
  </oj-dialog>
  <oj-button id="buttonOpener" on-obj-action="[[open]]">
    Open Modal Dialog
  </oj-button>
</div>
```

You must also add code to handle the events on the OK and button opener buttons, in addition to the Knockout `applyBindings()` call that completes the component binding.

```
require(['knockout', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/ojdialog'],
function(ko)
{
    ko.applyBindings(null, document.getElementById('dialogWrapper'));
});
```

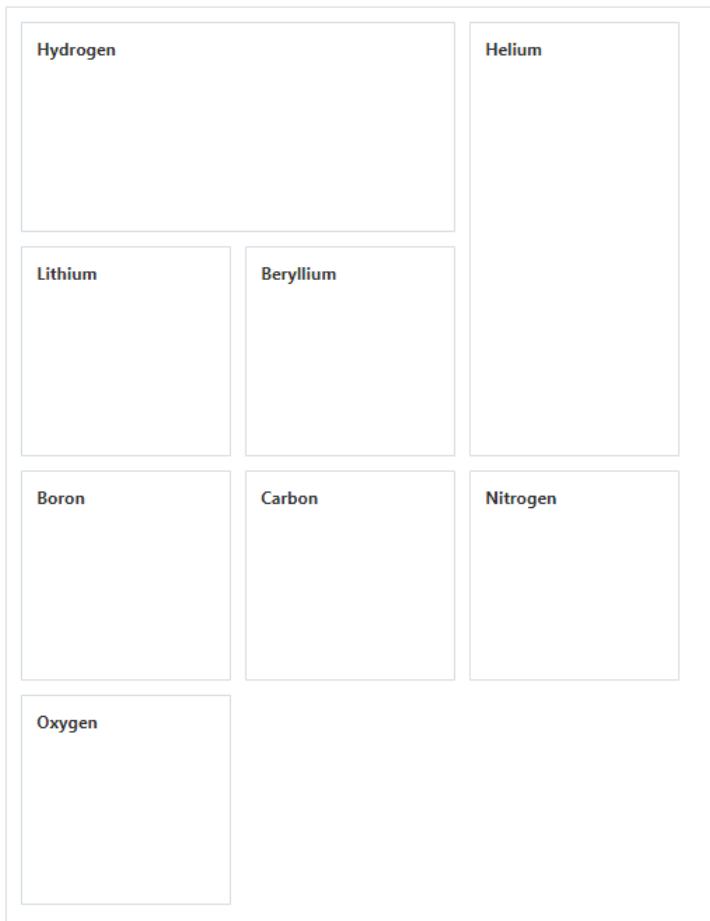
The Oracle JET Cookbook includes the complete code for this example at [Dialogs](#). You can also find samples for creating a modeless `oj-dialog` and dialogs rendered with custom headers and other display options.

Work with Masonry Layouts

Use the Oracle JET `oj-masonry-layout` component to create a responsive grid of tiles containing arbitrary content. You can specify the size of each tile by defining the number of rows and columns that the tile will span relative to the other tiles in the layout.

Masonry layouts are advanced dashboard components that are useful when tiles are of different sizes because the component will shift tiles to fill in gaps, if possible. Masonry layouts are not intended for page layout, and you typically use them when you want to compact tiles in both horizontal and vertical directions at the same time. You can also use masonry layout when you want to allow users to operate on tiles, such as resizing, inserting, deleting, and reordering of tiles. Additionally, masonry layout animates the resulting layout changes to help preserve the context of the user.

The image below shows a basic masonry layout with eight tiles. In this example, the first tile in the layout occupies two columns and one row, the second tile occupies one column and two rows, and the remaining tiles occupy one column and one row.



Topics:

- [Configure Masonry Layouts](#)
- [About the oj-masonry-layout Layout Process](#)
- [oj-masonry-layout Size Style Classes](#)

Configure Masonry Layouts

Define the masonry layout by directly adding `oj-masonry-layout` element in the HTML file. In the `oj-masonry-layout` element, add the group of sibling child elements that the `oj-masonry-layout` will manage. To specify the relative size of the tiles, add one of the `oj-masonry-layout` size style classes to each child element.

The code sample below shows the markup for the basic `oj-masonry-layout` shown in this section. The example uses the `oj-bind-for-each` element to iterate through the child elements.

```
<div id="masonrylayout-basic-example">
<div class="demo-scroll-container">
    <oj-masonry-layout id="masonryLayout">
        <oj-bind-for-each data="[[chemicals]]">
            <template>
                <div :class="[$current.data.sizeClass + ' oj-panel demo-title']">
```

```
<oj-bind-text value="[$current.data.name]"></oj-bind-text>
</div>
</template>
</oj-bind-for-each>
</oj-masonry-layout>
</div>
</div>
```

The size of each tile is determined by the value in the `sizeClass` property which is defined in the application's main script shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojmasonrylayout'],
function(ko, Bootstrap)
{
    function MyModel() {
        var self = this;

        self.chemicals = [
            { name: 'Hydrogen',
              sizeClass: 'oj-masonrylayout-tile-2x1' },
            { name: 'Helium',
              sizeClass: 'oj-masonrylayout-tile-1x2' },
            { name: 'Lithium',
              sizeClass: 'oj-masonrylayout-tile-1x1' },
            { name: 'Beryllium',
              sizeClass: 'oj-masonrylayout-tile-1x1' },
            { name: 'Boron',
              sizeClass: 'oj-masonrylayout-tile-1x1' },
            { name: 'Carbon',
              sizeClass: 'oj-masonrylayout-tile-1x1' },
            { name: 'Nitrogen',
              sizeClass: 'oj-masonrylayout-tile-1x1' },
            { name: 'Oxygen',
              sizeClass: 'oj-masonrylayout-tile-1x1' }
        ];
    }

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new MyModel(), document.getElementById('masonrylayout-basic-example'));
    });
});
```

The Oracle JET Cookbook at [Masonry Layouts](#) contains the complete example for the basic masonry layout shown in this section. You can also find demos that illustrate the different tile sizes, and masonry layouts that let you resize, reorder, and flip tiles.

Note:

The `oj-masonry-layout` component does not provide accessibility features such as keyboard navigation. It is the responsibility of the application developer to make the items in the layout accessible. For tips and additional detail, see the [oj-masonry-layout API documentation](#).

About the `oj-masonry-layout` Layout Process

`oj-masonry-layout` lays out its child tiles based on the size of the screen, the size of the tiles, and the order in which you define them.

To determine the layout, `oj-masonry-layout`:

- Processes the tiles in the order in which they originally appear in the DOM.
- Determines the number of columns to display based on the width of the `oj-masonry-layout` element and the width of a `1x1` tile.
- Determines the number of rows to display based on the number of columns and the number and sizes of tiles to lay out.
- Lays out the grid cells in a left-to-right, top-to-bottom order (or right-to-left, top-to-bottom order when the reading direction is right-to-left).

Tiles will be positioned in the first empty cell in which they fit. This can result in empty cells in the layout. Subsequent tiles may fill those earlier gaps if they fit.

If the element is resized, `oj-masonry-layout` will redo the layout, and the number of columns and rows may change.

`oj-masonry-layout` Size Style Classes

`oj-masonry-layout` supports tile sizes ranging from one to three columns and one to three rows, as shown in the following table.

Style Class	Description
<code>oj-masonrylayout-tile-1x1</code>	A tile that spans one column and one row
<code>oj-masonrylayout-tile-1x2</code>	A tile that spans one column and two rows
<code>oj-masonrylayout-tile-1x3</code>	A tile that spans one column and three rows
<code>oj-masonrylayout-tile-2x1</code>	A tile that spans two columns and one row
<code>oj-masonrylayout-tile-2x2</code>	A tile that spans two columns and two rows
<code>oj-masonrylayout-tile-2x3</code>	A tile that spans two columns and three rows
<code>oj-masonrylayout-tile-3x1</code>	A tile that spans three columns and one row
<code>oj-masonrylayout-tile-3x2</code>	A tile that spans three columns and two rows

Work with Nav Lists

The Oracle JET `oj-navigation-list` component enhances the HTML list (`ul`) element to provide a themable, WAI-ARIA compliant component that displays a list of vertical or horizontal navigation links. You can configure the nav list to slide in and out of view, expand and collapse, and respond to changes in screen size.

Topics:

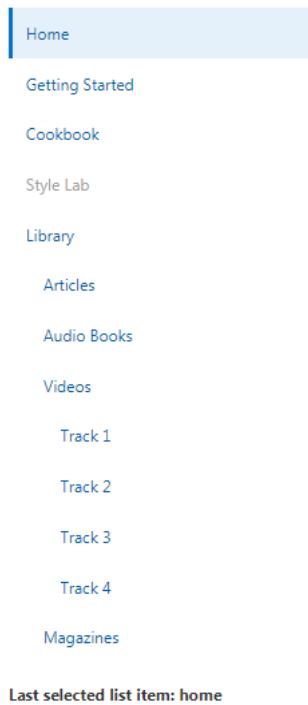
- [Understand the Data Requirements for Nav Lists](#)
- [Work with Nav Lists and Knockout Templates](#)

Understand the Data Requirements for Nav Lists

The data source for the `oj-navigation-list` component can be one of the following:

- Flat or hierarchical static HTML

The following image shows a vertical nav list that displays static, hierarchical HTML.



The following code sample shows a portion of the markup used to create the `oj-navigation-list` element with hierarchical static content and the Oracle JET component binding. The markup uses `ul` and `li` HTML elements to define the nav list, with nested lists for the hierarchical data. The markup also specifies `selectedItem` as a Knockout observable in the component's `selection` attribute that will update when the user selects a list item.

```
<div id="navlistdemo">
  <p>
    <div id="navlistcontainer" style="max-width:300px">
      <oj-navigation-list aria-label="Choose a navigation item"
selection="{{selectedItem}}>
        <ul >
          <li id="home" >
            <a href="#" >Home</a>
          </li>
          <li id="gettingStarted" >
            <a href="#" >Getting Started</a>
          </li>
          <li id="cookbook">
            <a href="#" >Cookbook</a>
          </li>
```

```

<li id="stylelab" class="oj-disabled" >
    <a href="#">Style Lab</a>
</li>
<li id="library" >
    <a href="#">Library</a>
    <ul>
        <li id="articles">
            <a href="#">Articles</a>
        </li>
        <li id="audios">
            <a href="#">Audio Books</a>
        </li>
        <li id="videos">
            <a href="#">Videos</a>
            <ul>
                <li id="track1">
                    <a href="#">Track 1</a>
                </li>
                <li id="track2">
                    <a href="#">Track 2</a>
                </li>
                <li id="track3">
                    <a href="#">Track 3</a>
                </li>
                <li id="track4">
                    <a href="#">Track 4</a>
                </li>
            </ul>
        </li>
        <li id="magazines">
            <a href="#">Magazines</a>
        </li>
    </ul>
</li>
</ul>
</oj-navigation-list>
</div>
<br>
<div>
    <p class="bold">Last selected list item:</p>
    <span><oj-bind-text value="[[selectedItem]]"></oj-bind-text></span>
</div>
</div>

```

The code to apply the binding and define `selectedItem` is shown below. In this example, the initial value of the Knockout observable is set to `home`, and the nav list will initially display with the **Home** item selected.

```

require(['knockout', 'ajs/objbootstrap','ajs/ojknockout','ajs/ojnavigationslist'],
    function(ko, Bootstrap)
    // this callback gets executed when all required modules are loaded
    {
        function ViewModel(){
            this.selectedItem = ko.observable("home");
        }
        var vm = new ViewModel();

        Bootstrap.whenDocumentReady().then(
            function ()

```

```

    {
        ko.applyBindings(vm, document.getElementById('navlistdemo'));
    }
);

```

- Oracle JET ArrayTreeDataProvider.

Typically, you use one of the Oracle JET `ArrayTreeDataProvider` class when your list data contains groups. To use a `ArrayTreeDataProvider`, specify the method that returns the tree data in the `data` attribute for the `oj-navigation-list` element.

```
<div id="navlistdemo">
    <div style="max-width:300px">
        <oj-navigation-list
            drill-mode="sliding"
            selection="{{selectedListItem}}"
            data="[[dataProvider]]"
            item.renderer="[[KnockoutTemplateUtils.getRenderer('folder_template',
true)]]">
            </oj-navigation-list>
        </div>
    </div>
```

- Oracle JET ArrayDataProvider

Use the `ArrayDataProvider` when you want to use an observable array or array as data for nav list.

The following example shows the HTML markup using `data` attribute in the `oj-navigation-list` element:

```
<oj-navigation-list
    selection="{{selectedItem}}"
    edge="top"
    data="[[dataProvider]]"
    item.renderer="[[KnockoutTemplateUtils.getRenderer('server_template',
true)]]">
</oj-navigation-list>
```

In the following JavaScript sample code, data in an array is passed to a variable, `dataProvider`, using the `ArrayDataProvider` object:

```
this.dataProvider = new ArrayDataProvider(data, {keyAttributes: 'id'});
```

For the complete example, see [Navigation List using ArrayDataProvider](#).

Note:

If you do not specify a data source in the navigation list component's `data` attribute, Oracle JET will examine the child elements inside the root element and use it for static content. If the root element has no children, Oracle JET will render an empty list.

- Oracle JET CollectionTreeDataSource

Use `CollectionTreeDataSource` when `Collection` is the model for the underlying data. The nav list will automatically react to model events from the underlying `Collection`.

To use a CollectionTreeDataSource, specify the method that returns the tree data in the data attribute for the oj-navigation-list element.

```
<div id="navlistdemo">
  <div style="max-width:300px">
    <oj-navigation-list
      selection="{{selectedItem}}"
      data="[[dataSource]]"
      item.renderer="[[KnockoutTemplateUtils.getRenderer('server_template',
true)]]">
      </oj-navigation-list>
    </div>
  </div>
```

The Oracle JET Cookbook contains a number of oj-navigation-list examples, including ones that illustrate the use of each supported data source. For details, see [Nav Lists](#).

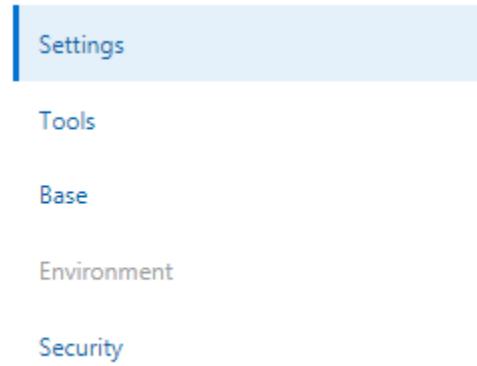
Work with Nav Lists and Knockout Templates

You can use a Knockout template to contain the markup for your list item content. Reference the name of the template in the template element's slot attribute.

The code sample below shows a portion of the markup and template for a nav list using an ArrayDataProvider object for its data. In this example, the template is named itemTemplate.

```
<div id="navlistdemo">
  <div>
    <oj-navigation-list selection="{{selectedItem}}"
      edge="top"
      data="[[dataProvider]]">
      <template slot="itemTemplate" data-oj-as="item">
        <li :id="[[item.data.id]]"
          :class="[[{'oj-disabled' : item.data.disabled === 'true'}]]">
          <a href="#"><oj-bind-text value="[[item.data.name]]"> </oj-bind-
text></a>
        </li>
      </template>
    </oj-navigation-list>
  </div>
  <br>
  <div>
    <p class="bold">Last selected list item:</p>
    <span id="results"> <oj-bind-text value="[[selectedItem]]"> </oj-bind-
text> </span>
  </div>
</div>
```

At runtime, the nav list iterates through the array and displays the content contained in name and applies styling for disabled items.



You can find the complete code sample for this nav list at [Nav List \(ArrayDataProvider\)](#).

Specify Initial Expansion in Nav Lists

You can specify the initial expansion of the list nodes when it first loads by using the `expanded` attribute.

By default, each list item's children items are hidden upon initial display. To specify that one or more items are expanded when the navigation list first loads, use the `expanded` attribute and specify the key set containing the keys of the items that should be expanded.

```
<ojs-navigation-list aria-label="Choose a navigation item"
    drill-mode="collapsible"
    selection="{{selectedItem}}"
    expanded="[[expanded]]"
    item.selectable="[[itemOnly]]">
```

In your application's `viewModel`, define the key set. The following example sets the navigation list to expand the `cookbook` item on initial display. Note that you must also add the `ojkeyset` module and `keySet` function to the require definition.

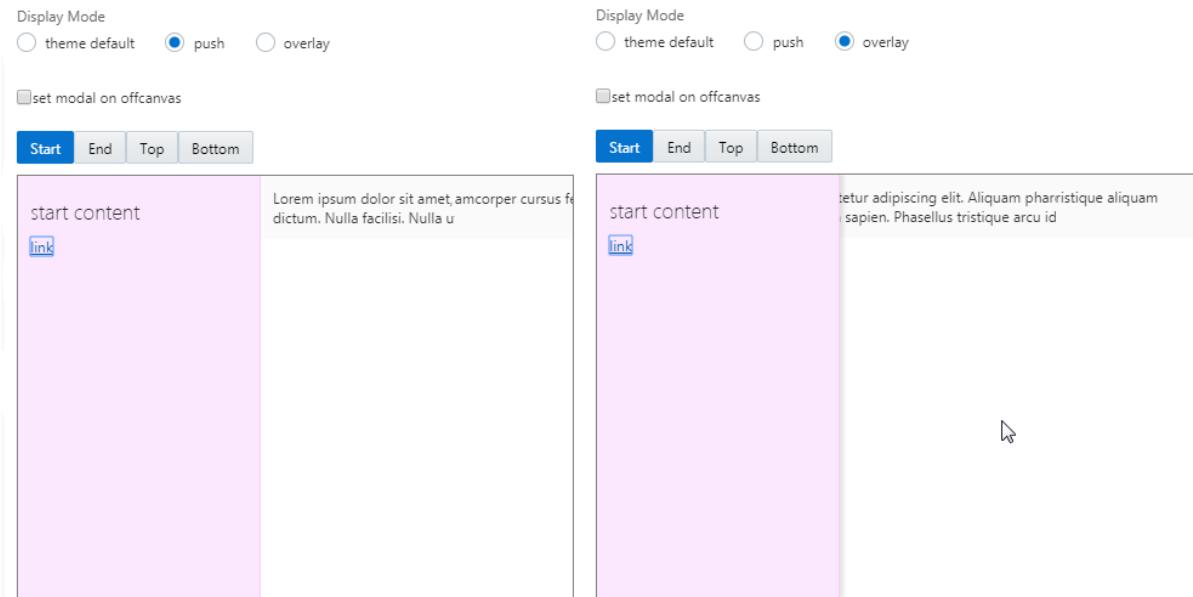
```
require('knockout', 'ojs/ojbootstrap', 'ojs/ojkeyset', 'ojs/ojknockout', 'ojs/ojnavigationlist', 'ojs/ojsswitch'),
function(ko, Bootstrap, keySet)
{
    function ViewModel(){
        this.itemOnly = function(context) {
            return context['leaf'];
        };
        this.selectedItem = ko.observable('home');
        this.isContrastBackground = ko.observable(false);
        this.expanded = new keySet.ExpandedKeySet(['cookbook']);
        ...contents omitted
    };
}
```

For the complete example, see [oj-navigation-list using expanded attribute](#).

Work with offCanvasUtils

Oracle JET offCanvasUtils are useful if you want to store content in a region that is not part of the viewport. With the offCanvasUtils, you can configure the region to slide in and out of view at the start, end, top, or bottom of the viewport, either by pushing existing content out of view or by overlaying the region.

The following image shows two examples of regions configured to slide in at the start of the viewport. In the region configured for push, offCanvasUtils pushes aside as much of the viewport content as needed to display the region. In the region configured for overlay, the content is displayed over the main content, obscuring the content.



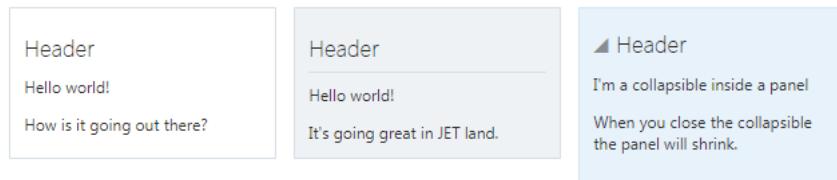
Configure an Off-Canvas Partition

To configure an off-canvas partition, create a HTML div wrapper to contain both the content and the off-canvas partition and use offCanvasUtils to show, hide, or toggle the partition view.

The Oracle JET Cookbook [Drawer Utilities](#) demos contain the sample code that implements the off-canvas partitions shown in this section and links to the [OffCanvasUtils API documentation](#). You can also find examples that implement the off-canvas partitions as responsive design patterns that change their position from off screen to fixed inside the viewport based on a media query. For more information about responsive design and media queries, see [Designing Responsive Applications](#).

Work with Panels

Panels use Oracle JET styles included with the Redwood theme to display a bounded section on the page. You can customize the size, color, and content of the panel.



In this example, three panels are displayed with three different colors. The content for the first two panels is defined using the HTML header and paragraph elements. The content for the third panel is defined as an `oj-collapse` element. The `oj-panel-*` classes are defined on the `div` elements that represent the panels.

```
<div id="panelPage">
  <div class="oj-flex">
    <div class="oj-panel oj-sm-margin-2x demo-mypanel">
      <h3>Header</h3>
      <p>Hello world!</p>
      How is it going out there?
    </div>

    <div class="oj-panel oj-panel-alt1 oj-sm-margin-2x demo-mypanel">
      <h3 class="oj-header-border">Header</h3>
      <p>Hello world!</p>
      It's going great in JET land.
    </div>

    <div class="oj-panel oj-panel-alt2 oj-sm-margin-2x demo-mypanel">
      <oj-collapse id="collapsiblePage">
        <h3>Header</h3>
        <div>
          <p>I'm a collapsible inside a panel</p>
          When you close the collapsible the panel will shrink.
        </div>
      </oj-collapse>
    </div>
  </div>
```

When panels are on the same row they will have equal height because they are in a `div` element defined with the `oj-flex` style class. You can change the default behavior by adding responsive classes to set alignment. For an example, see [Flex Layouts - Align](#).

You can customize the size or display using styles, or you can create a custom style class. In this example, the panel is customized using the `demo-mypanel` class, shown below.

```
.demo-mypanel {
  width: 200px;
}
```

The Oracle JET Cookbook contains the complete code for this example at [Panel Content](#). You can also find an example that shows the available panel colors at [Panel Colors](#).

For additional information about using and customizing the Oracle JET themes included with Oracle JET, see [Using CSS and Themes in Applications](#). For more information about the `oj-collapse` component, see [Work with Collapsibles](#).

Work with Popups

The Oracle JET popup framework includes the `oj-popup` element that you can use to create popups on your page. To manage stacking order of `oj-popup` and all Oracle JET elements that use popups internally, such as `oj-dialog`, `oj-messages`, and `oj-menu`, the popup framework uses the CSS `z-index` property and re-parents the popup DOM into a `zorder` container.

Topics:

- [Work with oj-popup](#)
- [Work with the Oracle JET Popup Framework](#)

Work with oj-popup

You can use the Oracle JET `oj-popup` component to create themable popups that follow [WAI-ARIA authoring practices](#) for accessibility. The image below shows a popup displayed with default options, but you can customize the popup's modality, position, tail, and chrome.



You can create the popup by directly adding the `oj-popup` element in the HTML file. Add the popup's content as the child of that element. For the image shown above, the popup is defined on the `oj-popup` element, with the content defined in the `span` element.

```
<oj-popup style="display:none" id="popup1">
  <span class="blink-rainbow">Hello World!!!</span>
</oj-popup>
```

In this example, the popup displays when the user clicks **Go**. The **Go** button is defined as an `oj-button` element that references the popup's `id` attribute to display the popup when clicked.

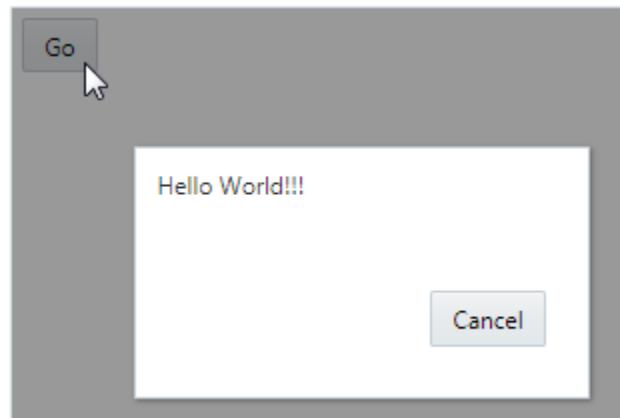
```
<div id="popupWrapper">
  <oj-popup style="display:none" id="popup1">
    <span class="blink-rainbow">Hello World!!!</span>
  </oj-popup>
  <oj-button id="btnGo"
    on-oj-action="[[function(data)
    {
      document.querySelector('#popup1').open('#btnGo');
    }]]">
    Go
  </oj-button>
</div>
```

In this example, the popup's modality defaults to modeless which means that the popup will not block user input on the page behind the popup. You can change this

to modal by setting the element's `modality` attribute as described in the [oj-popup API documentation](#).

```
<oj-popup class="demo-popup" id="popup1"
    tail="none" position.my.horizontal="center"
    position.my.vertical="bottom"
    position.at.horizontal="center" position.at.vertical="bottom"
    position.of="window" position.offset.y="-10"
    modality="modal"
    on-oj-animate-start='[[listener]]'>
```

When the user clicks the **Go** button now, the popup blocks user input of the page behind the popup with a blocking overlay pane.



The popup's default modality is defined in the `$popupModalityOptionDefault` SCSS variable and is specific to the chosen theme. For the Oracle JET theme, the SCSS variable is set to `null` which defaults to `modeless`, and the code sample below shows the generated CSS:

```
.oj-popup-option-defaults {
    font-family: "{}"; }
```

The Oracle JET Cookbook contains the complete example for this popup at [Popups](#). You can also find examples that customize the popup and that use the popup as a tooltip for accessibility.

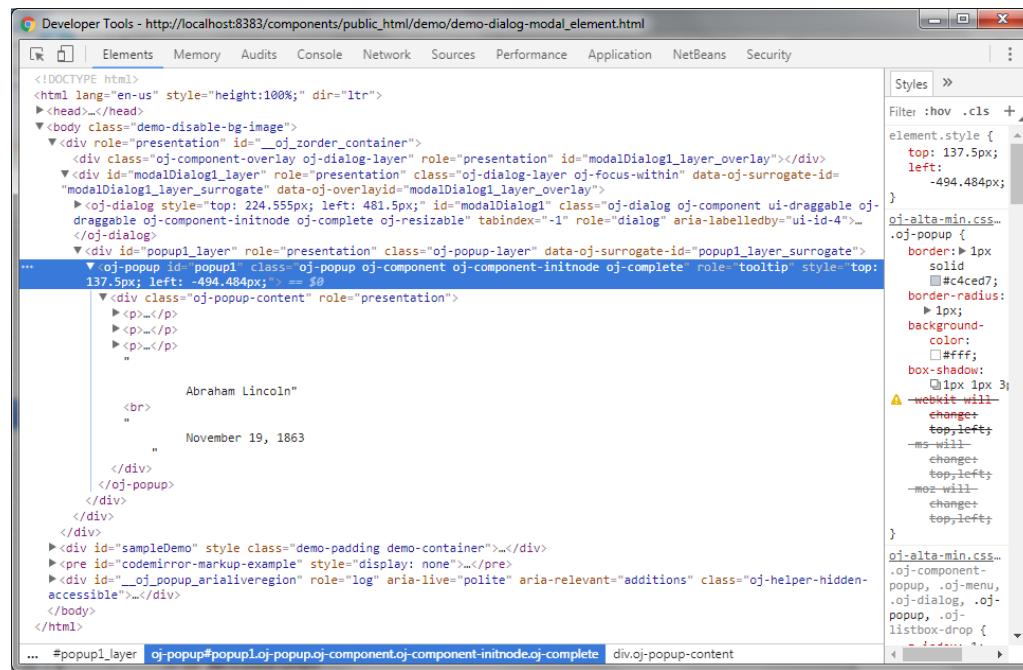
Note:

The `oj-popup` component defines default roles for accessibility and keyboard navigation. You can find details about the `oj-popup` component's accessibility support in the [oj-popup API documentation](#). In addition, you can find some general recommendations about best practices for making the gestures that launch the popup accessible.

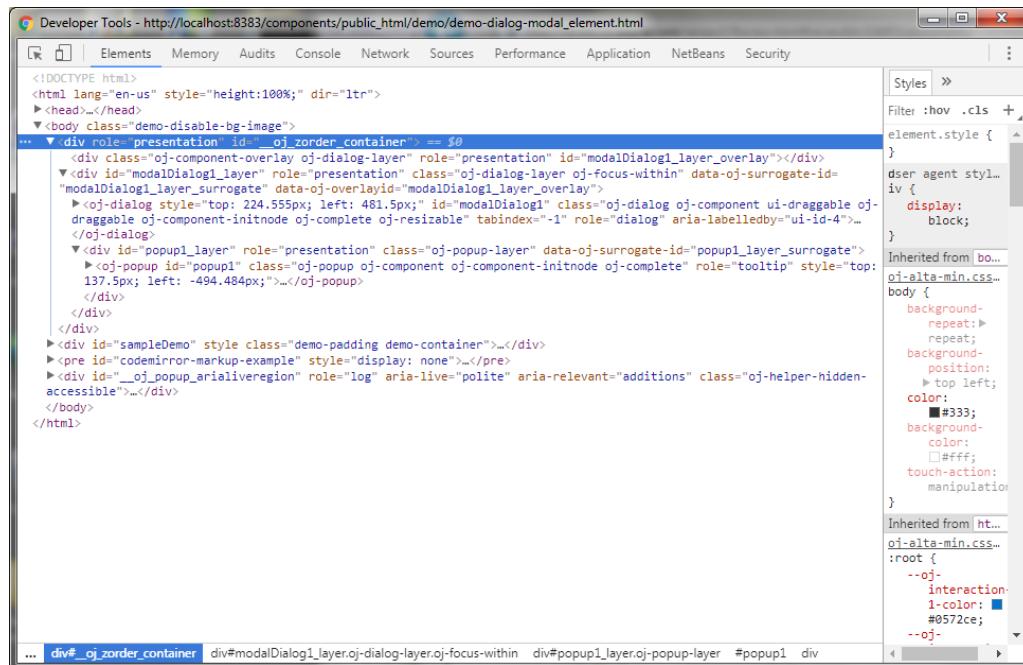
Work with the Oracle JET Popup Framework

The popup framework uses the CSS `z-index` property and re-parents the popup DOM to manage the stacking order of `oj-popup` and all Oracle JET components that use popups internally, such as `oj-dialog`, `oj-messages`, and `oj-menu`. `z-index` defaults should be sufficient for most purposes, but you can modify them using SCSS variables to generate new CSS or modify the CSS styles directly.

When a popup of any type opens, Oracle JET re-parents it into a `zorder` container in the document body and re-parents it back to its original location when closed. For example, if you add the `oj-popup` element shown in [Work with oj-popup](#) to the main content in an Oracle JET Web Nav Drawer Starter Template, the browser DOM reflects its initial placement within the document body.



When the user clicks the **Go** button to display the popup, Oracle JET re-parents it into the `oj_zorder_container` as a direct child of the `body` element.



The purpose of re-parenting is to better manage the stacking context as it relates to how popups are used versus where they are defined in the document. The `zorder` container has a default root container that holds popups open. When the popup is re-parented, it is wrapped in a `div` containing the `oj-popup-layer` style and assigned a z-index weight. If there are multiple open popups on the page, each popup is re-parented to the `zorder` container, and the active popup will be the popup with the highest z-index weight.

For example, at initial display, the `oj-popup`'s layer is marked with the `oj-popup-layer` style which has a z-index value of 1000. Popups of the same type are assigned the same z-index values. If there are multiple popups open, the popup that has active focus will be assigned a greater z-index value of 1001, and the `oj-popup-layer.oj-focus-within` style is applied.

In most cases, you should never need to modify the CSS z-index defaults since the resulting behavior may be unpredictable. If needed, however, you can update the SCSS variables used to generate the application's CSS or modify the CSS styles directly. The following table shows the default CSS style selectors, SCSS variables, and z-index values.

CSS Selector	SCSS Variable	Z-Index Value
<code>oj-popup-layer</code>	<code>\$popupZindex</code>	1000
<code>oj-popup-layer.oj-focus-within</code>	<code>\$popupZindex + 1</code>	1001
<code>oj-popup-layer.oj-popup-tail-simple</code>	<code>\$noteWindowZindex</code>	1030
<code>oj-popup-layer.oj-popup-tail-simple.oj-focus-within</code>	<code>\$noteWindowZindex + 1</code>	1031
<code>oj-listbox-drop-layer</code>	<code>\$popupZindex</code>	1000
<code>oj-menu-layer</code>	<code>\$popupZindex</code>	1000
<code>oj-dialog-layer</code>	<code>\$dialogZindex</code>	1050

CSS Selector	SCSS Variable	Z-Index Value
oj-dialog-layer.oj-focus-within	\$dialogZindex + 1	1051
oj-messages-layer	\$messagesZindex	2000
oj-messages-layer.oj-focus-within	\$messagesZindex + 1	2001
oj-component-overlay	Uses z-index value of associated oj- <i>popup_type</i> -layer. For example, the z-index of oj-dialog-layer is 1050, and oj-component-overlay will use 1050 when the associated popup is a dialog.	Varies according to oj- <i>popup_type</i> - layer.

When popup elements are initially defined on the page, they are defined with a default z-index value of 1 using the CSS selectors shown in the following table. The root popup is absolutely positioned on the page in relation to its parent container. Setting the value to 1 ensures that the root popup's children will display above the popup.

Oracle JET Components	CSS Selector	SCSS Variable	Z-Index Value
oj-menu	oj-menu	\$defaultZindex	1
oj-messages	oj-messages	\$defaultZindex	1
oj-popup	oj-popup	\$defaultZindex	1
oj-dialog	oj-dialog	\$defaultZindex	1
oj-combobox, oj-select	oj-listbox-drop	\$defaultZindex	1
oj-input-date-time, oj-input-date, oj-input-time	oj-popup	\$defaultZindex	1
Editable elements using note windows for messaging	oj-popup.oj-popup-tail-simple	\$defaultZindex	1

For additional information about Oracle JET's popup strategy, see the [oj-popup API documentation](#).

For additional information about Oracle JET's use of CSS and SCSS, see [Using CSS and Themes in Applications](#). For more information about the CSS z-index property, see <http://www.w3.org/wiki/CSS/Properties/z-index>.

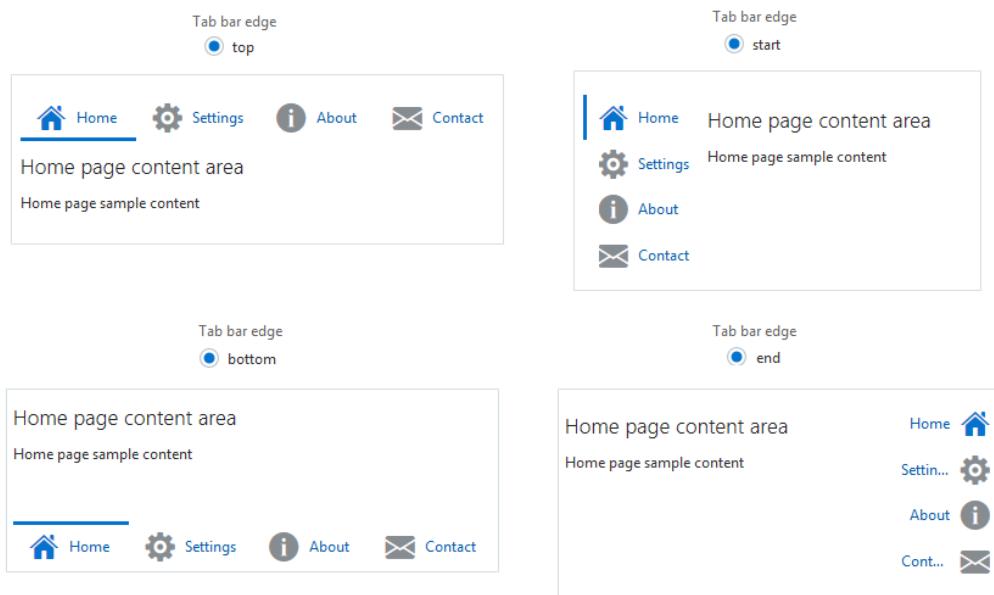
Work with Tab Bars

Use the Oracle JET oj-tab-bar element to create themable, sortable, and removable tabs that toggle between related content sections.

Tip:

Tab bars are designed for toggling between content only. If you want to perform operations on the content within a tab, use `oj-toolbar` instead. See [Work with Toolbars](#).

The Oracle JET tab bar enhances a HTML list element into a WAI-ARIA compliant, mobile friendly component with advanced interactive features. The tab bar is customizable, and you can configure the tabs to show icons, text, or both. The tab bar's position is also customizable, and you can display the tab bar on the top, bottom, or side (start or end) of the tab's content.



To create a tab bar, add the `oj-tab-bar` element to a HTML container element, typically a `div` element, on your page. Add the HTML unordered list element (`ul`) as a child to the tab bar, and add `li` child elements to the list for each tab. Each list item must include an anchor which contains the list item's text and optional icon.

The code sample below shows the markup for an `oj-tab-bar` that contains the four tabs shown in the image. The `edge` attribute specifies the position of the `oj-tab-bar`. In this example, it's set to `top`, and you can also set it to `bottom`, `start`, or `end`.

```
<div id="demo-container" class="oj-flex demo-edge-top">
  <oj-tab-bar id="hnavigationlist"
    edge="top"
    selection="{{selectedItem}}">
    <ul>
      <li id="home">
        <a href="#" aria-controls="home-tab-panel" id="home-tab">
          <span class="oj-tabbar-item-icon demo-home-icon-24 demo-icon-font-24">
            </span>
```

```

        Home
    </a>
</li>
<li id="settings">
    <a href="#" aria-controls="settings-tab-panel" id="settings-
tab">
        <span class="oj-tabbar-item-icon demo-gear-icon-16 demo-icon-
font-24">
            </span>
        Settings
    </a>
</li>
<li id="about">
    <a href="#" aria-controls="about-tab-panel" id="about-tab">
        <span class="oj-tabbar-item-icon demo-info-icon-24 demo-icon-
font-24">
            </span>
        About
    </a>
</li>
<li id="contact">
    <a href="#" aria-controls="contact-tab-panel" id="contact-tab">
        <span class="oj-tabbar-item-icon demo-email-icon-24 demo-icon-
font-24">
            </span>
        Contact
    </a>
</li>
</ul>
</oj-tab-bar>
</div>

```

Tip:

You can also use the tab bar's data attribute to populate the tab bar with data from a table data source or array data provider. See the [Tab Bars](#) cookbook demos for examples.

Add Content in Tab Bars

You can add the `oj-switcher` element that can dynamically decide which child element should be made visible.

To provide content to the tabs, add the `oj-switcher` element to the tab bar's container. `oj-switcher` dynamically decides which child element to display based on the value in its `value` attribute, which must match the value in the tab bar's `selection` attribute. The switcher's `slot` attribute also determines which tab's content to display and must match an `id` in the tab bar's list.

```
<div id="demo-container" class="oj-flex demo-edge-top">
    <oj-tab-bar id="hnnavlist"
        edge="top">
```

```
selection="{{selectedItem}}>
<ul>
  <li id="home">
    <a href="#" aria-controls="home-tab-panel" id="home-tab">
      <span class="oj-tabbar-item-icon demo-home-icon-24 demo-icon-
font-24">
        </span>
      Home
    </a>
  </li>
  ... contents omitted
</oj-tab-bar>
<oj-switcher value="[[selectedItem]]">
  <div slot="home"
    id="home-tab-panel"
    role="tabpanel"
    aria-labelledby="home-tab">
    <div class="demo-tab-content">
      <h2>Home page content area</h2>
      <p>Home page sample content</p>
    </div>
  </div>
  <div slot="settings"
    id="settings-tab-panel"
    role="tabpanel"
    aria-labelledby="settings-tab">
    <div class="demo-tab-content">
      <h2>Settings content area</h2>
      <p>Settings sample content</p>
    </div>
  </div>
  <div slot="about"
    id="about-tab-panel"
    role="tabpanel"
    aria-labelledby="about-tab">
    <div class="demo-tab-content">
      <h2>About content area</h2>
      <p>About sample content</p>
    </div>
  </div>
  <div slot="contact"
    id="contact-tab-panel"
    role="tabpanel"
    aria-labelledby="contact-tab">
    <div class="demo-tab-content">
      <h2>Contact page content area</h2>
      <p>Contact sample content</p>
    </div>
  </div>
</oj-switcher>
</div>
```

In this example, `selectedItem` is an observable, which is initially set to `home` in the application's view model.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojnavigationlist', 'ojs/ojswitcher'
],
function(ko, Bootstrap) // this callback gets executed when all
required modules are loaded
{
    function ViewModel(){
        this.selectedItem = ko.observable("home");
    }
    var vm = new ViewModel();

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(vm, document.getElementById('tabbardemo'));
    });
});
});
```

For a complete example that illustrates the use of `oj-switcher` with `oj-tab-bar`, see [Switcher](#).

Add Interactivity in Tab Bars

You can add interactivity feature in tab bars by using event handlers.

You can add the ability to add, remove, or reorder a tab within the tab bar. To enable this functionality, first create the tab bar using a supported data source.

- Removing a tab

Add the `oj-removable` class to each removable tab. Add the `on-oj-remove` attribute to the `oj-tab-bar` to specify an event handler to remove the tab from the data source.

! Important:

For accessibility, add the `oj-menu` child element to the tab bar as a context menu slot with an option marked as `data-oj-command="oj-tabbar-remove"`.

```
<oj-tab-bar contextmenu="tabmenu" id="hnavlist"
    selection="{{selectedItems}}"
    current-item="{{currentItem}}"
    edge="top"
    data="[[dataProvider]]"

    item.renderer="[[KnockoutTemplateUtils.getRenderer('server_template', true)]]"
    on-oj-remove="[[onRemove]]">
    <oj-menu slot="contextMenu"
        style="display:none"
        aria-label="Actions">
        <oj-option data-oj-command="oj-tabbar-
remove">
            Removable
        </oj-option>
    </oj-menu>
</oj-tab-bar>
```

- Adding a tab

To add a tab, add code to your view model to push the new tab to the data source. Provide the user with a dialog or other prompt to add the tab and reference the event handler that will add the tab to the data source.

```
<oj-button id="addTab" on-oj-action="[[openDialog]]">Add Tab</oj-
button>
```

- Reordering tabs

Add the `reorderable` attribute to the `oj-tab-bar` definition and set it to `enabled`. You must also add the `on-oj-reorder` attribute to specify an event handler to move the tab location in the data source.

! Important:

For accessibility, define a context menu on the page that includes entries for cutting and pasting items by specifying the `data-oj-command` attribute on the menu item with one of the values: `oj-tabbar-cut`, `oj-tabbar-paste-before`, and `oj-tabbar-paste-after`.

```
<oj-tab-bar id="tabbar"
            aria-label="Tabs using json data"
            selection="{{selectedItems}}"
            data="[[dataProvider]]"
            item.renderer="[[renderer]]"
            reorderable="enabled"
            edge="[[edge]]"
            on-oj-reorder="[[handleReorder]]">
  <oj-menu id="itemMenu" slot="contextMenu"
  style="display:none" aria-label="Item Reorder">
    <oj-option id="cutItem" data-oj-command="oj-tabbar-cut"></oj-option>
    <oj-option id="pasteBeforeItem" data-oj-command="oj-tabbar-paste-before"></oj-option>
    <oj-option id="pasteAfterItem" data-oj-command="oj-tabbar-paste-after"></oj-option>
  </oj-menu>
</oj-tab-bar>
```

For the code that implements the event handlers for adding, removing, and reordering tabs, see the [Tab Bar - Add and Remove](#) and [Tab Bar - Reorder](#) demos in the Oracle JET Cookbook.

Understand the Data Requirements for Tab Bars

The data source for the `oj-tab-bar` component can be one of the following:

- Flat or hierarchical static HTML

The following code sample shows a portion of the markup used to create the tab bar with hierarchical static content using the `oj-tab-bar` element.

```
<oj-tab-bar>
  <ul>
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
    <li><a href="#">Item 3</a></li>
  </ul>
</oj-tab-bar>
```

For information about static content in tab bar, see .

[oj-tab-bar](#)

- Oracle JET ArrayDataProvider

Use the `ArrayDataProvider` when you want to use an observable array or array as data for tab bar.

The following example shows the HTML markup using data attribute in the `oj-tab-bar` element:

```
<oj-tab-bar
    selection="{{selectedItem}}"
    data="[[dataProvider]]"
    item.renderer="[[KnockoutTemplateUtils.getRenderer('server_template',
true)]]">
</oj-tab-bar>
```

In the following JavaScript sample code, data in an array is passed to a variable, `dataProvider`, using the `ArrayDataProvider` object:

```
this.dataProvider = new ArrayDataProvider(data, {keyAttributes: 'id'});
```

For the complete example, see [Tab Bar using ArrayDataProvider](#).

 **Note:**

If you do not specify a data source in the tab bar component's `data` attribute, Oracle JET will examine the child elements inside the root element and use it for static content. If the root element has no children, Oracle JET will render an empty tab bar.

Work with Visualizations

The Oracle JET visualization components include charts, gauges, and other components that you can use and customize to present flat or hierarchical data in a graphical display for data analysis.

Topics:

- [Choose a Data Visualization Component for Your Application](#)
- [Use Attribute Groups With Data Visualization Components](#)

Choose a Data Visualization Component for Your Application

The visualization components include charts, gauges, and a variety of other visualizations including diagrams, timelines, thematic maps, and so on that you can use for displaying data. You may find the following usage suggestions helpful for determining which visualization to use in your application.

Charts

Charts show relationships among data and display values as lines, bars, and points within areas defined by one or more axes.

Chart Type	Image	Description	Usage Suggestions
Area		Displays series of data whose values are represented by filled-in areas. Areas can be stacked or unstacked. The axis is often labeled with time periods such as months.	<p>Use to show cumulative trends over time, such as sales for the last 12 months.</p> <p>Area charts require at least two groups of data along an axis.</p> <p>If you are working with multiple series and want to display unstacked data, use line or line with area charts to prevent values from being obscured.</p>
Bar		Displays data as a series of rectangular bars whose lengths are proportional to the data values. Bars display vertically or horizontally and can be stacked or unstacked.	<p>Use to compare values across products or categories, or to view aggregated data broken out by a time period.</p>
Box Plot		Displays the minimum, quartiles, median, and maximum values of groups of numerical data. Groups display vertically or horizontally. You can also vary the box width to make the width of the box proportional to the size of the group.	<p>Use to analyze the distribution of data. Box plots are also called box and whisker diagrams.</p>
Bubble		Displays three measures using data markers plotted on a two-dimensional plane. The location of the markers represents the first and second measures, and the size of the data markers represents the proportional values of the third measure.	<p>Use to show correlations among three types of values, especially when you have a number of data items and you want to see the general relationships.</p> <p>For example, use a bubble chart to plot salaries (x-axis), years of experience (y-axis), and productivity (size of bubble) for your work force. Such a chart enables you to examine productivity relative to salary and experience.</p>
Combination		Displays series of data whose values are represented by a combination of bars, lines, or filled-in areas.	<p>Combination charts are commonly configured as lines with bars for lines with stacked bars.</p> <p>For example, you can use a line to display team average rating with bars to represent individual team member ratings on a yearly basis.</p>
Funnel		Visually represents data related to steps in a process as a three-dimensional chart that represents target and actual values, and levels by color. The steps appear as vertical slices across a horizontal cone-shaped section. As the actual value for a given step or slice approaches the quota for that slice, the slice fills.	<p>Use to watch a process where the different sections of the funnel represent different stages in the process, such as a sales cycle.</p> <p>The funnel chart requires actual values and target values against a stage value, which might be time.</p>

Chart Type	Image	Description	Usage Suggestions
Line		Displays series of data whose values are represented by lines.	Use to compare items over the same time. Charts require data for at least two points for each member in a group. For example, a line chart over months requires at least two months. Typically a line of a specific color is associated with each group of data such as the Americas, Europe, and Asia. Lines should not be stacked which can obscure data. To display stacked data, use area or line with area charts.
Line with Area		Displays series of data whose values are represented as lines with filled-in areas.	Use for visualizing trends in a set of values over time and comparing those values across series.
Pie		Represents a set of data items as proportions of a total. The data items are displayed as sections of a circle causing the circle to look like a sliced pie.	Use to show relationship of parts to a whole such as how much revenue comes from each product line. Consider treemaps or sunbursts if you are working with hierarchical data or you want your visual to display two dimensions of data.
Polar		Displays series of data on a polar coordinate system. The polar coordinate system can be used for bar, line, area, combination, scatter, and bubble charts. Polygonal grid shape (commonly known as radar) is supported for polar line and area charts.	Use to display data with a cyclical x-axis, such as weather patterns over months of the year, or for data where the categories in the x-axis have no natural ordering, such as performance appraisal categories.
Pyramid		Displays values as slices in a pyramid. The area of each slice represents its value as a percentage of the total value of all slices.	Use to display hierarchical, proportional and foundation-based relationships, process steps, organizational layers, or topics interconnections.
Range		Displays a series of data whose values are represented either as an area or bar proportional to the data values.	Use to display a range of temperatures for each day of a month for a city.

Chart Type	Image	Description	Usage Suggestions
Scatter		Displays two measures using data markers plotted on a two-dimensional plane.	Use to show correlation between two different kinds of data values, such as sales and costs for top products. Scatter charts are especially useful when you want to see general relationships among a number of items.
Spark		Display trends or variations as a line, bar, floating bar, or area. Spark charts are simple and condensed.	Use to provide additional context to a data-dense display. Sparkcharts are often displayed in a table, dashboard, or inline with text.
Stock		Display stock prices and, optionally, the volume of trading for one or more stocks. When any stock or candlestick chart includes the volume of trading, the volume appears as bars in the lower part of the chart.	

Gauges

Gauges focus on a single value, displayed in relation to minimum, maximum, or threshold values.

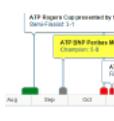
Gauge Type	Image	Description	Usage Suggestions
Dial		Displays a metric value plotted on a circular axis in relation to the minimum and maximum possible values for the metric. An indicator points to the dial gauge's metric value on the axis	The circular status meter is usually a better choice because of its more modern look and feel and efficient use of space.
LED		Graphically depicts a measurement, such as a key performance indicator (KPI). Several styles of shapes are available, including round or rectangular shapes that use color to indicate status, and triangles or arrows that point up, left, right, or down in addition to the color indicator.	Use to highlight a specific metric value in relation to its threshold.
Rating		Displays and optionally accepts input for a metric value.	Use to show ratings for products or services, such as the star rating for a movie.

Gauge Type	Image	Description	Usage Suggestions
Status Meter		Displays a metric value on a horizontal, vertical, or circular axis. An inner rectangle shows the current level of a measurement against the ranges marked on an outer rectangle. Optionally, status meters can display colors to indicate where the metric value falls within predefined thresholds.	

Other Data Visualizations

Other data visualizations include maps, timelines, Gantt charts and various other components that don't fit into the chart or gauge category.

Data Visualization Component	Image	Description	Usage Suggestions
Diagram		Models, represents, and visualizes information using a shape called a node to represent data, and links to represent relationships between nodes.	Use to highlight both the data objects and the relationships between them.
Gantt		Displays bars that indicate the start and end date of tasks.	Use to display project schedules.
Legend		Displays a panel which provides an explanation of the display data in symbol and label pairs.	Consider using the legend component when multiple visualizations on the same page are sharing a coloring scheme. For an example using <code>objLegend</code> with a bubble chart, see Use Attribute Groups With Data Visualization Components .
NBox		Displays data items across two dimensions. Each dimension can be split into multiple ranges, whose intersections result in distinct cells representing data items.	Use to visualize and compare data across a two-dimensional grid, represented visually by rows and columns.

Data Visualization Component	Image	Description	Usage Suggestions
PictoChart		Uses stamped images to display discrete data as a visualization of an absolute number or the relative size of different parts of a population.	<p>Common in infographics. Use when you want to use icons to:</p> <ul style="list-style-type: none"> visualize a discrete value, such as the number of people in a sample that meets a specified criteria. highlight the relative sizes of the data, such as the number of people belonging to each age group in a population sample.
Sunburst		Displays quantitative hierarchical data across two dimensions, represented visually by size and color. Uses nodes to reference the data in the hierarchy. Nodes in a radial layout, with the top of the hierarchy at the center and deeper levels farther away from the center.	<p>Use for identifying trends for large hierarchical data sets, where the proportional size of the nodes represents their importance compared to the whole. Color can also be used to represent an additional dimension of information.</p> <p>Use sunbursts to display the metrics for all levels in the hierarchy.</p>
Tag Cloud		Displays textual data where font style and size emphasizes the importance of each data item.	<p>Use for quickly identifying the most prominent terms to determine their relative importance.</p>
Thematic Map		Displays data that is associated with a geographic location.	<p>Use to show trends or patterns in data with a spatial element to it.</p>
Timeline		Displays a set of events in chronological order and offers rich support for graphical data rendering, scale manipulation, zooming, resizing, and objects grouping.	<p>Use to display time specific events in chronological order.</p>
Treemap		Displays quantitative hierarchical data across two dimensions, represented visually by size and color. Uses nodes to reference the data in the hierarchy. Nodes are displayed as a set of nested rectangles.	<p>Use for identifying trends for large hierarchical data sets, where the proportional size of the nodes represents their importance compared to the whole. Color can also be used to represent an additional dimension of information</p> <p>Use treemaps if you are primarily interested in displaying two metrics of data using size and color at a single layer of the hierarchy.</p>

For examples that implement visualization components, see the Oracle JET Cookbook at [Data Visualizations](#).

 **Note:**

To use an Oracle JET data visualization component, you must configure your application to use RequireJS. For details about adding RequireJS to your application, [Use RequireJS in an Oracle JET Application](#).

Use Attribute Groups With Data Visualization Components

Attribute groups allow you to provide stylistic values for color and shape that can be used as input for supported data visualization components, including bubble and scatter charts, sunbursts, thematic maps, and treemaps. In addition, you can share the attribute values across components, such as a thematic map and a legend, using an attribute group handler.

Using attribute groups is also one way that you can easily provide visual styling for data markers for a given data set. Instead of manually choosing a color for each unique property and setting a field in your data model, you can use an attribute group handler to get back a color or shape value given a data value. Once an attribute group handler retrieves a color or shape value given a data value, all subsequent calls that pass in the same data value will always return that color or shape.

Oracle JET provides the following classes that you can use for adding attribute groups to your data visualization components:

- `ColorAttributeGroupHandler`: Creates a color attribute group handler that will generate color attribute values.

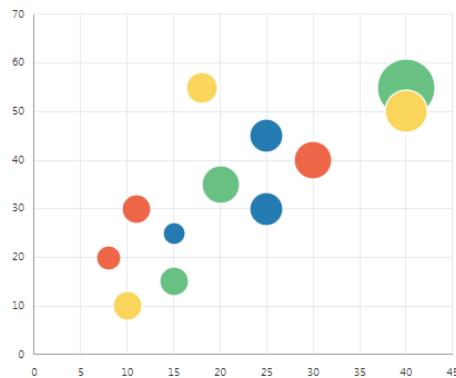
Colors are generated using the values in the `.oj-dvt-category-index*` tag selectors.

- `ShapeAttributeGroupHandler`: Creates a shape attribute group handler that will generate shape attribute values.

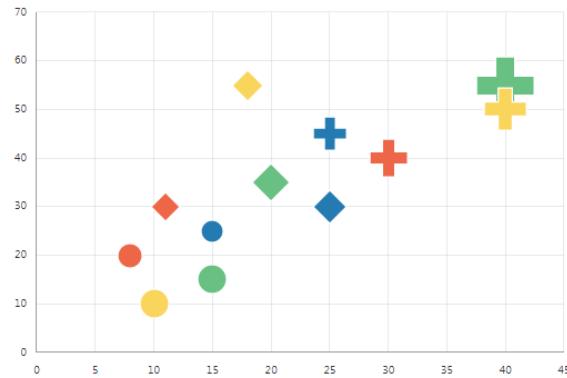
Supported shapes include `square`, `circle`, `human`, `triangleUp`, `triangleDown`, `diamond`, and `plus`.

You can see the effect of applying attribute groups to a bubble chart in the following figure. In this example, the shape of the markers (round, diamond, and plus) indicates the year for which the data applies. The color differentiates the brand. The example also uses the `Legend` data visualization component to provide a legend for the bubble chart.

Bubble Chart with Default Colors and Shapes



Bubble Chart with Attribute Groups for Color and Shape



The code excerpt below shows the JavaScript to create the bubble chart with color and shape attribute groups. The code relating to the attribute groups is highlighted in bold font.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojchart','ojs/ojlegend'],
    function(ko, Bootstrap)
    {
        var colorHandler = new ColorAttributeGroupHandler();
        var shapeHandler = new ShapeAttributeGroupHandler();
        shapeHandler.getValue();

        function ChartModel() {
            var data = [
                {x: 15, y: 25, z: 5, company: "Coke", year: "2010"}, {x: 25, y: 30, z: 12, company: "Coke", year: "2011"}, {x: 25, y: 45, z: 12, company: "Coke", year: "2012"}, {x: 15, y: 15, z: 8, company: "Pepsi", year: "2010"}, {x: 20, y: 35, z: 14, company: "Pepsi", year: "2011"}, {x: 40, y: 55, z: 35, company: "Pepsi", year: "2012"}, {x: 10, y: 10, z: 8, company: "Snapple", year: "2010"}, {x: 18, y: 55, z: 10, company: "Snapple", year: "2011"}, {x: 40, y: 50, z: 18, company: "Snapple", year: "2012"}, {x: 8, y: 20, z: 6, company: "Nestle", year: "2010"}, {x: 11, y: 30, z: 8, company: "Nestle", year: "2011"}, {x: 30, y: 40, z: 15, company: "Nestle", year: "2012"}];

            this.seriesItems = [
                {name: "Coke", displayInLegend: 'off', items: []}, {name: "Pepsi", displayInLegend: 'off', items: []}, {name: "Snapple", displayInLegend: 'off', items: []}, {name: "Nestle", displayInLegend: 'off', items: []}
            ];
            this.bubbleGroups = [];

            for(var i = 0; i < data.length; i++){
                var seriesIndex = Math.floor(i/3);
                this.seriesItems[seriesIndex].items.push({
                    x: data[i].x, y: data[i].y, z: data[i].z,
                    color: colorHandler.getValue(data[i].company),
                    markerShape: shapeHandler.getValue(data[i].year),
                    categories: [ data[i].company, data[i].year],

```

```

        shortDesc: data[i].company + " " + data[i].year + "<br/>X: " +
            data[i].x + "<br/>Y: " + data[i].y + "<br/>Z: " + data[i].z
    });
}

this.bubbleGroups= ["2010", "2011", "2012"];

var legendSections = {sections: [
    {title: "Year", items: [
        {markerShape:shapeHandler.getValue("2010"), text: "2010", id:
        "2010"},

        {markerShape:shapeHandler.getValue("2011"), text: "2011", id:
        "2011"},

        {markerShape:shapeHandler.getValue("2012"), text: "2012", id:
        "2012"}]}]};
```

{title: "Brand", items:[
 {color: colorHandler.getValue("Coke"), text: "Coke", id: "Coke"},
 {color: colorHandler.getValue("Pepsi"), text: "Pepsi", id:
 "Pepsi"},
 {color: colorHandler.getValue("Snapple"), text: "Snapple", id:
 "Snapple"},
 {color: colorHandler.getValue("Nestle"), text: "Nestle", id:
 "Nestle"}]}];

```

    self.legendSections = ko.observable(legendSections);
}

var chartModel = new ChartModel();

Bootstrap.whenDocumentReady().then(
    function ()
    {
        ko.applyBindings(chartModel, document.getElementById('chart-
container'));
    }
);
});
```

The bubble chart's legend uses the same attribute group handlers for color and shape. Since the color and shape values were initially set in the `colorHandler.getValue()` and `shapeHandler.getValue()` calls above, the calls to `getValue()` below will return the same values for color and shape.

```

...content omitted
var legendSections = {sections: [
    {title: "Year", items: [
        {markerShape:shapeHandler.getValue("2010"), text: "2010", id:
        "2010"},

        {markerShape:shapeHandler.getValue("2011"), text: "2011", id:
        "2011"},

        {markerShape:shapeHandler.getValue("2012"), text: "2012", id:
        "2012"}]}]};
```

{title: "Brand", items:[
 {color: colorHandler.getValue("Coke"), text: "Coke", id: "Coke"},
 {color: colorHandler.getValue("Pepsi"), text: "Pepsi", id:
 "Pepsi"},
 {color: colorHandler.getValue("Snapple"), text: "Snapple", id:
 "Snapple"},
 {color: colorHandler.getValue("Nestle"), text: "Nestle", id:
 "Nestle"}]}];

```
        {color: colorHandler.getValue("Nestle"), text: "Nestle", id:  
"Nestle"}  
    ]}  
];  
self.legendSections = ko.observable(legendSections);  
}  
  
var chartModel = new ChartModel();  
  
Bootstrap.whenDocumentReady().then(  
    function()  
    {  
        ko.applyBindings(chartModel, document.getElementById('chart-  
container'));  
    }  
)
```

The Oracle JET Cookbook provides the complete code for implementing both bubble charts at [Bubble Charts](#).

You can also initialize an attribute group with match rules which consist of a map of key value pairs for categories and the matching attribute values. For example, if you wanted to specify colors for specific categories instead of using the default colors, you could define the color attribute group with match rules.

```
var colorHandler = new ColorAttributeGroupHandler({ "soda": "#336699",  
                                                 "water": "#CC3300",  
                                                 "iced tea": "#F7C808" });
```

For detailed information about `ColorAttributeGroupHandler`, see the [ColorAttributeGroupHandler](#) API documentation. For more information about `ShapeAttributeGroupHandler`, see the [ShapeAttributeGroupHandler](#) API documentation.

Working with Oracle JET Web Components

Oracle JET Web Components are reusable pieces of user interface code that you can embed as custom HTML elements. Web Components can contain Oracle JET components, other Web Components, HTML, JavaScript, and CSS. You can create your own Web Component or add one to your page.

Topics

- [Typical Workflow for Working with Oracle JET Web Components](#)
- [Design Custom Web Components](#)
- [About Web Components](#)
- [Best Practices for Web Component Creation](#)
- [Create Web Components](#)
- [Test Web Components](#)
- [Add Web Components to Your Page](#)
- [Build Web Components](#)
- [Package Web Components](#)
- [Create a Project to Host a Shared Oracle Component Exchange](#)
- [Publish Web Components to Oracle Component Exchange](#)
- [Upload and Consume Web Components on a CDN](#)

Typical Workflow for Working with Oracle JET Web Components

Understand Oracle JET Web Components and the steps required to create and add them to your page.

To work with Web Components, refer to the typical workflow described in the following table.

Task	Description	More Information
Understand the Oracle JET Web Component architecture	Understand the structure and functionality of Web Components.	About Web Components
Decide how you want to organize custom Web Components	Learn about the various custom component types and how they are used by developers,	Design Custom Web Components
Review best practices for Web Component creation	Understand best practices for creating your Web Component, including design, coding, versioning, and styling standards.	Best Practices for Web Component Creation

Task	Description	More Information
Create your Web Components	Identify the steps you must take to create the various types of Oracle JET Web Components.	Create Web Components
Test your Web Components	Understand the testing guidelines for Web Components.	Test Web Components
Use standalone Web Components in your application	Identify the steps you must take to add a standalone Web Component to your page.	Add Web Components to Your Page
Build a Web Component	Understand the command used to build and optimize a Web Component that you want to share.	Build Web Components
Package your Web Components	Understand the command used to package a Web Component as a sharable zip archive file.	Package Web Components
Publish your Web Components	Identify the steps you can take to publish to Oracle Component Exchange or to upload to a CDN.	Publish Web Components to Oracle Component Exchange and Upload and Consume Web Components on a CDN

Design Custom Web Components

Oracle JET Web Components are custom components that include multiple component types. Web components that you create can be used in your application or they can be uploaded to Oracle Component Exchange to share with other developers.

The variety of component types supported by Oracle JET and Oracle Component Exchange are:

- **Standalone Web Components** are classic UI components with some kind of UI along with a defined set of properties, methods, events and slots. They can represent everything from a simple better-button type of widget all the way to a super complex whole page component such as a calendar.
- **Pack components**, also called a JET Pack, represents a versioned stripe of related components designed to be used together. When consumers pick up a component that is a member of a JET Pack they associate their application with the version of the pack as a whole, not the individual components within it. The JET Pack simplifies the setup of such projects and the dependency management as a whole.
- **Resource components** are re-usable libraries of assets used by Web Components contained in JET Packs. Resource components typically contain things like shared images, resource bundles, utility classes and so forth. A resource component has no hard and fast predefined structure so can contain anything that you want. However, it does not itself provide any UI components. Instead conventional standalone components would depend on one of these for shared resources. We only expect resource components to be used in concert with JET Packs.
- **Reference components** define a look-up reference to third party code. As such, reference components are a pointer to that code either as a NPM module and/or as a CDN location. Reference components don't actually include the third party libraries, they just point to it.

You can create standalone Web Components to support your specific application needs. You can also create sets of Web Components that you intend to be used together and assemble those in a JET Pack, or pack component type. The pack component contains the Web Components and configuration files that define the version stripe of each component in the pack. When Web Components are part of a pack, changes to their definition file are required to differentiate them from the same component used as a standalone component.

 **Tip:**

When you assemble components into a small number of packs, from the consumer's point of view, it makes path setup and dependency management much simpler.

You can enhance JET Pack components by using resource components when you have re-usable libraries of assets that are themselves not specifically UI components. The resource component structure is flexible so you add anything that you want, such as shared images, resource bundles, utility classes and so forth.

If you need to reference third-party code in a Web Component, you can create a reference component. The reference component doesn't include any third-party libraries, but it can define a pointer to that code either as a NPM module and/or as a CDN location. Although it is possible to embed third party library code into the packaged distribution of a given component, by separating it out you get two particular benefits:

- The dependency is clear and declared up front. This is an important consideration for organizations that care about third party liability and license usage. It also makes reacting to security vulnerabilities in third party code much easier.
- You can maximize re-use of these libraries - particularly with common libraries, such as moment.js.

The only component type that is not allowed in packs are reference components.

When creating JET Packs it is important to think about how their components can evolve over time in relation to the consuming applications. A common mistake is to start out syncing the component version numbers to the version number of the primary consuming application. This relationship can break down, however, for example when a breaking API change in one of your components forces a major version change which now takes you out of sync. The best practice is to adopt Semantic Versioning (SemVer) of components from the start and not to sync versions with the consuming application. Take the time to understand how SemVer works, particularly in relation to re-release versions. For more information, see [Version Numbering Standards](#).

You should maintain the source code for your component sets separately from the applications that will consume them. Remember that components will evolve over time at a separate rate from the consuming application so having the source code decoupled into a separate source code project and repository makes a huge amount of sense.

The recommended project layout for your component source project is based on the default project layout created by Oracle JET CLI tooling. The JET CLI supports the creation of TypeScript components as well as standard ES5/6 based components. If

you follow adhere to the project layout generated by JET tooling, then you derive the following tooling benefits:

- Automatic creation of the correct requireJS path mappings for your components and their upstream dependencies when testing within the context of this component source project
- Support for live editing when using the `ojet serve` command (both JS and TS components)
- Auto transpilation of Typescript based components to ES5 or ES6 as desired (ES6 by default)
- Automatic creation of both debug and minified components by the `ojet build` command
- Automatic creation of component bundles for JET Packs where bundling is specified
- Ability to directly publish to the Component Exchange using the `ojet publish component` command
- Ability to directly package your components into distributable zip files using the `ojet package component` command

About Web Components

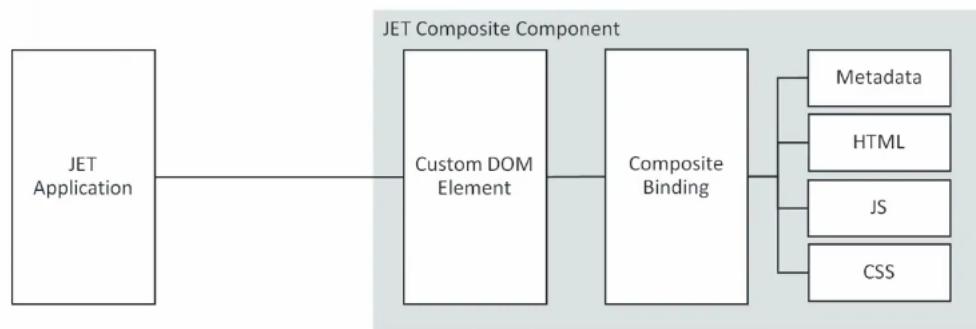
Oracle JET Web Components are packaged as standalone modules that your application can load using RequireJS. The framework supplies APIs that you can use to register Web Components. Knockout currently provides one and two way data binding, template markup, and Web Component activation.

Web Components follow much of the W3C Web Components - Custom Elements specification which describes a mechanism for enabling authors to define and use custom DOM elements. If you are new to Web Components and would like to learn more, visit this Oracle Blogs page for a series of articles that will introduce you to important concepts: <https://blogs.oracle.com/groundside/cca>.

Web Component Architecture

The following image shows a high-level view of the JET Web Component architecture. In this example, an Oracle JET application is consuming the Web Component, but you can add Web Components to other JavaScript or Oracle applications where supported.

JET Composite Component Architecture (CCA)



Web Components contain:

- A custom DOM element: Named element that functions as a HTML element.

```
<my-web-component attribute1="value1" attribute2="value2" ...>  
</my-web-component>
```

- Web Component binding definition: Knockout expression for setting Web Component attributes.

```
<my-web-component attribute1="value1" attribute2="[[value2]]"  
attribute3="{{value3}}">  
</my-web-component>
```

`attribute1`'s value can be a primitive JavaScript type (boolean, number, string) or a JSON object, `attribute2`'s value uses one way data binding, and `attribute3`'s value uses a two way binding. One way bindings on Web Components specify that the expression will not update the application's ViewModel if the value changes. In two way bindings, the expression will update and the value written back to the application's ViewModel.

In the following code sample, the Web Component is declared with three attributes: `type`, `data`, and `axis-labels`.

```
<my-chart type=bubble data="[[salesData]]" axis-  
labels={{showAxisLabels}} ... </my-chart>
```

Because the `salesData` expression is declared using one way binding (`[[salesData]]`), it will not be written back to if the `data` property is updated by the Web Component's ViewModel. The `data` property will contain the current value, but the `salesData` expression will not be updated. Alternatively, if the `axisLabels` property is updated by the ViewModel, both the `axisLabel` property and the `{{showAxisLabels}}` expression will contain the updated value.

- Metadata: Data provided in JSON format which defines the Web Component's required properties: `name`, `version`, and `jetVersion`. Metadata may also define optional properties, including `description`, `displayName`, `dependencies`, `icon`, `methods`, `events`, and `slots`.

Web Components support both runtime and design time metadata. Design time metadata isn't required at runtime and is useful for design time tools and property editors. Design time tools can define tools-specific metadata extensions to the Web Component's metadata. For any tool-specific metadata extensions, refer to the documentation for that specific tool. For additional information about metadata properties, see [Composite](#) in the API documentation.

The following sample shows some of the available metadata fields with descriptions of their content and whether they are not used at run time. Required metadata are highlighted in bold.

```
{  
  "name": "The component tag name",  
  "version": "The component version. Note that changes to the  
  metadata even for minor updates like updating the jetVersion should  
  result in at least a minor Web Component version change, e.g. 1.0.0
```

```

-> 1.0.1.",
  "jetVersion": "The semantic version of the supported JET version(s).  
Web Component authors should not specify a semantic version range  
that includes unreleased JET major versions as major releases may  
contain non backwards compatible changes. Authors should instead  
recertify Web Components with each major release and update the  
component metadata or release a new version that is compatible with  
the new release changes.",
  "description": "A high-level description for the component. Not  
used at run time.",
  "displayName": "A user friendly, translatable name of the  
component. Not used at run time.",

  "properties": {
    "property1": {
      "description": "A description for the property. Not used at  
run time.",
      "displayName": "A user friendly, translatable name of the  
property. Not used at run time.",
      "readOnly": "Boolean that determines whether a property can  
be updated outside of the ViewModel. False by default.",
      "type": "The type of the property, following Google's Closure  
Compiler syntax.",
      "value": "Object containing an optional default value for a  
property.",
      "writeback": "Boolean that determines whether an expression  
bound to this property should be written back to. False by  
default.",
      "enumValues": "An optional array of valid enum values for a  
string property. An error is thrown if a property value does not  
match one of the provided enumValues.",
      "properties": "A nested properties object for complex  
properties. Subproperties exposed using nested properties objects  
in the metadata can be set using dot notation in the attribute.  
See the Subproperties section for more details on working with  
subproperties."
    },
    "property2": {
      ... contents omitted
    }
  },
  "methods": {
    "method1": {
      "description": "A description for the method. Not used at run  
time.",
      "displayName": "A user friendly, translatable name of the  
method. Not used at run time.",
      "internalName": "An optional ViewModel method name that is  
different from, but maps to this method.",
      "params": "An array of objects describing the method  
parameter. Not used at run time.",
      "return": "The return type of the method, following Closure  
Compiler syntax. Not used at run time."
    }
  }
}

```

```

        },
        "method2": {
          ... contents omitted
        }
      },

      "events": {
        "event1": {
          "bubbles": "Boolean that indicates whether the event bubbles up through the DOM or not. Defaults to false. Not used at run time.",
          "cancelable": "Boolean that Indicates whether the event is cancelable or not. Defaults to false. Not used at run time.",
          "description": "A description for the event. Not used at run time.",
          "displayName": "A user friendly, translatable name of the method. Not used at run time.",
          "detail": {
            "field name": "Describes the properties available on the event's detail property which contains data passed when initializing the event. Not used at run time."
          }
        },
        "event2": {
          ... contents omitted
        }
      },

      "slots": {
        "slot1": {
          "description": "A description for the slot. Not used at run time.",
          "displayName": "A user friendly, translatable name of the method. Not used at run time."
        }
      }
    }
  }
}

```

- HTML markup: (Required) Contains the View definition which describes how to render the Web Component. The Web Component's View has access to several \$ variables along with any public variables defined in the Web Component's ViewModel. Some of the variables that can be used in the Web Component's View are:

Variables	Description
\$properties	A map of the Web Component's current property values
\$slotCounts	A map of slot names containing a number of associated child nodes assigned to that slot
\$unique	A unique string value provided for every component instance that can be used for unique ID generation
\$uniqueId	The ID of the Web Component, if specified. Otherwise, it is the same as unique

Variables	Description
\$props	Deprecated since 5.0.0, use \$properties instead
\$slotNodeCounts	Deprecated since 5.0.0, use \$slotCounts instead

- JavaScript: Optional script for defining the ViewModel and custom events.

The ViewModel is also where you define callbacks for various stages of the Web Component's lifecycle. Web Components support the following optional lifecycle methods: activated (context), connected (context), bindingsApplied (context), propertyChanged (context), and disconnected (element). For more information on lifecycle methods, see [Composite - Lifecycle](#).

- CSS: Optional styling for the Web Component.

CSS is not scoped to Web Components, and you must define styles appropriately.

- SCSS: Optional files containing Sass variables to generate the Web Component's CSS.

If you're defining only a few styles for your component, then adding the CSS manually may be sufficient. However, there may be use cases where you want to use Sass variables to generate the CSS. In those cases, create and place the SCSS files in the Web Component's folder and use the tooling to add node-sass to your application. See [Step 8 - Creating Web Components](#).

Important:

You must add the Sass files manually to the Web Component's folder. The tooling will compile any Sass files if they exist, but it will not create them for you.

Web Component Files

Web Components can contain CSS, HTML, JavaScript, and metadata files that you can modify according to your application requirements.

You can create a Web Component manually by creating a folder and adding the required files within the folder. You can also create a Web Component by using the Oracle JET CLI command `ojet create component <component-name>` that automatically generates the Web Component folder with the required files for your application.

When you create a Web Component manually, place the Web Component files in a folder with the same name as the Web Component tag. Typically, you place the folder within your application in a `jet-composites` folder: `application-path/jet-composites/my-web-component/`.

You can also place your Web Component in a different file location or reference a Web Component on a different server using RequireJS path mapping. For examples, see [Composite - Packaging and Registration](#).

Each Web Component file should use the following naming convention to match the purpose:

- `my-web-component-view.html`: view template

- *my-web-component-viewModel.js*: ViewModel
- *component.json*: metadata
- *my-web-component-styles.css*: CSS styling
- *my-web-component-styles.scss*: Sass variables to generate CSS for Web Components
- *loader.js*: RequireJS module defining the dependencies for its metadata, View, ViewModel, and CSS This file should also include the Web Component registration.

Web Component Slotting

Slots are used as placeholders in a Web Component that users can fill in with their markup. Slot is defined in the component JSON file of your Web Component.

Use slotting to add child components (which can also be Web Components) that get slotted into specified locations within the Web Component's View markup. The following example contains a portion of the View markup for a Web Component named `demo-columns`.

```
<div class="oj-flex oj-flex-item-pad">
  <div role="group" :aria-label="[$properties.headers[0]]"
    class="oj-flex-item demo-columns-col-a oj-flex oj-sm-flex-direction-
    column oj-sm-align-items-center">
    <h3>
      <oj-bind-text value="[$properties.headers[0]]"></oj-bind-
    text>
    </h3>
    <oj-bind-slot name="columnA">
    </oj-bind-slot>
  </div>
  ... content omitted
</div>
```

In this example, the `demo-columns` Web Component defines an `oj-bind-slot` named `columnA`. As shown below, a developer can specify a child component with a slot named `columnA` when adding the `demo-columns` Web Component to the page.

```
<demo-columns id="composite-container" headers='["Sales", "Human
Resources", "Support"]'>
  <!-- ko foreach: sales -->
  <demo-card slot="columnA" name="[[name]]" work-title="[[title]]"></
  demo-card>
  <!-- /ko -->
  ... contents omitted
</demo-columns>
```

Web Component Template Slots

You can define placeholders in your template using template slots that can be filled with any markup fragment you want when the template element is used within a markup of your component.

When you need to reuse a stamped template with varying data, you can use a template slot to expose the additional data from the component's binding context.

You can define placeholders in your template using template slots that can be filled with any markup fragment you want when the template element is used within a markup of your component. When you need to reuse a stamped template with varying data, you can use a template slot to expose the additional data from the component's binding context.

Template slots for Web Components are used to define additional binding contexts for a slotted content within an application. To declaratively define a template slot, use the `oj-bind-template-slot` element in the Web Component's View markup for the slot that contains a stamped template DOM. The `oj-bind-template-slot` element is similar to the `oj-bind-slot` element, but its slotted content should be wrapped inside a `template` element within the application DOM.

In the below example, the `demo-list` Web Component defines an `oj-bind-template-slot` named `item`. This template slot provides the `data` attribute that exposes additional properties to the template DOM and an `data-oj-as` attribute that is used as an alias for the `$current` variable. Note that the `data-oj-as` attribute for `template` element can be referenced only inside a default template.

```
<table>
  <thead>
    <tr>
      <th>
        <oj-bind-text value="[$properties.header]"></oj-bind-text>
      </th>
    </tr>
  </thead>
  <tbody>
    <oj-bind-for-each data="{{$properties.data}}">
      <template>
        <tr>
          <td>
            <!-- Template slot for list items with default template and
an optional alias -->
            <oj-bind-template-slot name="item" data="{{$value:
$current.data}}">
              <!-- Default template -->
              <template data-oj-as="listItem">
                <span>
                  <oj-bind-text value='[[listItem.value]]'></oj-bind-text>
                </span>
              </template>
            </oj-bind-template-slot>
          </td>
        </tr>
      </template>
    </oj-bind-for-each>
  </tbody>
</table>
```

```
</template>
</oj-bind-for-each>
</tbody>
... contents omitted
</table>
```

The `oj-bind-template-slot` children are resolved when the Web Component View bindings are applied and are then resolved in the application's binding context extended with additional properties provided by the Web Component. These additional properties are available on the `$current` variable in the application provided template slot. The application can use an optional `data-oj-as` attribute as an alias in the template instead of the `$current` variable. The following example contains a portion of the application's markup named `demo-list`.

```
<demo-list data="{{groceryList}}" header="Groceries">
<template slot="item" data-oj-as="groceryItem">
  <oj-checkboxset>
    <oj-option value="bought"><oj-bind-text
      value='[[groceryItem.value]]'></oj-bind-text></oj-option>
  </oj-checkboxset>
</template>
... contents omitted
</demo-list>
```

The Oracle JET Cookbook at [Web Component - Template Slots](#) includes complete examples for using template slots. `oj-bind-template-slot` API documentation describes the attributes and other template slot properties.

Web Component Events

Oracle JET Web Components can fire automatic property changed events that are mapped to the properties defined in the component metadata. These components will also fire custom events for the events declared in the component metadata.

Web Components can internally listen to the automatically generated `propertyChanged` events that are mapped to the properties in the component metadata using the `propertyChanged` lifecycle method. For example, a `propertyChanged` event is fired when a property is updated. This `propertyChanged` event contains the following properties:

- `property`: Name of the property that changed.
- `value`: Current value of the property.
- `previousValue`: Previous value of the property that changed.
- `updatedFrom`: The location from where the property was updated.
- `Subproperty`: An object holding information about the subproperty that changed.

When there is a need to declaratively define a custom event for a Web Component, you must declare the event in the component's metadata file. These events will only be fired if the code of the Web Component calls the `dispatchEvent()` method. The application can listen to these events by declaring the `event listener` attributes and `property setters`. These event listeners can be added declaratively or programmatically.

For the declarative specification of event listeners, use the `on-[event-name]` syntax for the attributes. For example, `on-click`, `on-value-changed`, and so on.

```
<ojs-element-name value="{{currentValue}}" on-value-changed="{{valueChangedListener}}"></ojs-element-name>
```

The programmatic specification of event listeners may use the DOM `addEventListener` mechanism or by using the custom element property.

The DOM `addEventListener` mechanism uses the `elementName.addEventListener` method. For example:

```
elementName.addEventListener("valueChanged", function(event) {...});
```

The custom element property uses the `elementName.onEventName` syntax for the property setter. For example:

```
elementName.onValueChanged = function(event) {...};
```

For more information, see the [Web Components - Events and Listeners](#) API documentation.

Web Component Examples

Web Components can contain slots, data binding, template slots, nested Web Components, and events. You can use the examples provided in the Oracle JET Cookbook for these Web Component features.

The Oracle JET Cookbook contains complete examples for creating basic and advanced Web Components. You can also find examples that use slotting and data binding. For details, see [Web Component - Basic](#).

For additional information about Web Component fields and methods, see [Composite](#) in the API documentation.

Best Practices for Web Component Creation

Best practices for creating Oracle JET Web Components include required and recommended patterns, configuration, coding practices, version numbering, and styling standards. Follow best practices to ensure interoperability with other Web Components and consuming frameworks.

Topics

- [Recommended Standard Patterns and Coding Practices](#)
- [CSS and Theming Standards](#)
- [Version Numbering Standards](#)

Recommended Standard Patterns and Coding Practices

Recommended patterns and coding practices for Oracle JET Web Components include standards for configuration, versioning, coding, and archival.

Component Versioning

Your Web Component must be assigned a version number in semantic version format.

When assigning and incrementing the version number associated with your components, be sure to follow semantic version rules and update Major, Minor and Patch version numbers appropriately. By doing so, component consumers will have a clear understanding about the compatibility and costs of migrating between different versions of your component.

To assign a version number to your Web Component, see [semantic version](#).

JET Version Compatibility

You must use the semantic version rules to specify the `jetVersion` of the supported JET version(s). Web Component authors should not specify a semantic version range that includes unreleased JET major versions as major releases may contain non backwards compatible changes. Authors should instead recertify Web Components with each major release and update the component metadata or release a new version that is compatible with the new release changes.

Translatable Resources

Developers who want to localize Web Component translatable resources now get a resource bundle (template) when they create their Web Component. These components should use the standard Oracle JET mechanism using the `oJL10n requireJS` plugin. You must store the translation bundles in a `resources/nls` subdirectory within your Web Component's root folder. You can declare the languages and locales that you support in the Web Component metadata.

Peer-to-Peer Communication

Components must prefer a shared observable provided by the consumer over any kind of secret signaling mechanism when you are dealing with a complex integration. For example, a filter component and a data display component. By using a shared observable you can pre-seed and programmatically interact with the components through the filter.

Alternatively, you can use events and public methods based on one of the following approaches being used:

- A hierarchical relationship between the source and receiver of the event.
- The identity of the source being passed to the receiver.

Note that in some runtime platforms, the developer doing the wiring may not have access to component IDs to pass the relevant identity.

- Listeners attached by components at the document level.

In this case, you are responsible for the cleanup of those listeners, management of duplicates, and so on. Also, such listeners should preferably be based on Web

Component events, not common events such as click, which might be overridden by intermediate nodes.

 **Note:**

Under the web-component standards (shadow DOM), events will be re-targeted as they transition the boundary between the component and the consuming view. That is, the apparent identity of the raising element might be changed, particularly in the case of Nested Web Component architecture where the event would get tagged with the element representing the outer Web Component rather than the inner Web Component. Therefore, you should not rely on the `event.target` attribute to identify the Web Component source when listening at the document level. Instead, the `event.deepPath` attribute can be used to understand the actual origin of the event.

Access to External Data

Web Components do not permit the usage of the knockout binding hierarchy to obtain data from outside the Web Component context, for example, `$root`, `$parent[1]`, and so on. All data transfer in and out of the component must be through the formal properties, methods, and events.

Object Scope

All properties and functions of Web Components should be confined to the scope of the view model. No window or global scope objects should be created. Similarly, the existence of window scope objects should not be assumed by the Web Component author. If a consumer Web Component defined externally at window or global level is required for read or write then that component must be passed in by the consuming view model through a formal property. Even if a well known global reference is needed from outside of the component, it should be formally injected using the require `define()` function and declared as a dependency in the Web Component metadata.

External References

If a Web Component must reference an external component, it should be part of the formal API of the component. The formal API passes the component reference through a property. For example, to allow the registration of a listener, the Web Component code requires a component reference defined externally. You must not allow Web Components to obtain IDs from hard-coded values, global storage, or walking the DOM.

Subcomponent IDs

Within the framework if any component needs a specific ID, use `context.unique` or `context.uniqueId` value to generate the ID. This ID is unique within the context of the page.

ID Storage

Any generated IDs should not be stored across invocation, such as in local storage or in cookies. The `context.unique` value for a particular Web Component may change each time a particular view is executed.

LocalStorage

It is difficult to consistently identify a unique instance of a Web Component within an application. So, it is advised not to allow a Web Component to utilize the local storage of a browser for persisting information that is specific to an instance of that Web Component. However, if the application provides a unique key through the public properties of the component you can then identify the unique instance of the component.

Additionally, do not use local storage as a secret signaling mechanism between composites. You cannot assure the availability of the capability and so it is recommended to exchange information through a shared JavaScript object or events as part of the public API for the component(s).

String Overrides

Web Components will often contain string resources internally to service their default needs for UI and messages. However, sometimes you may want to allow the consumer to override these strings. To do this, expose a property for this purpose on the component. By convention such a property would be called `translations`, and within it you can have sub-properties for each translatable string that relates to a required property on the component, for example `requiredHint`, `requiredMessageSummary`, and so on. These properties can then be set on the component tag using sub-property references. For example:

```
"properties" : {
  "translations": {
    "description": "Property to allow override of default messages",
    "writeback" : false,
    "properties" : {
      "requiredHint": {
        "description": "Change the text of the required hint",
        "type": "string"
      },
      "requiredMessageSummary": {
        "description": "...",
        "type": "string"
      },
      "requiredMessageDetail": {
        "description": "...",
        "type": "string"
      }
    }
  }
}
```

Logging

Use `Logger` to write log messages from your code in preference to `console.log()`. The Web Components should respect the logging level of the consuming application and not change it. You should ideally prefix all log messages emitted from the component with an identification of the source Web Component. As a preference, you can use the Web Component name. The components should not override the writer defined by the consuming application.

Expensive Initialization

Web Components should carry out minimum work inside the constructor function. Expensive initialization should be deferred to the `activated` lifecycle method or later. The constructor of a Web Component is invoked even if the component is not actually added to the visible DOM. For example, if a constructor is invoked within a Knockout `if` block. The further lifecycle phases will only occur when the component is actually needed.

Service Classes

The use of global service classes, that is functionality shared across multiple Web Components, can constitute an invisible contract that the consumer of your Web Component has to know about. To avoid this, we recommend:

- Create the service as a module that every Web Component can explicitly set it as `require()` block, thus removing the need for the consumer to do this elsewhere.
- Consider the timing issues that might occur if your service class needs some time to initialize, for example fetching data from a remote service. In such cases, you should be returning promises to the service object so that the components can safely avoid trying to use the information before it is actually available.

Using `ojModule`

If you use `ojModule` in a Web Component and plan to distribute the Web Component outside of your application, you must take additional steps to ensure that the contained `ojModule` could be loaded from the location relative to the location of the Web Component. Unless the View and ViewModel instances are being passed to `ojModule` directly, you will need to provide the require function instance and the relative paths for views and view models. The require function instance should be obtained by the component loader module by specifying `require` as a dependency.

```
<div data-bind="ojModule: {require: {instance:  
    require_instance, viewPath: "path_to_Web_Component_Views", modelPath:  
    "path_to_cWeb_Component_ViewModels"}}"></div>
```

require Option	Type	Description
instance	Function	Function defining the require instance
viewPath	String	String containing the path to the Web Component's Views
modelPath	String	String containing the path to the Web Component's ViewModels

For additional information about working with `ojModule`, see [ojModule](#).

Archiving Web Components for Distribution

If you want to create a zip file for packaging, create an archive with the same name as the component itself. You may add version-identifying suffixes to the zip file name for operational reasons. The Web Component artifacts must be placed in the root of the zip file, and there should be no intermediate directory structure before reaching the files.

Using Lifecycle Methods

If a ViewModel is provided for a Web Component, the following optional callback methods can be defined on its ViewModel that will be called at each stage of the Web Component's lifecycle. Some of the callback methods that can be used are listed below:

- `activated(context)`: Invoked after the ViewModel is initialized.
- `connected(context)`: Invoked after the View is first inserted into the DOM and then each time the Web Component is reconnected to the DOM after being disconnected.
- `bindingsApplied(context)`: Invoked after the bindings are applied on the View.
- `propertyChanged(context)`: Invoked when properties are updated before the `[property]Changed` event is fired.
- `disconnected(element)`: Invoked when this Web Component is disconnected from the DOM.

For additional information on Web Component lifecycle methods, see [Composite - Lifecycle](#).

Template Aliasing

JET components that support inline templates can now use an optional `data-oj-as` attribute to provide template specific aliases for `$current` variable at the application level. In the instances where the component must support multiple template slots as in the case of chart and table components, a single alias may not be sufficient. In such cases, you can use an optional `data-oj-as` attribute on the template element. For more information on the usage of this optional attribute with template slots, see [oj-bind-template-slot](#) API documentation.

CSS and Theming Standards

Oracle JET Web Components should comply with all recommended styling standards to ensure interoperability with other Web Components and consuming applications.

For information on the generic best practices for using CSS and Themes, see [Best Practices for Using CSS and Themes](#).

Standard	Details	Example
Prevent flash of unstyled content	<p>Oracle JET will add the <code>oj-complete</code> class to the Web Component DOM element after metadata properties have been resolved. To prevent a flash of unstyled content before the component properties have been setup, the component's CSS should include a rule to hide the component until the <code>oj-complete</code> class is set on the element.</p> <p>Note that this is an <code>element</code> selector, and there should <i>not</i> be a dot (.) before <code>acme-branding</code>.</p>	<pre>acme-branding:not(.oj-complete) { visibility: hidden; }</pre>

Standard	Details	Example
Add scoping	Use an element selector to minimize the chance that one of your classes is used by someone outside of your component and becomes dependent on your internal implementation. In the example to the right, if someone tries to apply the class acme-branding-header, it will have no effect if it's not within an acme-branding tag.	<pre>acme-branding .acme- branding-header { color: white; background: blue; }</pre> <p>IMPORTANT: If your component also includes a dialog, then when displayed, that dialog will be attached to the main document DOM tree and will not be a child of your component. Therefore, if you define a style to apply to a dialog defined by your component, you cannot scope it to the component name as that's not the actual container for the dialog when displayed.</p> <p>To resolve, use the component name as the prefix instead:</p> <pre>.acme-branding-dialog- background{ color: white; background: blue; }</pre>
Avoid element styling	The application will often style HTML tag elements like headers, links, and so on. In order to blend in with the application, avoid styling these elements in your Web Component. For information about Oracle JET's use of styles on HTML tags, see Use Tag Selectors or Style Classes with Alta .	 Avoid styling on elements like headers. <pre>acme-branding .acme- branding-header h3{ font-size: 45px; }</pre>

Version Numbering Standards

Web Components including standalone, JET Pack, and Resource components, require a version number and that number should adhere to a semantic versioning (SemVer) scheme that ensures a standard for development teams to follow similar to the approach adopted by Oracle JET release versioning.

Reference components are a slightly special case as the version of the reference component will always match the version of the NPM library that it references.

All other Web Component types, rely on semantic versioning to designate a version of the component. Semantic versioning defines a version number which has three primary segments and an optional fourth segment. The first three segments are defined as MAJOR.MINOR.PATCH with these meanings:

- MAJOR version when you make incompatible API changes

- MINOR version when you add functionality in a backward compatible manner
- PATCH version when you make backward compatible bug fixes

 **Note:**

For background on semantic versioning, visit <https://semver.org>.

When defining a version number for your Web Component, you must define all three of these core segments:

1.0.0

Additionally after the PATCH version, you can append an optional extension segment which can consist of two additional pieces of information:

- A segment delimited by a leading hyphen which allows you to assign a pre-release version
- A purely informational segment preceded by a plus sign (+) that you might use to hold a GIT commit hash or other control information. This segment plays no part in the comparison of component versions.

Here's an example of a fully-defined version number for a pre-release version of a component:

1.0.0-beta.1+332

In this case beta.1 is a pre-release indicator and 332 is a build number.

The change in version number for a Web Component should indicate to consumers the risk level of consuming that new version. Consumers should know that they can drop in a new MINOR or PATCH release of your components without needing to revise their code. If you make changes to the Web Component source code that forces the consuming application to do more than just refresh the Web Component's directory or change a CDN reference, then you should revise the MAJOR version to indicate this.

The Web Component metadata file `component.json` lets you define the supported version of Oracle JET that the component can work with. This is the `jetVersion` attribute which can be set to a specific version or version range, as defined by npm-semver. For example, the `jetVersion` attribute will be set to one of the following:

- Preferred: All MINOR and PATCH versions from the specified version forward until there is a change in MAJOR release number. For example: "`jetVersion: '^8.1.0'`", which indicates support for that release and all subsequent MINOR and PATCH versions, up to (but not including) the next MAJOR release and it is equivalent to "`>=8.1.0 <9.0.0`".
- The exact semantic version of exact version of Oracle JET that the Web Component supports. For example: "`jetVersion : '8.1.0'`", which implies that this Web Component supports only Oracle JET 8.1.0.
- All PATCH versions of Oracle JET within a specific MAJOR.MINOR release. For example: "`jetVersion : '8.0.x'`", which implies that this Web Component

supports every release of JET between JET 8.0.0 and JET 8.1.0 (but specifically not JET 8.1.0 itself).

- A specific range of Oracle JET versions. For example: "jetVersion" : "8.0.0 - 8.1.0", which implies that this Web Component supports every release of JET between JET 8.0.0 and JET 8.1.0 inclusive (so not including, for example, JET 8.2.0 or JET 7.0.0).

 **Tip:**

The Oracle JET recommended format is the first case defined similar to "⁸8.1.0". Given that JET itself follows the semantic versioning rules, changes that occur in MINOR or PATCH versions ought not break your Web Component source code. This also means that you don't have to release an update to all your Web Components for every MINOR release of Oracle JET (unless you choose to make use of a new feature).

 **Note:**

For background on npm-semver, visit npm documentation at this web site <https://docs.npmjs.com/misc/semver>.

In the case of JET Packs, you should pay attention to the dependencies attribute in the `component.json` file located in the pack. You should strive to require the various components bundled into the pack to be consumed together, and as such, you should define such dependencies with absolute version matches rather than version ranges to indicate this. For example, in this case the `demo-memory-game` component at version 1.0.2 will only expect to host a `demo-memory-card` at exactly version 1.0.2.

```
{  
  "name": "demo-memory-game",  
  "version": "1.0.2",  
  "type": "pack",  
  "displayName": "JET Memory Game",  
  "description": "A memory game element with a configurable card set  
and attempt counter.",  
  "dependencies": {  
    "demo-memory-card": "1.0.2"  
  }  
}
```

Create Web Components

Oracle JET supports a variety of custom Web Component component types. You can create standalone Web Components or you can create sets of Web Components that you intend to be used together and you can then assemble those in a JET Pack, or pack component type. You can enhance JET Packs by creating resource components when you have re-usable libraries of assets that are themselves not

specifically UI components. And, if you need to reference third-party code in a standalone component, you can create reference components to define pointers to that code.

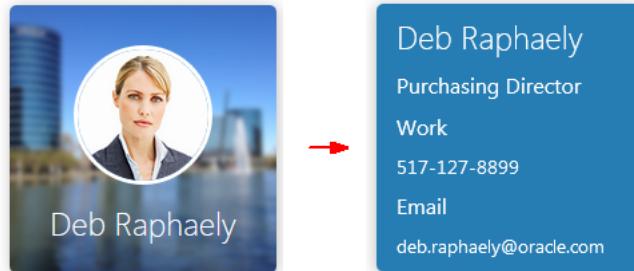
Topics

- [Create Standalone Web Components](#)
- [Create JET Packs](#)
- [Create Resource Components for JET Packs](#)
- [Create Reference Components for Web Components](#)

Create Standalone Web Components

Use the Oracle JET command-line interface (CLI) to create an Oracle JET Web Component template implemented as JavaScript or TypeScript that you can populate with content. If you're not using the tooling, you can add the Web Component files and folders manually to your Oracle JET application.

The following image shows a simple Web Component named `demo-card` that displays contact cards with the contact's name and image if available. When the user selects the card, the content flips to show additional detail about the contact.



The procedure below lists the high level steps to create a Web Component using this `demo-card` component as an example. Portions of the code are omitted for the sake of brevity, but you can find the complete example at [Web Component - Basic](#). You can also download the demo files by clicking the download button (in the cookbook.

Before you begin:

- Familiarize yourself with the steps to install the Oracle JET CLI, see [Install Oracle JET Tooling](#)
- Familiarize yourself with the steps to add a Web Component to an Oracle JET application using the Oracle JET CLI, see [Understanding the Web Application Workflow](#) or [Create a Hybrid Mobile Application Using the Oracle JET Command-Line Interface](#)
- Familiarize yourself with the list of reserved names for a Web Component that are not available for use, see [valid custom element name](#)
- Familiarize yourself with the list of existing Web Component properties, events, and methods, see [HTMLElement properties, event listeners, and methods](#)

- Familiarize yourself with the list of global attributes and events, see [Global attributes](#)

To create a Web Component:

1. Determine a name for your Web Component.

The Web Component specification restricts custom element names as follows:

- Names must contain a hyphen.
- Names must start with a lowercase ASCII letter.
- Names must not contain any uppercase ASCII letters.
- Names should use a unique prefix to reduce the risk of a naming collision with other components.

A good pattern is to use your organization's name as the first segment of the component name, for example, `org-component-name`. Names must not start with the prefix `oj-` or `ns-`, which correspond to the root of the reserved `oj` and `ns` namespaces.

- Names must not be any of the reserved names. Oracle JET also reserves the `oj` and the `ns` namespace and prefixes.

For example, use `demo-card` to duplicate the contact card example.

2. Determine where to place your Web Component, using one of the following options.

- Add the Web Component to an existing Oracle JET application that you created with the Oracle JET CLI.

If you use this method, you'll use the CLI to create a Web Component template that contains the folders and files you'll need to store the Web Component's content.

- Manually add the Web Component to an existing Oracle JET application that doesn't use the Oracle JET CLI.

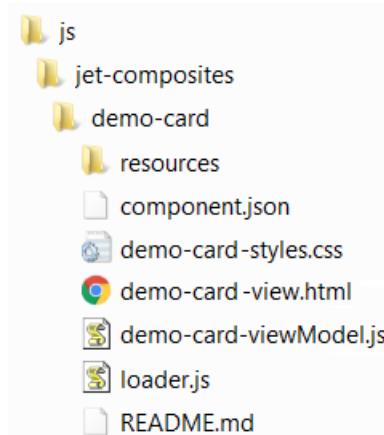
If you use this method, you'll create the folders and files manually to store the Web Component's content.

3. Depending upon the choice you made in the previous step, perform one of the following tasks to create the Web Component.

- If you used the Oracle JET CLI to create an application, then in the application's top level directory, enter the following command at a terminal prompt to generate the Web Component template:

```
ojet create component component-name
```

For example, enter `ojet create component demo-card` to create a Web Component named `demo-card` in a base application created with JavaScript. The command will add `jet-composites/demo-card` to the application's `js` folder and files containing stub content for the Web Component.



The base application's implementation of JavaScript or TypeScript determines the folder and implementation of the Web Component. If your base application has a TypeScript implementation, then the `ts` folder, not the `js` folder, contains the stub contents of the Web Component.

- If you're not using the Oracle JET CLI, create a `jet-composites` folder in your application's `js` folder or `ts` folder, and add folders containing the name of each Web Component you will create.

For the `demo-card` example, create the `jet-composites` folder and add a `demo-card` folder to it. You'll create the individual Web Component files in the remaining steps.

4. Determine the properties, methods, and events that your Web Component will support and add them to the `component.json` file in the Web Component's root folder, creating the file if needed.

The name of the Web Component properties, event listeners, and methods should avoid collision with the existing HTML element properties, event listeners, and methods. Additionally, the property name `slot` should not be used. Also, you must not re-define the global attributes and events.

The `demo-card` example defines properties for the Web Component and the contact's full name, employee image, title, work number, and email address. The required properties are highlighted in bold.

```
{  
  "name": "demo-card",  
  "description": "A card element that can display an avatar or initials on one side and employee information on the other.",  
  "version": "1.0.2",  
  "displayName": "Demo Card",  
  "jetVersion": ">=6.0.0 <9.2.0",  
  "properties": {  
    "name": {  
      "description": "The employee's full name.",  
      "type": "string"  
    },  
    "avatar": {  
      "description": "The url of the employee's image.",  
      "type": "string"  
    }  
  }  
}
```

```
},
"workTitle": {
  "description": "The employee's job title.",
  "type": "string"
},
"workNumber": {
  "description": "The employee's work number.",
  "type": "number"
},
"email": {
  "description": "The employee's email.",
  "type": "string"
}
}
```

This basic demo-card example only defines properties for the Web Component. You can also add metadata that defines methods and events as shown below. The metadata lists the name of the method or event and supported parameters.

```
{
  "properties": {
    ... contents omitted
  },
  "methods": {
    "flipCard": {
      "description": "Method to toggle flipping a card"
    },
    "enableFlip": {
      "description": "Enables or disables the ability to flip a
card.",
      "params": [
        {
          "name": "bEnable",
          "description": "True to enable card flipping and false
otherwise.",
          "type": "boolean"
        }
      ]
    },
    ...
  },
  "events": {
    "cardClick": {
      "description": "Triggered when a card is clicked and
contains the value of the clicked card..",
      "bubbles": true,
      "detail": {
        "value": {
          "description": "The value of the card.",
          "type": "string"
        }
      }
    }
  }
}
```

```

        }
    }
}
```

5. If your Web Component contains a ViewModel, add its definition to `web-component-name-viewModel.js` in the Web Component's root folder, creating the file if needed.

The code sample below shows the ViewModel for the demo-card Web Component. Comments describe the purpose, parameters, and return value of each function.

```

define(['knockout', 'ojs/ojknockout'],
    function(ko) {
        function model(context) {
            var self = this;
            self.initials = null;
            self.workFormatted = null;
            var element = context.element;

            /**
             * Formats a 10 digit number as a phone number.
             * @param {number} number The number to format
             * @return {string}           The formatted phone number
             */
            var formatPhoneNumber = function(number) {
                return Number(number).toString().replace(/(\d{3}) (\d{3}) (\d{4})/, '$1-$2-$3');
            }

            if (context.properties.name) {
                var initials = context.properties.name.match(/\b\w/g);
                self.initials = (initials.shift() +
                    initials.pop()).toUpperCase();
            }
            if (context.properties.workNumber)
                self.workFormatted =
                    formatPhoneNumber(context.properties.workNumber);

            /**
             * Flips a card
             * @param {MouseEvent} event The click event
             */
            self.flipCard = function(event) {
                if (event.type === 'click' || (event.type ===
                    'keypress' && event.keyCode === 13)) {
                    // It's better to look for View elements using
                    // instead of by DOM node order which isn't
                    // guaranteed.
                    $(element).children('.demo-card-flip-
                    container').toggleClass('demo-card-flipped');
                }
            };
        }
    }
}
```

```

        return model;
    }
)

```

- In the Web Component's root folder, add the View definition to `web-component-name-view.html`, creating the file if needed.

The View for the demo-card Web Component is shown below. Any property defined in the component's metadata is accessed using the `$properties` property of the View binding context.

```

<div tabindex="0" role="group" class="demo-card-flip-container"
  on-click="[[flipCard]]" on-keypress="[[flipCard]]" :aria-
  label="[[ $properties.name + ' Press Enter for
  more info.' ]]">
  <div class="demo-card-front-side">
    <oj-avatar class="demo-card-avatar" role="img" size="lg"
    initials="[[initials]]"
    src="[[ $properties.avatar }}" :aria-label="['Avatar of ' +
    $properties.name]]">
      </oj-avatar>
    <h2>
      <oj-bind-text value="[[ $properties.name ]]"></oj-bind-text>
    </h2>
  </div>

  <div class="demo-card-back-side">
    <div class="demo-card-inner-back-side">
      <h2>
        <oj-bind-text value="[[ $properties.name ]]"></oj-bind-
        text>
      </h2>
      <h5>
        <oj-bind-text value="[[ $properties.workTitle ]]"></oj-bind-
        text>
      </h5>
      <oj-bind-if test="[[ $properties.workNumber != null ]]">
        <h5>Work</h5>
        <span class="demo-card-text"><oj-bind-text
        value="[[workFormatted]]"></oj-bind-text></span>
      </oj-bind-if>
      <oj-bind-if test="[[ $properties.email != null ]]">
        <h5>Email</h5>
        <span class="demo-card-text"><oj-bind-text
        value="[[ $properties.email ]]"></oj-bind-text></span>
      </oj-bind-if>
    </div>
  </div>
</div>

```

For accessibility, the View's role is defined as `group`, with `aria-label` specified for the contact's name. In general, follow the same accessibility guidelines for

the Web Component View markup that you would anywhere else within the application.

7. If you're not using the Oracle JET CLI, create the `loader.js` RequireJS module and place it in the Web Component's root folder.

The `loader.js` module defines the Web Component dependencies and registers the component's `tagName`, `demo-card` in this example.

```
define(['ojs/ojcomposite', 'text!./demo-card-view.html', './demo-card-viewModel',
        'text!./component.json', 'css!./demo-card-styles'],
       function(Composite, view, viewModel, metadata) {
    Composite.register('demo-card', {
        view: view,
        viewModel: viewModel,
        metadata: JSON.parse(metadata)
    });
});
```

In this example, the CSS is loaded through a RequireJS plugin (`css!./demo-card-styles`), and you do not need to pass it explicitly in `Composite.register()`.

8. Configure any custom styling that your Web Component will use.

- If you only have a few styles, add them to `web-component-name-styles.css` file in the Web Component's root folder, creating the file if needed.

For example, the `demo-card` Web Component defines styles for the demo card's display, width, height, margin, padding, and more. It also defines the classes that will be used when the user clicks a contact card. A portion of the CSS is shown below.

```
/* This is to prevent the flash of unstyled content before the
Web Component properties have been setup. */
demo-card:not(.oj-complete) {
    visibility: hidden;
}

demo-card {
    display: block;
    width: 200px;
    height: 200px;
    perspective: 800px;
    margin: 10px;
    box-sizing: border-box;
    cursor: pointer;
}

demo-card h2,
demo-card h5,
demo-card a,
demo-card .demo-card-avatar {
    color: #fff;
    padding: 0;
```

```
}
```

... remaining contents omitted

- If you used the Oracle JET tooling to create your application and want to use Sass to generate your CSS:
 - a. If needed, at a terminal prompt in your application's top level directory, type the following command to add node-sass to your application: `ojet add sass`.
 - b. Create `web-component-name-styles.scss` and place it in the Web Component's top level folder.
 - c. Edit `web-component-name-styles.scss` with any valid SCSS syntax and save the file.

In this example, a variable defines the demo card size:

```
$demo-card-size: 200px;

/* This is to prevent the flash of unstyled content before
the Web Component properties have been setup. */
demo-card:not(.oj-complete) {
    visibility: hidden;
}

demo-card {
    display: block;
    width: $demo-card-size;
    height: $demo-card-size;
    perspective: 800px;
    margin: 10px;
    box-sizing: border-box;
    cursor: pointer;
}

demo-card h2,
demo-card h5,
demo-card a,
demo-card .demo-card-avatar {
    color: #fff;
    padding: 0;
}
... remaining contents omitted
```

- d. To compile Sass, at a terminal prompt type `ojet build` or `ojet serve` with the `--sass` flag and application-specific options.

```
ojet build|serve [options] --sass
```

`ojet build --sass` will compile your application and generate `web-component-name-styles.css` and `web-component-name-styles.css.map` files in the default platform's folder. For a web application, the command will place the CSS in `web/js/js-composites/web-component-name`.

`ojet serve --sass` will also compile your application but will display the web application in a running browser with livereload enabled. If you save a change to `web-component-name-styles.scss` in the application's `src/js/jet-composites/web-component-name` folder, Oracle JET will compile Sass again and refresh the display.

Tip:

For help with `ojet` command syntax, type `ojet help` at a terminal prompt.

9. If you want to add documentation for your Web Component, add content to `README.md` in your Web Component's root folder, creating the file if needed.

Your `README.md` file should include an overview of your component with well-formatted examples. Include any additional information that you want to provide to your component's consumers. The recommended standard for `README` file format is markdown.

For help with markdown, refer to <https://guides.github.com/features/mastering-markdown/>.

For the complete code of the `demo-card` Web Component CSS styles, see `demo-card-styles.css` in the [Web Component - Basic](#) cookbook sample.

For information on adding Web Component metadata that defines methods and events, see the [Web Component - Events](#) cookbook sample.

Create JET Packs

Create JET Packs to simplify project management for consumers who might pick up a component that is related to one or more components. You may require specific versions of the referenced components for individual JET Packs.

Fundamentally, the JET Pack is a library of related Web Components that does not directly include those assets, but is as an index to a particular versioned stripe of components.

Note:

Note there is one exception to the pack as a reference mechanism for related components. A pack might include one or more RequireJS bundle files which package up optimized forms of the component set into a small number of physical downloads. This, however, is always in addition to the actual components being available as independent entities in Oracle Component Exchange.

The components referenced by the JET Pack are intended to be used together and their usage is restricted by individual component version. Thus the JET Pack that you create will tie very specific versions of each component into a relationship with very specific, fixed versions of the other components in the same set. Thus a JET Pack itself has a "version stripe" which determines the specific components that users import into their applications. Since the version number of individual components may

vary, the JET Pack guarantees the consumer associates their application with the version of the pack as a whole, and not with the individual components contained by the pack.

For details about versioning JET Packs, see [Version Numbering Standards](#).

1. Create the JET Pack using the JET tooling from the root folder of your application.

```
ojet create pack my-pack
```

Consider the pack name carefully, as the name will determine the prefix to any components within that pack.

The tooling adds the folder structure with a single template `component.json` file that you will need to modify:

```
/(working folder)
/src
/js
/jet-composites
/my-pack
component.json
```

2. Create the components that you want to bundle with the JET Pack by using the JET tooling from the root folder of your application. The component name that you specify must be unique within the pack.

```
ojet create component my-widget-1 --pack=my-pack
```

The tooling nests the component folder `/my-widget-1` under the `my-pack` root folder and the new component files resemble those created for a standalone Web Component.

```
/(working folder)
/src
/js
/jet-composites
/my-pack
component.json
/my-widget-1
/resources
/nls
    my-widget-1-strings.js
component.json
loader.js
README.md
my-widget-1-viewModel.js
my-widget-1-styles.css
my-widget-1-view.html
```

Note the following about the created component files.

- component.js specifies the **name** of the component as **my-widget-1** and the **pack** is set to **my-pack**, providing the complete definition of the component's identity.
 - loader.js registers the HTML tag for the new component as **my-pack-my-widget-1**. This is the full name of the component, which is a concatenation of the pack name and the component name. When you need to refer to this component in a dependency from another component's metadata or in HTML, you will use the full name.
3. Optionally, for any Resource components that you created, as described in [Create Resource Components for JET Packs](#), add the component's working folder with its own component.json file to the pack file structure.

The tooling nests the component folder /my-widget-1 under the my-pack root folder and the new component files resemble those created for a standalone Web Component.

```
/(working folder)
/src
  /js
    /jet-composites
      /my-pack
        component.json
        /my-widget-1
          /resources
            /nls
              my-widget-1-strings.js
            component.json
            loader.js
            README.md
            my-widget-1-viewModel.js
            my-widget-1-styles.css
            my-widget-1-view.html
        /my-resource-component-1
          component.json
          /converters
            file1.js
            ...
          /resources
            /nls
              string-file1.js
            /root
              strings-file.js
          /validators
            file1.js
            ...
...
```

4. Optionally, generate any required bundled for desired components of the pack. Refer to RequireJS documentation for details at the <https://requirejs.org> web site.

 **Tip:**

You can use RequireJS to create optimized bundles of the pack components, so that rather than each component being downloaded separately by the consuming application at runtime, instead a single JavaScript file can be downloaded that contains multiple components. It's a good idea to use this facility if you have sets of components that are almost always used together. A pack can have any number of bundles (or none at all) in order to group the available components as required. Be aware that not every component in the pack has to be included in one of the bundles and that each component can only be part of one bundle.

5. Use a text editor to modify the `component.json` file in the pack folder root similar to the following sample, to identify pack dependencies and optional bundles. Added components must be associated by their full name and a specific version.

```
{  
    "name": "my-pack",  
    "version": "1.0.0",  
    "type": "pack",  
    "displayName": "My JET Pack",  
    "description": "An example JET Pack",  
    "dependencies": {  
        "my-pack-my-widget-1": "1.0.0",  
        ...  
    },  
    "bundles": {  
        "my-pack/my-bundle": [  
            "my-pack/my-bundle-file1/loader",  
            ...  
        ]  
    },  
    "extension": {  
        "catalog": {  
            "coverImage": "coverimage.png"  
        }  
    }  
}
```

Your pack component's `component.json` file must contain the following unique definitions:

- **name** is the name of the JET Pack has to be unique, and should be defined with the namespace relevant to your group. This name will be prepended to create the full name of individual components of the pack.
- **version** defines the exact version number of the pack, not a SemVer range.

 **Note:**

Changes in version number with a given release of a pack should reflect the most significant change in the pack contents. For example, if the pack contained two components and as part of a release one of these had a Patch level change and the other a Major version change then the pack version number change should also be a Major version change. There is no requirement for the actual version number of the pack to match the version number(s) of any of its referenced components. For more information see, [Version Numbering Standards](#).

- **type** must be set to `pack`.
 - **displayName** is the name of the pack component that you want displayed in Oracle Component Exchange. Set this to something readable but not too long.
 - **description** is the description that you want displayed in Oracle Component Exchange. For example, use this to explain how the pack is intended to be used.
 - **dependencies** defines the set of components that make up the pack, specified by the component full name (a concatenation of pack name and component name). Note that exact version numbers are used here, not SemVer ranges. It's important that you manage revisions of dependency version numbers to reflect changes to the referenced component's version and also to specify part of the path to reach the components within the pack.
 - **bundles** defines the available bundles (optional) and the contents of each. Note how both the bundle name and the contents of that bundle are defined with the pack name prefix as this is the RequireJS path that is needed to map those artifacts.
 - **catalog** defines the working metadata for Oracle Component Exchange, including a cover image in this case.
6. Use a text editor to modify the `component.json` file in the component folder root similar to the following sample, to identify the pack relationship. Components must be identified by a unique name (without the pack prefix) and a specific version.

```
{  
    "name": "my-widget-1",  
    "pack": "my-pack",  
    "displayName": "My Web Component",  
    "description": "Fully featured component",  
    "version": "1.0.0",  
    "jetVersion": "^8.1.0",  
    "dependencies": {  
        "my-widget-file1": "^1.0.0",  
        ...  
    },  
    ...  
}
```

Your pack component's component.json file must contain the following unique definitions:

- **name** is the name of the component has to be unique. This name will be prepended with the pack name to create the component full name. For example,
 - **pack** is the name of the JET Pack that the component is a part of.
 - **displayName** is the name of the component that you want displayed in Oracle Component Exchange.
 - **description** is the description that you want displayed in Oracle Component Exchange. For example, use this to explain the role of the components in the pack as they are intended to be used.
 - **version** defines the exact version number of the component, not a SemVer range.
 - **jetVersion** defines the compatible version(s) of Oracle JET, specified by a semantic version (SemVer) range. It's important that you manage revisions of this version number to inform consumers of the compatibility of a given change and also to specify part of the path to reach the components within the pack. For more information see, [Version Numbering Standards](#).
 - **dependencies** defines the set of libraries and other components that make up the component within the pack. In the case where a dependent component is also listed as a member of the JET Pack, specify components here by their full name (a concatenation of pack name and component name). For example, my-pack-my-component. Note that SemVer ranges are allowed. It's important that the range selected for a component within a particular JET Pack version overlaps with the members of that stripe.
7. Optionally, create a readme file in the root of your working folder. This should be defined as a plain text file called README.txt (or README.md when using markdown format).
 8. Optionally, create a cover image in the root of your working folder to display the component on Oracle Exchange. The file name can be the same as the name attribute in the component.json file.
 9. Use the JET tooling to create a zip archive of the JET Pack working folder when you want to upload the component to Oracle Component Exchange, as described in [Package Web Components](#).
 10. Support consuming the JET Pack in Oracle Visual Builder projects by uploading the component to Oracle Component Exchange, as describe in [Publish Web Components to Oracle Component Exchange](#).

Create Resource Components for JET Packs

Create a resource component when you want to reuse assets across Web Components that you assemble into JET Packs. The resource component can be reused by multiple JET Packs.

When dealing with complex sets of components you may find that makes sense to share certain assets between multiple components. In such cases, the components can all be included into a single JET Pack and then a resource component can be added to the pack in order to hold the shared assets. There is no constraint on what can be stored in a pack, typically it may expose shared JavaScript, CSS, and JSON

files and images. Note that third party libraries should generally be referenced from a reference component and should not included into a resource component.

You don't need any tools to create the resource component. You will need to create a folder in a convenient location. This folder will ultimately be zipped to create the distributable resource component. Internally this folder can then hold any content in any structure that you desire.

To create a resource component:

1. Create the working folder and populate it with the desired content. You can add content in any structure desired, with the exception of NLS content for translation bundles. In the case of NLS content, preserve the typical JET folder structure; this is important if your resource component is going to include such bundles.

```
/(working folder)
  /converters
    phoneConverter.js
    phoneConverterFactory.js
  /resources
    /nls
      oj-ext-strings.js
    /root
      oj-ext-strings.js
  /phone
    countryCodes.json
  /validators
    emailValidator.js
    emailValidatorFactory.js
    phoneValidator.js
    phoneValidatorFactory.js
    urlValidator.js
    urlValidatorFactory.js
```

In this sample notice how the `/resources/nls` folder structure for translation bundles is preserved according to the folder structured of the application generated by JET tooling.

2. Use a text editor to create a `component.json` file in the folder root similar to the following sample, which defines the resource utils for the JET Pack `oj-ext`.

```
{
  "name": "utils",
  "pack": "oj-ext",
  "displayName": "Oracle Jet Extended Utilities",
  "description": "A set of reusable utility classes used by the Oracle JET extended component set and available for general use. Includes various reusable validators",
  "license": "https://opensource.org/licenses/UPL",
  "type": "resource",
  "version": "2.0.2",
  "jetVersion": ">=6.0.0 <8.0.0",
  "publicModules": [
    "validators/emailValidatorFactory",
    "validators/urlValidatorFactory"
  ]}
```

```
"extension": {  
    "catalog": {  
        "category": "Resources",  
        "coverImage": "cca-resource-folder.svg"  
    }  
}
```

Your resource component's `component.json` file must contain the following unique definitions:

- **name** is the name of the resource component has to be unique, and should be defined with the namespace relevant to your group.
- **pack** is the name of the JET Pack containing the resource component.
- **displayName** is the name of the resource component as displayed in Oracle Component Exchange. Set this to something readable but not too long.
- **description** is the description that you want displayed in Oracle Component Exchange. For example, use this to explain the available assets provided by the component.
- **type** must be set to `resource`.
- **version** defines the semantic version (SemVer) of the resource component as a whole. It's important that you manage revisions of this version number to inform consumers of the compatibility of a given change.

Note:

Changes to the resource component version should roll up all of the changes within the resource component, which might not be restricted to changes only in `.js` files. A change to a CSS selector defined in a shared `.css` file can trigger a major version change when it forces consumers to make changes to their downstream uses of that selector. For more information see, [Version Numbering Standards](#).

- **jetVersion** defines the supported Oracle JET version range using SemVer notation. This is *optional* and depends on the nature of what you include into the resource component. If the component contains JavaScript code and any of that code makes reference to Oracle JET APIs, then you really should include a JET version range in that case. For more information about specifying semantic versions see [Version Numbering Standards](#).
- **publicModules** lists entry points within the resource component that you consider as being public and intend to be consumed by any component that depends on this component. Any API not listed in the array is considered to be pack-private and therefore can only be used by components within the same pack namespace, but may not be used externally.
- **catalog** defines the working metadata for Oracle Component Exchange, including a cover image in this case.

3. Optionally, create a readme file in the root of your working folder. A readme can be used to document the assets of the resource. This should be defined as a plain text file called `README.txt` (or `README.md` when using markdown format).

 **Tip:**

Take care to explain the state of the assets. For example, you might choose to include utility classes in the resource component that are deemed public and can safely be used by external consumers (for example, code outside of the JET Pack that the component belongs to). However, you may want to document other assets as private to the pack itself.

4. Optionally, create a change log file in the root of your working folder. The change log can detail significant changes to the pack over time and is strongly recommended. This should be defined as a text file called `CHANGELOG.txt` (or `CHANGELOG.md` when using markdown format).
5. Optionally, include a License file in the root of your working folder.
6. Optionally, create a cover image in the root of your working folder to display the component on Oracle Exchange. Using the third party logo can be helpful here to identify the usage. The file name can be the same as the name attribute in the `component.json` file.
7. Create a zip archive of the working folder when you want to upload the component to Oracle Component Exchange. Oracle recommends using the format `<fullName>-<version>.zip` for the archive file name. For example, `oj-ext-utils-2.0.2.zip`.

For information about using the resource component in a JET Pack, see [Create JET Packs](#).

Create Reference Components for Web Components

Create a reference component when you need to obtain a pointer to third-party libraries for use by Web Components.

Sometimes your JET Web Components need to use third party libraries to function and although it is possible to embed such libraries within the component itself, or within a resource component, it generally better to reference a shared copy of the library by defining a reference component.

Create the Reference Component

You don't need any tools to create the reference component. You will need to create a folder in a convenient location where you will define metadata for the reference component in the `component.json` file. This folder will ultimately be zipped to create the distributable reference component.

Reference components are generally standalone, so the `component.json` file you create must not be contained within a JET Pack.

To create a reference component:

1. Create the working folder and use a text editor to create a `component.json` file in the folder root similar to the following sample, which references the `moment.js` library.

```
{  
    "name": "oj-ref-moment",  
    "displayName": "Moment library",  
    "description": "Supplies reference information for moment.js used  
    to parse, validate, manipulate, and display dates and times in  
    JavaScript",  
    "license": "https://opensource.org/licenses/MIT",  
    "type": "reference",  
    "package": "moment",  
    "version": "2.24.0",  
    "paths": {  
        "npm": {  
            "debug": "moment",  
            "min": "min/moment.min"  
        },  
        "cdn": {  
            "debug": "https://static.oracle.com/cdn/jet/packs/3rdparty/  
            moment/2.24.0/moment.min",  
            "min": "https://static.oracle.com/cdn/jet/packs/3rdparty/  
            moment/2.24.0/moment.min"  
        }  
    },  
    "extension": {  
        "catalog": {  
            "category": "Third Party",  
            "tags": [  
                "momentjs"  
            ],  
            "coverImage": "coverImage.png"  
        }  
    }  
}
```

Your reference component's `component.json` file must contain the following unique definitions:

- **name** is the name of the reference component has to be unique, and should be defined with the namespace relevant to your group.
- **displayName** is the name of the resource component as displayed in Oracle Component Exchange. Set this to something readable but not too long.
- **description** is the description that you want displayed in Oracle Component Exchange. For example, use this to explain the function of the third party library.
- **license** comes from the third party library itself and must be specified.
- **type** must be set to `reference`.
- **package** defines the npm package name for the library. This will also be used as the name of the associated RequireJS path that will point to the library and so will be used by components that depend on this reference.

- **version** should reflect the version of the third party library that this reference component defines. If you need to be able to reference multiple versions of a given library then you will need multiple versions of the reference component in order to map each one.
 - **paths** defines the CDN locations for this library. See below for more information about getting access to the Oracle CDN.
 - **min** points to the optimal version of the library to consume. The debug path can point to a debug version or just the min version as here.
 - **catalog** defines the working metadata for Oracle Component Exchange including a cover image in this case.
2. Optionally, create readme file in the root of your working folder. A readme can be used to point at the third party component web site for reference. This should be defined as a plain text file called `README.txt` (or `README.md` when using markdown format).
 3. Optionally, create a cover image in the root of your working folder to display the component on Oracle Exchange. Using the third party logo can be helpful here to identify the usage. The file name can be the same as the name attribute in the `component.json` file.
 4. Create a zip archive of the working folder when you want to upload the component to Oracle Component Exchange. Oracle recommends using the format `<fullName>-<version>.zip` for the archive file name. For example, `oj-ref-moment-2.24.0.zip`.
 5. Support consuming the reference component in Oracle Visual Builder projects by uploading the component to a CDN. See below for more details.

Consume the Reference Component

When your Web Components need access to the third party library defined in one of these reference components, you use the dependency attribute metadata in the `component.json` to point to either an explicit version of the reference component or you can specify a semantic range. Here's a simple example of a component that consumes two such reference components at specific versions:

```
{  
  "name": "calendar",  
  "pack": "oj-sample",  
  "displayName": "JET Calendar",  
  "description": "FullCalendar wrapper with Accessibility added.",  
  "version": "1.0.2",  
  "jetVersion": "^8.0.0",  
  "dependencies": {  
    "oj-ref-moment": "2.24.0",  
    "oj-ref-fullcalendar": "3.9.0"  
  },  
  ...  
}
```

When the above component is added to an Oracle JET or Oracle Visual Builder project this dependency information will be used to create the correct RequireJS paths for the third party libraries pointed to be the reference component.

For more information about semantic version usage, see [Version Numbering Standards](#).

Alternatively, when you install a Web Component that depends on a reference component and you use Oracle JET CLI, the tooling will automatically do an npm install for you so that the libraries are local. However, with the same component used in Oracle Visual Builder, a CDN location must be used and therefore the reference component must exist on the CDN in order to be used in Visual Builder.

Test Web Components

Test Oracle JET Web Components using your favorite testing tools for client-side JavaScript applications.

Regardless of the test method you choose, be sure that your tests fully exercise the Web Component's:

- ViewModel (if it exists)

Ideally, your test results should be verifiable via code coverage numbers.

- HTML view

Be sure to include any DOM branches that might be conditionally rendered, and test all slots with and without default content.

- Properties and property values
- Events
- Methods
- Accessibility
- Security

For additional information about testing Oracle JET applications, see [Test Oracle JET Applications](#).

Add Web Components to Your Page

To use an Oracle JET Web Component, you must register the Web Component's loader file in your application and you must also include the Web Component element in the application's HTML. You can add any supporting CSS or files as needed.

1. In the Web Components's root folder, open `component.json` and verify that your version of Oracle JET is compatible with the version specified in `jetVersion`.

For example, the `demo-card` example specifies the following `jetVersion`:

```
"jetVersion": ">=3.0.0 <9.2.0"
```

This indicates that the component is compatible with JET versions greater than or equal to 3.0.0 and less than 9.2.0.

If your version of Oracle JET is lower than the `jetVersion`, you must update your version of Oracle JET before using the component. If your version of Oracle JET is greater than the `jetVersion`, contact the developer to get an updated version of the component.

2. In your application's index.html or main application HTML, add the component and any associated property declarations.

For example, to use the demo-card standalone Web Component, add it to your index.html file and add declarations for name, avatar, work-title, work-number, email, and background-image.

```
<div id="composite-container" class="oj-flex oj-sm-flex-items-initial">
  <oj-bind-for-each data="[[employees]]">
    <template>
      <demo-card class="oj-flex-item"
        name="[[${current.data.name}]]"
        avatar="[[${current.data.avatar}]]"
        work-title="[[${current.data.title}]]"
        work-number="[[${current.data.work}]]"
        email="[[${current.data.email}]]">
        </demo-card>
      </template>
    </oj-bind-for-each>
  </div>
```

In the case of components within a JET Pack, the HTML tag name is the component full name. The full name of a pack's member component is always a concatenation of the pack name and the component name, as specified by the **dependencies** attribute of the pack-level component.json file (located in the pack root folder under jet-composites). For example, a component widget-1 that is a member of the JET Pack my-pack, has the following full name that you can reference as the HTML tag name.

my-pack-widget-1

Note that the framework maps the attribute names in the markup to the component's properties.

- Attribute names are converted to lowercase. For example, a workTitle attribute will map to a worktitle property.
- Attribute names with dashes are converted to camelCase by capitalizing the first character after a dash and then removing the dashes. For example, the work-title attribute will map to a workTitle property.

You can access the mapped properties programmatically as shown in the following markup:

```
<h5><oj-bind-text value="[[properties.workTitle]]"></oj-bind-text></h5>
```

3. In your application's ViewModel, set values for the properties you declared in the previous step and add the component's loader file to the list of application dependencies.

For example, the following code adds the ViewModel to the application's RequireJS bootstrap file. The code also defines the `jet-composites/demo-card/loader` dependency.

```
require(['ojs/ojbootstrap', 'knockout', 'ojs/ojknockout', 'demo-
card/loader'],
function(Bootstrap, ko) {
    function model() {
        var self = this;
        self.employees = [
            {
                name: 'Deb Raphaely',
                avatar: 'images/composites/debraphaely.png',
                title: 'Purchasing Director',
                work: 5171278899,
                email: 'deb.raphaely@oracle.com'
            },
            {
                name: 'Adam Fripp',
                avatar: null,
                title: 'IT Manager',
                work: 6501232234,
                email: 'adam.fripp@oracle.com'
            }
        ];
    }

    Bootstrap.whenDocumentReady().then(function()
    {
        ko.applyBindings(new model(),
document.getElementById('composite-container'));
    });
});
```

In the case of a JET Pack, you add the loader file for the JET Pack by specifying the path based on the pack root and folder name of the component contained within the pack.

`'my-pack/widget-1/loader'`

4. Add any supporting CSS, folders, and files as needed.

For example, the demo card example defines a background image for the contact card in the application's `demo.css`:

```
#composite-container demo-card .demo-card-front-side {
    background-image: url('images/composites/card-
background_1.png');
}
```

Build Web Components

You can build your Oracle JET Web Component to optimize the files and to generate a minified folder of the component that can be shared with the consumers.

When your Web Component is configured and is ready to be used in different applications, you can build the Web Components of the type: standalone Web Component, JET Pack, and Resource component. Building these components using JET tooling generates a minified content with the optimized component files. This minified version of the component can be easily shared with the consumers for use. For example, you would build the component before publishing it to Oracle Component Exchange. To build the Web Component, use the following command from the root folder of the JET application containing the component:

```
ojet build component my-web-component-name
```

For example, if your Web Component name is demo-card-example, use the following command:

```
ojet build component demo-card-example
```

For a JET Pack, specify the pack name.

```
ojet build component my-pack-name
```

Note that the building individual components within the pack is not supported, and the whole pack must be built at once.

This command creates a `/min` folder in the `web/js/jet-composites/demo-card-example/x.x.x` directory of your Oracle JET web application, where `x.x.x` is the version number of the component. The `/min` folder contains the minified (release) version of your Web Component files.

Reference component do not require minification or bundling and therefore do not need to be built.

When you build Web Components:

- If your JET application contains more than one component, you can build the containing JET application to build and optimize all components together. The `build component` command with the component name provides the capability to build a single component.
- You can optionally use the `--release` flag with the `build` command, but it is not necessary since the `build` command generates both the debug and minified version of the component.
- You can optionally use the `--optimize=none` flags with the `build` command when you want to generate compiled output that is more readable and suitable for debugging. The component's `loader.js` file will contain the minified application source, but content readability is improved, as line breaks and white space will be preserved from the original source.

Package Web Components

You can create a sharable zip file archive of the minified Oracle JET Web Component from the Command-Line Interface.

When you want to share Web Components with other developers, you can create an archive file of the generated output contained in the `jet-composites` subfolder of the application's `/web`. After you build a standalone Web Component or a Resource component, you use the JET tooling to run the `package` command and create a zip file that contains the Web Component compiled and minified source.

```
ojet package component my-web-component-name
```

Similarly, in the case of JET Packs, you cannot create a zip file directly from the file system. It is necessary to use the JET tooling to package JET packs because the output under the `/jet-composites/<packName>` subfolder contains nested component folders and the tooling ensures that each component has its own zip file.

```
ojet package pack my-JET-Pack-name
```

The `package` command packages the component's minified source from the `/web/js/jet-composites` directory and makes it available as a zip file in a `/dist` folder at the root of the containing application. This zip file will contain both the specified component and a minified version of that component in a `/min` subfolder.

Reference component do not require minification or bundling and therefore do not need to be built. You can archive the Reference component by creating a simple zip archive of the component's folder.

The zip archive of the packaged component is suitable to share, for example, on Oracle Component Exchange, as describe in [Publish Web Components to Oracle Component Exchange](#). To help organize components that you want to publish, you can append the relevant component version number to the zip file name. For example, `my-pack-1.0.0.zip`

You can also generate an archive file when you want to upload the component to a CDN. In the CDN case, additional steps are required before you can share the component, as describe in [Upload and Consume Web Components on a CDN](#).

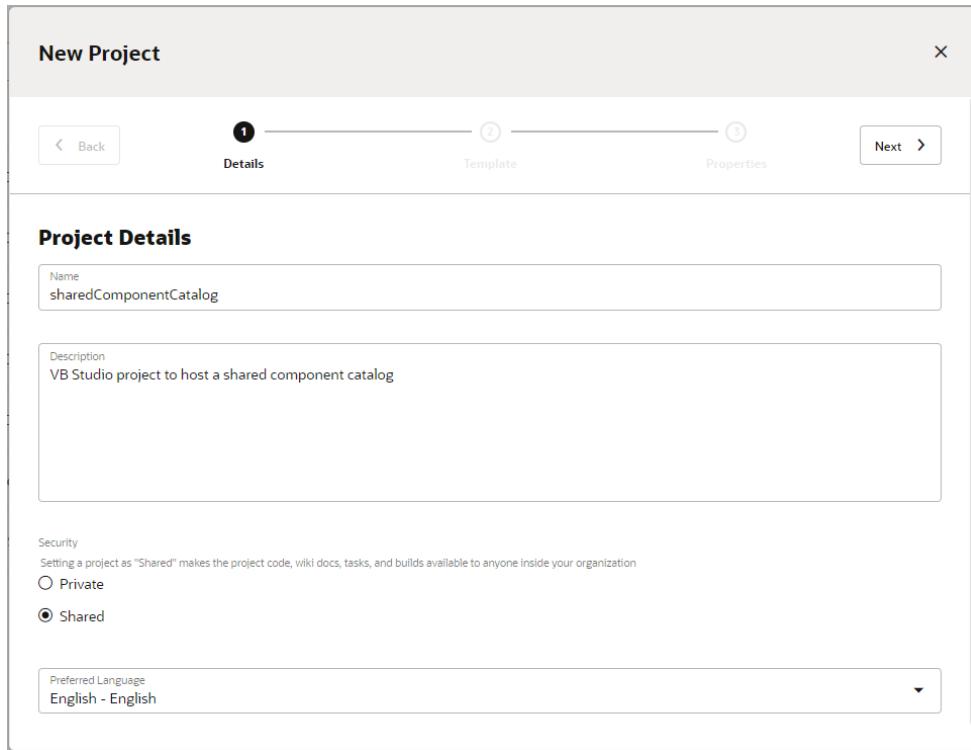
Create a Project to Host a Shared Oracle Component Exchange

When you want to store and share Web Components across machines for re-use by other developers, you can use a Component Exchange that you create in Oracle Visual Builder Studio.

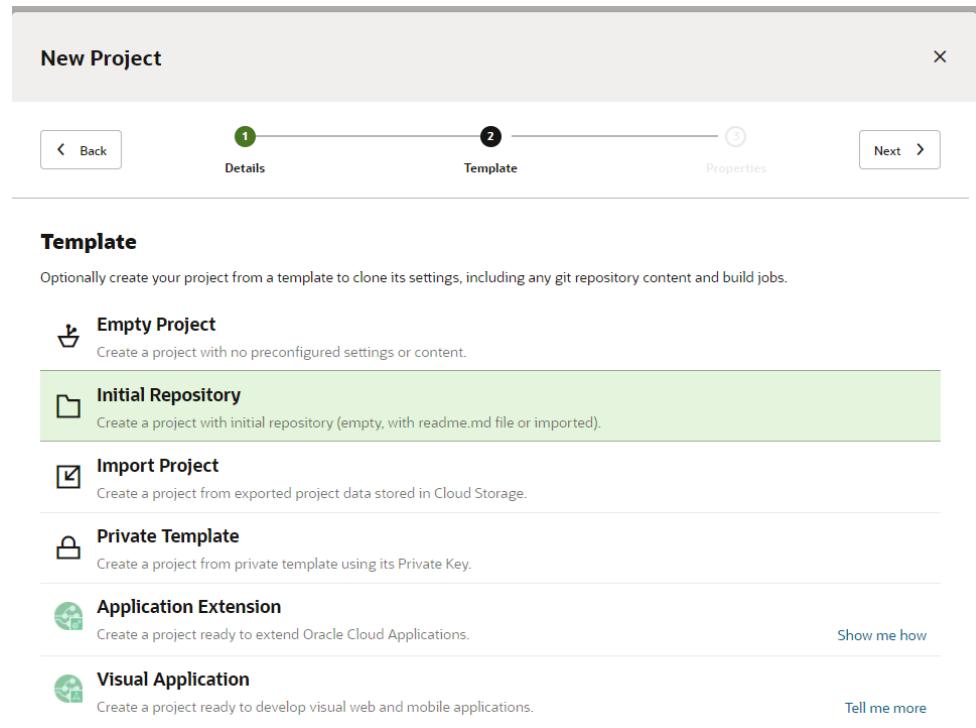
When you want to share Web Components, you generally need to set up a dedicated project to specifically host a Shared Exchange. Although every project within Oracle Visual Builder Studio has a private Component Exchange instance, the Shared Exchange makes uploaded components accessible to other developers. You can use the same project in your Shared Exchange to host the source code repository of your components.

To create the Shared Exchange:

1. Create a new project in Oracle Visual Builder Studio.



2. Select the **Initial Repository** template for the new project.



3. Leave the defaults settings unchanged or, optionally, select the settings for importing your component source code and click **Finish**.

New Project

1 Details 2 Template 3 Properties Back Finish

Project Properties Initial Repository

Initial Repository

Empty Repository

Initialize repository with README file

Import existing repository

Importing repository URL

> Credentials for non-public repos

Wiki Markup
Markdown

4. In the project provisioning screen, verify that **Component Exchange** is one of the services being created.

Visual Builder Studio

sharedComponentCatalog | Project Home

Project sharedComponentCatalog is being provisioned

Provisioning may take up to several minutes.
Please wait until all modules are provisioned.

<input checked="" type="checkbox"/> Agile	<input checked="" type="checkbox"/> Designer	<input checked="" type="checkbox"/> Merge Requests
<input checked="" type="checkbox"/> Build	<input checked="" type="checkbox"/> Docker	<input checked="" type="checkbox"/> Mobile Build
<input checked="" type="checkbox"/> Code	<input checked="" type="checkbox"/> Environments	<input checked="" type="checkbox"/> Project
<input checked="" type="checkbox"/> Component Exchange	<input checked="" type="checkbox"/> Issues	<input checked="" type="checkbox"/> Wiki
<input checked="" type="checkbox"/> Deploy	<input checked="" type="checkbox"/> Maven	

5. Share the project with all users who will need to access published components by adding them as members to the project. They will then use their own user name and password to access the Shared Component Exchange.

Once the project is created you can obtain the correct URL for use with the Oracle JET tooling to access the Shared Component Exchange, as described in [Publish Web Components to Oracle Component Exchange](#).

Publish Web Components to Oracle Component Exchange

When you want to store and share Web Components across machines for re-use by other developers, you can use the Oracle JET CLI (Command-Line Interface) to configure access to a Shared Component Exchange defined in Oracle Visual Builder Studio and then publish components to a public project.

When you want to share Web Components, you can use Oracle JET CLI to configure access to a Shared Component Exchange by supplying the URL to the target Component Exchange. Once you have configured the JET tooling for a specific Component Exchange, you can run the `publish component` command in the JET CLI to upload specified components. Users with access rights can use the CLI to search the Component Exchange for components by keyword and add components to their web application project.

Before you begin:

- Create a project in a Shared Component Exchange, as described in [Create a Project to Host a Shared Oracle Component Exchange](#).
- Share the project with all users who will need to access components by adding them as members to the project, as described in [Create a Project to Host a Shared Oracle Component Exchange](#). Users will need to create their own user name and password to access the Shared Component Exchange.

To publish components to a Shared Component Exchange:

1. Obtain the URL to the Shared Component Exchange that you can use to configure Oracle JET tooling.

- a. From the Shared Component Exchange that you created, copy the URL that you would use to clone the GIT repository. The URL will look similar to this.

```
https://john.doe@example.org@xxxxx-
cloud01.developer.ocp.oraclecloud.com/xxxxx-cloud01/s/
xxxxx-cloud01_sharedcomponentcatalog_8325/scm/
sharedcomponentcatalog.git
```

- b. Using the copied URL, remove the user name prefix (for example, `john.doe@example.org@`) and remove elements from `/scm` onwards to obtain a root of just the project, similar to this.

```
https://xxxxx-cloud01.developer.ocp.oraclecloud.com/xxxxx-
cloud01/s/xxxxx-cloud01_sharedcomponentcatalog_8325
```

- c. Next, append `/compcatalog/0.2.0`.

In this example, the Exchange URL that you need for the Oracle JET tooling looks like this.

```
https://xxxxx-cloud01.developer.ocp.oraclecloud.com/xxxxx-
cloud01/s/xxxxx-cloud01_sharedcomponentcatalog_8325/compcatalog/
0.2.0
```

2. Configure Oracle JET tooling to access the Component Exchange project by running the `ojet configure` command in the Oracle JET CLI. Set the `--exchange-url` flag on the command to pass the Component Exchange URL you obtained.

```
ojet configure --exchange-url=https://  
xxxxxxxxx-cloud01.developer.ocp.oraclecloud.com/xxxxxxxxx-cloud01/s/xxxxxxxxx-  
cloud01_sharedcomponentcatalog_8325/compcatalog/0.2.0
```

 **Tip:**

You can use the JET CLI to define an Exchange URL that is global to all projects for your user:

```
ojet configure --global --exchange-url=myExchange.org
```

The URL that you pass to the `ojet configure` command at the project level, overrides the global definition.

3. In the Oracle JET CLI, publish a component to the configured Component Exchange by running the `publish component` command.

```
ojet publish component my-demo-card
```

Optionally, you can supply Component Exchange login user name and password with the `publish` command. For more information, enter `ojet help publish` in the CLI.

Oracle JET tooling supports searching the configured Component Exchange by running the `search exchange` command with a keyword, such as the component name. Additionally, you can add components to your web application by running the `add component` command for the configured Component Exchange. For more information, use `ojet help` in the JET CLI.

Upload and Consume Web Components on a CDN

You can package a Web Component to make it available on a Content Delivery Network (CDN) and you can reuse the component in your application.

You can include the CDN location of the component in the component metadata as defined in the component's `component.json` file. By doing this, tools such as Oracle JET tooling and Oracle Visual Builder will be able to point to the CDN location when you build your applications in release mode.

CDN information is encoded by using the `paths` attribute in the `component.json` file. A typical example looks similar to this.

```
{  
  "name": "demo-samplecomponent",  
  "displayName": "Sample component",  
  "version": "1.0.0",
```

```

"paths": {
  "cdn": {
    "min": "https://static.example.com/cdn/jet/components/demo-
samplecomponent/1.0.0/min",
    "debug": "https://static.example.com/cdn/jet/components/demo-
samplecomponent/1.0.0"
  }
},
...

```

The following notes apply to the `paths` attribute that you specify:

- The exact CDN root location will depend on your CDN provider, and it is the final part of the location that is needed.
- The location has a folder with the same name as the component (in this example `demo-samplecomponent`) followed by the version number of the component (`1.0.0`). As you release new versions of the component, you will create a new version-number folder under the component root.
- You can provide both a `min` and `debug` path for the component, where the `debug` path is optional.

To prepare for CDN distribution, package the component by running the `package component` command in the Oracle JET Command-Line Interface. This will produce a zip file with the same name as the component (for example, `demo-samplecomponent.zip`) in the `/dist` folder of the containing Oracle JET application. This zip file will contain both the specified component and a minified version of that component in a `/min` subfolder.

Unpack the zip file that you created under the `/<component-name>/<version>` folder for your CDN as identified in the `component.json` file.

After a component is available on a CDN, you can then use it in your JET application by pointing the `requireJS` path for the component to the CDN location that you identified for the component. If you use Oracle JET tooling, this will be done for you; however, if you need to define the `requireJS` paths manually, such a mapping will look similar to this in `main.js` file:

```

requirejs.config(
{
  baseUrl: 'js',
  paths: {
    'demo-samplecomponent': '"https://static.example.com/cdn/jet/
components/demo-samplecomponent/1.0.0/min',
    ...

```

References to the component can be made using the path `<component-name>/loader`.

10

Using the Common Model and Collection API

The Oracle JET Common Model and Collection API provides a two-way data binding model using Knockout. Use the Model and Collection classes to build Create, Read, Update, Delete (CRUD) applications that integrate REST services or data from any web service that returns data in the form of JSON objects.

Topics:

- [Typical Workflow for Binding Data in Oracle JET](#)
- [About Oracle JET Data Binding](#)
- [About the Oracle JET Common Model and Collection Framework](#)
- [Integrate REST Services](#)
- [Create a CRUD Application Using Oracle JET](#)

Typical Workflow for Binding Data in Oracle JET

Understand Oracle JET's support for the Model-View-ViewModel design. Learn how to use the Oracle JET Common Model and Collection API and how to integrate REST services into your Oracle JET application.

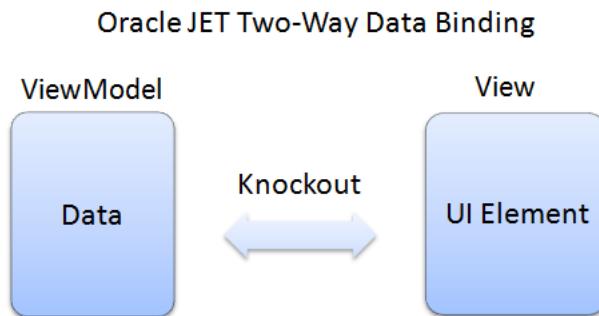
To use the Oracle JET Common Model for building CRUD applications, refer to the typical workflow described in the following table:

Task	Description	More Information
Understand Oracle JET support for data binding	Identify Oracle JET's support for the Model-View-ViewModel design.	About Oracle JET Data Binding
Use Oracle JET's Common Model and Collection	Identify the Oracle JET Common Model and Collection API, its relationship with Knockout, and how to use it in your Oracle JET application.	About the Oracle JET Common Model and Collection Framework
Integrate REST services	Understand how to integrate REST services into your Oracle JET application.	Integrate REST Services
Create a CRUD application	Create a CRUD application using the Oracle JET Common Model and Collection API.	Create a CRUD Application Using Oracle JET

About Oracle JET Data Binding

Oracle JET supports two-way data binding between the View and Model layers in the Model-View-ViewModel (MVVM) design. Data changes in the ViewModel are sent to

the UI components, and user input from the UI components is written back into the ViewModel.



Oracle JET uses Knockout to perform the data binding between the UI elements and the ViewModel. The ViewModel normally contains data fields for the UI state as well as references to external data. One of the ways to provide external data is to use the Common Model and Collection API.

Data for an Oracle JET application can come from any web data source that generates JSON data, such as a REST service, Server Sent Event (SSE), or WebSocket. In addition, Oracle JET also provides specific support for integrating web service data based on the Oracle REST standard.

Oracle JET also provides UI components for the View layer that include properties for data binding with Knockout. For additional information about Oracle JET's UI components and data binding options, see [Understanding Oracle JET User Interface Basics](#).

About the Oracle JET Common Model and Collection Framework

The Oracle JET Common Model and Collection API provides a collection-of-records object model that includes classes for bringing external data into an Oracle JET application and mapping the data to the application's view model.

Topics:

- About the Oracle JET Common Model and Collection API
 - About Oracle JET Data Binding and Knockout
 - Use the Oracle JET Common Model and Collection API

About the Oracle JET Common Model and Collection API

The Oracle JET Common Model and Collection API provides a collection-of-records object model that includes the following classes:

- Model: Represents a single record from a data service such as a REST service
 - Collection: Represents a set of data records and is a list of `Model` objects of the same type

- **Events:** Provides methods for event handling
- **KnockoutUtils:** Provides methods for mapping the attributes in a `Model` or `Collection` object to Knockout observables for use with component ViewModels

`Model` and `Collection` include client-side API that provides one way to bring external data into an Oracle JET application. `KnockoutUtils` provides the `map()` method to map the attributes in a model object or the attributes of all models in a collection object to the application's view data model.

About Oracle JET Data Binding and Knockout

Knockout provides bindings between components as well as binding data from a `ViewModel` to components or HTML elements in an Oracle JET application. Knockout is an integral part of Oracle JET's toolkit and is included in the Oracle JET distribution.

The Oracle JET Model and Collection API includes the `KnockoutUtils` class which contains the `map()` method to convert the attributes in a model object (or the attributes of all models in a collection object) into Knockout observables for use with components' `ViewModels`.

The example below maps the `data` collection to the `tasks` `ViewModel`.

```
renderTaskViews = function(tasksData) {
  this.tasks = KnockoutUtils.map(data);
}
```

To utilize the data model in Oracle JET applications, Oracle JET provides custom elements which support Knockout variables in the `data` attribute. For additional information about working with Oracle JET UI components, see [Understanding Oracle JET User Interface Basics](#).

```
<ojs-table data="[[dataProvider]]">
</ojs-table>
```

For additional information about Knockout, see <http://www.knockoutjs.com>.

Use the Oracle JET Common Model and Collection API

To use a `Model` or `Collection` class, an application must extend `Model` or `Collection` to create a foundation object to represent a data record or list of records from its data service. The application provides the data service's URL used for fetching and updating task records (when a task record's ID is appended), along with various options giving users the ability to map data service records to their client-side `ViewModel` and vice versa.

To use the Oracle JET Common Model and Collection API:

1. Add JavaScript code to your application that extends `Model`.

The following script shows a simple example of extending `Model` to create a model object which defines a `Department`. In this example, the data is returned from the REST service at the indicated URL. The `parse` callback parses the data and maps the attributes received from the REST service to desired `ViewModel` attribute names.

```
var self = this;
self.serviceURL = 'http://RESTServerIP:port/stable/rest/Departments';
self.Department = Model.extend({
```

```

        urlRoot: self.serviceURL,
        parse: self.parseDept,
        idAttribute: 'DepartmentId'
    });

/**
 * Callback to map attributes returned from RESTful data service to desired
view model attribute names
 */
self.parseDept = function(response) {
    return {DepartmentId: response['DepartmentId'],
            DepartmentName: response['DepartmentName'],
            LocationId: response['LocationId'],
            ManagerId: response['ManagerId']};
};


```

2. Add JavaScript code to your application that extends Collection.

The following code example creates a collection object for its entire data set (or list) of task records and ties that to a specific instance of a task record model. The `fetch()` method tells the collection to go to the data service and asynchronously retrieve the data services' data set using the given URL, through jQuery AJAX calls. The application's `success` callback method is invoked when the call successfully returns and the collection has been populated with model records.

```

self.DeptCol = ko.observable();
self.myDept = new self.Department();

// Create a base object "class" for the entire dataset
self.DeptCollection = Collection.extend({
    url: self.serviceURL + "?limit=50",
    model: self.myDept
});

// Create a specific instance for the departments. This will be filled with
instances of the
// model "department" for each record when the data is retrieved from the
data service
self.DeptCol(new self.DeptCollection());
self.dataProvider(new CollectionDataProvider(self.DeptCol()));


```

3. Bind the returned data collection to a Knockout ViewModel to make it ready for consumption by one or more components on the application page.

The following code sample maps the `vm` Collection object to the element in the view with the ID 'table' using the `ko.applyBindings()` function.

```

var vm = new viewModel;

Bootstrap.whenDocumentReady().then(
    function ()
    {
        ko.applyBindings(vm, document.getElementById('table'));
    }
);


```

4. Add code to the application's index.html or main page that consumes the Knockout ViewModel.

The following code examples shows a simple table that is defined with four columns: Department Id, Department Name, Location Id, and Manager Id. The data attribute binds the dataProvider collection to the ViewModel using Knockout.

```
<ojs-table id="table" summary="Department List" aria-label="Departments Table"
           data='[[dataProvider]]'
           columns='[{"headerText": "Department
Id",
            "field": "DepartmentId"}, {"headerText": "Department
Name",
            "field": "DepartmentName"}, {"headerText": "Location
Id",
            "field": "LocationId"}, {"headerText": "Manager
Id",
            "field": "ManagerId"}]'>
</ojs-table>
```

For a complete list of Oracle JET Common Model and Collection API properties and functions, see the [Model](#) and [Collection](#) API documentation.

Integrate REST Services

Oracle JET is designed to work with any web service that returns data in the form of JSON objects. Because the content of JSON objects can vary widely from service type, the application developer is responsible for examining the JSON object's content and defining the ViewModel appropriately.

Topics:

- [About Oracle JET Support for Integrating REST Services](#)
- [Pass Custom AJAX Options in Common Model CRUD API calls](#)
- [Supply a customURL Callback Function](#)
- [Replace sync or ajax Functions](#)

About Oracle JET Support for Integrating REST Services

The Model and Collection classes provide native support for mapping JSON objects obtained from REST web services to Knockout observables. The `Collection.url` property specifies the path to the web service data.

```
Collection.extend({url: "http://myserver/departments", ...})
```

Model objects also include the `url` property. By default, this is set to the model ID appended to the `Collection.url` property: `http://myserver/departments/modelID`.

To override the default path set by the Collection object, set the `model.urlRoot` property, and the application will use the `url.Root/modelID` as the path to the data.

Oracle JET also provides three ways to customize the AJAX requests that Oracle JET makes when accessing REST services through the Common Model.

- Pass custom AJAX options in Common Model CRUD API calls.

- Supply a `customURL` callback function.
- Replace the `sync` or `ajax` functions.

Pass Custom AJAX Options in Common Model CRUD API calls

Use this method if the default URL behavior is acceptable, but your application needs to pass custom properties such as headers to the REST server.

To customize the AJAX options, add property/value pairs to the `options` argument of the `Collection.create(create)`, `Collection.fetch(read)`, `Model.save(update)`, and `Model.destroy(delete)` functions.

The following code example shows how an application could pass in a custom header when doing a read from the REST service.

```
myObjCollection.fetch(  
    {  
        headers:{my-custom-header:"header-value"},  
        beforeSend: myBeforeSendCallbackFunc,  
        success:function(collection){  
            }});
```

Supply a `customURL` Callback Function

If you need to customize the URL or AJAX behavior, set the `Collection.customURL` property to a user-defined callback function when defining the collection.

```
function myCustomFunction(operation, collection, options) {  
    ...  
}  
var myCollection = Collection.extend({customURL:myCustomFunction,...})
```

The callback function should include the following parameters which are passed to it by default:

- `operation`: `create`, `read`, `update`, `patch`, or `delete`
- `collection`: The `Model` or `Collection` object requesting the operation
- `options`: Information relevant to the operation

Typical options include:

- `recordID`: Id of the record involved in the operation
- `fetchSize`: The number of records to return; returns all if not set
- `startIndex`: Starting record number of the data set
- `fromID`: Retrieve records starting with or after the record with the given unique ID. Based on your REST server's interpretation, the record with the given ID may or may not be included.
- `since`: Retrieve records with timestamps after the given timestamp
- `until`: Retrieve records with timestamps up to the given timestamp
- `sort`: Field or fields by which to sort
- `sortDir`: Specific ascending or descending order

The following example shows a `customURL` callback. By setting this user-created callback function on the Collection's `customURL` property, the Collection will call out to this function when it needs information to make a REST call.

```
function myCustomFunction(operation, collection, options) {
    // Use the default if operation is create
    if (operation === "create") {
        return null;
    }
    if (operation === "delete") {
        // Just set a URL if it's delete
        return "http://destroy/model='+options['recordID']";
    }
    var retObj = {};
    if (operation === "read") {
        retObj['url'] = "http://fetch";
        if (options['sort']) {
            retObj['url'] += "/order="+options['sort'] + ";" + options['sortDir'];
        }
        retObj['headers'] = {my-custom-header:"header-value"};
        retObj['mimeType'] = "text/plain";
        return retObj;
    }
    // Update or patch
    retObj['url'] = "http://update/model=" + options['recordID'];
    retObj['type'] = "POST";
    retObj['beforeSend'] = myBeforeSendCallback;
    return retObj;
}
```

Replace sync or ajax Functions

You can replace `sync()` or `ajax()` when you need to change the entire client-server transport mechanism. For example, this option is best if you want to use WebSockets instead of AJAX calls to the server.

The `sync` method is the primary server access method for all models and collections and accepts the following parameters:

```
sync = function(method, model, options)
```

Valid values for the `method` parameter are the CRUD operations: `create`, `read`, `update`, `patch`, and `delete`. The `model` parameter accepts either the `Model` being created, `read`, `updated`, `patched`, or `deleted`, or the `Collection` being `read`. Options are passed down from the higher-level common model API call and vary with the type of operation.

The replacement `sync()` method is completely responsible for implementing all of these operations using whatever transport mechanism is being used, whether it be the application's own AJAX routines, WebSockets, a JavaScript server, or something else. The method must return a Promise object similar to the AJAX XMLHttpRequest (XHR) in order to maintain compatibility with potential virtual API calls being made by the `Model` and `Collection` object.

 **Note:**

Replacing `sync()` replaces the transport mechanism for the JET common model used by all calls within the application. This is a very advanced use of the Oracle JET common model.

The `ajax()` method is the primary AJAX entry point for all Model and Collection server interactions, when they are using the default sync implementations. `ajax()` passes its parameters and return value through to the `jQuery.ajax()` method by default. For additional information about the `jQuery.ajax()` method's expected parameters and return value, see <http://api.jquery.com/jquery.ajax>.

For additional information about the `sync()` or `ajax()` methods, see the Oracle JET `sync()` and `ajax()` API documentation.

Create a CRUD Application Using Oracle JET

Use Knockout and the Oracle JET Common Model API to create applications that perform CRUD (Create, Read, Update, Delete) operations on data returned from a REST Service API.

Topics:

- [Define the ViewModel](#)
- [Read Records](#)
- [Create Records](#)
- [Update Records](#)
- [Delete Records](#)

Define the ViewModel

Identify the data source for your application and create the ViewModel.

1. Identify your data source and examine the data. For data originating from a REST service, identify the service URL and navigate to it in a browser.

The following example shows a portion of the output of a REST service that returns department data for a fictitious organization from a REST server named MockRESTServer.

```
{  
  "Departments" : [ {  
    "DepartmentId" : 10,  
    "DepartmentName" : "Administration",  
    "ManagerId" : null,  
    "LocationId" : null,  
    "version" : "ACED00057...contents truncated",  
    "links" : {  
      "self" : {  
        "rel" : "self",  
        "href" : "http://mockrest/stable/rest/Departments/10"  
      },  
      "parent" : {  
        "rel" : "parent",  
        "href" : "http://mockrest/stable/rest/Departments/  
      }  
    }  
  }  
]
```

```
        "canonical" : {
            "rel" : "canonical",
            "href" : "http://mockrest/stable/rest/Departments/10"
        },
        "Employees" : {
            "rel" : "child",
            "href" : "http://mockrest/stable/rest/Departments/10/Employees"
        }
    },
    {
        "DepartmentId" : 20,
        "DepartmentName" : "Marketing",
        "ManagerId" : null,
        "LocationId" : null,
        "version" : "ACED00057...contents truncated",
        "links" : {
            "self" : {
                "rel" : "self",
                "href" : "http://mockrest/stable/rest/Departments/20"
            },
            "canonical" : {
                "rel" : "canonical",
                "href" : "http://mockrest/stable/rest/Departments/20"
            },
            "Employees" : {
                "rel" : "child",
                "href" : "http://mockrest/stable/rest/Departments/20/Employees"
            }
        }
    },
    {
        ... contents omitted
    }],
    "links" : {
        "self" : {
            "rel" : "self",
            "href" : "http://mockrest/stable/rest/Departments"
        }
    },
    "_contextInfo" : {
        "limit" : 25,
        "offset" : 0
    }
}
```

In this example, each department is identified by its `DepartmentId` and contains information about its name (`DepartmentName`), manager (`ManagerId`), and location (`LocationId`). Each department also contains employees (`Employees`) which are children of each department.

Tip:

The Oracle JET Common Model CRUD sample application uses JSON data returned from a mock rest service. You can find the mock rest service scripts in the `public_html/js/rest` folder. You can find the sample JSON data in the `departments.json` file located in the `public_html/js` folder.

2. Determine the data you will need for your collection.

For example, the following figure shows a simple Oracle JET table that uses the DepartmentId, DepartmentName, LocationId, and ManagerId returned from the REST service identified in the previous step to display a table of department IDs, names, location IDs, and manager IDs. The table element is defined as an `oj-table` element, which is included in the Oracle JET UI component library.

Department Id	Department Name	Location Id	Manager Id
10	Administration		
20	Marketing		
30	Transportation		
40	Shipping		
50	Human Resources		
60	Operations		
70	Inventory		
80	Sales	2500	2500
100	Finance		
110	Documentation		
130	Billing	1700	1700
140	Control And Credit	1700	1700

3. Add a JavaScript function to your application that will contain your ViewModel.

The following code shows a skeleton function that defines the ViewModel in the Oracle JET Common Model CRUD application. In this example, the function is stored in a file named `app.js`. The code to complete the ViewModel will be defined in upcoming steps.

```
define(['knockout'],
  function(ko)
  {
    function viewModel() {
      // To be defined
    };
    return {'deptVM': viewModel};
  }
)
```

 **Note:**

This example uses RequireJS for modular development. The RequireJS bootstrap file will be shown in a later step. For additional information about using RequireJS, see [Using RequireJS for Modular Development](#).

4. Add a JavaScript function to the function you defined in the previous step that defines the data model using `Model.extend()`.

The highlighted code in the example below defines the data model for the application shown in the preceding figure. The `Department` variable represents a single record in the database and is displayed as a row in the table. `Department` is declared using the `Model.extend` function call and instantiated in the declaration for `myDept`. The `urlRoot` property defines the data source, which in this case is the REST service URL.

```
define(['knockout', 'ojs/ojknockouttemplateutils', 'ojs/ojmodel',
'MockRESTServer'],
    function(ko, KnockoutTemplateUtils, model)
{
    function viewModel() {
        var self = this;
        self.KnockoutTemplateUtils = KnockoutTemplateUtils;
        self.serviceURL = 'http://mockrest/stable/rest/
Departments';

        function parseDept(response) {
            if (response['Departments']) {
                var innerResponse = response['Departments'][0];
                if (innerResponse.links.Employees == undefined) {
                    var empHref = '';
                } else {
                    empHref = innerResponse.links.Employees.href;
                }
                return {DepartmentId: innerResponse['DepartmentId'],
                        DepartmentName: innerResponse['DepartmentName'],
                        links: {Employees: {rel: 'child', href:
empHref}}};
            }
            return {DepartmentId: response['DepartmentId'],
                    DepartmentName: response['DepartmentName'],
                    LocationId: response['LocationId'],
                    ManagerId: response['ManagerId'],
                    links: {Employees: {rel: 'child', href:
response['links']['Employees'].href}}};
        }

        // Think of this as a single database record or a single
        // table row.
        var Department = model.Model.extend({
            urlRoot: self.serviceURL,
            parse: parseDept,
            idAttribute: 'DepartmentId'
        });

        var myDept = new Department();
    };

    return {'deptVM': viewModel};
});
}
```

The `parse` property is an optional user callback function to allow parsing of JSON record objects as they are returned from the data service. In this example, `parseDept` is the callback function and simply maps the `DepartmentId` and `DepartmentName` returned from the REST service to `DepartmentId` and `DepartmentName`. If the `LocationId` or `ManagerId` records contain data, `parseDept` maps the attributes to `LocationId` and `ManagerId`.

The parse callback can be useful for mapping database attribute names to names that may make more sense. For example, if your database uses `Id` and `Name` as the attributes that represent the department ID and department name, you could replace the return call in the `parseDept` function with:

```
return {DepartmentId: response['Id'], DepartmentName: response['Name']};
```

For a complete list of `Model` properties and functions, see the [Model API](#) documentation.

5. Define the collection that will hold the data model object you defined in the previous step using `Collection.extend()`.

The highlighted code in the example below defines the collection object for the `Department` model object. The `DeptCollection` variable is declared in the `viewModel()` function using the `Collection.extend` function and instantiated in the declaration for `self.DeptCol`. The `url` property defines the data source, which in this case is the REST service URL, and limits the collection to 50 records.

```
define(['knockout', 'ojks/knockouttemplateutils', 'ojks/ojmodel', 'ojks/ojcollectiondataprovider', 'MockRESTServer'],
    function(ko, KnockoutTemplateUtils, model, CollectionDataProvider){
        function viewModel() {
            var self = this;
            self.KnockoutTemplateUtils = KnockoutTemplateUtils;
            self.serviceURL = 'http://mockrest/stable/rest/Departments';
            self.Departments = ko.observableArray([]);
            self.DeptCol = ko.observable();
            self.dataProvider = ko.observable();

            var parseDept = function(response) {
                ... contents omitted
            };

            var Department = model.Model.extend({
                urlRoot: self.serviceURL,
                parse: parseDept,
                idAttribute: 'DepartmentId'
            });

            var myDept = new Department();

            // this defines our collection and what models it will hold
            var DeptCollection = model.Collection.extend({
                url: self.serviceURL + "?limit=50",
                model: myDept,
            });

            self.DeptCol(new DeptCollection());
            self.dataProvider(new
CollectionDataProvider(self.DeptCol()));

        }
        return {'deptVM': viewModel};
    );
}
```

Both the `DeptCol` and `dataProvider` objects are defined as Knockout observables so that changes to the data collection can be handled. The `dataProvider` object

will contain the column data needed by the `oj-table` element and is passed the `DeptCol` observable as a parameter to `CollectionDataProvider()`.

The `Departments` object is defined as a Knockout observable array and will be populated in a later step.

For a complete list of `Collection` properties and functions, see the [Collection API documentation](#).

6. Populate the collection with data by calling `Collection.fetch()` to read the data from the data service URL.

The highlighted code in the code sample below calls `collection.fetch()` to add data to the `DeptCol` data collection and complete the `ViewModel`.

```
define(['knockout', 'ojs/ojknockouttemplateutils', 'ojs/ojmodel', 'ojs/ojcollectiondataProvider', 'MockRESTServer'],
    function(ko, KnockoutTemplateUtils, model, CollectionDataProvider){
        function viewModel() {
            var self = this;
            self.KnockoutTemplateUtils = KnockoutTemplateUtils;
            self.serviceURL = 'http://mockrest/stable/rest/Departments';
            self.Departments = ko.observableArray([]);
            self.DeptCol = ko.observable();
            self.dataProvider = ko.observable();

            self.fetch = function(successCallBack) {
                // populate the collection by calling fetch()
                self.DeptCol().fetch({
                    success: successCallBack,
                    error: function(jqXHR, textStatus, errorThrown){
                        console.log('Error in fetch: ' + textStatus);
                    }
                });
            };

            var parseDept = function(response) {
                ... contents omitted
            };

            var Department = model.Model.extend({
                urlRoot: self.serviceURL,
                parse: parseDept,
                idAttribute: 'DepartmentId'
            });

            var myDept = new Department();

            var DeptCollection = model.Collection.extend({
                url: self.serviceURL + "?limit=50",
                model: myDept
            });

            self.DeptCol(new DeptCollection());
            self.dataProvider(new
CollectionDataProvider(self.DeptCol()));

        }
        return {'deptVM': viewModel};
    }
);
```

The `fetch()` function also defines an error callback that will log a message to the console if the `fetch()` call fails.

7. Add the ViewModel or the file containing the name of your ViewModel to your RequireJS bootstrap file, typically `main.js`.

If you created your Oracle JET application using an Oracle JET Starter template, you should already have a `main.js` file. Locate the line that defines the require modules and add your file to the list.

For example, the code below lists the modules defined for the Common Model Sample. The application stores its ViewModel in the `app.js` file. The reference to the `app.js` file is highlighted in bold.

```
require(['knockout',
        'app',
        'ojs/bootstrap',
        'footer',
        'ojs/ojknockout-model',
        'MockRESTServer',
        'ojs/ojmodel',
        'ojs/ojknockout',
        'ojs/ojdialog',
        'ojs/ojinputtext',
        'ojs/ojinputnumber',
        'ojs/ojbutton',
        'ojs/ojtable'],
```

You must also add the `app` reference in the callback definition as shown in the following example.

```
// this callback gets executed when all required modules are loaded
function(ko, app, Bootstrap, footer, KnockoutUtils, MockRESTServer)
{
    ...
}
```

8. Update your RequireJS bootstrap file to instantiate the ViewModel, create the Knockout bindings, and display the content on the page.

The highlighted code in the `main.js` code sample below creates a Knockout observable for each element in the `deptData` collection and assigns the resulting array to the `Departments` Knockout observable array you defined in a previous step.

```
require(['knockout', 'app', 'ojs/ojbootstrap', 'footer', 'ojs/ojknockout-
model',
        'MockRESTServer', 'ojs/ojmodel', 'ojs/ojknockout','ojs/ojdialog', 'ojs/
ojinputtext',
        'ojs/ojinputnumber', 'ojs/ojbutton', 'ojs/ojtable'],
        function(ko, app, Bootstrap, footer, KnockoutUtils,
        MockRESTServer) // this callback gets executed when all required modules are
loaded
{
    var fvm = new footer.footerVM();

    Bootstrap.whenDocumentReady().then(
        function(){
            var xhttp = new XMLHttpRequest();
            xhttp.overrideMimeType('application/json');
            xhttp.open('GET', 'js/departments.json', true);
            xhttp.onreadystatechange = function() {
```

```

        if (xhttp.readyState == XMLHttpRequest.DONE && xhttp.status
== 200) {
            new MockRESTServer(JSON.parse(xhttp.responseText),
{id:"DepartmentId",
url:/^http:\/\/mockrest\/stable\/rest\/Departments(\?
limit=([\d]*)?)$/i,
idUrl:/^http:\/\/mockrest\/stable\/rest\/Departments\(
([\d]+)$/i);

        var vm = new app.deptVM();
ko.applyBindings(fvm,
document.getElementById('footerContent'));
        vm.fetch(
            function(collection, response, options){
                var deptData = collection;
                // This will create a ko.observable() for each element
                // in the deptData response and assign the resulting array
                // to the Departments ko observableArray.
                vm.Departments = KnockoutUtils.map(deptData, null,
true);
                //perform a Knockout applyBindings() call binding this
                // viewModel with the current DOM
                ko.applyBindings(vm,
document.getElementById('mainContent'));
                //Show the content div after the REST call is completed.
                document.getElementById('mainContent').style.display
= 'block';
            });
        }
        xhttp.send(null);
    );
}
);

```

Read Records

To read the records, define the Oracle JET elements that will read the records in your main HTML5 page.

The following sample code shows a portion of the index.html file that displays a table of records using the ViewModel defined in the previous steps and the oj-table element. In this example, the mainContent div includes the table definition that creates Department Id, Department Name, Location Id, and Manager Id as the table header text and defines the content for each row.

```

<div id="mainContent" class="oj-md-12 oj-flex-item page-padding" style="display:
none;">
    <div class="page-padding">
        <oj-table id="table" data="[[dataProvider]]"
            columns='[{"headerText": "Department Id",
                "field": "DepartmentId", "sortable": "enabled"},
            {"headerText": "Department Name",
                "field": "DepartmentName", "sortable": "enabled"},
            {"headerText": "Location Id",
                "field": "LocationId"}, {"headerText": "Manager Id",
                "field": "ManagerId"}]'>
    </oj-table>

```

```
</div>  
</div>
```

The `data` attribute reads the variable `dataProvider` from the REST service and binds it to the `oj-table` element.

Create Records

To add the ability to create new records, add elements to your HTML5 page that accept input from the user and create a function that sends the new record to the REST server.

The figure below shows the result of adding a form using `oj-input-*` elements to the Oracle JET Common Model sample application. The user can enter a new department number in the provided field or use the side arrows to increment or decrement the value. The user then enters the name and clicks **Add**.

Department Id	Department Name	Location Id	Manager Id
10	Administration		
20	Marketing		
30	Transportation		
40	Shipping		
50	Human Resources		
60	Operations		
70	Inventory		
80	Sales	2500	2500
100	Finance		
110	Documentation		
130	Billing	1700	1700
140	Control And Credit	1700	1700

New Department

Department Id

Department Name

To add the ability to create new records to the application's ViewModel and HTML5 page:

1. Add elements to the application's main page that accept input from the user.

The highlighted code in the example below adds `oj-input-*` elements to the `index.html` page shown in the previous task.

```

<div id="mainContent" class="oj-md-12 oj-flex-item page-padding"
style="display: none;">
    <div class="page-padding">
        <div id="deptList" class="oj-md-9 oj-flex-item">
            <oj-table id="table" summary="Demo Table"
                ... contents omitted
            </oj-table>
            <br/>
            <div id="addDept" class="oj-flex-item oj-md-3 oj-sm-12 right">
                <div id="quickUpdate" class="frame">
                    <h3>New Department</h3><hr/>
                    <oj-form-layout id="newDeptForm" label-edge="inside">
                        <oj-input-number id="newDepartId" value="555" label-
                        hint="Department Id"></oj-input-number>
                        <oj-label-value>
                            <oj-label slot="label" for="newDepartName">Department
                        Name</oj-label>
                        <oj-input-text slot="value" id="newDepartName"
                        maxlength='30' placeholder="enter new name"></oj-input-text>
                        <oj-button slot="value" id="saveBtn" on-oj-
                        action="[[ addDepartment ]]" label='Add Department'></oj-button>
                        </oj-label-value>
                    </oj-form-layout>
                </div>
            </div>
        </div>
    </div>
</div>

```

The form is defined to contain two input fields: an `oj-input-number` for the department ID, and an `oj-input-text` for the department name. The `oj-button` component is used for the form's button.

The `oj-button`'s `on-oj-action` attribute is bound to the `addDepartment` function which is defined in the next step.

2. Add code to the ViewModel to add the user's input as a new record (model) in the data collection.

The highlighted code in the example below shows the `addDepartment()` function that adds the new department number and department name to the `DeptCol` data collection. In this example, the function calls the `Collection.create()` method which creates the new model instance, adds it to the data collection, and saves the `new DepartmentId` and `DepartmentName` to the data service.

```

define(['knockout', 'ojs/ojknockouttemplateutils', 'ojs/ojmodel', 'ojs/
ojcollectiondataProvider', 'MockRESTServer'],
    function(ko, KnockoutTemplateUtils, model,
CollectionDataProvider) {
    function viewModel() {
        var self = this;
        self.KnockoutTemplateUtils = KnockoutTemplateUtils;
        self.serviceURL = 'http://mockrest/stable/rest/Departments';
        self.Departments = ko.observableArray([]);
        self.DeptCol = ko.observable();
        self.dataProvider = ko.observable();

        self.fetch = function(successCallBack) {
            ... contents omitted
        };

        function parseDept(response){

```

```

        ... contents omitted
    };

    var Department = model.Model.extend({
        urlRoot: self.serviceURL,
        parse: parseDept,
        parseSave:parseSaveDept,
        idAttribute: 'DepartmentId'
    });

    var myDept = new Department();

    var DeptCollection = model.Collection.extend({
        url: self.serviceURL + "?limit=50",
        model: myDept,
        comparator: 'DepartmentId'
    });

    self.DeptCol(new DeptCollection());
    self.dataProvider(new
CollectionDataProvider(self.DeptCol()));

    function parseSaveDept(response){
        return {DepartmentId: response['DepartmentId'],
                DepartmentName: response['DepartmentName'],
                LocationId: response['LocationId'],
                ManagerId: response['ManagerId'],
                links: {Employees: {rel: 'child', href:
response['links']['Employees'].href}}};
    };

    self.addDepartment = function (formElement, event) {
        var id = document.getElementById("newDepartId").value;
        var recordAttrs = {DepartmentId: id,
                           DepartmentName:
document.getElementById("newDepartName").value,
                           ManagerId: "", LocationId: "",
                           links: {Employees: {rel: 'child', href:
self.serviceURL + '/' + id + '/Employees'}}};

        this.DeptCol().create(recordAttrs,{
            wait: true,
            contentType: 'application/
vnd.oracle.adf.resource+json',
            success: function (model, response) {
                console.log('Success in Create');
            },
            error: function(jqXHR, textStatus, errorThrown){
                console.log('Error in Create: ' + textStatus);
            }
        });
    }
}

return {'deptVM': viewModel};
}
);

```

The Collection.create() function accepts options to control the record save. In this example, simple error checking is added. In addition, the create() function

sends a `contentType` to the REST server. Depending upon the REST service you are using, this option may not be required or may need modification.

Update Records

To add the ability to update records, add elements to your HTML5 page that accept input from the user and create a function that sends the updated record to the REST server.

The figure below shows the Oracle JET Common Model sample application configured to allow updates to the department name. When the user moves the focus over a department name, a tooltip appears that prompts the user to click to edit. If the user clicks the department name, a dialog appears that enables the user to change the name. The user can type in a new department name and click **Change** to update the record or Cancel to keep the existing department name.

Department Id	Department Name	Location Id	Manager Id
10	Administration		
20	Marketing		
30	Transportation		
40	Shipping		
50	Human Resources		
60	Operations		
70	Inventory		
80	Sales	250	
100	Finance		
110	Documentation		
130	Billing	170	
140	Control And Credit	170	

(Click on a Department Name to edit it)

Change Department Name

Department Name

Change Cancel

To add the ability to update records to the application's `ViewModel` and `HTML5` page:

- Add elements to the application's main page that identifies updatable elements and enables the user to perform an action to update them.

The highlighted code in the example below adds the `oj-dialog` element to the page and provides the prompt to the user to click to edit the page.

```
<div id="mainContent" class="oj-flex-item oj-sm-12 oj-md-12 demo-page-content-area page-padding" style="display: none;">
    <div class="page-padding">
        <div id="deptList" class="oj-flex-item oj-md-9 oj-sm-12">
            <oj-table id="table" data="[[dataProvider]]"
                columns='[{"headerText": "Remove", "id": "column1",
```

```

"sortable": "disabled"} ,
        { "headerText": "Department Id",
          "field": "DepartmentId", "sortable": "enabled"} ,
        { "headerText": "Department Name",
          "field": "DepartmentName", "sortable": "enabled"} ,
        { "headerText": "Location Id",
          "field": "LocationId"} ,
        { "headerText": "Manager Id",
          "field": "ManagerId"} ]'
      selectionMode='{"row": "none", "column": "none"}'
      row-
    renderer='[[KnockoutTemplateUtils.getRenderer("row_tmpl", true)]]'>
      </oj-table>
    </div>
    <oj-dialog id="editDialog" style="display:none" title="Change Department
Name" drag-affordance="title-bar"
      modality="modeless" resize-behavior="none">
      <div slot="header" class="oj-helper-clearfix" aria-
labelledby="dialog-title-id">
        <div>
          <span id="infoIcon" class="oj-fwk-icon oj-fwk-icon-status-info"
style="float:left; margin-right: 10px"></span>
          <span id="dialog-title-id" class="oj-dialog-title">Change
Department Name</span>
        </div>
        </div>
      <div slot="body">
        <div class="oj-md-odd-cols-4">
          <oj-label for="newName" class="oj-label">Department Name</oj-label>
          <oj-input-text id="newName" value="{{currentDeptName}}"></oj-input-
text>
        </div>
        </div>
        <div slot="footer">
          <oj-button id="submitBtn" on-oj-
action="[[updateDeptName]]">Change</oj-button>
          <oj-button id="resetBtn" on-oj-action="[[cancelDialog]]">Cancel</
oj-button>
        </div>
      </oj-dialog>
    </div>
  </div>
<script type="text/html" id="row_tmpl">
  <tr>
    <td>
      <div id='deptId' on-click="[[function(data, event)
{$root.showChangeNameDialog(DepartmentId,data,event)}]]">
        <oj-bind-text value="[[DepartmentId]]"></oj-bind-text>
      </div>
    </td>
    <td>
      <div id="deptName" on-click="[[function(data, event)
{$root.showChangeNameDialog(DepartmentId,data,event)}]]">
        <oj-bind-text value="[[DepartmentName]]"></oj-bind-text>
      </div>
    </td>
    <td>
      <div id="locId">
        <oj-bind-text value="[[LocationId]]"></oj-bind-text>
      </div>
    </td>
  </tr>
</script>

```

```

</td>
<td>
<div id="mgrId">
    <oj-bind-text value="[[ManagerId]]"></oj-bind-text>
</div>
</td>
</tr>
</script>

```

The `oj-dialog` element includes the `oj-dialog-header`, `oj-dialog-body`, and `oj-dialog-footer` formatting classes. The `oj-dialog-header` class is optional and is used in this example to format the Info icon and title.

The `oj-dialog-body` class formats the dialog body which includes the `oj-label` for Department Name and an `oj-input-text` element to capture the user's input. The `oj-dialog-footer` defines two `oj-button` components which add the Change and Cancel buttons to the dialog. When the user clicks **Change**, the `updateDepartmentName()` function handles the updates to the record.

The original table is also modified to define the `row-renderer` attribute which specifies the `row_tmpl` template to use for the row display. The template script calls the `showChangeNameDialog()` function to display the dialog.

The `updateDepartmentName()` and `showChangeNameDialog()` functions are defined in the next step. For additional information about the `oj-dialog` component, see the [oj-dialog API documentation](#).

2. Add code to the ViewModel to update the record.

The highlighted code in the example below shows the `updateDepartment()` and `showChangeNameDialog()` functions.

```

define(['knockout', 'ojs/ojknockouttemplateutils', 'ojs/ojmodel', 'ojs/
ojcollectiondataprovider', 'MockRESTServer'],
    function(ko, KnockoutTemplateUtils, model,
CollectionDataProvider) {
    function viewModel() {
        var self = this;
        self.KnockoutTemplateUtils = KnockoutTemplateUtils;
        self.serviceURL = 'http://mockrest/stable/rest/Departments';
        self.Departments = ko.observableArray([]);
        self.DeptCol = ko.observable();
        self.dataProvider = ko.observable();
        self.currentDeptName = ko.observable('default');
        self.workingId = ko.observable('');

        self.fetch = function(successCallBack) {
            ... contents omitted
        };

        function parseDept(response){
            ... contents omitted
        }

        function parseSaveDept(response){
            ... contents omitted
        }

        var Department = model.Model.extend({
            urlRoot: self.serviceURL,

```

```

        parse: parseDept,
        parseSave:parseSaveDept,
        idAttribute: 'DepartmentId'
    });

var myDept = new Department();

var DeptCollection = model.Collection.extend({
    url: self.serviceURL + "?limit=50",
    model: myDept,
    comparator: 'DepartmentId'
});

self.DeptCol(new DeptCollection());
self.dataProvider(new
CollectionDataProvider(self.DeptCol()));

self.showChangeNameDialog = function(deptId, data,
event) {
    var currName = data.DepartmentName;
    self.workingId(deptId);
    self.currentDeptName(currName);
    document.getElementById("editDialog").open();
}

self.updateDeptName = function(formData, event) {
    var currentId = self.workingId();
    var myCollection = self.DeptCol();
    var myModel = myCollection.get(currentId);
    var newName = self.currentDeptName();
    if (newName != myModel.get('DepartmentName') &&
newName != '') {
        myModel.save({'DepartmentName': newName}, {
            success: function(myModel, response,
options) {

document.getElementById("editDialog").close();
},
error: function(jqXHR, textStatus,
errorThrown) {
            alert("Update failed with: " +
textStatus);

document.getElementById("editDialog").close();
}
});
} else {
    alert('Department Name is not different or the
new name is not valid');
    document.getElementById("editDialog").close();
}
};

return {'deptVM': viewModel};
};

);

```

The `showChangeNameDialog()` function stores the selected department detail and opens the dialog with the existing department name shown in the `oj-input-text` field.

The `updateDepartment()` function calls the `Model.save()` method to save the current `Model` object to the data source. The function also defines success and error callbacks to close the dialog upon success or issue an error message if the record was not updated.

Delete Records

To add the ability to delete records, add elements to your HTML5 page that accept input from the user and create a function that sends the new record for deletion to the REST server.

The figure below shows the Oracle JET Common Model CRUD application configured to allow record deletion. The user can check one or more departments in the list and click **Remove Department** to delete the record or records.

Remove	Department Id	Department Name	Location Id	Manager Id
<input type="checkbox"/>	10	Administration		
<input type="checkbox"/>	20	Marketing		
<input checked="" type="checkbox"/>	30	Transportation		
<input type="checkbox"/>	40	Shipping		
<input type="checkbox"/>	50	Human Resources		
<input type="checkbox"/>	60	Operations		
<input type="checkbox"/>	70	Inventory		
<input type="checkbox"/>	80	Sales	2500	2500
<input type="checkbox"/>	100	Finance		
<input type="checkbox"/>	110	Documentation		
<input type="checkbox"/>	130	Billing	1700	1700
<input type="checkbox"/>	140	Control And Credit	1700	1700

(Click on a Department Name to edit it)

To add the ability to delete records to the application's ViewModel and HTML5 page:

1. Add elements to the application's main page that identifies records marked for deletion and enables the user to perform an action to delete them.

The highlighted code in the example below adds the `Remove` column with a check box to the department list and adds the `Remove Department` button below the list.

```
<div id="mainContent" class="oj-flex-item oj-sm-12 oj-md-12 demo-page-content-area page-padding" style="display: none;">
    <div class="page-padding">
        <div id="deptList" class="oj-flex-item oj-md-9 oj-sm-12">
            <oj-table id="table" data="[[dataProvider]]"
                columns='[{"headerText": "Remove", "id": "column1",
                "sortable": "disabled"},
```

```

        {"headerText": "Department Id",
         "field": "DepartmentId", "sortable": "enabled"}, 
        {"headerText": "Department Name",
         "field": "DepartmentName", "sortable": "enabled"}, 
        {"headerText": "Location Id",
         "field": "LocationId"}, 
        {"headerText": "Manager Id",
         "field": "ManagerId"}]
    selectionMode='{"row": "none", "column": "none"}'
    row-
    renderer='[KnockoutTemplateUtils.getRenderer("row_tmpl", true)]]'>
    </oj-table>
    <br/>
    <oj-button id="deleteDept_btn" disabled="[[!somethingChecked()]]" on-oj-action="[[deleteDepartment]]">Remove
        Department</oj-button>
    </div>
    </div>
    </div>

<script type="text/html" id="row_tmpl">
    <tr>
        <td>
            <oj-checkboxset aria-hidden='true' on-value-changed="[$parent.enableDelete]" class='oj-checkboxset-no-chrome'>
                <oj-option :id="[[DepartmentId]]" value="[[DepartmentId]]"></oj-option>
            </oj-checkboxset>
        </td>
        <td>
            <div id='deptId' on-click="[[function(data, event) {$root.showChangeNameDialog(DepartmentId,data,event)}]]">
                <oj-bind-text value="[[DepartmentId]]"></oj-bind-text>
            </div>
        </td>
        <td>
            <div id="deptName" on-click="[[function(data, event) {$root.showChangeNameDialog(DepartmentId,data,event)}]]">
                <oj-bind-text value="[[DepartmentName]]"></oj-bind-text>
            </div>
        </td>
        <td>
            <div id="locId">
                <oj-bind-text value="[[LocationId]]"></oj-bind-text>
            </div>
        </td>
        <td>
            <div id="mgrId">
                <oj-bind-text value="[[ManagerId]]"></oj-bind-text>
            </div>
        </td>
    </tr>
</script>

```

The original table is modified to include the Remove column. The `row-renderer` attribute specifies the `row_tmpl` template to use for the row display. The template script adds the `checkbox` input element to the first column and the value of `DepartmentId`, `DepartmentName`, and `LocationId` to the remaining columns.

The button's click action is bound to the `deleteDepartment` function which is created in the next step.

2. Add code to the ViewModel to delete the record or records submitted by the user.

The highlighted code in the example below shows the `deleteDepartment()` function. In this example, the function calls the `collection.remove()` method which removes the model or models from the data collection. To delete the record from the data source, the function calls the `model.destroy()` method.

```
define(['knockout', 'ojs/ojknockouttemplateutils', 'ojs/ojmodel', 'ojs/ojcollectiondatatypeprovider', 'MockRESTServer'],
    function(ko, KnockoutTemplateUtils, model,
CollectionDataProvider) {
    function viewModel() {
        var self = this;
        self.KnockoutTemplateUtils = KnockoutTemplateUtils;
        self.serviceURL = 'http://mockrest/stable/rest/Departments';
        self.Departments = ko.observableArray([]);
        self.DeptCol = ko.observable();
        self.dataProvider = ko.observable();
        self.currentDeptName = ko.observable('default');
        self.workingId = ko.observable('');
        self.somethingChecked = ko.observable(false);

        self.fetch = function(successCallBack) {
            ... contents omitted
        }

        var Department = model.Model.extend({
            urlRoot: self.serviceURL,
            idAttribute: 'DepartmentId'
        });

        var myDept = new Department();

        var DeptCollection = Collection.extend({
            url: self.serviceURL + "?limit=50",
            model: myDept,
            comparator: 'DepartmentId'
        });

        self.DeptCol(new DeptCollection());
        self.dataProvider(new
CollectionDataProvider(self.DeptCol()));

        self.enableDelete = function(event) {
            self.somethingChecked(event && event.target &&
event.target.value && event.target.value.length);
            }.bind(this);

        self.deleteDepartment = function(event, data,
bindingContext) {
            var deptIds = [];
            deptIds = self.findDeptIds();
            var collection = data.DeptCol();
            deptIds.forEach(function(value, index, arr) {
                var model = collection.get(parseInt(value));
                if (model) {
                    collection.remove(model);
                    model.destroy();
                }
            });
        });
    }
});
```

```
        self.enableDelete();
        document.getElementById("table").refresh();
    }.bind(this);

    self.findDeptIds = function() {
        var selectedIdsArray = [];
        var divs =
document.querySelectorAll('input[type=checkbox]:checked');
        for (var i = 0; i < divs.length; i++) {
            selectedIdsArray.push(divs[i].id);
        }
        return selectedIdsArray;
    }
}
return {'deptVM': viewModel};
};


```

The `deleteDepartment()` function calls the `findDeptIds()` function which returns the list of selected departments marked for deletion. The `enableDelete()` function resets the check box after the department list is deleted.

11

Validating and Converting Input

Oracle JET includes validators and converters on a number of Oracle JET editable elements, including `oj-combobox`, `oj-input*`, and `oj-text-area`. You can use them as is or customize them for validating and converting input in your Oracle JET application. Some editable elements such as `oj-checkboxset`, `oj-radio-set`, and `oj-select` have a simple attribute for required values that implicitly creates a built-in validator.

 **Note:**

The `oj-input*` mentioned above refers to the family of input components such as `oj-input-date-time`, `oj-input-text`, and `oj-input-password`, among others.

Topics:

- [Typical Workflow for Validating and Converting Input](#)
- [About Oracle JET Validators and Converters](#)
- [About Oracle JET Converters](#)
- [About Oracle JET Validators](#)

Typical Workflow for Validating and Converting Input

Understand Oracle JET's validation and conversion framework before working with it. Optionally, learn how to create custom converters and validators to extend the framework.

To validate and convert input in Oracle JET, refer to the typical workflow described in the following table:

Task	Description	More Information
Understand Oracle JET's validation and conversion framework	Understand the validators and converters included in Oracle JET and identify when you might need a custom validator or converter.	About Oracle JET Validators and Converters
Use Oracle JET converters in your application	Use <code>ColorConverter</code> , <code>IntlNumberConverter</code> , and <code>IntlDateTimeConverter</code> to convert color, number, and date-time inputs to values expected by the view model.	About Oracle JET Converters
Use a custom converter in your Oracle JET application	How to reference a custom converter in an Oracle JET application.	Use Custom Converters in Oracle JET

Task	Description	More Information
Use Oracle JET validators in your application	Use <code>DateTimeRangeValidator</code> , <code>DateRestrictionValidator</code> , <code>LengthValidator</code> , <code>NumberRangeValidator</code> , <code>RegExpValidator</code> , and <code>RequiredValidator</code> to validate user input.	About Oracle JET Validators
Use a custom validator in your Oracle JET application	How to reference a custom validator in an Oracle JET application.	Use Custom Validators in Oracle JET

About Oracle JET Validators and Converters

Oracle JET provides converter classes that convert user input strings into the data type expected by the application and validator classes that enforce a validation rule on those input strings.

For example, you can use Oracle JET's `IntlDateTimeConverter` to convert a user-entered date to a `Date` object for use by the application's `ViewModel` and then use `DateTimeRangeValidator` to validate that input against a specified time range. You can also use converters to convert `Date` or `Number` objects to a string suitable for display or convert color object formats.

To retrieve the converter or validator factory for a registered type, Oracle JET provides the `Validation` class which includes methods to register and retrieve converter and validator factories.

If the converters or validators included in Oracle JET are not sufficient for your application, you can create custom converters or validators. Optionally, you can provide a custom factory that implements the contract for a converter using `ConverterFactory` or a validator using `ValidatorFactory` and register the converter or validator with the `Validation` class. The `Validation` class enables you to access your custom converter or validator using the same mechanisms as you would use with the Oracle JET standard converters and validators.

Topics:

- [About Validators](#)
- [About Converters](#)

About Validators

All Oracle JET editable elements support a `value` attribute and provide UI elements that allow the user to enter or choose a value. These elements also support other attributes that page authors can set that instruct the element how it should validate its value.

An editable element may implicitly create a built-in converter and/or built-in validators for its normal functioning when certain attributes are set.

For example, editable elements that support a `required` property create the required validator implicitly when the property is set to `true`. Other elements like `oj-input-`

date, oj-input-date-time, and oj-input-time create a datetime converter to implement its basic functionality.

Topics

- [About the Oracle JET Validators](#)
- [About Oracle JET Component Validation Attributes](#)
- [About Oracle JET Component Validation Methods](#)

About the Oracle JET Validators

The following table describes the Oracle JET validators and provides links to the API documentation:

Validator	Description	Link to API	Module
DateTimeRangeValidator	Validates that the input date is between two dates, between two times, or within two date and time ranges	DateTimeRangeValidator	ojvalidation-datetime-range
DateRestrictionValidator	Validates that the input date is not a restricted date	DateRestrictionValidator	ojvalidation-daterestriction
LengthValidator	Validates that an input string is within a specified length	LengthValidator	ojvalidation-length
NumberRangeValidator	Validates that an input number is within a specified range	NumberRangeValidator	ojvalidation-numberrange
RegExpValidator	Validates that the regular expression matches a specified pattern	RegExpValidator	ojvalidation-regexp
RequiredValidator	Validates that a required entry exists	RequiredValidator	ojvalidation-required

About Oracle JET Component Validation Attributes

The attributes that a component supports are part of its API, and the following validation specific attributes apply to most editable elements.

Element Attribute	Description
converter	When specified, the converter instance is used over any internal converter the element might create. On elements such as oj-input-text, you may need to specify this attribute if the value must be processed to and from a number or a date value.
countBy	When specified on LengthValidator, countBy enables you to change the validator's default counting behavior. By default, this property is set to codeUnit, which uses JavaScript's String length property to count a UTF-16 surrogate pair as length === 2. Set this to codePoint to count surrogate pairs as length ===1.
max	When specified on an Oracle JET element like oj-input-date or oj-input-number, the element creates an implicit range validator.
min	When specified on an Oracle JET element like oj-input-date or oj-input-number, the component creates an implicit range validator.

Element Attribute	Description
pattern	When specified on an Oracle JET element like <code>oj-input-text</code> , <code>oj-input-password</code> , or <code>oj-text-area</code> , the component creates an implicit <code>RegExp</code> validator using the pattern specified. If the regular expression pattern requires a backslash, while specifying the expression within an Oracle JET element, you need to use double backslashes. For more information, see Use Oracle JET Validators with Oracle JET Components .
placeholder	When specified, it displays placeholder values in most elements.
required	When specified on an Oracle JET element, the element creates an implicit required validator.
validators	When specified, the element uses these validators along with the implicit validators to validate the UI value. Can be implemented with <code>Validators</code> or <code>AsyncValidators</code> to validate the user input on the server asynchronously.

Some editable elements do not support specific validation attributes as they might be irrelevant to its intrinsic functioning. For example, `oj-radio-set` and `oj-checkbox-set` do not support a converter attribute since there is nothing for the converter to convert. For an exact list of attributes and how to use them, refer to the `Attributes` section in the element's API documentation. For Oracle JET API documentation, see [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#). Select the component you're interested in viewing from the API list.

About Oracle JET Component Validation Methods

Oracle JET editable elements support the following methods for validation purposes. For details on how to call this method, its parameters and return values, refer to the component's API documentation.

Element Method	Description
<code>refresh()</code>	Use this method when the DOM the element relies on changes, such as the <code>help</code> attribute tooltip on an <code>oj-label</code> changing due to a change in locale.
<code>reset()</code>	Use this method to reset the element by clearing all <code>messages</code> and <code>messagesCustom</code> attributes - and update the element's display value using the attribute value. User entered values will be erased when this method is called.
<code>validate()</code>	Use this method to validate the component using the current display value.

For details on calling a element's method, parameters, and return values, See the `Methods` section of the element's API documentation in [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#). You can also find detail on how to register a callback for or bind to the event and for information about what triggers the events. Select the component you're interested in viewing from the API list.

About Converters

The Oracle JET converters include date-time, number, and color converters and are described in the following table.

Converter	Description	Link to API
<code>ColorConverter</code>	Converts Color object formats	ColorConverter

Converter	Description	Link to API
IntlDateTimeConverter	Converts a string to a Date or a Date to a string	IntlDateTimeConverter
IntlNumberConverter	Converts a string to a number or formats a number or Number object value to a string	IntlNumberConverter

About Oracle JET Component Converter Options

Oracle JET Converters use the `options` attribute to allow a range of formatting options for color, number, and date and time values.

Color Converters

Color converters support the following options.

Option Name	Description
format	Sets the format of the converted color specification. Allowed values are "rgb" (default), "hsl", "hsv", "hex", and "hex3" (3 digit hex code).

For an exact list of options and methods and how to use them, refer to API documentation of the `ColorConverter` class. See [ColorConverter](#).

Number Converters

Number converters support a number of options that can be used to format decimal numbers, currency, percentages, or units. Some of the important options are:

Option Name	Description
style	Sets the style of number formatting. Allowed values are "decimal" (default), "currency", "percent" or "unit". For decimals, percentages, and units, the locale-appropriate symbols are used for decimal characters (point or comma) and grouping separators (if grouping is enabled).
currency	Mandatory when <code>style</code> is "currency". Specifies the currency that will be used when formatting the number. The value should be a ISO 4217 alphabetic currency code (such as USD or EUR).
currencyDisplay	Allowed values are "code", "name", and "symbol" (default). When <code>style</code> is <code>currency</code> , this option specifies if the currency is displayed as its name (such as Euro) an ISO 4217 alphabetic currency code (such as EUR), or the commonly recognized symbol (such as €).

Option Name	Description
unit	Mandatory when style is "unit". Allowed values are "byte" or "bit". This option is used for formatting only and cannot be used for parsing. Use this option to format digital units like 10Mb for bit unit or 10MB for byte unit. Note that scale is formatted automatically. For example, 1024 (with the byte unit set) is interpreted as 1 KB.
minimumIntegerDigits	Sets the minimum number of digits before the decimal place. The number is padded with leading zeros if it would not otherwise have enough digits. Allowed values are any integer from 1 to 21.
minimumFractionDigits	Sets the minimum number of digits after the decimal place. The number is padded with trailing zeros if it would not otherwise have enough digits. Allowed values are any integer from 0 to 20.
maximumFractionDigits	Sets the maximum number of digits after the decimal place. The number is rounded if it has more than the set maximum number of digits. Allowed values are any integer from 0 to 20.
pattern	Sets a pattern to use for the number, overriding other options. The pattern uses the symbols specified in the Unicode CLDR for numbers, percent, and currency formats. For the currency format, the currency option must be set. Use of the currency symbol (¤) indicates whether it will be displayed or not. For example, {pattern: '¤#,##0', currency: 'USD'}. For the percentage format, if the style option is set to 'percent', use of the percentage symbol (%) indicates whether it will be displayed or not. If the style option is not set to percent, the percentage symbol is required. For example, {style: 'percent', pattern: "#,##0"}. For the decimal or exponent patterns, the example is {pattern: "#,##0.00"} or {pattern: "0.##E+0"}.
roundingMode	Specifies the rounding behavior. Allowed values are HALF_UP, HALF_DOWN, and HALF_EVEN.
roundDuringParse	Specifies whether or not to round during parse. Defaults to false; the number converter rounds during format but not during parse.

For an exact list of options and methods and how to use them, refer to API documentation of the `IntlNumberConverter` class. See [IntlNumberConverter](#).

For an example illustrating the use of Number Converter options, see [Number Converter](#).

Date Time Converters

Date Time converters support a wide array of options that can be used to format date and time values in all common styles across locales. Some of the important options are:

Option Name	Description
year	Allowed values are the strings "2-digit" (00–99) and "numeric" (full year value, default).
month	Allowed values are the strings "2-digit" (01–12), "numeric" (variable digit value such as 1 or 11, default), "narrow" (narrow name such as J for January), "short" (abbreviated name such as Jan), and "long" (wide name such as January).
day	Allowed values are the strings "2-digit" (01–31) and "numeric" (variable digit value such as 1 or 18, default).
formatType	Determines the standard date and/or time format lengths to use. Allowed values are "date", "time", "datetime". When set, a value for dateFormat or timeFormat must be specified where appropriate.
dateFormat	Specifies the standard date format length to use when formatType is set to "date" or "datetime". Allowed values are "short" (default), "medium", "long", "full".
timeFormat	Specifies the standard time format length to use when formatType is set to "time" or "datetime". Allowed values are "short" (default), "medium", "long", "full".
hour	Allowed values are the strings "2-digit" (01–12 or 01–24) and "numeric" (variable digit value such as 1 or 23).
minute	Allowed values are the strings "2-digit" and "numeric". This value is always displayed as 2 digits (00–59).
second	Allowed values are the strings "2-digit" and "numeric". This value is always displayed as 2 digits (00–59).

For an exact list of options and methods and how to use them, refer to API documentation of the `IntlDateTimeConverter` class. See [IntlDateTimeConverter](#).

For an example illustrating the use of `DateTime` Converter options, see [DateTime Converter](#).

About Oracle JET Converters

The Oracle JET color, date-time, and number converters, `ColorConverter`, `IntlDateTimeConverter`, and `IntlNumberConverter`, extend the `Converter` object which defines a basic contract for converter implementations.

The converter API is based on the ECMAScript Internationalization API specification (ECMA-402 Edition 1.0) and uses the Unicode Common Locale Data Repository (CLDR) for its locale data. Both converters are initialized through their constructors, which accept options defined by the API specification. For additional information about the ECMA-402 API specification, see <http://www.ecma-international.org/ecma-402/1.0>. For information about the Unicode CLDR, see <http://cldr.unicode.org>.

The Oracle JET implementation extends the ECMA-402 specification by introducing additional options, including an option for user-defined patterns. For the list of additional options, see the [ColorConverter](#), [IntlDateTimeConverter](#), and [IntlNumberConverter](#) API documentation.

For examples that illustrate the date-time and number converters in action, see the [Converters](#) section in the Oracle JET Cookbook. For examples using the color converter, see the [Color Palette](#) and [Color Spectrum](#) Cookbook samples.

 **Note:**

The bundles that hold the locale symbols and data used by the Oracle JET converters are downloaded automatically based on the locale set on the page when using RequireJS and the `ojs/ojvalidation-datetime` or `ojs/ojvalidation-number` module. If your application does not use RequireJS, the locale data will not be downloaded automatically.

You can use the converters with an Oracle JET component or instantiate and use them directly on the page.

Topics:

- [Use Oracle JET Converters with Oracle JET Components](#)
- [About Oracle JET Converters Lenient Parsing](#)
- [Understand Time Zone Support in Oracle JET](#)
- [Use Custom Converters in Oracle JET](#)
- [Use Oracle JET Converters Without Oracle JET Components](#)

Use Oracle JET Converters with Oracle JET Components

Oracle JET elements that accept user input, such as `oj-input-date`, already include an implicit converter that is used when parsing user input. However, you can also specify an explicit converter on the element which will be used instead when converting data from the model for display on the page and vice versa. An explicit converter is required if you want to include time zone data.

For example, the following code sample shows a portion of a form containing an `oj-input-date` component that uses the default converter supplied by the component implicitly. The highlighted code shows the `oj-input-date` component.

```
<oj-form-layout id="datetime-converter-example">
    ...
    <oj-input-date id="date1" value="{{date}}" name="date1"
        label-hint="input date with no converter"
        help.instruction="enter a date in your preferred format and we
        will attempt to figure it out">
    </oj-input-date>
</oj-form-layout>
```

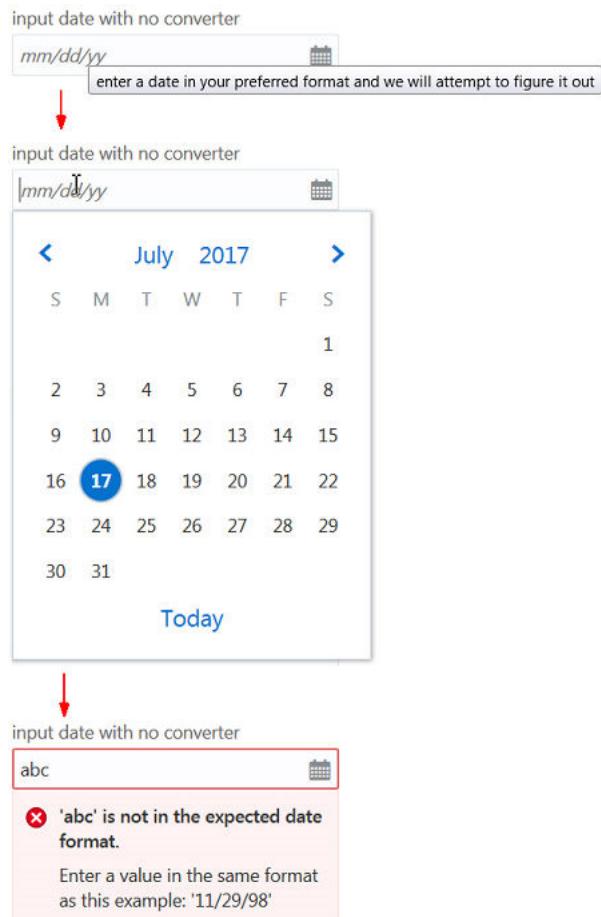
The script to create the view model for this example is shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/
ojdatetimepicker', 'ojs/ojlabel'],
```

```
function(ko, Bootstrap)
{
    function MemberViewModel()
    {
        var self = this;
        self.date = ko.observable();
        self.datetime = ko.observable();
        self.time = ko.observable();
    };

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new MemberViewModel(),
document.getElementById('datetime-converter-example'));
        }
    );
});
```

When the user runs the page, the `oj-input-date` element displays an input field with the expected date format. In this example, the element also displays a hint when the user hovers over the input field, and displays a calendar when the user clicks in the input field. If the user inputs data that is not in the expected format, the built-in converter displays an error message with the expected format.

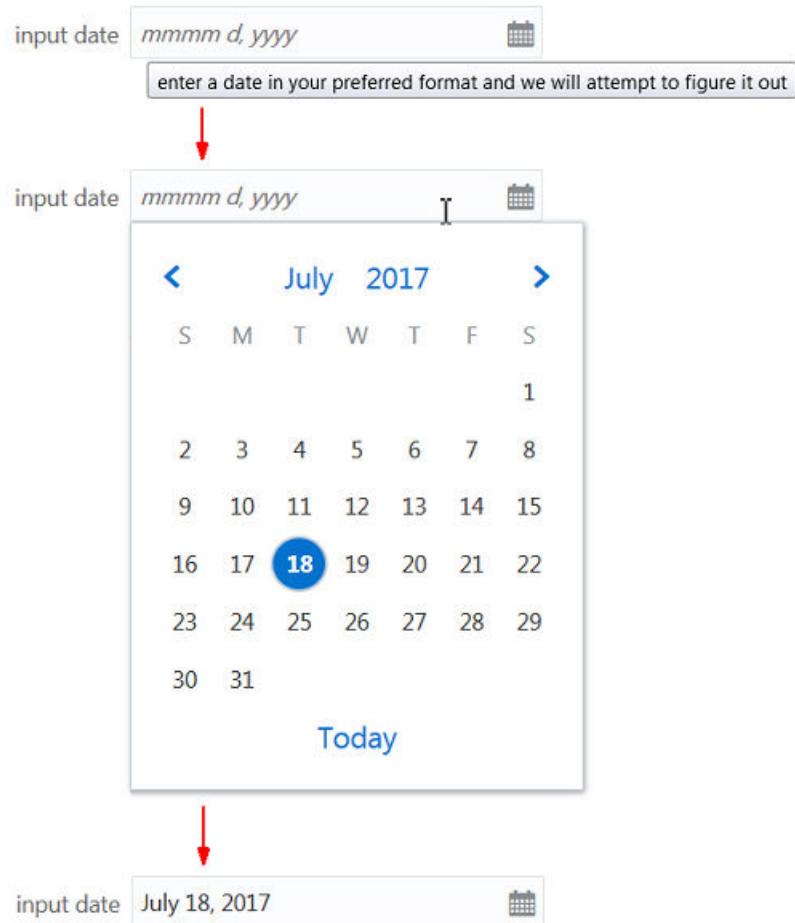


The error that the converter throws when there are errors during parsing or formatting operations is represented by the `ConverterError` object, and the error message is represented by an object that duck-types `Message`. The messages that Oracle JET converters use are resources that are defined in the translation bundle included with Oracle JET. For more information about messaging in Oracle JET, see [Working with User Assistance](#).

You can also specify the converter directly on the element's `converter` attribute, if it exists. The code excerpt below defines another `oj-input-date` element on the sample form and specifies the `IntlDateTimeConverter` converter with options that will convert the user's input to a numeric year, long month, and numeric day according to the conventions of the locale set on the page. The `options` parameter is an object literal that contains the ECMA-402 options as name-value pairs.

```
<div class="oj-flex">
  <div class="oj-flex-item">
    <oj-label for="date2">input date</oj-label>
  </div>
  <div class="oj-flex-item">
    <oj-input-date id="date2" value="{{date}}" name="date2"
      help.instruction="enter a date in your preferred format and
      we will attempt to figure it out"
      converter= '{
        "type": "datetime",
        "options": {"year": "numeric", "month": "long", "day":
        "numeric"}}'>
    </oj-input-date>
  </div>
</div>
```

When the user runs the page in the `en-us` locale, the `oj-input-date` element displays an input field that expects the user's input date to be in the `mmmm d, yyyy` format. The converter will accept alternate input if it makes sense, such as `18/07/17` (`MM/dd/yy`), and perform the conversion, but will throw an error if it cannot parse the input. For details about Oracle JET converters and lenient parsing support, see [About Oracle JET Converters Lenient Parsing](#).



Parsing of narrow era, weekday, or month name is not supported because of ambiguity in choosing the right value. For example, if you initialize the date time converter with options {weekday: 'narrow', month: 'narrow', day: 'numeric', year: 'numeric'}, then for the en-US locale, the converter will format the date representing May 06, 2014 as T, M 6, 2014, where T represents Tuesday. If the user inputs T, M 6, 2014, the converter can't determine whether the user meant Thursday, March 6, 2014 or Tuesday, May 6, 2014. Therefore, Oracle JET expects that user input be provided in either their short or long forms, such as Tues, May 06, 2014.

For additional details about the `IntlDateTimeConverter` and `IntlNumberConverter` component options, see [IntlDateTimeConverter](#) and [IntlNumberConverter](#).

About Oracle JET Converters Lenient Parsing

The Oracle JET converters support lenient number and date parsing when the user input does not exactly match the expected pattern. The parser does the lenient parsing based on the leniency rules for the specific converter.

`IntlDateTimeConverter` provides parser leniency when converting user input to a date and enables the user to:

- Input any character as a separator irrespective of the separator specified in the associated pattern. For example, if the expected date pattern is set to y-M-d, the

date converter will accept the following values as valid: 2020-06-16, 2013/06-16, and 2020aaa06xxx16. Similarly, if the expected time pattern is set to HH:mm:ss, the converter will accept the following values as valid: 12.05.35.

- Specify a 4-digit year in any position relative to day and month. For example, both 11-2013-16 and 16-11-2013 are valid input values.
- Swap month and day positions, as long as the date value is greater than 12 when working with the Gregorian calendar. For example, if the user enters 2020-16-06 when y-M-d is expected, the converter will autocorrect the date to 2020-06-16. However, if both date and month are less or equal to 12, no assumptions are made about the day or month, and the converter parses the value against the exact pattern.
- Enter weekday and month names or mix short and long names anywhere in the string. For example, if the expected pattern is E, MMM, d, y, the user can enter any of the following dates:

```
Tue, Jun 16 2020
Jun, Tue 2020 16
2020 Tue 16 Jun
```

- Omit weekdays. For example, if the expected pattern is E, MMM d, y, then the user can enter Jun 16, 2020, and the converter autocorrects the date to Tuesday, Jun 16, 2020. Invalid weekdays are not supported. For instance, the converter will throw an exception if the user enters Wednesday, Jun 16, 2020.

`IntlNumberConverter` supports parser leniency as follows:

- If the input does not match the expected pattern, Oracle JET attempts to locate a number pattern within the input string. For instance, if the pattern is #,##0.0, then the input string abc-123.45de will be parsed as -123.45.
- For the currency style, the currency symbol can be omitted. Also, the negative sign can be used instead of a negative prefix and suffix. As an example, if the pattern option is specified as "\u00a4#,##0.00;(\u00a4#,##0.00)", then (\$123), (123), and -123 will be parsed as -123.
- When the style is percent, the percent sign can be omitted. For example, 5% and 5 will both be parsed as 0.05.

Understand Time Zone Support in Oracle JET

By default, the `oj-input-date-time` and `oj-input-time` elements and `IntlDateTimeConverter` support only local time zone input. You can add time zone support by including the `ojs/ojtimezonedata` module and creating a converter with the desired pattern.

Oracle JET supports time zone conversion and formatting using the following patterns:

Token	Description	Example
z, zz, zzz	Abbreviated time zone name, format support only	PDT, PST
zzzz	Full time zone name, format support only	Pacific Standard Time, Pacific Daylight Time
Z, ZZ, ZZZ	Sign hour minutes	-0800
X	Sign hours	-08

Token	Description	Example
XX	Sign hours minutes	-0800
XXX	Sign hours:minutes	-08:00
VV	Time Zone ID	America/Los Angeles

The image below shows the basic `oj-input-date-time` element configured for time zone support. In this example, the component is converted using the Z pattern.

InputDateTime Timezone converter

06/16/20 12:00:00 PM +0800


Pattern options:

MM/dd/yy hh:mm:ss a Z
▼

isoStrFormat options:

offset
▼

Current dateTime value is:

2020-06-16T12:00:00+08:00

The `oj-input-date-time` element is initialized with its `converter` attribute, in this case a method named `dateTimeConverter`.

```
<div id="div1">
  <oj-label for="timezone">InputDateTime Timezone converter</oj-label>
  <oj-input-date-time id="timezone" value="{{dateTimeValue}}"
    converter=[[dateTimeConverter]]>
  </oj-input-date-time>
  <br/><br/>
  <p>
    <oj-label for="patternSelector">Pattern options:</oj-label>
    <oj-combobox-one id="patternSelector" value="{{patternValue}}>
      <oj-option value="MM/dd/yy hh:mm:ss a Z">MM/dd/yy hh:mm:ss a Z</oj-option>
      <oj-option value="MM-dd-yy hh:mm:ss a VV">MM-dd-yy hh:mm:ss a VV</oj-option>
      <oj-option value="MM-dd-yy hh:mm X">MM-dd-yy hh:mm X</oj-option>
    </oj-combobox-one>
  </p>
  <p>
    <oj-label for="isoStrFormatSelector">isoStrFormat options:</oj-label>
    <oj-combobox-one id="isoStrFormatSelector"
      value="{{isoStrFormatValue}}>
```

```

<oj-option value="offset">offset</oj-option>
<oj-option value="zulu">zulu</oj-option>
<oj-option value="local">local</oj-option>
</oj-combobox-one>
</p>
<br/>
<span class="oj-label">Current dateTime value is: </span>
<span><oj-bind-text value="[[dateTimeValue]]"></oj-bind-text></span>
//...contents omitted
</div>
```

The ViewModel contains the `dateTimeConverter()` definition. Note that you must also add the `ojs/ojconverter-datetime` module to use its `DateTimeConverter` API and add the `ojs/timezonedata` module to your `RequireJS` definition to access the time zone data files.

```

require(['knockout', 'ojs/ojbootstrap',
        'ojs/ojconverter-datetime', 'ojs/ojknockout',
        'ojs/ojdatetimepicker', 'ojs/ojselectcombobox',
        'ojs/ojtimezonedata', 'ojs/ojlabel'],
       function (ko, Bootstrap, DateTimeConverter,
       {
           function FormatModel()
           {
               var self = this;

               this.dateTimeValue = ko.observable("2020-06-16T20:00:00-08:00");
               this.patternValue = ko.observable("MM/dd/yy hh:mm:ss a Z");
               this.isoStrFormatValue = ko.observable("offset");
               this.dateTimeConverter = ko.observable(
                   new DateTimeConverter.IntlDateTimeConverter(
                   {
                       pattern : self.patternValue(),
                       isoStrFormat: self.isoStrFormatValue(),
                       timeZone:'Etc/GMT-08:00'
                   }));
               //Note that ojCombobox's value is always encapsulated in an
               array
               this.patternValue.subscribe(function (newValue) {
                   this.dateTimeConverter
                   new DateTimeConverter.IntlDateTimeConverter(
                   {
                       pattern : newValue,
                       isoStrFormat: self.isoStrFormatValue(),
                       timeZone:'Etc/GMT-08:00'
                   });
                   }.bind(this));
               this.isoStrFormatValue.subscribe(function (newValue) {
                   this.dateTimeConverter(
                   new DateTimeConverter.IntlDateTimeConverter(
                   {
                       pattern : self.patternValue(),

```

```
        isoStrFormat: newValue,
        timeZone:'Etc/GMT-08:00'
    }));
}.bind(this));

//....contents omitted
Bootstrap.whenDocumentReady().then(
    function ()
    {
        ko.applyBindings(new FormatModel(),
document.getElementById('div1'));
    }
);
});
```

For an additional example illustrating how to add time zone support to `oj-input-date-time` and `oj-input-time` elements, see [Input Date and Time - Time Zone](#).

Use Custom Converters in Oracle JET

You can create custom converters in Oracle JET by extending `Converter` or by duck typing it. You can also create a custom converter factory to register the converter with Oracle JET and make it easier to instantiate the converter.

Custom converters can be used with Oracle JET components, provided they don't violate the integrity of the component. As with the built-in Oracle JET converters, you can also use them directly on the page.

The figure below shows an example of a custom converter used to convert the current date to a relative term. The `Schedule For` column uses a `RelativeDateTimeConverter` to convert the date that the page is run in the `en-US` locale to display Today, Tomorrow, and the date in two days.

Schedule For	Start Time	Class Name	Instructor	Duration	
Today	2:30 PM	Early Morning Power Hour	Marcus Levi	1 hour	
	4:00 PM	Beginners Yoga	Rachel Donald	1 hour & 15 minutes	
	8:00 PM	Spinning	Sign Up	Chris Sharp	45 minutes
	9:00 PM	Pilates Mat	Sue Miller	1 hour	
	10:30 AM	Salsa Dance	Paige Davis	1 hour	
	11:30 AM	Zumba	Lisa Rogers	1 hour	
Tomorrow	12:30 PM	Barre Fit	Sign Up	Paige Davis	1 hour
	4:00 PM	Hatha Yoga	Ramu	1 hour & 15 minutes	
	8:00 PM	Body Conditioning	Birgette Peters	1 hour	
	9:00 PM	Stretch and Roll	Sue Leon	1 hour	
	10:30 AM	Core	Jose Marquis	1 hour	
	11:30 AM	Spinning	Sign Up	Giselle Chan	1 hour
In 2 Days	12:30 PM	Body Conditioning	Chris Sharp	1 hour	
	4:00 PM	Yoga Flow	Chris Carington	1 hour & 15 minutes	
	8:00 PM	Pilates Mat	Sue Miller	1 hour	
	9:00 PM	Basic Step	Jose Marquis	1 hour	
	10:30 AM	Zumba	Lisa Rogers	1 hour	
	11:30 AM	Hip-Hop	Sign Up	Dione	1 hour

To create and use a custom converter in Oracle JET:

1. Define the custom converter.

The code sample below shows a portion of the `RelativeDateTimeConverter` definition. The converter wraps the Oracle JET `IntlDateTimeConverter` by providing a specialized `format()` method that turns dates close to the current date into relative terms for display. For example, in the `en-US` locale, the relative terms will display `Today`, `Yesterday`, and `Tomorrow`. If a relative notation for the date value does not exist, then the date is formatted using the regular Oracle JET `format()` method supported by the Oracle JET `IntlDateTimeConverter`.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojconverter-datetime',
        'ojs/ojarraydataprovider', 'ojs/ojconverterutils-i18n',
        'ojs/ojknockout', 'ojs/ojtable', 'ojs/ojbutton'],
       function (ko, Bootstrap, DateTimeConverter, ArrayDataProvider,
ConvertI18nUtils)
{
    ...
    ...contents omitted
    function RelativeDateTimeConverter(options) {
        this.Init(options);
    };
    // Defines default option values
});
```

```

        RelativeDateTimeConverter._DEFAULT_RELATIVE_DATETIME_CONVERTER_OPTIONS
=
{
    'formatUsing' : "displayName",
    'dateField' : "week"
};

// Initializes converter instance with the set options
RelativeDateTimeConverter.prototype.Init = function(options)
{
    this._options = options;
    //create datetime converter and use this as the wrapper converter.
    this._wrappedConverter = new
DateTimeConverter.IntlDateTimeConverter(options);
};

// Returns the options set on the converter instance
RelativeDateTimeConverter.prototype.getOptions = function ()
{
    return this._options;
};

// Does not support parsing
RelativeDateTimeConverter.prototype.parse = function(value)
{
    return null;
};

// Formats a value using the relative format options. Returns the
formatted
// value and a title that is the actual date, if formatted value is a
relative date.
RelativeDateTimeConverter.prototype.format = function(value)
{
    var base;
    var formattedRelativeDate;

    // We get our wrapped converter and call its formatRelative function
    and store the
    // return value ("Today", "Tomorrow" or null) in formatted variable
    // See IntlDateTimeConverter#formatRelative(value, relativeOptions)
    // where relativeOptions has formatUsing and dateField options.
    dateField is
    // 'day', 'week', 'month', or 'year'.
    formattedRelativeDate = this._getWrapped().formatRelative(value,
this._getRelativeOptions());

    // We get our wrapped converter and call its format function and
    store the returned
    // string in base variable. This will be the actual date, not a
    relative date.
    base = this._getWrapped().format(value);

    // Capitalize the first letter of the string
    if (formattedRelativeDate && typeof formattedRelativeDate
== "string")
    {
        formattedRelativeDate = formattedRelativeDate.replace(/(\w)(\w*)/g,
function (match, i, r) {
            return i.toUpperCase() + (r !== null ? r : "");
        });
    }
};

```

```

        }
        return {value: formatted || base, title: formattedRelativeDate ?
base : ""}
    };

    // Returns a hint
    RelativeDateTimeConverter.prototype.getHint = function ()
    {
        return null;
    };

    RelativeDateTimeConverter.prototype._getWrapped = function ()
    {
        return this._wrappedConverter;
    };

    RelativeDateTimeConverter.prototype._getRelativeOptions = function ()
    {
        var options = this._options;
        var relativeOptions = {};

        if (options && options["relativeField"])
        {
            relativeOptions['formatUsing'] = "displayName";
            relativeOptions['dateField'] = options["relativeField"];
        }
        else
        {
            relativeOptions =
RelativeDateTimeConverter._DEFAULT_RELATIVE_DATETIME_CONVERTER_OPTIONS;
        }

        return relativeOptions;
    };
    ... contents omitted
});

```

The custom converter relies on the `IntlDateTimeConverter` converter's `formatRelative()` method. For additional details about the `IntlDateTimeConverter` converter's supported methods, see the [IntlDateTimeConverter API documentation](#).

2. Add code to your application that uses the custom converter.

The code sample below shows how you could add code to your script to use the custom converter.

```

var relativeDayOptions = {relativeField: 'day', year: "numeric",
                           month: "numeric", day: "numeric"};
var rdConverter = new RelativeDateTimeConverter(relativeDayOptions);

```

3. Add the Oracle JET element or elements that will use the custom converter to your page.

For the code sample that creates the `obj-table` element and displays the `Schedule` For column in relative dates, see [Converters \(Custom\)](#).

Use Oracle JET Converters Without Oracle JET Components

If you want to use a converter without binding it to an Oracle JET component, create the converter using the constructor for the converter of your choice.

The Oracle JET Cookbook includes the [Converters Factory](#) demo that shows how to use the number and date time converters directly in your pages without binding them to an Oracle JET component. In the demo image, the salary is a number formatted as currency, and the start date is an ISO string formatted as a date.



The sample code below shows a portion of the viewModel that defines a salaryConverter to format a number as currency and a dateConverter that formats the start date using the date format style and medium date format.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojconverter-number',
        'ojs/ojconverter-datetime', 'ojs/ojknockout'],
       function(ko, Bootstrap, NumberConverter, DateTimeConverter)
{
    function DemoViewModel()
    {
        var self = this;
        // for salary fields
        var salOptions = {style: 'currency', currency: 'USD'};
        var salaryConverter = new
NumberConverter.IntlNumberConverter(salOptions);

        self.amySalary = ko.observable(salaryConverter.format(125475.00));
        self.garySalary = ko.observable(salaryConverter.format(110325.25));

        // for date fields
        var dateOptions = {formatStyle: 'date', dateFormat: 'medium'};
        var dateConverter = new
DateTimeConverter.IntlDateTimeConverter(dateOptions);

        self.amyStartDate = ko.observable(dateConverter.format("2020-01-02"));
        self.garyStartDate = ko.observable(dateConverter.format("2009-07-25"));
        ... contents omitted
    });
}
```

The code sample below shows the portion of the markup that sets the display output to the formatted values contained in amySalary, amyStartDate, garySalary, garyStartDate.

```
<td>
<div class="oj-panel oj-panel-alt4 demo-panel-customizations">
  <h3 class="oj-header-border">Amy Flanagan</h3>
  
  <p>Product Manager</p>
  <span style="white-space: nowrap;"><b>Salary</b>:</span>
  <span>
    <ojs-bind-text value="[[amySalary]]"></ojs-bind-text>
  </span>
</div>
```

```
</span>
<br />
<span style="white-space:nowrap;"><b>Joined</b>:<br />
<span>
<ojo-bind-text value="[[amyStartDate]]"></ojo-bind-text>
</span>
<br />
</div>
</td>
<td>
<div class="oj-panel oj-panel-alt2 demo-panel-customizations">
<h3 class="oj-header-border">Gary Fontaine</h3>

<p>Sales Associate</p>
<span style="white-space:nowrap;"><b>Salary</b>:<br />
<span>
<ojo-bind-text value="[[garySalary]]"></ojo-bind-text>
</span>
<br />
<span style="white-space:nowrap;"><b>Joined</b>:<br />
<span>
<ojo-bind-text value="[[garyStartDate]]"></ojo-bind-text>
</span>
<br />
</div>
</td>
```

About Oracle JET Validators

Oracle JET validators provide properties that allow callers to customize the validator instance. The properties are documented as part of the validators' API. Unlike converters where only one instance of a converter can be set on an element, page authors can associate one or more validators with an element.

When a user interacts with the element to change its value, the validators associated with the element are run in order. When the value violates a validation rule, the `value` attribute is not populated, and the validator highlights the element with an error.

You can use the validators with an Oracle JET element or instantiate and use them directly on the page.

Topics:

- [Use Oracle JET Validators with Oracle JET Components](#)
- [Use Custom Validators in Oracle JET](#)
- [About Asynchronous Validators](#)

Use Oracle JET Validators with Oracle JET Components

Oracle JET editable elements, such as `oj-input-text` and `oj-input-date`, set up validators both implicitly, based on certain attributes they support such as `required`, `min`, `max`, and so on, and explicitly by providing a means to set up one or more validators using the component's `validators` attribute. As with the Oracle JET

converters, the validators attribute can be specified either using JSON array notation or can be an array of actual validator instances.

For example, the following code sample shows a portion of a form containing an `oj-input-date` element that uses the default validator supplied by the component implicitly. The highlighted code shows the HTML5 attribute set on the `oj-input-date` element. When the `oj-input-date` reads the `min` attribute, it creates the implicit `DateTimeRangeValidator`.

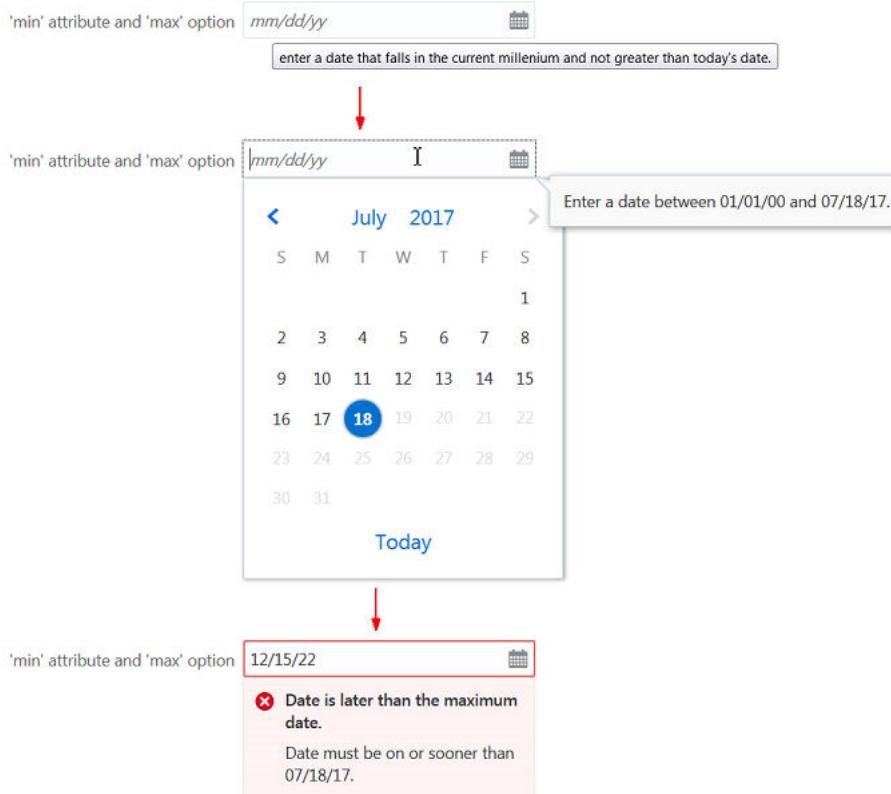
```
<oj-form-layout id="validator-example" label-edge="inside">
  <oj-input-date id="dateTimeRange1" value="{{dateValue1}}"
min="2000-01-01T08:00:00.000"
    help.instruction="Enter a date that falls in the current
millennium and not greater than today's date."
    max= '[{todayIsoDate}]' label-hint="'min' attribute and 'max'
option">
  </oj-input-date>
</oj-form-layout>
```

The script to create the view model for this example is shown below.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojconverterutils-i18n',
        'ojs/ojasyncvalidator-datetimepicker', 'ojs/ojconverter-datetime',
        'ojs/ojknockout', 'ojs/ojinputnumber', 'ojs/ojinputtext',
        'ojs/ojdatetimepicker', 'ojs/ojlabel', 'ojs/ojformlayout'],
       function(ko, Bootstrap, ConverterUtilsI18n, AsyncDateTimeRangeValidator,
DateTimeConverter)
{
  function DemoViewModel()
  {
    var self = this;
    self.dateValue1 = ko.observable();
    self.todayIsoDate =
ko.observable(ConverterUtilsI18n.IntlConverterUtils.dateToLocalIso(new Date()));
  }

  Bootstrap.whenDocumentReady().then(
    function ()
    {
      ko.applyBindings(new DemoViewModel(),
document.getElementById('validator-example'));
    }
  );
});
```

When the user runs the page, the `oj-input-date` element displays an input field with the expected date format. The `help.instruction` attribute set on the element displays as a tooltip upon hovering. When the user clicks on the field, the validator hint provided by the implicitly created `DateTimeRangeValidator` is shown in a note window, along with a calendar popup. If the user inputs data that is not within the expected range, the built-in validator displays an error message with the expected range.



The error thrown by the Oracle JET validator when validation fails is represented by the `ValidatorError` object, and the error message is represented by an object that duck-types `Message`. The messages and hints that Oracle JET validators use when they throw an error are resources that are defined in the translation bundle included with Oracle JET. For more information about messaging in Oracle JET, see [Working with User Assistance](#).

You can also specify the validator on the element's `validators` attribute, if it exists. The code sample below adds another `oj-input-date` element to the sample form and calls a function which specifies the `DateTimeRangeValidator` validator (`dateTimeRange`) in the `validators` attribute.

```
<oj-form-layout id="validator-example" label-edge="inside">
  <oj-input-date id="dateTimeRange2" value="{{dateValue2}}"
    validators="[[validators]]"
    label-hint="'validators' attribute">
  </oj-input-date>
</oj-form-layout>
```

The highlighted code below shows the additions to the `viewModel`, including the defined function, with options that set the valid minimum and maximum dates and a hint that displays when the user sets the focus in the field.

```
require(['knockout', 'ojs/ojbootstrap', 'ojs/ojconverterutils-i18n',
        'ojs/ojasyncvalidator-datetime-range', 'ojs/ojconverter-datetime',
        'ojs/ojknockout', 'ojs/ojinputnumber', 'ojs/ojinputtext',
        'ojs/ojdatetimepicker', 'ojs/ojlabel', 'ojs/ojformlayout'],
       function(ko, Bootstrap, ConverterUtilsI18n, AsyncDateTimeRangeValidator,
               DateTimeConverter)
```

```

{
    function DemoViewModel()
    {
        var self = this;
        self.dateValue1 = ko.observable();
        self.todayIsoDate =
            ko.ConverterUtilsI18n.IntlConverterUtils.dateToLocalIso(new Date());
        self.millenniumStartIsoDate =
            ko.observable(ConverterUtilsI18n.IntlConverterUtils.dateToLocalIso(new
            Date(2000, 00, 01)));
        self.validators = ko.computed(function()
        {
            return [
                new AsyncDateTimeRangeValidator({
                    max: this.todayIsoDate(),
                    min: this.millenniumStartIsoDate(),
                    hint: {
                        'inRange': 'Custom Hint: Enter a date that falls in the current
millennium.'
                    },
                    messageSummary: {'rangeUnderflow': 'Custom: Oops!', 'rangeOverflow': 'Custom: Oops!'},
                    converter: new DateTimeConverter.IntlDateTimeConverter({"day": "2-
digit", "month": "2-digit", "year": "2-digit"})
                })
            ];
        }).bind(this));
    };

    Bootstrap.whenDocumentReady().then(
        function ()
        {
            ko.applyBindings(new DemoViewModel(),
            document.getElementById('validator-example'));
        }
    );
});

```

When the user runs the page for the en-US locale, the `oj-input-date` element displays an input field that expects the user's input date to be between 01/01/00 and the current date. When entering a date value into the field, the date converter will accept alternate input as long as it can parse it unambiguously. This offers end users a great deal of leniency when entering date values.

For example, typing 1-2-3 will convert to a Date that falls on the 2nd day of January, 2003. If the Date value also happens to fall in the expected Date range set in the validator, then the value is accepted. If validation fails, the component will display an error.

Oracle JET elements can also use a `regExp` validator. If the regular expression pattern requires a backslash, while specifying the expression within an Oracle JET element, you need to use double backslashes. For example:

```

<oj-input-text
    ...
    validators='[{
        "type": "regExp",
        "options": {
            "pattern": "\\\d+(\\.\\\\d\\{1,2\\})?"
        }
    }]

```

```
"messageDetail": "Enter a valid number with up to 2 digits of decimal"
}
}]'>
</oj-input-text>
```

The options that each validator accepts are specified in [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#).

The Oracle JET Cookbook contains the complete example used in this section as well as examples that show the built-in validators for date restrictions, length, number range, regular expression, and required fields. For details, see [Validators](#).

For more information about Oracle JET component validation, see [Understand How Validation and Messaging Works in Oracle JET Editable Components](#).

Use Custom Validators in Oracle JET

You can create custom validators in Oracle JET by extending `Validator` or by duck typing it.

Custom validators can be used with Oracle JET components, provided they don't violate the integrity of the component. As with the built-in Oracle JET validators, you can also use them directly on the page. For information about messaging in Oracle JET, see [Use the `messages-custom` Attribute](#).

The figure below shows an example of a custom validator used to validate password entries. If the user's password doesn't match, the validator displays an error message.

Password *

Enter at least 6 characters including a number, one uppercase and lowercase letter

Create

>Password *

Confirm Password

•••••

✖ Error

The passwords must match!

Create

To create and use a custom validator in Oracle JET:

- 1.** Define the custom validator.

The highlighted code in the sample below shows the definition for the `equalToPassword` custom validator used in the figure above. The validator duck types the Validator contract. Because the validator provides the methods expected of a validator, Oracle JET accepts it.

```
function DemoViewModel ()  
{  
    var self = this;  
    self.password = ko.observable();  
    self.passwordRepeat = ko.observable();  
  
    // When password observable changes, validate the Confirm Password  
    // component  
    // if it holds a non-empty value.  
    self.password.subscribe (function (newValue)  
    {  
        var cPassword = document.getElementById("cpassword");  
        var cpUIVal = cPassword.value;  
  
        if (newValue && cpUIVal)  
        {  
            cPassword.validate();  
        }  
    });  
  
    // A validator associated to the Confirm Password field, that compares  
    // the value in the password observable matches the value entered in the  
    // Confirm Password field. getHint is a required method and may return  
    null.  
  
    self.equalToPassword = {  
  
        validate: function(value)  
        {  
            var compareTo = self.password.peek();  
            if (!value && !compareTo)  
                return true;  
            else if (value !== compareTo)  
            {  
                throw new Error(bundle['app']['validator-equalTo']['summary']);  
            }  
            return true;  
        },  
        getHint: function()  
        {  
            return null;  
        }  
    };  
}  
  
Bootstrap.whenDocumentReady().then(  
    function ()  
    {  
        ko.applyBindings(new DemoViewModel(), document.getElementById('custom-  
validator-example'));  
    }  
);  
});
```

2. Add code to your application that uses the custom validator.

The code sample below shows how you could add code to your page to use the custom validator. In this example, both input fields are defined as `oj-input-password` elements. The first `oj-input-password` element uses `RegExpValidator` to validate that the user has input a password that meets the application's password requirements. The second `oj-input-password` element uses the `equalToPassword` validator to verify that the password in the second field is equal to the password entered in the first field.

```
<oj-form-layout id="custom-validator-example" label-edge="inside">
  <oj-input-password id="password" name="password" required
  value="{{password}}"
  label-hint="Password"
  help.instruction="Enter at least 6 characters including a number, one
uppercase and lowercase letter"
  validators='[{
    "type": "regExp",
    "options": {
      "pattern": "(?=.*\\d)(?=.*[a-z])(?=.*[A-Z]).{6,}",
      "label": "Password",
      "messageSummary": "{label} too Weak",
      "messageDetail": "You must enter a password that meets our minimum
security requirements."}]'
  </oj-input-password>
  <oj-input-password id="cpassword" name="cpassword"
  value="{{passwordRepeat}}"
  label-hint="Confirm Password"
  validators="[[equalToPassword]]"></oj-input-password>
</oj-form-layout>
```

For the complete code sample used in this section, see [Validators \(Custom\)](#).

About Asynchronous Validators

Oracle JET input components support asynchronous server-side validation via the `validators` attribute. That means you can check input values against server data without the need to submit a form or refresh a page.

For example, in a form that collects new user data, you can have a validator on the e-mail field to check if the input value has been registered previously.

You can also set number range validators that check against volatile data. For example, on an e-commerce website, you can check the user's cart against the available inventory, and inform the user if the goods are unavailable without them submitting the cart for checkout.

The Oracle JET Cookbook has a sample that uses the `validators` attribute and dummy data to simulate server-side validation.

The following code shows an `oj-input-text` element with the `validators` attribute set to `validators` and `asyncValidator` observables in the JavaScript code. The `validators` attribute must duck-type `AsyncValidator` to fulfill the API contract required to create the asynchronous validator.

```
<oj-form-layout id="f11" label-edge="top">
  <oj-input-text id="text-input" required
  label-hint="Quantity Limit"
  on-valid-changed="[[validChangedListener]]"
```

```

    validators="[[validators, asyncValidator]]"
    value="{{quantityLimit}}"
    converter= '{
      "type": "number",
      "options": {"style": "currency", "currency": "USD",
      "currencyDisplay": "code", "pattern": "\u00a3 ##,##0.00"} }'>
  </oj-input-text>
</oj-form-layout>
```

The following JavaScript code shows the number range validator created in the `asyncValidator` object that returns a Promise.

```

self.asyncValidator = {
  // 'validate' is a required method
  // that is a function that returns a Promise
  validate: function (value) {
    // used to format the value in the validation error message.
    var converterOption =
    {
      style: 'currency',
      currency: 'USD',
      currencyDisplay: 'code',
      pattern: '\u00a3 ##,##0.00'
    };

    // As of JET 8.0, by default JET's validators are AsyncValidators.
    var numberRangeValidator =
      new AsyncNumberRangeValidator(
        { min: 100, max: 10000, converter: converterOption });

    // eslint-disable-next-line no-undef
    return new Promise(function (resolve, reject) {
      // We could go to the server and check the
      // user's credit score and based on that
      // credit score use a specific number range validator.
      // We mock a server-side delay
      setTimeout(function () {
        numberRangeValidator.validate(value).then(
          // eslint-disable-next-line no-unused-vars
          function (result) {
            /* handle a successful result */
            resolve();
          },
          function (e) {
            /* handle an error */
            var converterInstance =
              new NumberConverter.IntlNumberConverter(converterOption);
            reject({
              detail: e + ' Your value is ' +
                converterInstance.format(value) + '.'
            });
          });
        }, 1000);
      });
    },
    // 'hint' is an optional field that returns a Promise
    'hint': new Promise(function(resolve, reject) {
      ...
    })
};
```

 **Note:**

A Promise object in JavaScript represents a value that may not be available yet, but will be resolved at some point in the future. In asynchronous validation, the `AsyncValidator.validate()` function returns a Promise that evaluates to Boolean `true` if validation passes and if validation fails, it returns an Error. For more, see the `validators` attribute section of [ojInputText](#) or see [Promise \(MDN\)](#).

For the full Cookbook sample, see [Async Validators](#).

12

Working with User Assistance

The Oracle JET user assistance framework includes support for user assistance on the editable components in the form of help, hints, and messaging that you can customize as needed for your application. Editable components include `oj-checkboxset`, `oj-color*`, `oj-combobox*`, `oj-input*`, `oj-radioset`, `oj-select*`, `oj-slider`, `oj-switch`, and `oj-text-area`.

Note:

The `oj-input*` mentioned above refers to the family of input components such as `oj-input-date-time`, `oj-input-text`, and `oj-input-password`, among others. `oj-color*`, `oj-combobox*`, and `oj-select*` each represent two components.

Topics:

- [Typical Workflow for Working with User Assistance](#)
- [Understand Oracle JET's Messaging APIs on Editable Components](#)
- [Understand How Validation and Messaging Works in Oracle JET Editable Components](#)
- [Use Oracle JET Messaging](#)
- [Configure an Editable Component's `oj-label` Help Attribute](#)
- [Configure an Editable Component's `help.instruction` Attribute](#)
- [Control the Display of Hints, Help, and Messages](#)

Tip:

To add tooltips to plain text or other non-editable components, use `oj-popup`. See [Work with oj-popup](#).

Typical Workflow for Working with User Assistance

Understand Oracle JET's user assistance capabilities and how to add user assistance to your Oracle JET application.

To add user assistance to your Oracle JET application, refer to the typical workflow described in the following table:

Task	Description	More Information
Identify Oracle JET's user assistance capabilities on editable components	Identify the editable components and their messaging APIs.	Understand Oracle JET's Messaging APIs on Editable Components
Understand Oracle JET editable component validation and messaging process	Understand the normal, deferred, and mixed validation processes and the messaging associated with each.	Understand How Validation and Messaging Works in Oracle JET Editable Components
Use Oracle JET validation and messaging on editable components	Learn how to track the validity of a group of editable components and configure an Oracle JET application to notify editable components of business validation failures and to receive notification of an editable component's events and messages.	Use Oracle JET Messaging
Configure the help attribute on the oj-label component	Add a help icon to your oj-label that includes a URL for additional information or adds explanatory text that appears when the user hovers over the icon.	Configure an Editable Component's oj-label Help Attribute
Configure the help.instruction attribute on editable components	Configure the help.instruction attribute to add advisory information to an input field that appears when the field has focus.	Configure an Editable Component's help.instruction Attribute
Control the display of hints, help, and messages	Use the editable component's display-options attribute to control the display type of converter and validator hints, messaging, and help instruction properties.	Control the Display of Hints, Help, and Messages

Understand Oracle JET's Messaging APIs on Editable Components

Oracle JET provides a messaging API to support messaging on Oracle JET editable components.

Editable components include the following:

- oj-checkboxset
- oj-color-palette
- oj-color-spectrum
- oj-comboobox-many
- oj-comboobox-one
- oj-input-date
- oj-input-date-time
- oj-input-number
- oj-input-password

- oj-input-text
- oj-input-time
- oj-radio-set
- oj-select-many
- oj-select-one
- oj-slider
- oj-switch
- oj-text-area

Topics:

- [About Oracle JET Editable Component Messaging Attributes](#)
- [About Oracle JET Component Messaging Methods](#)

The Oracle JET Cookbook also includes descriptions and examples for working with each component at: [Form Controls](#).

About Oracle JET Editable Component Messaging Attributes

The following attributes impact messaging on editable elements.

Element Attribute	Description
converter	Default converter hint displayed as placeholder text when a placeholder attribute is not already set.
display-options	JSON object literal that specifies the location where the element should display auxiliary content such as messages, converterHint, validatorHint, and helpInstruction in relation to itself. Refer to the element's API documentation for details.
help	<p>Help message displayed on an <code>oj-label</code> element when the user hovers over the Help icon. No formatted text is available for the message. The <code>oj-label</code> has two exclusive attributes, <code>help.definition</code> and <code>help.source</code>.</p> <p>The <code>help.definition</code> attribute's value appears and the attribute's value is read by a screen reader when you hover with a mouse, when you tab into the Help icon, or when you press and hold on a device. The default value is <code>null</code>.</p> <p>The <code>help.source</code> attribute's value is a link, which is opened when you click with a mouse or tap on a device. The default value is <code>null</code>.</p>
help.instruction	Displays text in a note window that displays when the user sets focus on the input field. You can format the text string using standard HTML formatting tags.
messages-custom	List of messages that the application provides when it encounters business validation errors or messages of other severity type.
placeholder	The placeholder text to set on the element.
translations	Object containing all translated resources relevant to the component and all its superclasses. Use sub-properties to modify the component's translated resources.
validators	List of validators used by element when performing validation. Validator hints are displayed in a note window by default.

See the [Attributes](#) section of the element's API documentation in [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) for additional details

about its messaging properties. Select the component you're interested in viewing from the API list.

About Oracle JET Component Messaging Methods

Editable value components support the following method for messaging purposes.

Component Event	Description
showMessages	Takes all deferred messages and shows them. If there were no deferred messages this method simply returns. When the user sets focus on the component, the deferred messages will display inline.

See the Methods section of the component's API documentation in the [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#) for details on how to call the method, its parameters, and return value. Select the component you're interested in viewing from the list.

Understand How Validation and Messaging Works in Oracle JET Editable Components

The actions performed on an Oracle JET component, the properties set on it, and the methods called on it, all instruct the component on how it should validate the value and what content it should show as part of its messaging.

Editable components always perform either normal or deferred validation in some situations. In other situations, the editable component decides to perform either normal or deferred validation based on the component's state. Understanding the normal and deferred validation process may be helpful for determining what message properties to set on your components.

- **Normal Validation:** During normal validation, the component clears all messages properties (`messages-custom`), parses the UI value, and performs validation. Validation errors are reported to the user immediately. If there are no validation errors, the `value` attribute is updated, and the value is formatted and pushed to the display.

The editable component always runs normal validation when:

- The user interacts with an editable component and changes its value in the UI.
- The application calls `validate()` on the component.

Note:

When the application changes certain properties, the component might decide to run normal validation depending on its current state. See Mixed Validation below for additional details.

- **Deferred Validation:** Uses the `required` validator to validate the component's value. The `required` validator is the only validator that participates in deferred validation. During deferred validation all `messages` properties are cleared unless

specified otherwise. If the value fails deferred validation, validation errors are not shown to the user immediately.

The editable component always runs deferred validation when:

- A component is created. None of the `messages` properties are cleared.
- The application calls the `reset()` method on the component.
- The application changes the `value` property on the component programmatically.

 **Note:**

When the application changes certain properties programmatically, the component might decide to run deferred validation depending on its current state. See Mixed Validation below for additional details.

- Mixed Validation: Runs when the following properties are changed or methods are called by the application. Either deferred or normal validation is run based on the component's current state, and any validation errors are either hidden or shown to the user. Mixed validation runs when:
 - converter property changes
 - disabled property changes
 - readOnly properties change
 - required property changes
 - validators property changes
 - refresh() method is called

Topics:

- [Understand How an Oracle JET Editable Component Performs Normal Validation](#)
- [Understand How an Oracle JET Editable Component Performs Deferred Validation](#)

The Oracle JET Cookbook includes additional examples that show normal and deferred validation at [Validators \(Component\)](#). For additional information about the validators and converters included with Oracle JET, see [Validating and Converting Input](#).

Understand How an Oracle JET Editable Component Performs Normal Validation

An Oracle JET editable component runs normal validation when the user changes the value in the UI or when the application calls the component's `validate()` method. In both cases, error messages are displayed immediately.

Topics:

- [About the Normal Validation Process When User Changes Value of an Editable Component](#)

- About the Normal Validation Process When Validate() is Called on Editable Component

About the Normal Validation Process When User Changes Value of an Editable Component

When a user changes an editable value:

1. All `messages-custom` messages are cleared. An `onMessagesCustomChanged` event is triggered if applicable and if the change in value is obvious.
 2. If a converter is set on the component, the UI value is parsed. If there is a parse error, then processing jumps to step 5.
 3. If one or more validators are set on the component:
 - a. The parsed (converted) value is validated in sequence using the specified validators, with the implicit required validator being the first to run if present. The value that is passed to the implicit required validator is trimmed of white space.
 - b. If the validator throws an error, the error is remembered, and the next validator runs if it exists.
- After all validators complete, if there are one or more errors, processing jumps to step 5.
4. If all validations pass:
 - a. The parsed value is written to the component's `value` attribute, and an `onValueChanged` event is triggered for the `value` attribute.
 - b. The new value is formatted for display using the converter again and displayed on the component.

Note:

If the component's `value` property happens to be bound to a Knockout observable, then the value is written to the observable as well.

5. If one or more errors occurred in an earlier step:
 - a. The component's `value` property remains unchanged.
 - b. Errors are displayed by default inline with the component. The user can also view the details of the error by setting focus on the component.
6. When the user fixes the error, the validation process begins again.

About the Normal Validation Process When Validate() is Called on Editable Component

The `validate()` method validates the component's current display value using the converter and all validators registered on the component and updates the `value` attribute if validation passes.

The method is only available on editable components where it makes sense, for example, `oj-input-number`. For details about the `validate()` method, see [validate\(\)](#).

Understand How an Oracle JET Editable Component Performs Deferred Validation

An Oracle JET editable component runs deferred validation when the component is created, when its `value` or `required` property is changed programmatically, or when the component's `reset()` method is called. This section provides additional detail about the deferred validation process when an Oracle JET editable component is created and when the `value` property is changed programmatically.

Topics:

- [About the Deferred Validation Process When an Oracle JET Editable Component is Created](#)
- [About the Deferred Validation Process When value Property is Changed Programmatically](#)

You can also find additional detail in the [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#). Select the component you're interested in from the navigation list.

About the Deferred Validation Process When an Oracle JET Editable Component is Created

When an editable element is created, as one of the last steps, it runs deferred validation on the component's initialized value.

1. The required validator is run, and a validation error raised if the value is empty or null.
2. If a validation error is raised, the component updates the messaging framework. No messaging themes are applied on the component nor does it show the error message in the note window because the validation error message is deferred.

Note:

Page authors can call `showMessages()` at any time to reveal deferred messages.

About the Deferred Validation Process When value Property is Changed Programmatically

An Oracle JET editable element's `value` property can change programmatically if:

- The page has code that changes the element's `value` attribute, or
- The page author refreshes the `ViewModel` observable with a new server value.

In both cases, the element will update itself to show the new value as follows:

1. All messages properties are cleared on the editable element and `onMessagesCustomChanged` events triggered if applicable.
2. An `onValueChanged` event is triggered for the `value` attribute if applicable.
3. If a converter is set on the element, the `value` attribute is retrieved and formatted before it's displayed. If there is a format error, then processing jumps to step 5. Otherwise the formatted value is displayed on the element.
4. Deferred validators are run on the new value. Any validation errors raised are communicated to the messaging framework, but the errors themselves are not shown to the user.
5. If there was a formatting error, the validation error message is processed and the component's `messages-custom` attribute updated. Formatting errors are shown right away.

 **Note:**

Page authors should ensure that the value you set is the expected type as defined by the component's API and that the value can be formatted without any errors for display.

Use Oracle JET Messaging

Use the Oracle JET messaging framework to notify an Oracle JET application of a component's messages and validity as well as notify an Oracle JET component of a business validation failure.

Topics:

- [Notify an Oracle JET Editable Component of Business Validation Errors](#)
- [Understand the `oj-validation-group` Component](#)
- [Create Page Level Messaging](#)

Notify an Oracle JET Editable Component of Business Validation Errors

You can notify Oracle JET editable elements of business validation errors using the `messages-custom` attribute and the `showMessages()` method.

Topics:

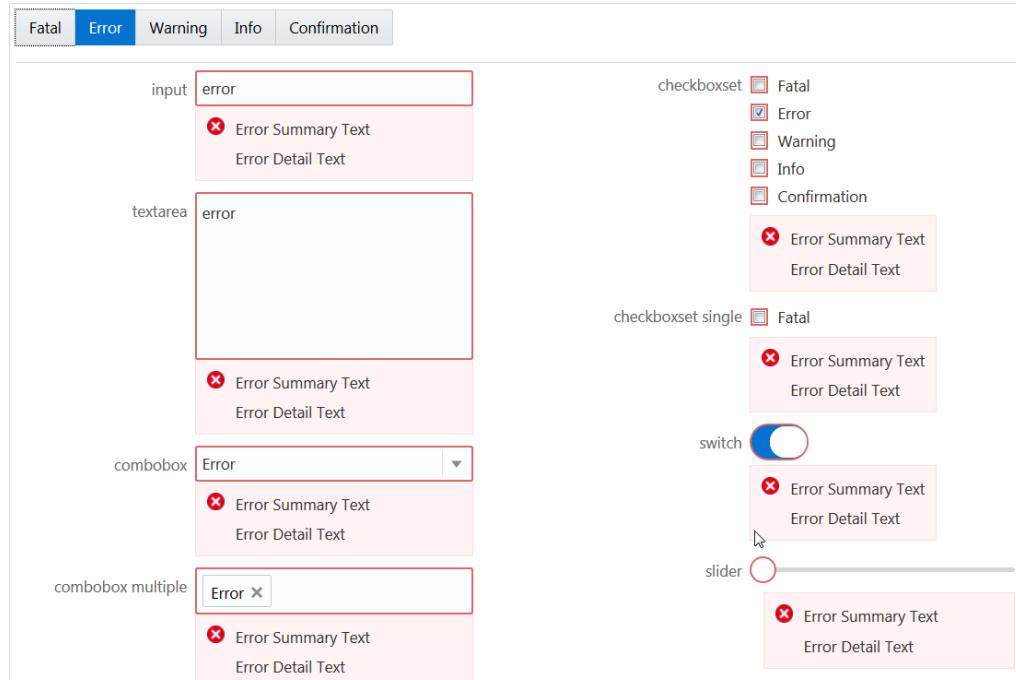
- [Use the `messages-custom` Attribute](#)
- [Use the `showMessages\(\)` Method on Editable Components](#)

Use the `messages-custom` Attribute

Your application can use this attribute to notify Oracle JET components to show new or updated custom messages. These could be a result of business validation that originates in the `viewModel` layer or on the server. When this property is set, the

message shows to the user immediately. The `messages-custom` attribute takes an Object that duck-types `Message` with `detail`, `summary`, and `severity` fields.

In this example, the severity type button is toggled and a message of the selected severity type is pushed onto the `messages-custom` array. The `messages-custom` attribute is set on every form control in this example. When the `messages-custom` attribute is changed, it is shown immediately. In this example, the user selected the Error severity type, and the associated messages are shown for the various text input and selection components.



In this example an observable, `appMessages`, is declared and is bound to the `messages-custom` attribute for various elements. The following code describes how you can associate the observable with the `messages-custom` attribute for the `oj-switch` element .

```
<oj-switch id="switch" value="{{switchValue}}"
  messages-custom="{{appMessages}}">
</oj-switch>
```

In the corresponding JavaScript file, set the severity type and pass it to the observable, `appMessages`, to display associated messages.

```
if (summary && detail)
{
  msgs.push({summary: summary, detail: detail, severity: type});
}
self.appMessages(msgs);
```

In the example below, an instance of a cross-field custom validator is associated with the `emailAddress` observable, ensuring that its value is non-empty when the user chooses **Email** as their contact preference.

The screenshot shows a user interface with three fields:

- Best Reached By:** A radio button group where the "Email" option is selected.
- Email Address:** An input field containing "john_doe@example.com". Below it is a red error message box with a red X icon and the text "Email Address Required" followed by "You must enter a value for field Email Address".
- Phone Number:** An input field containing "ten digit phone number".

In the corresponding JavaScript file you must set the `messages-custom` attribute as `emailAddressMessages`.

```
<oj-input-text id="emailId" type="email" name="emailId"
               placeholder="john_doe@example.com"
               value="{{emailAddress}}"
               messages-custom="{{emailAddressMessages}}"
               disabled="[[contactPref() !== 'email']]">
</oj-input-text>
```

For the complete example and code used to create the custom validator, see [Cross-Field Validation](#). The demo uses a custom validator to validate an observable value against another. If validation fails the custom validator updates the `messages-custom` attribute.

Use the `showMessages()` Method on Editable Components

Use this method to instruct the component to show its deferred messages. When this method is called, the Oracle JET editable component automatically takes all deferred messages and shows them. This causes the component to display the deferred messages to the user.

For an example, see [Show Deferred Messages](#).

Understand the `oj-validation-group` Component

The `oj-validation-group` component is an Oracle JET element that tracks the validity of a group of components and allows a page author to enforce validation best practices in Oracle JET applications.

Applications can use this component to:

- Determine whether there are invalid components in the group that are currently showing messages using the `invalidShown` value of the `valid` property.

- Determine whether there are invalid components that have deferred messages, such as messages that are currently hidden in the group, using the `invalidHidden` value of the `valid` property.
- Set focus on the first enabled component in the group using the `focusOn()` method. They can also focus on the first enabled component showing invalid messages using `focusOn("@firstInvalidShown")`.
- Show deferred messages on all tracked components using the `showMessages()` method.

For details about the `oj-validation-group` component's attributes and methods, see [oj-validation-group](#).

Track the Validity of a Group of Editable Components Using `oj-validation-group`

You can track the validity of a group of editable components by wrapping them in the `oj-validation-group` component.

The `oj-validation-group` searches all its descendants for a `valid` property, and adds them to the list of components it is tracking. When it adds a component, it does not check the tracked component's children since the component's valid state should already be based on the valid state of its children, if applicable.

When it finds all such components, it determines its own `valid` property value based on all the enabled (including hidden) components it tracks. Any disabled or readonly components are ignored in calculating the `valid` state.

The most invalid component's `valid` property value will be the `oj-validation-group` element's `valid` property value. When any of the tracked component's `valid` value changes, `oj-validation-group` will be notified and will update its own `valid` value if it has changed.

The following code sample shows how an `oj-validation-group` can be used to track the overall validity of a typical form.

```
<div id="validation-usecase">
  <oj-validation-group id="tracker" valid="{{groupValid}}>
    <oj-form-layout label-edge="inside" id="f11">

      <oj-input-text id="firstname" required
                     autocomplete="off"
                     label-hint="First Name"
                     name="firstname" >
      </oj-input-text>
      <oj-input-text id="lastname" required
                     value="{{lastName}}"
                     autocomplete="off"
                     label-hint="Last Name">
      </oj-input-text>
      <oj-input-text id="email"
                     on-value-changed="[[firstEmailValueChanged]]"
                     autocomplete="off"
                     label-hint="Email"
                     value="{{email}}" >
      </oj-input-text>
      <oj-input-text id="email2"
                     autocomplete="off"
                     label-hint="Confirm Email"
                     validators="[[emailMatchValidator]]"
```

```

        value="{{email2}}">
    </oj-input-text>
    <oj-checkboxset id="colors" label-hint="Favorite Colors">
        <oj-option id="blueopt" value="blue">Blue</oj-option>
        <oj-option id="greenopt" value="green">Green</oj-option>
        <oj-option id="pinkopt" value="pink">Pink</oj-option>
    </oj-checkboxset>
</oj-form-layout>
</oj-validation-group>
<hr/>
<div class="oj-flex">
    <div class="oj-flex-item"> </div>
    <div class="oj-flex-item">
        <oj-button id="submitBtn"
            on-oj-action="[[submit]]">Submit</oj-button>
    </div>
</div>
<hr/>
<span>oj-validation-group valid property: </span>
<span id="namevalid">
    <oj-bind-text value="[[groupValid]]"></oj-bind-text>
</span>
</div>

```

A portion of the script to create the view model for this example is shown below. This portion pertains to the `oj-validation-group` used above. The full script is contained in the [Cookbook sample linked below](#).

```

require([ 'knockout', 'ojs/ojbootstrap', 'ojs/ojknockout', 'ojs/ojbutton', 'ojs/
ojcheckboxset', 'ojs/ojformlayout', 'ojs/ojinputtext', 'ojs/ojvalidationgroup'],
    function (ko, Bootstrap)
{
    // this callback gets executed when all required modules
    // for validation are loaded
    {
        function DemoViewModel() {
            var self = this;
            self.tracker = ko.observable();

            ...

            // to show the oj-validation-group's valid property value
            self.groupValid = ko.observable();

            // User presses the submit button
            self.submit = function () {

                var tracker = document.getElementById("tracker");

                if (tracker.valid === "valid") {
                    // submit the form would go here
                    alert("everything is valid; submit the form");
                }
                else {
                    // show messages on all the components that have messages hidden.
                    tracker.showMessages();
                    tracker.focusOn("@firstInvalidShown");
                }
            };
        };

        Bootstrap.whenDocumentReady().then(

```

```

        function ()
        {
            ko.applyBindings(new DemoViewModel(),
            document.getElementById('validation-usecase'));
        }
    });
})
);

```

The figure below shows the output for the code sample. The status text at the bottom of each instance shows the valid state of the `oj-validation-group`, and by extension, the form.

The figure consists of three screenshots of a web form. The first screenshot shows the initial state with all fields empty. The second screenshot shows the 'Last Name' field highlighted in red with an error message. The third screenshot shows the form submitted successfully with all fields filled and a confirmation message.

Screenshot 1 (Initial State):

- Fields: First Name (John), Last Name (empty), Email (empty), Confirm Email (empty), Favorite Colors (Blue, Green, Pink checkboxes).
- Submit button.
- Status text: `oj-validation-group valid property: invalidHidden`.

Screenshot 2 (Invalid State):

- Fields: First Name (John), Last Name (empty), Email (empty), Confirm Email (empty), Favorite Colors (Blue, Green, Pink checkboxes).
- Last Name field has a red border and an error message: "Value is required. You must enter a value."
- Submit button.
- Status text: `oj-validation-group valid property: invalidShown`.

Screenshot 3 (Valid State):

- Fields: First Name (John), Last Name (Doe), Email (empty), Confirm Email (empty), Favorite Colors (Blue, Green, Pink checkboxes).
- All fields are filled.
- A modal dialog box displays the message: "everything is valid; submit the form".
- OK button.
- Status text: `oj-validation-group valid property: valid`.

The Oracle JET Cookbook contains the complete example used in this section. See [Form Fields](#).

For an example showing the `oj-validation-group` used for cross-field validation, see [Cross-Field Validation](#).

Create Page Level Messaging

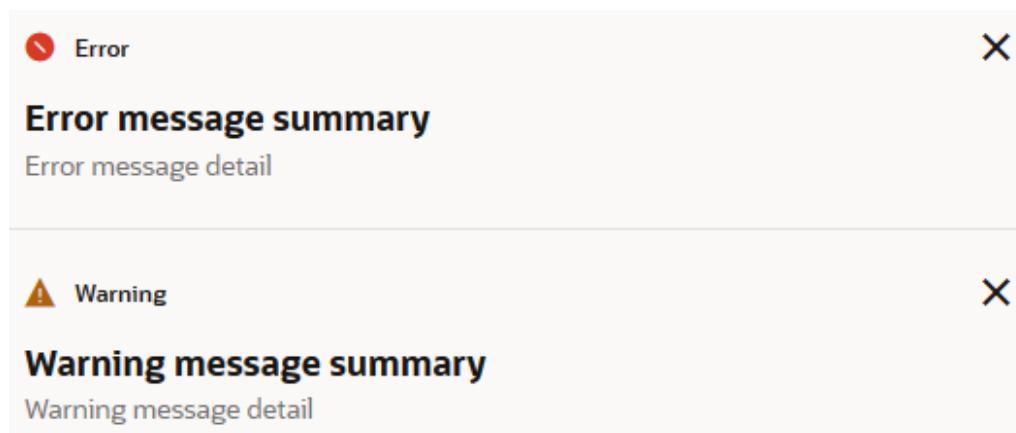
The Oracle JET messaging framework includes the `oj-messages` and `oj-message` elements that you can use to create different types of messages on your page.

The `oj-message` element represents a single message. This tag can be used to create three types of messages:

- **Notification Messages:** These are small popups, usually in one corner of the view port.
- **Inline Messages:** These are messages in line with other components on the page, such as messages below the page header.
- **Overlay Messages:** These are messages that pop up on top of other components, such as below the page header overlapping the page contents.

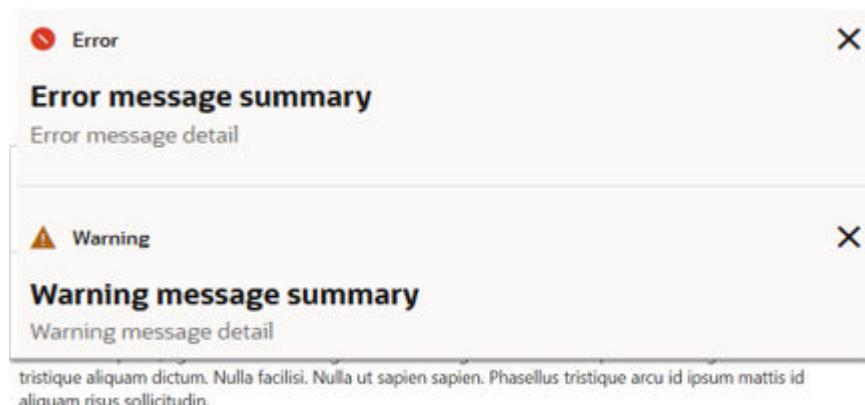
The `oj-messages` element is a wrapper tag that manages the layout of individual messages. This wrapper tag allows for mass dismissal of messages. For single page applications, it is recommended that no more than one instance of `oj-messages` be defined for these three layouts to avoid clutter on the page.

Inline-style messages are typically used as page level inline messages just below the page header.



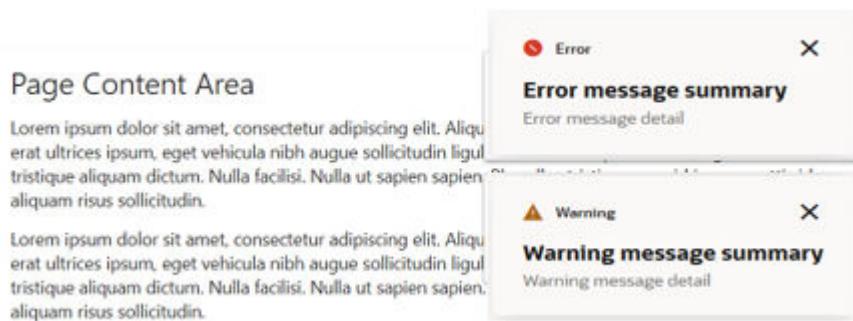
The Cookbook contains an example that showcases inline messages. See [Inline messages in page layout](#).

Overlay-style messages are typically used as a page level overlay message just below the page header. These messages usually pertain to the page or region of the application.



The Cookbook contains an example that showcases overlay messages. See [Overlay messages in page layout](#).

Notification-style messages are typically used for asynchronous page-level messages in a corner of the viewport.



The Cookbook contains an example that showcases notification messages. See [Notification messages in page layout](#).

Create Messages with `oj-message`

The Oracle JET messaging framework contains a number of properties to customize how messages are created and displayed.

To create messages:

1. To specify that the message displays inline, create an `oj-messages` element and set the `display` attribute to `general` and do not set the `position` attribute.

```
<oj-messages id="oj-messages-id" messages="[{inlineMessages}]"
display="general">
```

2. To specify that the message displays as an overlay, set the `display` attribute to `general` and the `position` attribute to `{}`.

```
<oj-messages id="oj-messages-id" messages="[[appMessages]]"
display="general" position="{}">
```

3. To create notification style messages, set the `display` attribute to `notification` and the `position` attribute to `{}`.

```
<oj-messages id="notificationMessages" messages="[[emailMessages]]"
position="{}" display="notification"></oj-messages>
```

Note:

For overlay and notification messages, you can specify the `position` attribute fully instead of `{}`. Theming variables are available for setting default positioning at application level. See the API doc for more.

4. There are three ways to create `oj-message` children for any type of message:

- a. Include `oj-message` as direct children of `oj-messages`.

```
<oj-messages id="inlineMessages">
  <oj-message message='{"summary": "Some summary", "detail": "Some detail", "autoTimeout": 5000}'></oj-message>
  <oj-message message="[[surveyInstructions]]"></oj-message>
  <oj-message message="[[surveySubmitConfirmation]]"></oj-
message>
</oj-messages>
```

- b. Use `oj-bind-for-each` to generate `oj-message` children.

```
<oj-messages id="pageOverlayMessages" position="{}"
display="notification">
  <oj-bind-for-each
data="[[serviceRequestStatusUpdateMessages]]">
    <template>
      <oj-message message="[$current.data]"></oj-message>
    </template>
  </oj-bind-for-each>
  <oj-bind-for-each data="[[criticalIncidentMessages]]">
    <template>
      <oj-message message="[$current.data]"></oj-message>
    </template>
  </oj-bind-for-each>
</oj-messages>
```

- c. Specify the `messages` attribute on `oj-messages` to a message object that programmatically generates messages.

```
<oj-messages id="notificationMessages"
messages="[[emailMessages]]" position="{}"
display="notification">
</oj-messages>
```

5. Set the `message.auto-timeout` subproperty to the number of milliseconds before it should close itself, or set to `-1` to disable auto timeout.
6. Set the `message.close-affordance` subproperty to `'defaults'` to allow users to dismiss messages, or `'none'` to disallow it.

 **Note:**

The `close-affordance` attribute should be set to `none` only when the user cannot dismiss messages manually. Messages can still be closed programmatically by calling the `close()` method on `oj-message`.

7. Set the `message.timestamp` subproperty with an ISOString to specify a timestamp displayed in the message header.
8. Set the `message.sound` subproperty to a URL to play a custom sound for accessibility purposes. Set it to `'defaults'` for a default sound, or `'none'` to disable it.

 **Note:**

Sound is an accessibility feature required for low-vision users who view a zoomed section of the UI. Because messages may be shown outside of the zoomed section, users require sound to be played to notify them of new messages.

For details about the `oj-messages` component's attributes and methods, see [oj-messages](#). For `oj-message`, see [oj-message](#).

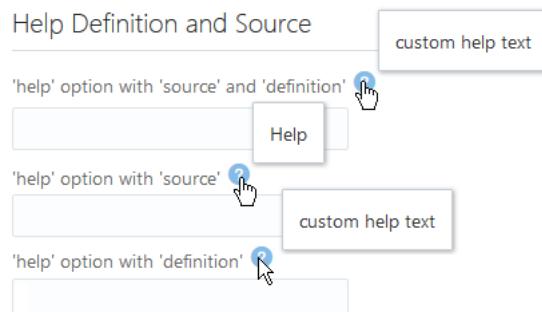
Configure an Editable Component's oj-label Help Attribute

Configure an `oj-label` element's `help` attribute to add a **Help** icon that displays descriptive text, includes a clickable link to a URL for additional information, or displays both the help text and clickable link when the user hovers over it.

The `help` attribute includes two sub-properties that control the help definition text and help icon:

- `definition`: Contains the help definition text that displays when the user does one of the following:
 - hovers over the help icon
 - tabs into the help icon with the keyboard
 - presses and holds the help icon on a mobile device
- `source`: Contains the URL to be used in the help icon's link

The following image shows three `oj-label` components configured to use the `help` attribute. The top component is configured with both a `definition` and `source` help sub-property, and the image shows the text and clickable pointer that displays when the user hovers over the help icon. In the middle image, the `oj-label` component includes a help icon that links to a URL when the user clicks it. In the bottom image, the `oj-label` displays custom help text when the user hovers over the label or help icon.



Before you begin:

- Familiarize yourself with the process of adding an Oracle JET element to your page. See [Add an Oracle JET Component to Your Page](#).

To configure an `oj-label` element's `help` property:

1. Add the `oj-label` element to your page.
2. In the markup, add the `help` attribute and the `definition` or `source` sub-property.

The markup for the `ojInputText` components is shown below. Each `ojInputText` component uses the `ojComponent` binding to define the component and set the `help` sub-properties. In this example, the user will be directed to `http://www.oracle.com` after clicking **Help**.

```
<div id="form-container" class="oj-form">
  <h3 class="oj-header-border">Help Definition and Source</h3>
  <oj-label id="ltext10" for="text10" help.definition="custom help text"
            help.source="http://www.oracle.com">'help' property with 'source' and
            'definition'</oj-label>
  <oj-input-text id="text10" name="text10" value="{{text}}"></oj-input-
text>

  <oj-label id="ltext11" for="text11"
            help.source="http://www.oracle.com">'help' property with 'source'</oj-
label>
  <oj-input-text id="text11" name="text11" value="{{text}}"></oj-input-text>

  <oj-label id="ltext12" for="text12"
            help.definition="custom help text">'help' property with 'definition'</
oj-label>
  <oj-input-text id="text12" name="text12" value="{{text}}"></oj-input-text>
</div>
```

See the Oracle JET Cookbook at [Help and Title](#) for the complete example to configure the `help` property on the `ojInputText` component.

Configure an Editable Component's help.instruction Attribute

Configure an editable component's help text using the `help.instruction` attribute. This will display a note window with advisory text (often called a tooltip) when the input component has focus.

The following image shows two `oj-input-text` elements configured to display text when the user sets focus in the input field. In the first example, the `help.instruction` attribute is defined for the `oj-input-text` element without formatting. In the second example, the attribute value has HTML formatting added to the advisory text.



Before you begin:

- Familiarize yourself with the process of adding an Oracle JET element to your page. See [Add an Oracle JET Component to Your Page](#).

To configure an editable element's `help.instruction` attribute:

1. Add the editable element to your page.
2. In the markup, add the `help.instruction` attribute in the element tag.

The following code sample shows the markup for defining the three `oj-input-text` components.

```
<div id="form-container" class="oj-form">
    <h3 class="oj-header-border">Help Instruction</h3>
    <oj-label for="text20">input with 'help.instruction' attribute</oj-label>
    <oj-input-text id="text20" name="text20" autocomplete="off"
    validators="[[validators]]"
        help.instruction="enter at least 3 alphanumeric characters"
    value="{{text}}"></oj-input-text>

    <oj-label for="text21">input with 'help.instruction' attribute with
    binding</oj-label>
    <oj-input-text id="text21" name="text21" autocomplete="off"
    validators="[[validators]]"
        help.instruction="{{helpInstruction}}"
    value="{{text}}"></oj-input-text>

    <oj-label for="text22">input with formatted text in 'help.instruction'
    attribute</oj-label>
    <oj-input-text id="text22" name="text22" autocomplete="off"
    validators="[[validators]]"
        help.instruction="<html>enter <span style='color:red'>at least 3
    alphanumeric</span> characters</html>"
    value="{{text}}"></oj-input-text>
</div>
```

3. In your application script, bind the component's `value` to a Knockout observable.

In this example, each `oj-input-text` element's `value` attribute is defined as `text` which is set to a Knockout observable in the following script. The script also defines regular expression validators to ensure that the user enters at least three letters or numbers.

```
require(['knockout', 'ojs/objbootstrap', 'ojs/ojknockout', 'ojs/ojinputtext',
'ajs/ajlabel'],
    function(ko, Bootstrap)
{
    function MemberViewModel()
    {
        var self = this;
        self.text = ko.observable();

        self.validators = ko.computed(function()
        {
            return [
                {
                    type: 'regExp',
                    options: {
                        pattern: '[a-zA-Z0-9]{3,}',
                        messageDetail: 'You must enter at least 3 letters or
numbers'}];
        });
    }
});
```

```
    });
    self.helpInstruction = "enter at least 3 alphanumeric characters";
};

Bootstrap.whenDocumentReady().then(
  function ()
  {
    ko.applyBindings(new MemberViewModel(),
document.getElementById('form-container'));
  }
);
});
```

For the complete example, see [Help and Title](#) in the Oracle JET Cookbook. For additional detail about the `oj-input-text` component, see the [ojInputText API documentation](#).

For additional information about the regular expression validator, see [About Oracle JET Validators and Converters](#).

Control the Display of Hints, Help, and Messages

Use the `display-options` attribute to control the placement and visibility of converter and validator hints, messages, and help instructions.

The following image shows the default placement and visibility of help, converter and validator hints, and messages. This example uses the `oj-input-date` component, but the behavior is the same on all editable components where it makes sense:

- `validator hint`: Displays in a note window on focus
- `converter hint`: Used as the input field's placeholder, or displays in a note window if the `placeholder` attribute is defined.
- `messages`: Displays inline on error
- `help.instruction`: Displays in a note window on focus

The `oj-label` exclusive attribute `help.definition` displays in a note window on hover.

Default Display of Messages, Hints, Help Instruction

Default Display on Hover

Default Display on Focus

Default Display on Required Error

Default Display on Validator Error

Default Display on Converter Error

The code sample below shows the markup for the `oj-input-date` component used in this example. The example includes definitions for `help.instruction`, `validator hints`, and a data value for custom messages on validation failure. The sample also shows the markup for a `oj-label` element with the `help` attribute.

```
<div id="form-container" class="oj-form">
  <h3 class="oj-header-border">Default Display of Messages, Hints, Help
  Instruction</h3>
  <oj-label for="date10" help.definition="custom help text"> Input Date</oj-
  label>
  <oj-input-date id="date10" size="30" name="date10" required placeholder="month
  day, year"
    help.instruction='enter a date in your preferred format
    and we will attempt to figure it out'
    converter="[[longDateConverter]]"
    value="{{birthdate}}" validators="[[validators]]"
    translations='{
      "required": {
        "hint": "validator hint: required",
        "messageSummary": "<html>custom summary: {label}
Required</html>",
        "messageDetail": "<html>custom detail: A value is
        required for this field</html>"}}'
  </oj-input-date>
```

The code sample below shows the custom messages on validation failure set in the application's script.

```
function MemberViewModel()
{
```

```

var self = this;
self.validators = ko.computed(function()
{
    return [
        {
            type: 'datetimeRange',
            options: {
                min: ValidationBase.IntlConverterUtils.dateToLocalIso(new Date(1930,
00, 01)),
                max: ValidationBase.IntlConverterUtils.dateToLocalIso(new Date(1995,
11,31)),
                hint: {
                    inRange: 'Validator hint: datetimeRange: January 1, 1930 -
November 30, 1995 years',
                    messageSummary:{},
                    rangeOverflow: 'Date later than max.',
                    rangeUnderflow: 'Date earlier than min.'},
                    messageDetail: {
                        rangeOverflow: 'The value \'{value}\' is not in the expected
range; it is too high.',
                        rangeUnderflow: 'The value \'{value}\' is not in the
expected range; it is too low.'}
                }]];
});
//...Contents Omitted
}

Bootstrap.whenDocumentReady().then(
    function ()
    {
        ko.applyBindings(new MemberViewModel(), document.getElementById('form-
container'));
    }
);
});

```

Using the `display-options` element attribute in your markup, you can change the default behavior of the hints, help, and messaging properties of a single editable component on your page. To control the behavior of all editable components on the page, you can use the `Component.setDefaultOptions()` method in your application script to set `displayOptions` values.

`display-options` allows you to change the default behavior as follows:

- `helpInstruction`: Set to `none` to turn off the help instruction display.
- `converterHint`: Set to `none` to turn off the display or set to `notewindow` to change the default placement from placeholder text to a note window.
- `validatorHint`: Set to `none` to turn off the display.
- `messages`: Set to `none` to turn off the display or set to `notewindow` to change the default placement from inline to a note window.

Before you begin:

- Familiarize yourself with the process of adding an Oracle JET element to your page. See [Add an Oracle JET Component to Your Page](#).

To change the default display type (inline or note window) and display options for hints, help, and messages:

1. Add the editable element to your page.
2. To change the default display type (inline or note window) for an individual editable component, add the `display-options` attribute to your component definition and set it as needed.

For example, to turn off the display of hints and `help.instruction` and to display messages in a note window, add the highlighted markup to your component definition:

```
<oj-input-date id="date12" required value="{{birthdate}}"  
    converter="[[longDateConverter]]" validators="[[validators]]"  
    help.instruction="enter a date in your preferred format and we will  
    attempt to figure it out"  
    display-options='{"converterHint": "none", "validatorHint": "none",  
    "helpInstruction": "none", "messages": "notewindow"}'  
    ... contents omitted  
}"
```

3. To change the default display and location for all editable components in your application, add the `Component.setDefaultOptions()` method to your application's script and specify the desired `displayOptions`.

For example, to turn off the display of hints and help and to display messages in a note window, add the `ojComponent.setDefaultOptions()` method with the arguments shown below.

```
Components.setDefaultOptions({  
    'editableValue':  
    {  
        'displayOptions':  
        {  
            'converterHint': ['none'],  
            'validatorHint': ['none'],  
            'messages': ['notewindow'],  
            'helpInstruction': ['none']  
        }  
    }  
});
```

The Oracle JET cookbook contains the complete code for this example at [User Assistance](#). You can also find additional examples that illustrate hints, help, and messaging configuration.

13

Developing Accessible Applications

Oracle JET and Oracle JET components have built-in accessibility features for persons with disabilities. Use these features to create accessible Oracle JET web and hybrid mobile application pages.

Topics:

- [Typical Workflow for Developing Accessible Oracle JET Applications](#)
- [About Oracle JET and Accessibility](#)
- [About the Accessibility Features of Oracle JET Components](#)
- [Create Accessible Oracle JET Pages](#)

Typical Workflow for Developing Accessible Oracle JET Applications

Understand accessibility features included in Oracle JET and the components that it provides. You should also understand which tasks are needed for specific components and which tasks are needed at the page level to ensure accessibility.

To develop accessible Oracle JET applications, refer to the typical workflow described in the following table:

Task	Description	More Information
Identify accessibility features included with Oracle JET	Identify the accessibility features included in Oracle JET and Oracle JET components.	About Oracle JET and Accessibility
Create accessible Oracle JET components	Identify tasks needed for specific components to ensure accessibility.	About the Accessibility Features of Oracle JET Components
Create accessible Oracle JET pages	Identify tasks needed at the page level to ensure accessible Oracle JET pages.	Create Accessible Oracle JET Pages

About Oracle JET and Accessibility

Oracle JET components have built-in accessibility support that conforms with the Web Content Accessibility Guidelines version 2.1 at the AA level (WCAG 2.1 AA), developed by the World Wide Web Consortium (W3C).

Accessibility involves making your application usable for persons with disabilities such as low vision or blindness, deafness, or other physical limitations. This means, for example, creating applications that can be:

- Used without a mouse (keyboard only).
- Used with assistive technologies such as screen readers and screen magnifiers.

- Used without reliance on sound, color, animation, or timing.

Oracle JET components provide support for:

- Keyboard and touch navigation

Oracle JET components follow the Web Accessibility Initiative - Accessible Rich Internet Application (WAI-ARIA) [Developing a Keyboard Interface](#) guidelines. The API documentation for each Oracle JET component lists its keyboard and touch end user information when applicable, including a few deviations from the WAI-ARIA guidelines.

- Zoom

Oracle JET supports browser zooming up to 400%. For example, on the Firefox browser, you can choose **View -> Zoom -> Zoom In**.

- Screen reader

Oracle JET supports screen readers such as JAWS, Apple VoiceOver, and Google Talkback by generating content that complies with WAI-ARIA standards, and no special mode is needed.

- Oracle JET component roles and names

Each Oracle JET component has an appropriate role, such as button, link, and so on, and each component supports an associated name (label), if applicable.

- Sufficient color contrast

Oracle JET provides the Redwood theme, starting in Oracle JET release 9.0.0, which is designed to provide a luminosity contrast ratio of at least 4.5:1.

Oracle JET does not support the use of access keys due to their negative impact on assistive tooling and accessibility.

Oracle documents the degree of conformance of each product with the applicable accessibility standards using the [Voluntary Product Accessibility Template \(VPAT\)](#). You should review the appropriate VPAT for the version of Oracle JET that you are using for important information including known exceptions and defects, if any.

While Oracle JET is capable of rendering an application that conforms to WCAG 2.1 AA to the degree indicated by the VPAT, it is the responsibility of the application designer and developer to understand the applicable accessibility standards fully, use JET appropriately, and perform accessibility testing with disabled users and assistive technology.

About the Accessibility Features of Oracle JET Components

Oracle JET components are designed to generate content that conforms to the WCAG 2.1 AA standards. In most cases, you don't need to do anything to add accessibility to the Oracle JET component. However, there are some components where you may need to supply a label or other property.

For those components, the component's API documentation contains an Accessibility section that provides the information you need to ensure the component's accessibility.

 **Note:**

Some Oracle products have run-time accessibility modes that render content optimized for certain types of users, such as users of screen readers. For the most part, Oracle JET renders all accessibility-related content all of the time. There is only a mode for users that rely on the operating system's high contrast mode, which is described in [Create Accessible Oracle JET Pages](#).

Oracle JET components that provide keyboard and touch navigation list the keystroke and gesture end user information in their API documentation. Since the navigation is built into the component, you do not need to do anything to configure it.

You can access an individual Oracle JET component's accessibility features and requirements in the [JavaScript API Reference for Oracle® JavaScript Extension Toolkit \(Oracle JET\)](#). Select the component you're interested in from the list on the left. You can also find the list of supported keystrokes and gestures for each Oracle JET component that supports keystrokes and gestures in the [Oracle® JavaScript Extension Toolkit \(JET\) Keyboard and Touch Reference](#).

Create Accessible Oracle JET Pages

Content generated by Oracle JET is designed to conform to the WCAG 2.1 AA standards. However, many standards are not under the complete control of Oracle JET, such as overall UI consistency, the use of color, the quality of on-screen text and instructions, and so on.

A complete product development plan that addresses accessibility should include proper UI design, coding, and testing with an array of tools, assistive technology, and disabled users.

 **Note:**

In most cases, end-user documentation for your application must describe information about accessibility, such as example keystrokes needed to operate certain components.

Topics:

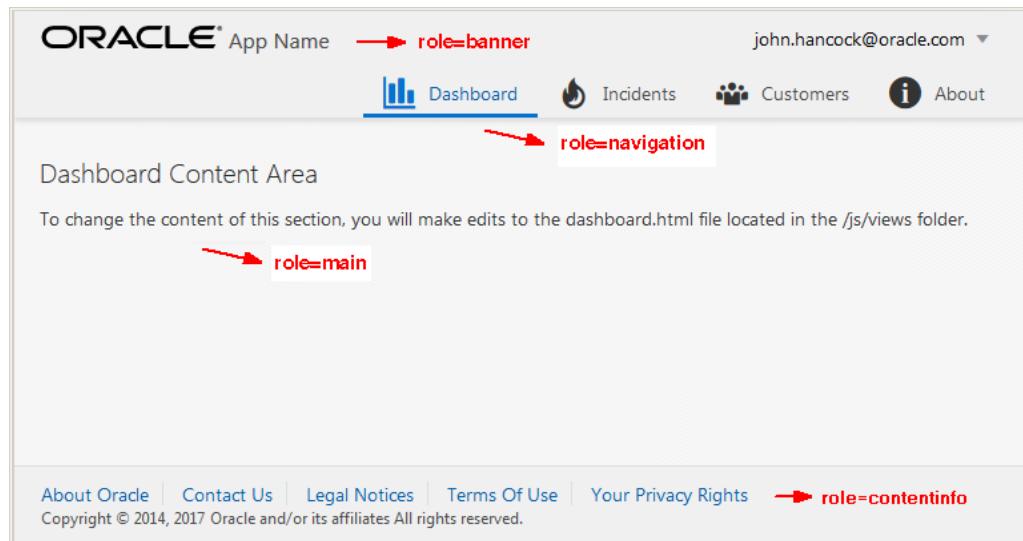
- [Configure WAI-ARIA Landmarks](#)
- [Configure High Contrast Mode](#)
- [Hide Screen Reader Content](#)
- [Use ARIA Live Region](#)

Configure WAI-ARIA Landmarks

WAI-ARIA landmarks provide navigational information to assistive technology users. Landmark roles identify the purpose of a page region and allow the user to navigate

directly to a desired region. Without landmarks, assistive technology users must use the TAB key to navigate through a page.

The Oracle JET team recommends the use of WAI-ARIA landmarks to ensure page accessibility and provides examples you can use in the Oracle JET Starter Template collection. The following figure shows the run-time view of the Oracle JET Web Nav Drawer Starter Template. In this example, the page is organized into regions compatible with WAI-ARIA landmark regions, including regions for the banner, navigation, main, and contentinfo landmarks.



The highlighted code in the following example shows the landmarks for the Web Nav Drawer Starter Template. Each landmark is placed on the HTML element that defines the landmark region: `div` for the navigation regions, `header` for the banner region, `oj-module` for the main region, and `footer` for the contentinfo region.

```
<!DOCTYPE html>
<html lang="en-us">

    <head>
        <title>Oracle JET Starter Template - Web Nav Drawer</title>
        ...contents omitted
    </head>

    <body class="oj-web-applayout-body">
        ...contents omitted

        <div id="globalBody" class="oj-offcanvas-outer-wrapper oj-offcanvas-page">
            <div id="navDrawer" role="navigation" class="oj-contrast-marker oj-web-
            applayout-offcanvas oj-offcanvas-start">
                <oj-navigation-list id="navDrawerList" data="[[navDataProvider]]"
                    edge="start"

                    item.renderer="[[KnockoutTemplateUtils.getRenderer('navTemplate', true)]]""
                    on-click="[[toggleDrawer]]"
                    selection="{{selection.path}}">
                </oj-navigation-list>
            </div>
        </div>
```

```
<div id="pageContent" class="oj-web-applayout-page">

    <header role="banner" class="oj-web-applayout-header">
        <div class="oj-web-applayout-max-width oj-flex-bar oj-sm-align-items-center">
            ...contents omitted
        </div>
        <div role="navigation" class="oj-web-applayout-max-width oj-web-applayout-navbar">
            <oj-navigation-list id="navTabBar" class="oj-sm-only-hide oj-md-condense oj-md-justify-content-flex-end"
                data="[[navDataProvider]]"
                edge="top"
            item.renderer="[[KnockoutTemplateUtils.getRenderer('navTemplate', true)]]"
                selection="{{selection.path}}">
                </oj-navigation-list>
            </div>
        </header>

        <oj-module role="main" class="oj-web-applayout-max-width oj-web-applayout-content" config="[[moduleAdapter.koObservableConfig]]">
            </oj-module>

            <footer class="oj-web-applayout-footer" role="contentinfo">
                ...contents omitted
            </footer>

        </div>
    </div>

    <script type="text/javascript" src="js/libs/require/require.js"></script>
    <script type="text/javascript" src="js/main.js"></script>

</body>
</html>
```

If your application includes a complementary region, add `role="complementary"` to the HTML `div` element:

```
<div role="complementary"></div>
```

For details about working with the Oracle JET Starter Templates, see [Scaffold a Web Application](#).

For additional information about WAI-ARIA landmark roles, see [landmark_roles](#).

Configure High Contrast Mode

High contrast mode is for people who require a very high level of contrast in order to distinguish components on the page. Operating systems such as Microsoft Windows and MacOS provide methods for users to run in high contrast mode.

The graphic below shows the effect of changing to high contrast mode on Oracle JET icon font images.

Icon Font Images - Normal Mode

Icons with a suffix "-16" were designed to look best at 16px.



Icons with a suffix "-24" were designed to look best at 24px.



Colorizing with css



Icon Font Images - High Contrast Mode

Icons with a suffix "-16" were designed to look best at 16px.



Icons with a suffix "-24" were designed to look best at 24px.



Colorizing with css



Oracle JET provides the `oj-hicontrast` class that you can use to configure high contrast mode in your application.

Topics:

- [Understand Color and Background Image Limitations in High Contrast Mode](#)
- [Add High Contrast Mode to Your Oracle JET Application](#)
- [Add High Contrast Images or Icon Fonts](#)
- [Test High Contrast Mode](#)

Understand Color and Background Image Limitations in High Contrast Mode

There are color and background image limitations in high contrast mode that your application may need to work around.

In high contrast mode the colors in the CSS may be ignored or overridden, including background, border, and text colors. Therefore, in high contrast mode you may need to find an alternative way to show state. For example, you might need to add or change the border to show that something is selected.

Also, your application may need to show alternate high contrast images that work on dark or light background color. Some operating systems, like Microsoft Windows 7, offer multiple display profiles for high contrast mode, including a black-on-white and white-on-black mode.

Consider providing an image that includes a background, so either black on a white background or white on a black background. That way it won't matter what the background color is on the page since the contrast is in the image itself.

Finally, on Windows, background images don't appear in high contrast mode. Therefore, you cannot use background images to communicate anything informative. You can use a background image to make something look nice, but don't use it to communicate information like the status of a process or whether something is required.

Add High Contrast Mode to Your Oracle JET Application

In most cases, you do not need to do anything to enable high contrast mode in your Oracle JET application. If you're using RequireJS to load Oracle JET component modules, Oracle JET will load a script that attempts to detect if a user is running in

high contrast mode. If the script succeeds at detection, it will place the `oj-hicontrast` class on the `body` element.

There is a serious limitation to this method, however. There is no standard way to detect high contrast mode, and we can't guarantee that the script works in all cases on all browsers. For example, on Windows, the script does detect high contrast mode on Internet Explorer, Microsoft Edge, and Firefox browsers, but it does not detect high contrast mode on Chrome.

To be guaranteed that the `.oj-hicontrast` styles are applied, add a user preference setting for high contrast to your application and configure the application to add the `oj-hicontrast` class to the `body` element when the preference is set.

When the class is added, the `.oj-hicontrast` CSS styles are applied to the page where defined. The code below shows an excerpt from the Redwood CSS which changes the `outline-width` to 3 on the `ojButton` component when the button has focus.

```
.oj-hicontrast .oj-button.oj-focus .oj-button-button {  
    outline-width: 3px; }
```

Note:

For disabled content, JET supports an accessible luminosity contrast ratio, as specified in [WCAG 2.1 - Success Criterion 1.4.3 Contrast \(Minimum\)](#), in the themes that are accessible.

Section 1.4.3 says that text or images of text that are part of an inactive user interface component have no contrast requirement. Because disabled content may not meet the minimum contrast ratio required of enabled content, it cannot be used to convey meaningful information. For example, a checkbox may still appear checked in disabled mode, but since the colors may not pass contrast ratio, you cannot rely on all users being able to see that it's checked. If the information is needed to interact with the page correctly, you must convey it using a different method, for example as plain text.

Add High Contrast Images or Icon Fonts

To support high contrast image files, Oracle JET provides Sass mixins that you can use to generate the correct CSS in high contrast mode to:

- Use an alternate image.
- Use images without using background images.

The Oracle JET cookbook provides examples that you can use at: [CSS Images](#).

You can also use icon fonts instead of image files to support high contrast mode. The limitation is that icon fonts use a single color. Since these icons are text, they will be guaranteed to contrast against the background color on systems that ignore colors in the CSS. However, if you use color to show state (for example, changing an icon to blue when selected), the colors may be ignored in high contrast modes. You may need to find an alternative like setting a border instead. The Oracle JET cookbook provides icon font demos at: [Icon Fonts](#).

Test High Contrast Mode

The recommended way to test high contrast mode in Oracle JET applications is to set high contrast mode at the operating system level on a Microsoft Windows platform. The Windows platform is recommended because Windows turns off background colors and images in high contrast mode. Also, the Google Chrome browser does not remove background images in high contrast mode, and unless this is the only browser you plan to support, you should test high contrast with Microsoft Internet Explorer or Firefox.

To turn high contrast mode on and off in Microsoft Windows, use the following key combination: Left Alt+Left Shift+PrtScn. You may need to refresh your browser to see the new mode.

Hide Screen Reader Content

Sometimes you want to have some text on the page that is read to the screen reader user, but the sighted user doesn't see. Oracle JET provides the `oj-helper-hidden-accessible` class that you can use to hide content.

You can find the `.oj-helper-hidden-accessible` style defaults in the Oracle JET CSS file. For the Redwood theme, the CSS file is: `web/css/redwood/x.x.x/web/redwood.css`. For the Alta theme, the CSS file is: `web/css/alta/x.x.x/web/oj-alta.css`. For additional information about theming and Oracle JET, see [Using CSS and Themes in Applications](#).

Use ARIA Live Region

An ARIA live region is a simple mechanism for notifying assistive technologies when a web page content is updated.

When using a single page application, if there are any changes to the page or the page region, the user using assistive technologies like screen readers are not notified about the changes in the page content since the URL of the application has not changed. To help provide a notification to the screen readers when a page or segments of a page change, an ARIA live region can be used to announce the dynamic changes within a web page. When the update takes place within an ARIA live region, a screen reader is automatically notified, wherever its focus is at the time, and it announces the updated content to the user. This can be achieved by using the `aria-live` attribute.

The `aria-live` attribute identifies an element as a live region. It takes three possible values:

- `off`: No notification
- `polite`: Screen reader notifies user once the current task is complete
- `assertive`: Screen reader interrupts the current task to notify user

If the value of the `aria-live` attribute defined for an element is set to `polite`, your screen reader will not be interrupted and will announce the changes in the ARIA live region when the user has no activity on the screen. If the value is set to `assertive`, the new information has high priority and should be notified or announced to the user immediately.

You can also use some of the advanced ARIA live region attributes to intimate information of the entire live region or only a portion of the live region to assistive technologies. Some of the advanced live region attributes to use are:

- **aria-atomic:** The `aria-atomic` attribute is used along with `aria-live` attribute when the page contains live regions. This attribute is used to set whether the assistive technologies should present the entire live region as a single unit or to only announce the regions that have been changed. The possible values are `true` or `false`. The default value is `false`.
- **aria-relevant:** This attribute is used in conjunction with the `aria-live` attribute to describe the types of changes that have occurred within a given live region that should be announced to the user. The possible settings are `additions`, `removals`, `text`, and `all`. The default setting is `additions text`.

The below example shows a sample of the ARIA live region defined in the `index.html` file of an application. In this example, the `aria-live` attribute is set to `assertive` and the `aria-atomic` attribute is set to `true`:

```
<div id="globalBody" class="oj-offcanvas-outer-wrapper oj-offcanvas-page">
    <!-- Region for announcing messages to Screen Readers -->
    <div id="announce" class="sendOffScreen" :aria-live="[[manner() ? manner() : 'assertive']]" aria-atomic="true">
        <p id="ariaLiveMessage"><oj-bind-text value="[[message]]"></oj-bind-text>
    </div>
    <div id="navDrawer" role="navigation" class="oj-contrast-marker oj-web-applayout-offcanvas oj-offcanvas-start">
        <oj-navigation-list id="navDrawerList" data="[[navDataProvider]]" edge="start" item.renderer="[[KnockoutTemplateUtils.getRenderer('navTemplate', true)]]" on-click="[[toggleDrawer]]" selection="{{selection.path}}">
            </oj-navigation-list>
    </div>
    <div id="pageContent" class="oj-web-applayout-page">
        ... contents omitted
    </div>
```

The following example shows how to set up the observables and event listeners in the `appController.js` file of the application.

```
define(['knockout', 'ojs/ojresponsiveutils', 'ojs/ojresponsiveknockoututils', 'ojs/ojcorerouter', 'ojs/ojarraydataprovider', 'ojs/ojknockout', 'ojs/ojoffcanvas'],
    function (ko, ResponsiveUtils, ResponsiveKnockoutUtils, CoreRouter, ArrayDataProvider) {
        function ControllerViewModel() {
            var self = this;

            self.manner = ko.observable('assertive');
            self.message = ko.observable();
            function announcementHandler(event) {
                self.message(event.detail.message);
                self.manner(event.detail.manner);
```

```

};

document.getElementById('globalBody').addEventListener('announce',
announcementHandler, false);

var smQuery =
ResponsiveUtils.getFrameworkQuery(ResponsiveUtils.FRAMEWORK_QUERY_KEY.SM_ONLY);
self.smScreen =
ResponsiveKnockoutUtils.createMediaQueryObservable(smQuery);
var mdQuery =
ResponsiveUtils.getFrameworkQuery(ResponsiveUtils.FRAMEWORK_QUERY_KEY.MD_UP);
self.mdScreen =
ResponsiveKnockoutUtils.createMediaQueryObservable(mdQuery);

let navData = [
    { path: '', redirect: 'dashboard' },
    { path: 'dashboard', detail: { label: 'Dashboard', iconClass: 'oj-ux-ico-bar-chart' } },
    { path: 'incidents', detail: { label: 'Incidents', iconClass: 'oj-ux-ico-fire' } },
    { path: 'customers', detail: { label: 'Customers', iconClass: 'oj-ux-ico-contact-group' } },
    { path: 'about', detail: { label: 'About', iconClass: 'oj-ux-ico-information-s' } }
];

// Router setup
let router = new CoreRouter(navData, {
    urlAdapter: new UrlParamAdapter()
});
router.sync();

this.moduleAdapter = new ModuleRouterAdapter(router);

this.selection = new KnockoutRouterAdapter(router);

// Setup the navDataProvider with the routes, excluding the first
redirected
// route.
this.navDataProvider = new ArrayDataProvider(navData.slice(1),
{keyAttributes: "path"});
... contents omitted

```

To send the announcement, you can use a dispatcher that can be defined within the page viewModel files or in the router enter method. The following example shows a page viewModel file that fires a dispatch in the `self.transitionCompleted` lifecycle method.

```
define(['knockout', 'ajs/ojpopup'],
function (ko) {
```

```
function DashboardViewModel() {
    var self = this;

    self.transitionCompleted = function (info) {
        var message = "Loaded Dashboard page"
        var params = {
            'bubbles': true,
            'detail': { 'message': message, 'manner': 'assertive' }
        };
        document.getElementById('globalBody').dispatchEvent(new
CustomEvent('announce', params));
    };
}
```

For more information on using an ARIA live region, see [ARIA Live Regions](#).

14

Internationalizing and Localizing Applications

Oracle JET supports internationalization and globalization of Oracle JET web and hybrid mobile applications. Configure your Oracle JET application so that the application can be used in a variety of locales and international user environments.

Topics:

- [Internationalize and Localize Oracle JET Applications - A Typical Workflow](#)
- [About Internationalizing and Localizing Oracle JET Applications](#)
- [Internationalize and Localize Oracle JET Applications](#)
- [Work with Oracle JET Translation Bundles](#)

Internationalize and Localize Oracle JET Applications - A Typical Workflow

Understand Oracle JET's internationalization and localization support. Optionally, learn how to merge translations into the Oracle JET translation bundle.

To add internationalization and localization to your Oracle JET application, refer to the typical workflow described in the following table:

Task	Description	More Information
Understand internationalization and localization support in Oracle JET	Understand internationalization and localization support and how it applies to Oracle JET applications.	About Internationalizing and Localizing Oracle JET Applications
Internationalize and localize an Oracle JET application	Use Oracle JET's internationalization features to add internationalization support to an Oracle JET application.	Internationalize and Localize Oracle JET Applications
Add translations to an Oracle JET application	Merge your translations into the Oracle JET translation bundle.	Work with Oracle JET Translation Bundles

About Internationalizing and Localizing Oracle JET Applications

Oracle JET includes support for internationalization (I18N) , localization (L10N), and Oracle National Language Support (NLS) translation bundles.

Internationalization (I18N) is the process of designing software so that it can be adapted to various languages and regions easily, cost effectively, and, in

particular, without engineering changes to the software. Localization (L10N) is the use of locale-specific language and constructs at run time. Oracle has adopted the industry standards for I18N and L10N such as World Wide Web Consortium (W3C) recommendations, Unicode technologies, and Internet Engineering Task Force (IETF) specifications to enable support for the various languages, writing systems, and regional conventions of the world.

Languages and locales are identified with a standard language tag and processed as defined in [BCP 47](#).

Oracle JET includes Oracle National Language Support (NLS) translation support for the languages listed in the following table.

Language	Language Tag
Arabic	ar
Brazilian Portuguese	pt
Canadian French	fr-CA
Chinese (Simplified)	zh-Hans (or zh-CN)
Chinese (Traditional)	zh-Hant (or zh-TW)
Croatian	hr
Czech	cs
Danish	da
Dutch	nl
Estonian	et
Finnish	fi
French	fr
German	de
Greek	el
Hebrew	he
Hungarian	hu
Icelandic	is
Italian	it
Latin_Serbian	sr-Latn
Latvian	lv
Lithuanian	lt
Japanese	ja
Korean	ko
Norwegian	no
Polish	pl
Portuguese	pt-PT
Romania	ro
Russian	ru
Serbian	sr
Slovak	sk

Language	Language Tag
Slovenian	sl
Spanish	es
Swedish	sv
Thai	th
Turkish	tr

Oracle JET translations are stored in a resource bundle. You can add your own translations to the bundles. For additional information see [Add Translation Bundles to Oracle JET](#).

Oracle JET also includes formatting support for over 180 locales. Oracle JET locale elements are based upon the Unicode Common Locale Data Repository (CLDR) and are stored in locale bundles. For additional information about Unicode CLDR, see <http://cldr.unicode.org>. You can find the supported locale bundles in the Oracle JET distribution:

```
js/libs/oj/vxxx/resources/nls
```

It is the application's responsibility to determine the locale used on the page. Typically the application determines the locale by calculating it on the server side from the browser locale setting or by using the user locale preference stored in an identity store and the supported translation languages of the application.

Once the locale is determined, your application must communicate this locale to Oracle JET for its locale sensitive operations such as loading resource bundles and formatting date-time data. Oracle JET determines the locale for locale sensitive operations in the following order:

1. Locale specification in the RequireJS configuration.
2. lang attribute of the `html` tag.
3. `navigator.language` browser property or `navigator.userLanguage` Internet Explorer property.

Setting the `lang` attribute on the `html` tag is the recommended practice because, in addition to setting the locale for Oracle JET, it sets the locale for all HTML elements as well. Oracle JET automatically loads the translations bundle for the current language and the locale bundle for the locale that was set. If you don't set a locale, Oracle JET will default to the browser property.

Note:

The locale and resource bundles are loaded automatically only when your application uses the RequireJS `ojI10n` plugin. For information about using RequireJS in your Oracle JET application, see [Using RequireJS for Modular Development](#).

Finally, Oracle JET includes validators and converters that use the locale bundles. When you change the locale on the page, an Oracle JET component has built in

support for displaying content in the new locale. For additional information about Oracle JET's validators and converters, see [Validating and Converting Input](#).

Internationalize and Localize Oracle JET Applications

Configure your application to use Oracle JET's built-in support for internationalization and localization.

Topics:

- [Use Oracle JET's Internationalization and Localization Support](#)
- [Enable Bidirectional \(BiDi\) Support in Oracle JET](#)
- [Set the Locale Dynamically](#)
- [Work with Currency, Dates, Time, and Numbers](#)

Use Oracle JET's Internationalization and Localization Support

To use Oracle JET's built-in internationalization and localization support, you can simply specify one of the supported languages or locales on the `lang` attribute of the `html` element on your page. For example, the following setting will set the language to the French (France) locale.

```
<html lang="fr-FR">
```

If you want to specify the French (Canada) locale, you would specify the following instead:

```
<html lang="fr-CA">
```

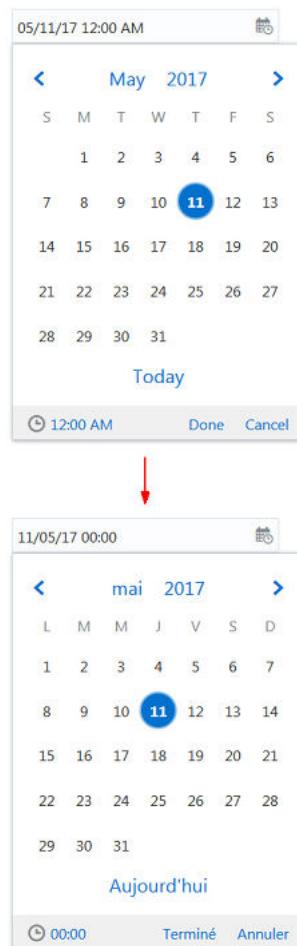
Tip:

The `lang` specification isn't case sensitive. Oracle JET will accept `FR-FR`, `fr-fr`, etc., and map it to the correct resource bundle directory.

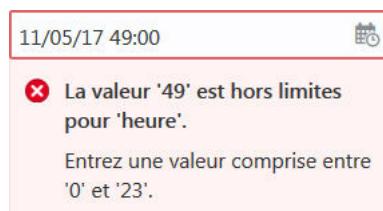
When you specify the locale in this manner, any Oracle JET component on your page will display in the specified language and use locale constructs appropriate for the locale.

If the locale doesn't have an associated resource bundle, Oracle JET will load the lesser significant language bundle. If Oracle JET doesn't have a bundle for the lesser significant language, it will use the default root bundle. For example, if Oracle JET doesn't have a translation bundle for `fr-CA`, it will look for the `fr` resource bundle. If the `fr` bundle doesn't exist, Oracle JET will use the default root bundle and display the strings in English.

In the image below, the page is configured with the `oj-input-date-time` component. The figure shows the effect of changing the `lang` attribute to `fr-FR`.



If you type an incorrect value in the `oj-input-date-time` field, the error text is displayed in the specified language. In this example, the error is displayed in French.

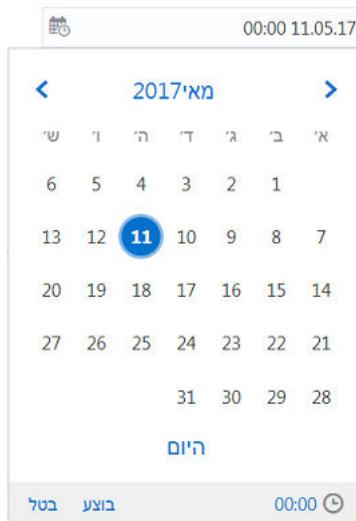


Enable Bidirectional (BiDi) Support in Oracle JET

If the language you specify uses a right-to-left (RTL) direction instead of the default left-to-right (LTR) direction, such as Arabic and Hebrew, you must specify the `dir` attribute on the `html` tag in addition to the `lang` attribute. The code below shows an example that specifies the Hebrew Israel (he-IL) locale with BiDi support enabled.

```
<html lang="he-IL" dir="rtl">
```

The image below shows the same `oj-input-date-time` field that displays if you specify the Hebrew Israel locale and change the `dir` attribute to `rtl`.



Once you have enabled BiDi support in your Oracle JET application, you must still ensure that your application displays properly in the desired layout and renders strings as expected.

 **Note:**

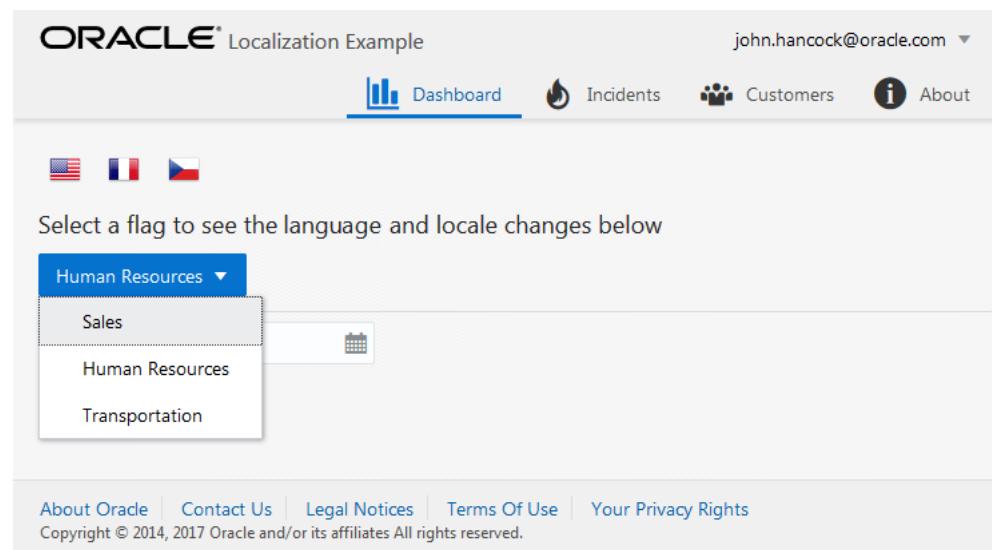
JET does not support the setting of the `dir` attribute on individual HTML elements which would cause a page to show mixed directions. Also, if you programmatically change the `dir` attribute after the page has been initialized, you must reload the page or refresh each JET component.

Set the Locale Dynamically

You can configure your application to change the page's locale dynamically by using the `Config.setLocale()` function:

```
setLocale(locale, callback)
```

The image below shows the Oracle JET application configured to display a menu that displays a department list when clicked and a date picker. By default, the page is set to the `en-US` locale. Both the menu and date picker are displayed in English.

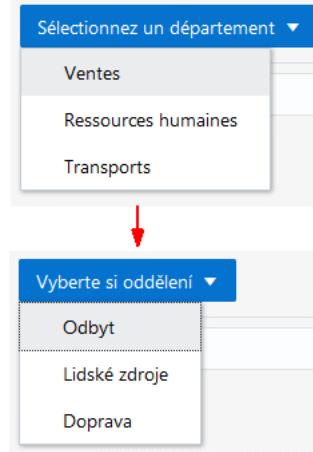


The application also includes a button set which shows the United States of America, France, and Czech Republic flags. When the user clicks one of the flags, the page locale is set to the locale represented by the flag: en-US, fr-FR, or cs-CZ.

 **Note:**

The flags used in this example are for illustrative use only. Using national flags to select a UI language is strongly discouraged because multiple languages are spoken in one country, and a language may be spoken in multiple countries as well. In a real application, you can use clickable text instead that indicates the preferred language to replace the flag icons.

The figure below shows the updated department list after the user clicks the French and Czech Republic flags.



The code that sets the locale in this example uses the `Config.setLocale()` function call highlighted below. The menu is refreshed in the `ViewModel` to reload the department list for the chosen locale.

```
// When the country flags are clicked we get a new language to set as the
// current locale
self.setLang = function(evt) {
    var newLang = '';
    var lang = evt.currentTarget.id;
    switch (lang){
        case '?čeština':
            newLang = 'cs-CZ';
            break;
        case 'français':
            newLang = 'fr-FR';
            break;
        default:
            newLang = 'en-US';
    }
    Config.setLocale(newLang,
        function() {
            document.getElementsByTagName('html')[0].setAttribute('lang',
newLang);
            // In this callback function we can update whatever is needed with the
            // new locale. In this example, we reload the menu items.
            loadMenu();
        }
    );
}
```

When the application changes the locale by calling `refresh()` on the `oj-input-date` component, the page will automatically update to use the new locale and display the menu and date in the new locale. However, you must explicitly define the strings that appear in the menu items and then retrieve those strings using the `Translations.getTranslatedString()` method.

```
getTranslatedString(key, var_args)
```



Note:

Do not use this functionality unless the application itself can switch the UI locale dynamically. Dynamically changing the UI locale often ends up with the UI in mixed languages or locales because the application may have cached data that are locale sensitive.

The code that loads the menu is shown below. The menu items and menu button labels are defined with a call to `getTranslatedString()`. The `refresh()` method of both the menu and date component are called after the translations and locale bundles are loaded for the new locale to refresh the display.

```
function DashboardViewModel() {
    var self = this;

    // Setting up knockout observables for the button label and the menu items
    self.localeLabel = ko.observable();
    self.menuNames = ko.observableArray([]);
```

```

        self.changeLabel = function (evt) {
            self.localeLabel(Translations.getTranslatedString(evt.target.value));
        }

        self.setLang = function (evt) {
            ...contents omitted
        }

        // This function loads the menu items.
        function loadMenu() {
            // These two lines are pulling the translated values for the menu items from
            // the
            // appropriate resource file in the /resources/nls directory
            self.menuNames(
                [
                    {'itemName': Translations.getTranslatedString('menu1')},
                    {'itemName': Translations.getTranslatedString('menu2')},
                    {'itemName': Translations.getTranslatedString('menu3')}
                ]
            );
            self.localeLabel(Translations.getTranslatedString('label'));

            // Since we've modified the children of a jqueryUI component, we need
            // to refresh it to get all of the build in styling again
            document.getElementById('buttonMenu').refresh();
            document.getElementById('dateInput').refresh();
        }
    }
}

```

For information about defining your own translation strings and adding them to the Oracle JET resource bundle, see [Add Translation Bundles to Oracle JET](#).

When you use this approach to internationalize and localize your application, you must consider every component and element on your page and provide translation strings where needed. If your page includes a large number of translation strings, the page can take a performance hit.

Also, if SEO (Search Engine Optimization) is important for your application, be aware that search engines normally do not run JavaScript and access static text only.

Tip:

To work around issues with performance or SEO, you can add pages to your application that are already translated in the desired language. When you use pages that are already translated, the Knockout bindings are executed only for truly dynamic pieces.

Work with Currency, Dates, Time, and Numbers

When you use the converters included with Oracle JET, dates, times, numbers, and currency are automatically converted based on the locale settings. You can also provide custom converters if the Oracle JET converters are not sufficient for your application. For additional information about Oracle JET converters, see [About Oracle JET Converters](#). For information about adding custom converters to your application, see [Use Custom Converters in Oracle JET](#).

Work with Oracle JET Translation Bundles

Oracle JET includes a translation bundle that translates strings generated by Oracle JET components into all supported languages. Add your own translation bundle by merging it with the Oracle JET bundle.

Topics

- [About Oracle JET Translation Bundles](#)
- [Add Translation Bundles to Oracle JET](#)

About Oracle JET Translation Bundles

Oracle JET includes a translation bundle that translates strings generated by Oracle JET components into all supported languages. You can add your own translation bundle following the same format used in Oracle JET.

The Oracle JET translation bundle follows a specified format for the content and directory layout but also allows some leniency regarding case and certain characters.

Translation Bundle Location

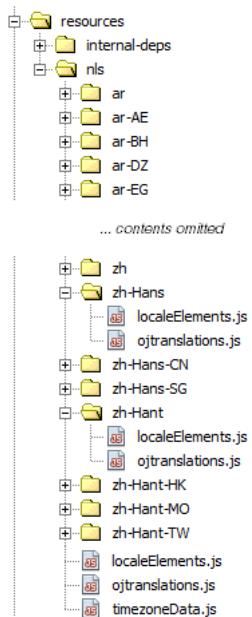
The RequireJS bootstrap file (usually `main.js`) contains the location of the Oracle JET translation bundle, which is named `ojtranslations.js`.

```
libs/obj/vxxx/resources/nls/ojtranslations
```

Each supported language is contained in a directory under the `nls` directory. The directory names use the following convention:

- lowercase for the language sub-tag (`zh`, `sr`, etc.)
- title case for the script sub-tag (`Hant`, `Latn`, etc.)
- uppercase for the region sub-tag (`HK`, `BA`, etc.)

The language, script, and region sub-tags are separated by hyphens (-). The following image shows a portion of the directory structure.



Top Level Module

The `ojtranslations.js` file contains the strings that Oracle JET translates and lists the languages that have translations. This is the top level module or root bundle. In the root bundle, the strings are in English and are the runtime default values when a translation isn't available for the user's preferred language.

Translation Bundle Format

Oracle JET expects the top level root bundle and translations to follow a specified format. The root bundle contains the Oracle JET strings with default translations and the list of locales that have translations. // indicates a comment.

```

define({
  // root bundle
  root: {
    "oj-message": {
      fatal:"Fatal",
      error:"Error",
      warning:"Warning",
      info:"Info",
      confirmation:"Confirmation",
      "compact-type-summary":"{0}: {1}"
    },
    // ... contents omitted
  },
  // supported locales.
  "fr-CA":1,
  ar:1,
  ro:1,
  "zh-Hant":1,
  nl:1,
}

```

```

    it:1,
    fr:1,
    // ... contents omitted
    tr:1,fi:1
});

```

The strings are defined in nested JSON objects so that each string is referenced by a name with a prefix: `oj-message.fatal`, `oj-message.error`, etc.

The language translation resource bundles contain the Oracle JET string definitions with the translated strings. For example, the following code sample shows a portion of the French (Canada) translation resource bundle, contained in `nls/fr-CA/ojtranslations.js`.

```

define({
  "oj-message": {
    fatal:"Fatale",
    error:"Erreur",
    warning:"Avertissement",
    info:"Infos",
    confirmation:"Confirmation",
    "compact-type-summary":"{0}: {1}"
  },
  // ... contents omitted
});

```

When there is no translation available for the user's dialect, the strings in the base language bundle will be displayed. If there are no translations for the user's preferred language, the root language bundle, English, will be displayed.

Named Message Tokens

Some messages may contain values that aren't known until runtime. For example, in the message "User foo was not found in group bar", the `foo` user and `bar` group is determined at runtime. In this case, you can define `{username}` and `{groupname}` as named message tokens as shown in the following code.

```
"MyUserKey": "User {username} was not found in group {groupname}."
```

At runtime, the actual values are replaced into the message at the position of the tokens by calling `Translations.getTranslatedString()` with the key of the message as the first argument and the parameters to be inserted into the translated pattern as the second argument.

```
var params = {'username': 'foo', 'groupname': 'bar'};
Translations.getTranslatedString("MyUserKey", params);
```

Numeric Message Tokens

Alternatively, you can define numeric tokens instead of named tokens. For example, in the message "This item will be available in 5 days", the number 5 is determined at runtime. In this case, you can define the message with a message token of `{0}` as shown in the following code.

```
"MyKey": "This item will be available in {0} days."
```

A message can have up to 10 numeric tokens. For example, the message "Sales order {0} has {1} items" contains two numeric tokens. When translated, the tokens can be reordered so that message token {1} appears before message token {0} in the translated string, if required by the target language grammar. The JavaScript code that calls `getTranslatedString()` remains the same no matter how the tokens are reordered in the translated string.

 **Tip:**

Use named tokens instead of numeric tokens to improve readability and reuse.

Escape Characters in Resource Bundle Strings

The dollar sign, curly braces and square brackets require escaping if you want them to show up in the output. Add a dollar sign (\$) before the characters as shown in the following table.

Escaped Form	Output
\$\$	\$
\${	{
\$}	}
\$[[
\$]]

For example, if you want your output to display [Date: {01/02/2020}, Time: {01:02 PM}, Cost: \$38.99, Book Name: John's Diary], enter the following in your resource bundle string:

```
"productDetail": "$[Date: ${01/02/2020$}, Time: ${01:02 PM$}, Cost: $38.99, Book Name: John's Diary$]"
```

Add Translation Bundles to Oracle JET

You can add a translation bundle to Oracle JET by merging your new bundle with the existing Oracle JET translation bundle.

To add translation bundles to Oracle JET:

1. Define the translations.

For example, the following code defines a translation set for a menu containing a button label and three menu items. The default language is set to English, and the default label and menu items will be displayed in English. The root object in the file is the default resource bundle. The other properties list the supported locales, `fr` and `cs`.

```
define({
  "root": {
    "label": "Select a department",
    "menu1": "Sales",
    "menu2": "Human Resources",
```

```

        "menu3": "Transportation"
    },
    "fr": true,
    "cs": true
}) ;

```

To add a prefix to the resource names (for example, department.label, department.menu1, and so on), add it to your bundles as shown below.

```

define({
  "root": {
    "department": {
      "label": "Select a department",
      "menu1": "Sales",
      "menu2": "Human Resources",
      "menu3": "Transportation"
    }
  }
},
  "fr": true,
  "cs": true
}) ;

```

When the locale is set to a French locale, the French translation bundle is loaded. The code below shows the definition for the label and menu items in French.

```

define({
  "label": "Sélectionnez un département",
  "menu1": "Ventes",
  "menu2": "Ressources humaines",
  "menu3": "Transports"
}) ;

```

You can also provide regional dialects for your base language bundle by just defining what you need for that dialect.

```

define({
  "label": "Canadian French message here"
}) ;

```

When there is no translation available for the user's dialect, the strings in the base language bundle will be displayed. In this example, the menu items will be displayed using the French translations. If there are no translations for the user's preferred language, the root language bundle, whatever language it is, will be displayed.

2. Include each definition in a file located in a directory named nls.

For example, the default translation in the previous step is placed in a file named menu.js in the js/resources/nls directory. The French translation is located in a file named menu.js in the js/resources/nls/fr directory.

3. In the application's requireJS bootstrap file (typically main.js), add the bundle to the merge option in the ojL10n definition.

```

// This section configures the i18n plugin.
// It is merging the Oracle JET built-in translation resources
// with a custom translation file for the menu items used in this example.
// Any resource file added, must be placed under a directory named "nls".
// You can use a path mapping or you can define
// a path that is relative to the location of this main.js file.

```

```
config: {
  ojL10n: {
    merge: {
      'ojtranslations/nls/ojtranslations': 'resources/nls/menu'
    }
  }
}
```

Oracle JET supports only one custom bundle to be merged with the default JET translation bundle. If your application has more than one translation bundle, combine them into one and then add the bundle to the `merge` option.

Using CSS and Themes in Applications

Oracle JET includes the Redwood theme that provides styling across web or hybrid mobile applications and implements Oracle Redwood Design System. The Redwood theme provides hundreds of custom properties (also called CSS variables) to achieve its look and feel. You can use the Redwood theme as provided, or you can customize the custom properties manually and through the tooling.

Note:

Starting in Oracle JET release 9.0.0, the Redwood theme is the default theme for all new JET web and mobile applications.

Topics:

- [About the Redwood Theme Included with Oracle JET](#)
- [Typical Workflow for Working with Themes in Oracle JET Applications](#)
- [CSS Files Included with the Redwood Theme](#)
- [Create an Application with the Redwood Theme](#)
- [Best Practices for Using CSS and Themes](#)
- [Work with Images](#)
- [Work with Custom Themes](#)
- [Experimental: Work with CSS Variables](#)

About the Redwood Theme Included with Oracle JET

Oracle Redwood Design System is the new Oracle standard for application look and feel. It is being implemented company-wide to unify the user interface of all Oracle product offerings and is implemented in Oracle JET release 9.0.0 as the new Redwood theme.

Oracle JET takes this opportunity to refresh the toolkit look and feel with this dynamic, forward thinking design system and also to introduce all new components that rely on the user experience of Oracle Redwood Design System, such as JET waterfall layout and JET stream list component.

All starter templates for web applications and hybrid mobile applications use the Redwood theme. Because all applications will be created with the Redwood theme by default, you no longer need to specify the theme type when using the JET Tooling create and serve commands. There are no theme variations specific to the mobile platforms. For details, see [Create an Application with the Redwood Theme](#).

If you have an existing application that you want to migrate from the Alta theme, you can migrate to JET release 9.0.0 and configure the application to run with the out of

the box CSS for the Redwood theme. For details, see [Migrate to the Redwood Theme CSS](#).

Currently, customizing Redwood theme by working with CSS variables remains experimental in JET release 9.0.0. In this release, CSS variable usage must be limited to test applications and prototyping of the UI. For details, see [About CSS Variables and Custom Themes in Oracle JET](#).

Typical Workflow for Working with Themes in Oracle JET Applications

Understand themes included in Oracle JET and how to work with them using Oracle JET Tooling.

Oracle JET includes cascading style sheets (CSS) files and syntactically awesome style sheets (Sass) files, variables, and tools. To understand these aspects of the Redwood theme included with Oracle JET, refer to the typical workflow described in the following table.

! Important:

Follow this workflow to maintain compatibility with future Oracle JET versions. Do not modify the styling definitions of Oracle JET classes. Doing so may prevent your application from migrating to a future Oracle JET version.

Content related only to the Alta theme has been removed from this chapter and can be found in [Alta Theme in Oracle JET v9.0.0 and Later](#).

Task	Description	More Information
Understand the Redwood theme	Learn about the Redwood theme that is the default theme for new applications, starting in release 9.0.0.	About the Redwood Theme Included with Oracle JET
Identify the theme CSS files included with Oracle JET	Identify how the theme CSS files support web and mobile development.	CSS Files Included with the Redwood Theme or CSS Files Included in Alta
Understand the best practices for using CSS and Themes	Use the recommended standards for CSS and themes for both Redwood and Alta.	Best Practices for Using CSS and Themes
Work with images	Understand Oracle JET's support for images.	Work with Images
Minimize application CSS	Use the tooling to customize the out of the box Redwood theme to minimize CSS.	Work with Custom Themes
Understand Oracle JET's custom property support	Identify the CSS or Sass files, variables and tools included in Oracle JET that support custom themes.	About CSS Variables and Custom Themes in Oracle JET or Work with Sass

Task	Description	More Information
Customize application look and feel	Override CSS variables to control JET component UI and optionally generate a custom CSS to process away the variables. Note: Customization with CSS variables is preview-only in Oracle JET release 9.0.0.	Experimental: Work with CSS Variables or Customize Alta Themes Using the Tooling
Re-generate your custom theme CSS	Migrate your application to a later JET release and rebuild the theme CSS using Oracle JET Tooling, then migrate your theme to work with the latest Redwood theme or Alta theme updates.	Oracle JET Application Migration for Release 9.2.0

CSS Files Included with the Redwood Theme

Oracle JET includes CSS files designed for display on web browsers and hybrid mobile applications that implement Oracle Redwood Design System. In JET, the Redwood theme includes minified and readable versions of the CSS.

Starting in Oracle JET release 9.0.0, application themes are based on Redwood theme, a single design system for both mobile and browser applications that replaces the multiple themes needed in prior releases.

The Redwood CSS is included with the Oracle JET distribution and is located in the `/<app_root>/node_modules/@oracle/oraclejet/dist/css/redwood` folder. The Redwood CSS distribution contains the following files:

- `oj-redwood.css`: Readable version of the CSS for the out of the box Redwood theme
- `oj-redwood-min.css`: Minified version of the CSS for the out of the box Redwood theme

In addition, the Redwood theme includes the following CSS:

- `oj-redwood-notag.css`: Readable version of the CSS without tag selectors
- `oj-redwood-notag-min.css`: Minified version of the CSS without tag selectors.

For additional details about Oracle JET theming and tag selectors, see [Disable JET Styling of Base HTML Tags](#).

If the CSS files provided by Oracle JET are sufficient and you only want to add a few application-specific styles, you may find that adding the classes to `app.css` in your application's `/src/css` folder will meet your needs.

If, however, you want to use a custom theme or add more than a few application-specific classes, then you can use Oracle JET Tooling to run your application with CSS variables that you override in your application CSS. When you use the tooling to create a custom theme, the tooling adds the following CSS from the Oracle JET distribution to your application:

- `oj-redwood-cssvars.css`: Readable version of the CSS variable definitions for Oracle JET components

- `oj-redwood-cssvars-min.css`: Minified version of the CSS variable definitions.

Currently, the process of using CSS variables to customize application look and feel is experimental, as described in [Experimental: Work with CSS Variables](#). Although CSS variables support customization, they are not supported on all browsers and resolving CSS variables at runtime can impact performance.

Always use the recommended standards to work with your CSS and themes. For more information, see [Best Practices for Using CSS and Themes](#).

! Important:

Do not override the style classes in the Oracle JET CSS distribution. The CSS files shipped with Oracle JET are considered private and must not be modified. Such modifications may prevent you from migrating your theme to a future release.

Create an Application with the Redwood Theme

The Redwood theme is the Oracle JET implementation of Oracle Redwood Design System and is the default theme for applications that you create in JET release 9.0.0 and later.

Oracle Redwood Design System is the new Oracle user experience design language, and Redwood theme is the recommended theme if you are creating a new JET web or hybrid mobile application.

You use the `ojet create` command to scaffold an application that by default uses the Redwood CSS files provided with the Redwood theme distribution. When you build the application, the JET Tooling loads one of two versions of the CSS. It will either load `redwood-cssvars.css` that contains the CSS variables that define the Redwood theme styles or it loads `redwood.css`, where the CSS variables are processed away. By default, your application will use the processed `redwood.css` for performance reasons.

When you create your application, JET Tooling will use the out of the box Redwood theme file as configured by the property `defaultTheme=redwood` in the `oraclejetconfig.json` file. There are no other configuration settings needed for the tooling to use the `redwood.css` when you build and run the application.

```
{  
  "paths": {  
    ...  
  },  
  "defaultBrowser": "chrome",  
  "sassVer": "x.x.x",  
  "defaultTheme": "redwood",  
  "defaultCssvars": "disabled",  
  "generatorVersion": "9.2.0"  
}
```

Working with CSS variables is currently supported as an experimental process in Oracle JET releases 9.x.0. For details about using the `redwood-cssvars.css` file and overriding CSS variables, see [Experimental: Work with CSS Variables](#).

To create an application with the Redwood theme:

1. Create the application.

```
ojet create my-web-app
```

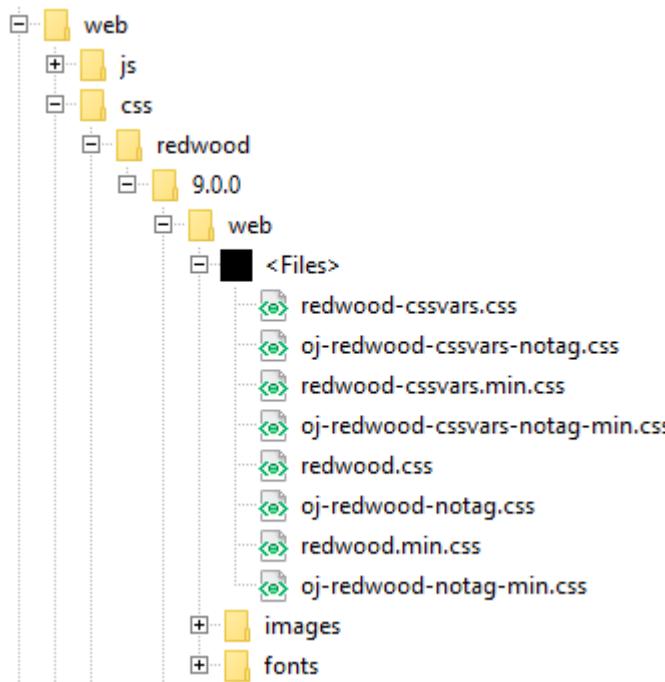
The tooling creates the `index.html` file in the `src` folder with a stylesheet link that will load the CSS for the out of the box Redwood theme.

```
<!-- injector:theme -->
<link rel="stylesheet" href="css/libs/oj/v9.2.0/redwood/oj-redwood-
min.css" id="css" />
<!-- endinjector -->
```

2. Build a development version of your application.

```
ojet build
```

The tooling outputs the built application source to the `web` build folder in the application's root and populates the `/web/css/redwood/9.2.0/web` folder with the set of available Redwood theme files, including the default theme `redwood.css`, as well as fonts and images.



Theme files with the `oj-` prefix and `-notag` suffix provide alternatives to the out of the box Redwood theme that allow you to work with custom themes, as described in [Work with Custom Themes](#) and in [Experimental: Work with CSS Variables](#).

Note that `index.html` in the build output folder by default loads the expanded CSS from `redwood.css`. To load minified CSS from `redwood-min.css`, build the application in release mode.

```
ojet build --release
```

3. Run the application to view the out of the box Redwood theme in the browser.

```
ojet serve
```

To run your application with CSS that you maintain, add the `--theme` option.

```
ojet serve --theme=themeName
```

Application CSS that you maintain is located by default in the `/src/css` folder of your project.

Best Practices for Using CSS and Themes

Use the recommended styling standards for creating CSS and themes in your web or hybrid mobile application. These practices apply to all themes in Oracle JET, including Redwood theme and Alta theme.

Standard	Details	Example
Never override Oracle JET classes	While there are ways to override Oracle JET CSS style classes containing the <code>oj-</code> prefix, none of them is allowed, whether public or private. Follow the workflow documented in Typical Workflow for Working with Themes in Oracle JET Applications .	 Here is one example of an overridden Oracle JET class that is not allowed: <pre>.acme-branding-header .oj-button{ color: white; background: blue; }</pre>
Use mobile-first design	Applications should be mobile first, meaning that they should work on a phone and tablet. This means they must be touch friendly, including sizing tap targets appropriately . There is no hover in mobile, so the design should not rely on the use of hover.	

Standard	Details	Example
Follow naming conventions	<p>Use the <code>.namespace-block-element</code> naming convention for your CSS file.</p> <p>For example, if you are writing a branding bar that contains a header element for your company acme, then choose acme as the namespace, branding as the block, and header as the element. So, the selector name will be <code>.acme-branding-header</code>. Oracle JET uses dashes in their selector names, but you may use other naming conventions such as Block, Element, Modifier (BEM).</p> <p>Including a namespace is important in order to minimize the chances of your base CSS (for example, one provided by a client) on the page affecting your application's CSS and vice-versa. For example, if your CSS and the client-provided CSS both have a class named <code>branding-header</code> as seen below, your branding header text color will be red.</p>	<code>.acme-branding-header</code>
	<p>Client CSS:</p> <pre>.branding-header{color: red}</pre> <p>Your CSS:</p> <pre>.branding-header{background-color: blue}</pre> <p>Using <code>.acme-branding-header</code> rather than just <code>.branding-header</code> will greatly reduce the chance that your client will use a class with the same name.</p>	
Use only Oracle JET public classes	Use only the public classes documented in the Oracle® JavaScript Extension Toolkit (JET) Styling Reference . All other Oracle JET classes are considered private and subject to change without notice.	
Minimize custom CSS	JET has quite a few CSS Utility classes that may be helpful. For example, you can style your application for responsive design using flex layout classes, hide content at various screen sizes, style panels for color, control padding for various screen widths, and so on. When you use JET style classes in your application, you also ensure compatibility with the Oracle JET provided theme.	
Don't set the font family in the CSS	The application should set the font family for text once on the root of the page which allows the application to change the font family as needed. In order to blend in with the font family chosen by the application, do not set the font family in the CSS.	<p> Do not set the font family like this:</p> <pre>.acme-branding-header { font-family: arial; }</pre>

Standard	Details	Example
Use REM for font sizes and CSS styles	Consider using REM (root em) for font sizes and other CSS style properties where relative sizing allows your application to scale based on the root html element font size. Oracle JET components rely on REM, which allows your application to adjust to any changes to the underlying CSS font sizes for the themes included with Oracle JET. Likewise, your application can benefit by working with rem units instead of pixels or some other absolute unit. You can use rem units where ever an HTML style can benefit from scalable length units, such as the CSS properties for line-height, width, height, padding, margin, and so on. If you do not want to set a style directly on the html tag, you can reference the oj-html class as described in Use Tag Selectors or Style Classes with Alta .	.acme-branding-header { font-size: 1.2rem; }
Add bi-directional (BIDI) styling support	Oracle JET applications are expected to set dir="rtl" for right-to-left (RTL) languages as described in Set the Text Direction . You can use this setting to support both left-to-right (LTR) and RTL languages in your CSS.	html:not([dir="rtl"]) .acme-branding-header { right: 0; } html[dir="rtl"] .acme-branding-header { left: 0; }
Use oj-hicontrast for high contrast styling	When Oracle JET detects high contrast mode, it places the oj-hicontrast selector on the body element which you can use to change the CSS as needed. See Configure High Contrast Mode .	.acme-branding-header { border: 1px; } .oj-hicontrast .acme-branding-header { border: 2px; }
Avoid !important	Avoid the use of !important in your CSS as it makes it problematic to override the value. Where possible, use higher specificity instead. See the Mozilla specificity page for more information.	 Avoid using !important. .acme-branding-header { font-size: 1.2rem ! important; }
Optimize image use	All image systems have advantages and disadvantages. See Work with Images to decide if icon fonts are right for you. Always consider performance when using images. For tips, see Add Performance Optimization to an Oracle JET Application	

DOCTYPE Requirement

In order for Oracle JET's theming to work properly, you must include the following line at the top of all HTML5 pages:

```
<!DOCTYPE html>
```

If you don't include this line, the CSS in your application may not function as expected. For example, you may notice that some elements aren't properly aligned.

Tip:

If you create an Oracle JET application using the tooling or one of the sample applications, this line is already added for you, and you do not need to add it yourself.

ThemeUtils

Oracle JET provides the `ThemeUtils` class that you can use to obtain information about the current theme and use that information to generate and apply a class.

The `ThemeUtils` class provides the following services:

- Return the name of the current theme.
- Return the target platform of the current theme.

Returns `all` for the Redwood theme, because the Redwood theme is the same for web applications and all mobile device platforms.

For an example that shows how to vary background color based on the current theme's name and target platform, see [Theme Info](#). For information about the specific services that the `ThemeUtils` class provides, see [ThemeUtils](#).

Set the Text Direction

If the language you specify uses a right-to-left (RTL) direction instead of the default left-to-right (LTR) direction, such as Arabic and Hebrew, you must specify the `dir` attribute on the `html` tag: `<html lang="name" dir="rtl">`.

For example, the following code specifies the Hebrew Israel (he-IL) locale with right-to-left direction enabled:

```
<html lang="he-IL" dir="rtl">
```

Oracle JET does not support multiple directions on a page. The reason this is not supported is that the proximity of elements in the document tree has no effect on the CSS specificity, so switching directions multiple times in the page may not work the way you might expect. The code sample below shows an example.

```
<style>
  [dir=ltr] .foo {color: blue}
  [dir=rtl] .foo {color: red}
</style>
```

```
<span dir="rtl">
  <span dir="ltr">
    <span class="foo">You might think I will be blue because dir=ltr is on,
      a closer ancestor than dir=rtl. But css doesn't care
      about proximity, so instead I am red
      (because [dir=rtl] .foo {color: red} was defined last).
    Isn't that surprising?
  </span>
</span>
</span>
```

For more information about localizing your application and adding bidirectional support, see [Enable Bidirectional \(BiDi\) Support in Oracle JET](#). For more information about CSS specificity, see <https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity>.

Work with Images

Oracle JET uses icon fonts whenever possible to render images provided by the Redwood theme. When icon fonts are not possible, Oracle JET uses SVG images.

You may also find the following topics helpful when working with images.

Topics:

- [Image Considerations](#)
- [Icon Font Considerations](#)

Image Considerations

There are a variety of ways to load icons, such as sprites, data URIs, icon fonts, and so on. Factors to consider when choosing an image strategy include:

- Themable: Can you use CSS to change the image? Can you replace a single image easily?
- High contrast mode: Does the image render properly in high contrast mode for accessibility?
- High resolution support: Does the image look acceptable on high resolution (retina) displays?
- Image limitations: Are there limitations that impact your use case? For example, icon fonts are a single color, and small SVG images often do not render well.
- Performance: Is image size a factor? Do you need alternate versions of an image for different resolutions or states such as disabled, enabled, hover, and active?

Icon Font Considerations

Oracle JET uses icon fonts whenever possible because icon fonts have certain advantages over other formats.

- Themable: You can use style classes to change their color instead of having to replace the image, making them very easy to theme.
- High contrast mode: Icon fonts are optimal for high contrast mode as they are considered text. However, keep in mind that you can't rely on color in high contrast

mode, and you may need to indicate state (active, hover, and so on) using another visual indicator. For example, you can add a border or change the icon font's size. For additional information about Oracle JET and high contrast mode, see [Configure High Contrast Mode](#).

- High resolution: Icon fonts look good on a high resolution (retina) display without providing alternate icons.
- Performance: You can change icon font colors using CSS so alternate icons are not required to indicate state changes. Alternate images are also not required for high resolution displays.

Icon fonts also have disadvantages. It can be difficult to replace a single image, and they only show one color. You can use text shadows to provide some depth to the icon font.

Work with Custom Themes

There may be cases where you want to create a custom theme without the need to customize Oracle JET component look and feel. For example, Oracle JET Tooling adds support to limit the CSS to only the components in your application or to enable styling in base HTML tags instead of the default styling provided by JET.

For look and feel customization, see [Experimental: Work with CSS Variables](#).

Topics:

- [Add Custom Theming Support](#)
- [Disable JET Styling for Unused JET Components](#)
- [Disable JET Styling of Base HTML Tags](#)

Add Custom Theming Support

You can use Oracle JET Tooling to add support for custom theming to your application.

Adding theming support is a prerequisite step to performing any modifications to the out of the box Redwood theme.

The Oracle JET distribution provides the CSS that your application will use by default to load the out of the box Redwood theme. When you want to theme your application, you can use the Oracle JET Tooling to add theming support to your application to include Redwood-specific CSS settings files.

At the most basic level you can work with the custom theme settings files to minimize the out of the box Redwood CSS. For example, you can minimize the JET component CSS that you want to load or you can specify not to load the CSS for base HTML tags.

Additionally, you can prepare your application to use your custom CSS when you want to directly modify CSS variable settings.

To add theming support:

1. In your application's top level directory, enter the following command at a terminal prompt to install the PostCSS tooling chain.

```
ojet add theming
```

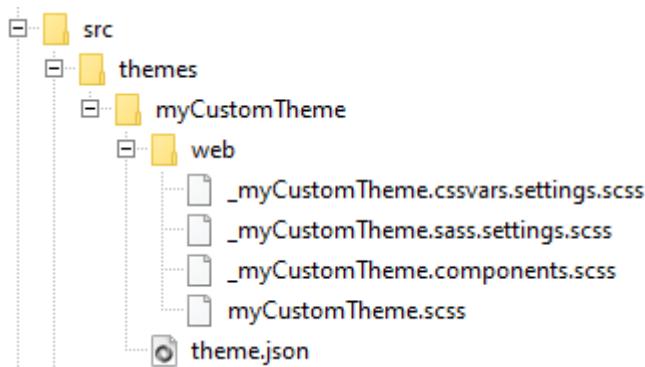
2. Add the custom theme settings files to your application in a folder named for your custom theme.

```
ojet create theme themeName
```

For example, the following command creates a custom theme named `myCustomTheme`.

```
ojet create theme myCustomTheme
```

The command creates a folder with the custom theme name in the application `/src/themes` directory.



In the figure above, the `web` folder shows the `.scss` custom theme settings files that you can modify. The `myCustomTheme.scss` file is the aggregating file for the custom theme settings. You can use the `_myCustomTheme.components.scss` file to customize the scope of the Redwood CSS for JET components and HTML base tags.

Note that you can optionally use the `_myCustomTheme.cssvars.settings.scss` and `_myCustomTheme.sass.settings.scss` files to customize CSS by altering custom property values (CSS variables and SCSS variables).

You'll find out more about working with these files at the end of this section.

The `theme.json` file contains the version number of the theme, starting with `0.0.1`.

3. In the application root, edit the `oraclejetconfig.json` file and set the `defaultTheme` property to the custom theme name that you specified with the `ojet create theme` command.

```
{
  "paths": {
    ...
  },
  "defaultBrowser": "chrome",
  "sassVer": "x.x.x",
  "defaultTheme": "myCustomTheme",
  "defaultCssvars": "disabled",
  "generatorVersion": "9.2.0"
}
```

In this sample, `myCustomTheme` is the name of the custom theme shown in the previous step.

If you make no other changes, when you build and run the application, the CSS will load the out of the box Redwood theme.

To work with the settings files to customize the theme, you can perform these additional tasks:

- [Disable JET Styling for Unused JET Components](#)
- [Disable JET Styling of Base HTML Tags](#)
- [Experimental: Work with CSS Variables \(preview only in JET release 9.x.0\)](#)

Disable JET Styling for Unused JET Components

By default when you build the application, JET Tooling loads CSS with all Redwood style classes enabled. You can configure the CSS to use just the styles required by the components of your application.

The Redwood CSS styles loaded at build time are controlled by import statements in custom theme settings files that you can optionally add to your project. To work with the CSS settings files in your project, you need to add theming support to the application and then create a custom theme.

With the custom theme settings files added to the `/src/themes` folder of your project, you can reduce the size of the CSS by commenting and uncommenting import statements in the settings files for specific JET component style classes. With the appropriate import settings completed, JET Tooling will process the CSS settings files and load CSS for only the JET components you specified.

Before you begin:

- Install the Redwood theme tooling chain and configure a custom theme, as described in [Add Custom Theming Support](#). The `themes` folder with custom theme settings files are added to your application source.

To reduce CSS by limiting component CSS:

1. In the `/src/themes/themeName/web` folder of your application, edit the aggregating `themeName.scss` file, and comment out the import statement that by default enables importing all JET component styles.

```
//@import "oj/all-components/themes/redwood/_oj-all-components.scss";
```

And, then remove the comment on the import to enable compiling with the file that controls the CSS to include.

```
@import "_themeName.components.scss";
```

2. Edit the `_themeName.components.scss` file and remove the comment on the import statements for the JET components in your application.

```
...
//@import "oj/avatar/themes/redwood/oj-avatar.scss";
//@import "oj/badge/themes/redwood/oj-badge.scss";
@import "oj/button/themes/redwood/oj-button.scss";
```

```
@import "oj/buttonset-one/themes/redwood/oj-buttonset-one.scss";  
@import "oj/buttonset-many/themes/redwood/oj-buttonset-many.scss";  
//@import "oj/card/themes/redwood/oj-card.scss";  
//@import "oj/chart/themes/redwood/oj-chart.scss";  
...
```

In this example, only the import statement for buttons and button sets are uncommented to include the CSS for those components. All import statements in the `_themeName.components.scss` file remain commented out, by default.

3. Build a development version of your application.

```
ojet build
```

4. Run your application with live reload enabled.

```
ojet serve
```

5. Use Oracle JET live reload to immediately observe the effect of commenting and uncommenting import statements, after you save the changes in the `_themeName.components.scss` file.

In this example, after CSS compilation, the CSS that is created based on the theme for buttons and button sets will contain only the CSS related to those components.

6. To exit the application, press Ctrl+C at the terminal prompt.

Disable JET Styling of Base HTML Tags

By default, Oracle JET applies styles to HTML tag elements such as `a`, `h1`, `h2`, and so on. This feature makes it easier to code a page since you do not have to apply selectors to each occurrence of the element. You can disable JET styling of HTML tag elements and optionally directly style these tag elements.

If you do not want to apply styles to all base HTML tags by default, you can specify that Oracle JET use style classes instead of tag selectors. The Redwood CSS styles loaded at build time are controlled by import statements in custom theme settings files that you can optionally add to your project. To work with the CSS settings files in your project, you need to add theming support to the application and then create a custom theme.

With the custom theme settings files added to the `/src/themes` folder of your project, you can comment out an import statement that by default enables importing all JET component styles and add in its place the no-tag version of the import statement. With the appropriate import setting completed, when you build your application, JET Tooling will process the CSS settings files and load CSS for all JET components, but without the JET style classes for HTML base tags. To apply JET style to the HTML tags with the no-tag import enabled, you must explicitly set the JET style class on the desired HTML tag. For example, you can apply the JET style for links on the HTML anchor tag, like this ``.

The following table lists the HTML tags with default Oracle JET tag styles and the corresponding Oracle JET style class that you can optionally apply when you enable the no-tag import statement.

HTML Tag	Oracle JET Style Class
html	oj-html
body	oj-body
a	oj-link
h1, h2, h3, h4	oj-header
hr	oj-hr
p	oj-p
ul, ol	oj-ul, oj-ol

Before you begin:

- Install the Redwood theme tooling chain and configure a custom theme, as described in [Add Custom Theming Support](#). The `themes` folder with custom theme settings files are added to your application source.

To disable JET styling of base HTML tag:

1. In the `/src/themes/themeName/web` folder of your application, edit the aggregating `themeName.scss` file, and comment out the import statement that enables importing all JET component styles.

```
//@import "oj/all-components/themes/redwood/_oj-all-
components.scss";
```

And, then add the import statement to enable compiling with Oracle JET style classes that you can optionally apply to HTML tags.

```
@import "oj/all-components/themes/redwood/_oj-all-components-
notag.scss";
```

This will generate style classes that you can apply to HTML tags, for example ``.

2. Build a development version of your application.

```
ojet build
```

3. Run your application with live reload enabled.

```
ojet serve
```

4. Use Oracle JET live reload to immediately observe any styling changes.
5. To exit the application, press **Ctrl+C** at the terminal prompt.

Experimental: Work with CSS Variables

When you add a custom theme to your scaffolded application, Oracle JET adds CSS variable definition files that you can optionally modify to customize your application's look and feel. You can override CSS variables in the CSS you create for your

application or you can modify one or more CSS variables and optionally generate a custom CSS to process away the variables.

 **Note:**

Currently, theming with CSS variables remains experimental in JET release 9.0.0. In this release, CSS variable overrides must be limited to test applications and prototyping of the UI.

Topics

- [About CSS Variables and Custom Themes in Oracle JET](#)
- [Experimental: Client-side Theming Using CSS Variables](#)
- [Experimental: Build-time Theming with JET Tooling](#)

About CSS Variables and Custom Themes in Oracle JET

JET supports theming based on the out of the box Redwood theme with CSS variables instead of Sass variables starting with JET release 9.0.0.

Using the Redwood CSS out of the box, without any changes, is production in JET 9.0.0. Support for theming using CSS variables to customize application look and feel is preview in JET 9.0.0. In a future JET release, CSS variables may be removed or renamed and must not be used in a production application at this time. Oracle JET hopes to have CSS variables production-ready in JET release 10.0.0.

There are a few options for theming applications using CSS variables.

- You can override CSS variables defined by JET in your application CSS at runtime.

In this scenario, we call "client-side theming", your application runs with the `redwood-cssvars.css` variable definition CSS file and you can add variable overrides in your application CSS file. The online demo application CSS Variable Theme Builder shows the variable names and provides general information on variables on the Instruction tab. Note that there is a small performance penalty to using CSS variables in the browser. See the next bullet if you want to improve performance by processing away CSS variables at build time.

- You can override CSS variables defined by JET and generate resolved CSS with your overrides at build time.

In this scenario, we call "build-time theming", you can rely on JET Tooling to create a theme with the CSS variables processed away. In the JET distribution `css` directory, JET provides two versions of the CSS files: `oj-redwood-cssvars.css` and `oj-redwood.css`. The latter file has gone through a preprocessor to remove the CSS variables. When you use JET Tooling to build your application, you will similarly generate files with and without CSS variables. Since using CSS variables at runtime has a small performance penalty, you can use the resolved CSS that processes away the variables at build time. The JET Theme Builder application lets you use JET Tooling to create a theme and provides a variety of components to easily see the look and feel changes. The Theme Builder - Instruction tab includes steps to try it out. You can click the link to the zip for the latest production version.

Sass is not completely eliminated in JET; JET continues to use Sass when you work with the Redwood theme for bundling and there are a few cases, for example in media queries, where CSS variables aren't supported. For the few exceptions, Sass variables are needed (such as `$screenSmallMinWidth` and `$screenSmallMaxWidth`). Therefore Sass is still part of the tooling chain when you use `ojet add theming` in the JET CLI. If you use Sass to generate your own styles and don't rely on any JET Sass variables, then you can continue to use Sass as before. However, if you do rely on JET Sass variables, then most of those Sass variables will no longer be available, see the Theme Builder - Migration tab for how to migrate variables.

Alta themes will continue to use Sass, they will not switch to use CSS variables. If you have an existing theme that extends Alta and you want to migrate it to extend the Redwood theme (and use CSS variables), it is a manual process. There is information on variable migration on the Theme Builder - Migration tab. Migrating a production-ready custom theme to Redwood should be performed only after Redwood theming is production.

Use these links to view Theme Builder pages:

- [CSS Variable Theme Builder](#)
- [CSS Variable Theme Builder - Instruction Tab](#)
- [CSS Variable Theme Builder - Migration Tab](#)

Experimental: Client-side Theming Using CSS Variables

When you want to resolve CSS on the client, you can override CSS variables defined by the Redwood theme in your own CSS.

The process of overriding CSS variables to achieve a custom look and feel is currently experimental in JET releases 9.x.0. Consider creating a test application to preview client-side theming with CSS variables.

Enabling client-side theming with CSS variables is controlled at the application level through the CSS injector in your application's `index.html`. You can run the `ojet build --cssvars=enabled` command to load CSS from `redwood-cssvars.css` at build time.

Because client-side theming requires processing on the client to resolve CSS, there is a small performance penalty to using CSS variables. Leave the `oraclejetconfig.json` file property to the default setting `defaultCssvar=disable` unless you have an application use case that can benefit from client-side theming. As an alternative, consider modifying JET CSS variable definitions and processing away the variables, as described in [Experimental: Build-time Theming with JET Tooling](#).

Before you begin:

- Install the theme tooling chain and configure a custom theme, as described in [Add Custom Theming Support](#). The `themes` folder with custom theme settings files are added to your application source.
- Optionally, download and install the [CSS Variable Theme Builder - Instruction Tab](#) application when you want to learn about the available CSS variables in this interactive demo application. Follow the instructions online to modify the theme definition files to learn how CSS variable overrides change the demo tool UI.

To override Redwood CSS variables in your application CSS:

1. In the `/src/css` folder of your application, edit the application CSS that you maintain and define the CSS variable overrides.

Most customization is done with CSS variables, but in a few cases, such as for media queries, Sass variables are needed.

For example, to change the default large font size for headings to `2.5rem` from `2rem`, in the provided `app.css` file, define your variable and override the CSS variable `--oj-typography-heading-lg-font-size`, like this:

```
:root {  
  --my-app-typography-heading-lg-font-size: 2.5rem;  
  --oj-typography-heading-lg-font-size: var(--my-app-typography-heading-lg-font-size);  
}
```

2. Edit the `/src/index.html` file, and add links to the application CSS files that contain your overrides.

```
<!-- This is where you would add any app specific styling -->  
<link rel="stylesheet" href="css/app.css" type="text/css"/>
```

If you edited the provided `app.css` file in the `/src/css` folder, the link shown above is already present in the `index.html` file and does not require editing.

3. Build a development version of your application with the `--cssvars=enabled` option and add `--theme` to specify the CSS that contains your overrides.

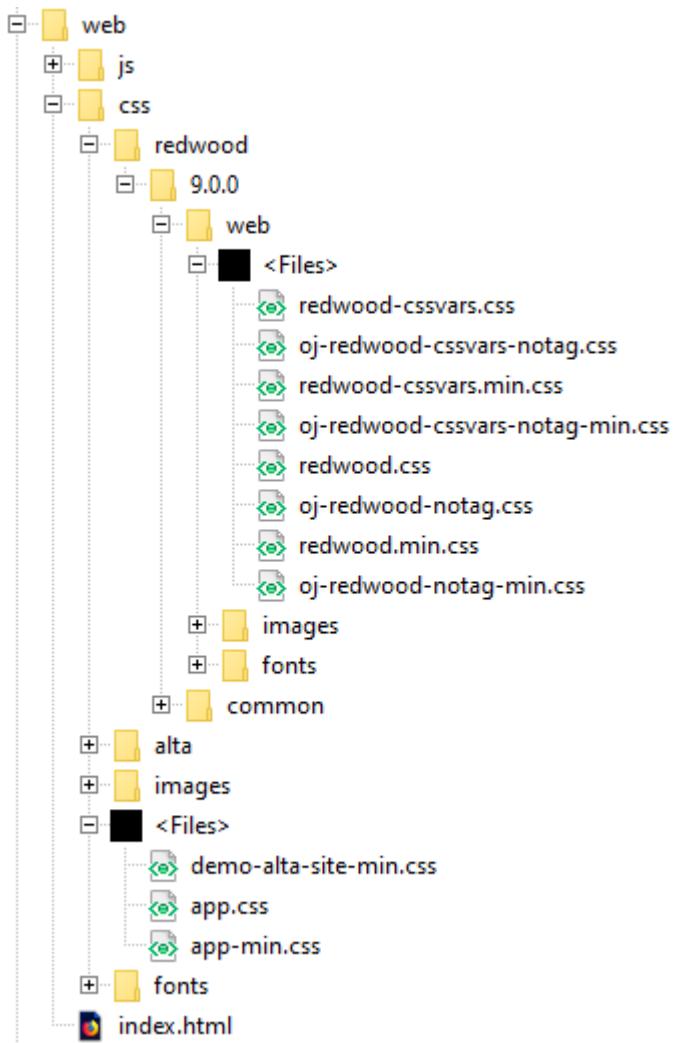
```
ojet build --cssvars=enabled --theme=themeName
```

Alternatively, you can edit the `oraclejetconfig.json` file and set the `defaultCssvars` property to `enabled`. When you edit the `.json` configuration file, each time you build or serve the application, the tooling will load the appropriate CSS variables file.

```
{  
  "paths": {  
    ...  
  },  
  "defaultBrowser": "chrome",  
  "sassVer": "x.x.x",  
  "defaultTheme": "myCustomThemeName",  
  "defaultCssvars": "enabled",  
  "generatorVersion": "9.2.0"  
}
```

Note that the property `defaultTheme` identifies the custom theme. This setting allows you to build the application without specifying the custom theme name.

The tooling outputs the built application source to the `web` build folder in the application's root and populates the `/web/css` folder with the CSS that contains your overrides, such as the provided `app.css` file. The tooling also populates the `/web/css/redwood/9.2.0/web` folder with the set of available Redwood theme CSS files, including the CSS variables defined in `redwood-cssvars.css`, as well as fonts and images.



Theme files with the `oj-` prefix and `-notag` suffix provide alternatives to the out of the box Redwood theme that allow you to work with a reduced CSS, as described in [Work with Custom Themes](#).

Note that `index.html` in the build output folder is now enabled to load CSS variables from `redwood-cssvars.css`. To load minified CSS variables from `redwood-cssvars.min.css`, build the application in release mode.

```
ojet build --cssvars=enabled --theme=themeName --release
```

4. To run your application with live reload enabled, enter `ojet serve` with the `--theme` option to specify the CSS that you maintain for your application.

```
ojet serve --theme=themeName
```

You can also serve the application and enable CSS variables, if not previously configured.

```
ojet serve --cssvars=enabled --theme=themeName
```

You do not need to specify the custom theme that you configured in the `oraclejetconfig.json` file.

5. Use Oracle JET live reload to immediately observe the effect of any CSS variable overrides.
6. To exit the application, press Ctrl+C at the terminal prompt.

Experimental: Build-time Theming with JET Tooling

When you use CSS variables on the client there is a small performance penalty. To improve performance, you can process away customized CSS variables at build time by using Oracle JET Tooling.

The process of overriding CSS variable to achieve a custom look and feel is currently experimental in JET releases 9.x.0. Consider creating a test application to preview build-time theming with CSS variables.

You use the command `ojet add theming` to add tooling support to your application to work with two versions of CSS: one that includes CSS variables and another that processes away the variables.

With CSS variables tooling support added, you can use the command `ojet create theme` to add CSS variable definitions files, including `_myTheme.cssvars.settings.css`, to your project. You can then directly modify the variable settings in this file to customize the theme.

This process of modifying CSS variables to theme your application ensures the underlying JET style classes remain unchanged. Therefore theming with variables allows you to customize parts of the Redwood theme without sacrificing migration of your theme changes.

With the variable settings defined, you can now build your application with the PostCSS tooling chain to process away the variables and load resolved CSS. To process away the CSS variables at build time, you will need to rerun the processing each time you pick up a new version of Oracle JET.

! Important:

Never override the underlying style classes of JET components since migration of such modifications to a future JET release cannot be guaranteed.

Before you begin:

- Install the PostCSS tooling chain and configure a custom theme, as described in [Add Custom Theming Support](#). The `themes` folder with custom theme settings files are added to your application source.
- Optionally, download and install the [CSS Variable Theme Builder - Instruction Tab](#) application when you want to preview customizations in this interactive demo application. Follow the instructions online to modify the theme definition files to learn how CSS variable overrides change the demo tool UI.

To override and processed away Redwood CSS variables:

1. In the `/src/themes/themeName/web` folder of your application, edit the `_themeName.cssvars.settings.scss` and `_themeName.sass.settings.scss` files to set CSS variables or Sass variables as needed for your application.

Most customization is done with CSS variables, but in a few cases, such as for media queries, Sass variables are needed.

For example, to change the default font size to 18px from 16px, in the `_themeNamecssvars.settings.scss` file, remove the comment from the `--oj-typography-root-font-size` variable and set it to `1.125em` like this:

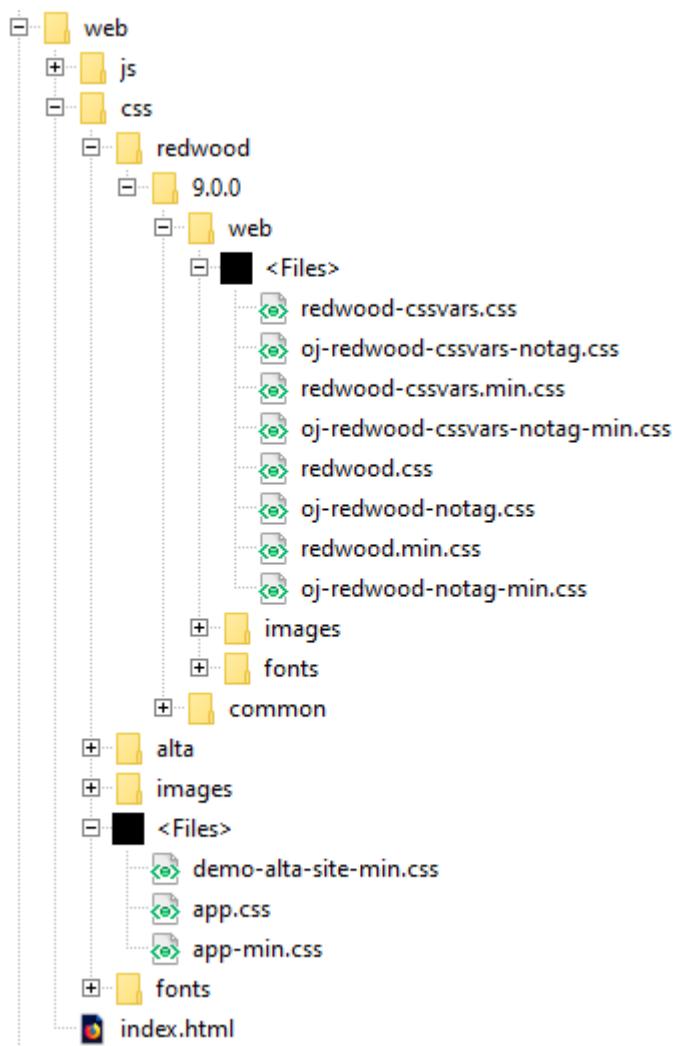
```
--oj-typography-root-font-size: 1.125em;
```

2. Build a development version of your application and process away the variables in the custom theme into resolved CSS.

```
ojet build
```

You do not need to specify the custom theme that you previously configured in the `oraclejetconfig.json` file.

The tooling outputs the built application source to the `web` build folder in the application root and populates the `/web/css` folder with the CSS that you maintain, such as in the provided `app.css` file. The tooling also populates the `/web/css/redwood/9.2.0/web` folder with the set of available Redwood theme CSS files, including the processed `redwood.css` that contains no CSS variables, as well as fonts and images.



Theme files with the `oj-` prefix and `-notag` suffix provide alternatives to the out-of-the-box Redwood theme that allow you to work with a reduced CSS, as described in [Work with Custom Themes](#).

Note that `index.html` in the build output folder is now enabled to load resolved CSS from `redwood.css`. To load minified CSS from `redwood.min.css`, build the application in release mode.

```
ojet build --release
```

3. To run your application with live reload enabled, enter `ojet serve` with the `--theme` option to specify any CSS that you maintain for your application.

```
ojet serve --theme=themeName
```

You do not need to specify the custom theme that you configured in the `oraclejetconfig.json` file.

4. Use Oracle JET live reload to immediately observe the effect of any CSS variable overrides.

5. To exit the application, press Ctrl+C at the terminal prompt.

16

Securing Applications

Oracle JET follows security best practices for Oracle JET components and provides the OAuth class to help you manage access to users' private data.

Topics:

- [Typical Workflow for Securing Oracle JET Applications](#)
- [About Securing Oracle JET Applications](#)
- [Use OAuth in Your Oracle JET Application](#)
- [About Securing Hybrid Mobile Applications](#)
- [About Cross-Origin Resource Sharing \(CORS\)](#)

Typical Workflow for Securing Oracle JET Applications

Understand how to secure your Oracle JET application. Optionally, understand how to use the OAuth plugin to manage access to client (end user) private data.

To develop secure Oracle JET applications, refer to the typical workflow described in the following table:

Task	Description	More Information
Understand Oracle JET application security	Identify Oracle JET security features and which tasks you should take to secure your Oracle JET application.	About Securing Oracle JET Applications
Use OAuth	Use Oracle JET OAuth plugin.	Use OAuth in Your Oracle JET Application

About Securing Oracle JET Applications

Oracle JET applications are client-side HTML applications written in JavaScript, and you should follow best practices for securing your Oracle JET applications.

There are a number of Internet resources available that can assist you, including the [Open Web Application Security Project \(OWASP\)](#), [Web Application Security Project \(WASP\)](#), [Web Application Security Working Group \(WASWG\)](#), and various commercial sites.

Topics:

- [Oracle JET Components and Security](#)
- [Oracle JET Security and Developer Responsibilities](#)
- [Oracle JET Security Features](#)
- [Oracle JET Secure Response Headers](#)

Oracle JET includes components that follow best practices for security and provides the OAuth plugin for providing secure access to a user's private data. However, the application developer is expected to perform tasks that are not included in Oracle JET.

Oracle JET Components and Security

Oracle JET components follow best practices for security. In particular:

- All JavaScript code is executed in strict mode using the `use strict` directive.
Strict mode changes warnings about poor syntax, such as using undeclared variables, into actual errors that you must correct. For more information, see http://www.w3schools.com/js/js_strict.asp.
- Oracle JET code does not use inline `script` elements.
Because browsers can't tell where the inline script originated, the World Wide Web Consortium (W3C) Content Security Policy prohibits the use of inline scripts. For additional information, see <https://w3c.github.io/webappsec/specs/content-security-policy>.
- Oracle JET code does not generate random numbers.
- Any HTML generated by an Oracle JET component is either escaped or sanitized.
For information about why this is needed, see https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet#Guidelines_for_Developing_Secure_Applications_Utilizing_JavaScript.

Oracle JET Security and Developer Responsibilities

Oracle JET components follow established security guidelines and ensure that strings provided as options and user input will never be executed as JavaScript to prevent XSS attacks. However, Oracle JET does not include a mechanism for sanitizing strings, and you should consult established guidelines for dealing with XSS attacks in your own code and content.

You can find more information about securing JavaScript applications at https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet#Guidelines_for_Developing_Secure_Applications_Utilizing_JavaScript.

Oracle JET Security Features

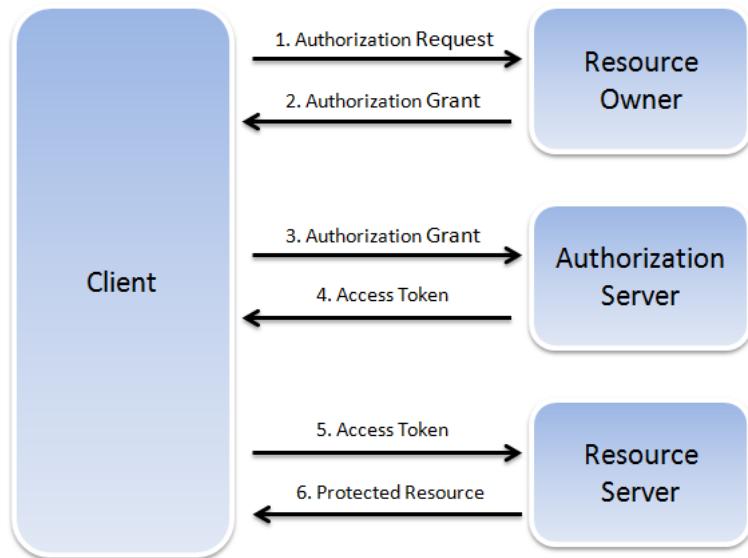
The Oracle JET API provides the OAuth authorization plugin which supports the OAuth 2.0 open protocol. OAuth standardizes the way desktop and web applications access a user's private data. It provides a mechanism for users to grant access to private data without sharing their private username and password credentials.

OAuth 2.0 defines the following roles:

- Resource owner: An entity that can grant access to a protected resource, such as the end user.
- Client: Application making protected and authorized resource requests on behalf of the resource owner.
- Resource server: Server hosting the protected resources that can accept and respond to protected resource requests using access tokens.

- Authorization server: Server that issues access tokens to the client after it successfully authenticates the resource owner and obtains authorization. The authorization server can be the same server as the resource server. In addition, an authorization server can issue access tokens accepted by multiple resource servers.

[OAuth 2.0 Request for Comments \(RFC\) 6749](#) describes the interaction between the four roles as an abstract flow.



1. The client requests authorization from the resource owner, either directly or through the authorization server. Note that the RFC specifies that the authorization server is preferred.
2. The client receives an authorization grant, which is defined as the credential representing the resource owner's authorization.
3. The client requests an access token from the authorization server by authenticating with the server and presenting the authorization grant.
4. The authorization server issues the access token after authenticating the client and validating the authorization grant.
5. The client presents the access token to the resource server and requests the protected resource.
6. The resource server validates the access token and serves the request if validated.

The access token is a unique identifier issued by the server and used by the client to associate authenticated requests with the resource owner whose authorization is requested or has been obtained by the client.

The Oracle JET OAuth plugin provides functions for the following tasks:

- Getting access token credentials if initialized by client credentials.
- Caching access token credentials.
- Creating the header array with bearer token.

For details about using the OAuth plugin, see [Use OAuth in Your Oracle JET Application](#). For additional information about OAuth 2.0, see <http://tools.ietf.org/html/rfc6749>.

Oracle JET Secure Response Headers

Oracle JET recommends the usage of HTTP response headers to securely host your JET web application. These response headers protect your web applications from cross scripting attacks (XSS) attacks, packet sniffing, and clickjacking.

You must configure these response headers on the server where the JET application is hosted. As the configuration of these response headers is dependent on the type of server, you must refer to the documentation of your server for configuration steps.

You must use the secure response headers to notify the user agent to only connect to a given site over HTTPS and load all resources over secure channels to control XSS attacks and packet sniffing. You can configure the response header in the server to specify the protocols that are allowed to be used; for example, a server can specify that all content must be loaded using HTTPS protocol.

It is highly recommended to host your JET application using the HTTPS protocol to reduce the cross scripting attacks or packet sniffing. Also some of the new browser features only work under HTTPS protocol. The below table lists some of the secure response headers along with the HTTPS column that indicates which of these headers are specific for HTTPS based configuration.

Table 16-1 Secure Response Header Options

Option	Value	HTTPS Related	Description
Content-Security-Policy	See Table 16-2, Content-Security-Policy Header Options .	No	Specifies fine-grained resource access.
X-XSS-Protection	1; mode=block	No	Blocks a page when cross site scripting attempt is detected. NOTE: <ul style="list-style-type: none"> • The X-XSS-Protection directive is a defense-in-depth mechanism to mitigate the effect of reflected XSS vulnerabilities and does not detect or block persistent or DOM based XSS attacks. Applications must still perform proper input validation on the server and output encoding as the primary defense against XSS. • Mozilla Firefox does not implement cross site scripting protection.
X-Permitted-Cross-Domain-Policies	none	No	Cross-domain policy file is an XML document that grants a web client permission to handle data across domains.

Table 16-1 (Cont.) Secure Response Header Options

Option	Value	HTTPS Related	Description
X-Frame-Options	deny	No	<p>Prevents clickjacking for browsers. This directive can only be set using an HTTP response header. To frame your content from the same origin, use <code>sameorigin</code>. If hosted by known host(s), specify <code>allow-frame</code> hostname.</p> <p>NOTE:</p> <ul style="list-style-type: none"> If a request contains both a CSP frame-ancestors and X-Frame-Options directive, browsers that support both will ignore the X-Frame-Options directive in favor of the standardized CSP frame-ancestors directive. The <code>allow-frame</code> directive does not support wildcards.
X-Content-Type-Options	nosniff	No	Ensures browser uses MIME type to determine the content type. Use of this directive with images requires the image format to match its specified MIME type. Use of this directive on JavaScript files requires the MIME type to be set to <code>text/javascript</code> .
Strict-Transport-Security	<code>max-age=<secs>; includeSubDomains</code>	Yes	Tells the browser to communicate only with the specified site (and any subdomains) over HTTPS and prevents the user from overriding an invalid or self-signed certificate.
Referrer-Policy	no-referrer	No	Tells the browser to include referrer information on outbound link requests.
Public-Key-Pins	<code>pin-sha256="<sha256>"; max-age=<secs></code>	Yes	Prevents use of incorrect or fraudulent certificates.
Expect-CT	<code>max-age=86400, enforce</code>	Yes	Signals to the browser that compliance to the Certificate Transparency Policy should be enforced.

Content Security Policy Headers

Content Security Policy (CSP) is delivered through an HTTP response header and controls the resources that an Oracle JET web application can use.

The CSP header provides a mechanism to restrict the locations from which JavaScript running in a browser can load the required resources, restrict the execution of JavaScript, and control situations in which a page can be framed. This can mitigate Cross Site Scripting (XSS) vulnerabilities as well as provide protection against clickjacking attacks.

You should enable CSP for browsers to help the server administrators reduce or eliminate the attacks by specifying the domains that the browser should consider to be valid sources for loading executable scripts, stylesheets, images, fonts, and so on. See the [Browser Compatibility Matrix](#) for the browser versions that support CSP.

 **Note:**

Internet Explorer 11 supports only the sandbox attribute and uses x-content-security-policy header instead of the standard content-security-policy header.

Configuring CSP involves adding the Content-Security-Policy HTTP header to a web page and giving it values to control resources the user agent is allowed to load for that page. To add Content-Security-Policy HTTP header to a web page, configure your web server to return the Content-Security-Policy HTTP response header. For example, here is a basic CSP response header, where script-src directive specifies an executable script as the resource type and 'self' is a constant that specifies the current domain as the approved source that the browser may load script from:

```
Content-Security-Policy: script-src 'self'
```

CSP has some of the following commonly used constants:

- 'none': Blocks the use of certain resource type.
- 'self': Matches the current origin (but not subdomains).
- 'unsafe-inline': Allows the use of inline JS and CSS.
- 'unsafe-eval': Allows the use of mechanisms like eval().

Note as an alternative to the unsafe-eval CSP domain, JET provides an expression evaluator that allows JET expression syntax to be evaluated in a way that is compliant with Content Security Policies that prohibit unsafe evaluations. The default CSP use of the unsafe-eval domain remains unchanged, but applications can opt into the JET behavior with some restrictions on the types of expressions that are supported. See details on creation, usage, supported expressions and limitations in the [CspExpressionEvaluator](#) API documentation.

Alternatively, you can also use the HTML meta tags to configure CSP. For example:

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';  
img-src https:///*; frame-src 'none';">
```

Note that some of the CSP directives do not work with the HTML meta tags, for example frame-ancestors.

The below table describes the out-of-box settings required by a JET web application to run in its most secure mode without changing the JET functionality. The JET web application may need to modify these settings for additional resource origins. The table lists the different response header directives that can be used while enabling CSP based on the two following scenarios:

- Co-Hosted: When the JET and the Application source codes are hosted on the same server
- Content Delivery Network (CDN): When the JET code is from the JET CDN and the Application code is from a different server

Table 16-2 Content-Security-Policy Header Options

CSP Version	Header Options	Co-Hosted	CDN	Description
CSP 1.0	default-src	'none'	'none'	Serves as a default setting that ensures resource loading is blocked if the resource type is not specified. All other settings need to be explicitly enabled for specific resource origins.
CSP 1.0	connect-src	'self'	'self'	Manages the REST and Web Sockets to be accessed.
CSP 1.0	font-src	'self'	'self' static.oracle.com	Specifies valid sources for fonts.
CSP 1.0	img-src	data: 'self'	'self' data: static.oracle.com	Specifies valid sources for images. Allows JET inline images.
CSP 1.0	media-src	'none'	'none'	Specifies valid sources for loading media using the <audio>, <video>, and <track> elements.
CSP 1.0	object-src	'none'	'none'	Specifies valid sources for the <object>, <embed>, and <applet> elements.

Table 16-2 (Cont.) Content-Security-Policy Header Options

CSP Version	Header Options	Co-Hosted	CDN	Description
CSP 1.0	script-src	'self' 'unsafe-eval'	'self' static.oracle.com 'unsafe-eval';	Specifies valid sources for JavaScript. This directive is used for knockout expressions and JET function creation. Note: The use of unsafe-eval is currently required by Knockout to resolve expressions, but JET provides an alternative expression evaluator when you require your application to run in a strict Content Security Policy environment. For usage information, see the CspExpressionEvaluator API documentation .
CSP 1.0	style-src	'self' 'unsafe-inline'	'self' static.oracle.com 'unsafe-inline'	Specifies valid sources for stylesheets. This directive is used to set styles from JavaScript.
CSP 1.0	sandbox	-	-	Runs the page as in a sandboxed iframe.
CSP 2.0	form-action	-	-	This directive is used for form submits. Not applicable for JET.

Table 16-2 (Cont.) Content-Security-Policy Header Options

CSP Version	Header Options	Co-Hosted	CDN	Description
CSP 2.0	frame-ancestors	'none'	'none'	Specifies valid sources for nested browsing contexts loading using elements such as <frame> and <iframe> and prevents clickjacking for browsers. This directive can only be set using an HTTP response header. To frame your content from the same origin, use 'self'. If hosted by known host(s), specify the hosts.

When default-src is set to none, you must explicitly enable all the other needed settings for specific resource origins.

The following example shows how to set up CSP if a website administrator wants to allow content from a trusted domain and all its subdomains:

```
Content-Security-Policy: default-src 'self' *.trusted.com
```

The following example shows how to set up CSP if a website administrator wants to allow users of a web application to include images from any origin in their own content, but to restrict audio or video media to trusted providers, and all scripts only to a specific server that hosts trusted code.

```
Content-Security-Policy: default-src 'self'; img-src *; media-src
media1.com media2.com; script-src userscripts.example.com
```

Use OAuth in Your Oracle JET Application

You can use the OAuth plugin to manage access to client (end user) private data. The Oracle JET API includes the OAuth class which provides the methods you can use to initialize the OAuth object, verify initialization, and calculate the authorization header based on client credentials or access token.

Topics:

- [Initialize OAuth](#)
- [Verify OAuth Initialization](#)
- [Obtain the OAuth Header](#)

- Use OAuth with Oracle JET Common Model
- Integrate OAuth with Oracle Identity Management (iDM) Server

Initialize OAuth

You can create an instance of a specific OAuth object using the OAuth constructor:

```
new OAuth(header, attributes)
```

The attributes and header parameters are optional.

Parameter	Type	Description
header	String	MIME Header name. Defaults to Authorization
attribute	Object	Contains client credentials or access/bearer token. Client credentials contain: <ul style="list-style-type: none">• client_id (required): public client Credentials• client_secret (required): secret client credentials• bearer_url (required): URL for token bearer and refresh credentials• Additional attributes as needed (optional) Access/bearer tokens contain: <ul style="list-style-type: none">• access_token (required): Bearer token• Additional attributes as needed (optional)

The code sample below shows three examples for initializing OAuth.

```
// Initialize OAuth with client credentials
var myOAuth = new OAuth('X-Header', {...Client credentials...});

// Initialize OAuth with token credentials
var myOAuth = new OAuth('X-Header', {...Access/Bearer token...});

// Initialize OAuth manually
var myOAuth = new OAuth();
```

If you choose to initialize OAuth manually, you can add the client credentials or access/bearer token using methods shown in the following code sample.

```
// Initializing client credentials manually
myOAuth.setAccessTokenRequest({...Client Credentials ...});
myOAuth.clientCredentialGrant();

// Initializing access bearer token manually
myOAuth.setAccessTokenResponse({...Access Token...});
```

The OAuth API also includes methods for getting and cleaning the client credentials or access tokens. For additional information, see the [OAuth API documentation](#).

Verify OAuth Initialization

Use the `isInitialized()` method to verify that the initialization succeeded.

```
var initFlag = myOAuth.isInitialized();
```

Obtain the OAuth Header

Use the `getHeader()` method to get the OAuth header. The method calculates the authorization header based on the client credentials or access token.

```
// Client credentials
var myOAuth = new OAuth('New-Header', {...Client credentials...});
var myHeaders = myOAuth.getHeader();

// Access token
var myOAuth = new OAuth('New-Header', {...Access/Bearer token...});
var myHeaders = myOAuth.getHeader();

// Manual initialization, client credentials
var myOAuth = new OAuth();
myOAuth.setAccessTokenRequest({...Client credentials...});
var myHeaders = myOAuth.getHeader();

// Manual initialization, access token
var myOAuth = new OAuth('New-Header', {...Access/Bearer token...});
var myHeaders = myOAuth.getHeader();
```

Use OAuth with Oracle JET Common Model

You can add the `OAuth` object to your `ViewModel`, either embedded or as an external plugin.

Topics:

- [Embed OAuth in Your Application's ViewModel](#)
- [Add OAuth as a Plugin in Your ViewModel](#)

For information about Oracle JET's Common Model, see [Using the Common Model and Collection API](#).

Embed OAuth in Your Application's ViewModel

The code sample below shows how you could embed the `OAuth` object in your `ViewModel`. This example initializes `OAuth` with client credentials.

```
function viewModel() {
  var self = this;
  ...
  self.myOAuth = new OAuth('X-Authorization', {...Client credentials...});

  var tweetModel = Model.extend({
    ...
  });
  var myTweet = new tweetModel();
  ...
  var tweetCollection = Collection.extend({
    model: myTweet,
    oauth: self.myOAuth, // using embedded feature
    ...
  });
  self.myTweetCol = new tweetCollection();
  ...
}
```

```

self.myTweetCol.fetch({
    success: function(collection, response, options) {
        ...
    },
    error: function(jqXHR, textStatus, errorThrown) {
        ... // process errors
    }
});
}

```

To embed the `OAuth` object in your `ViewModel` and initialize it with a bearer/access token:

```

function viewModel() {
    var self = this;
    ...
    self.myOAuth = new OAuth('X-Authorization', {...Access/Bearer token...});

    var tweetModel = Model.extend({
        ...
    });
    var myTweet = new tweetModel();
    ...
    var tweetCollection = Collection.extend({
        model: myTweet,
        oauth: self.myOAuth, // using embedded feature
        ...
    });
    self.myTweetCol = new tweetCollection();
    ...
    self.myTweetCol.fetch({
        success: function(collection, response, options) {
            ...
        },
        error: function(jqXHR, textStatus, errorThrown) {
            ... // process errors or insert new access_token and re-fetch
        }
    });
}

```

Add OAuth as a Plugin in Your ViewModel

The code sample below shows how you could add the `OAuth` object as a plugin in your `ViewModel`. This example initializes `OAuth` with client credentials.

```

var viewModel() {
    var self = this;
    ...
    self.myOAuth = new OAuth('X-Authorization', {...Client credentials...});

    var tweetModel = Model.extend({
        ...
    });
    var myTweet = new tweetModel();
    ...
    var tweetCollection = Collection.extend({
        model: myTweet,
        ...
    });
    self.myTweetCol = new tweetCollection();
    ...
}

```

```

self.prefetch = function() {
    var header = self.myOAuth.getHeader();
    $.ajaxSetup({
        beforeSend: function (xhr){
            for(var hdr in header ) {
                if(header.hasOwnProperty(hdr))
                    xhr.setRequestHeader(hdr, header[hdr]);
            }
        }
    });
    self.myTweetCol.fetch({
        success: function(collection, response, options) {
            ...
        },
        error: function(jqXHR, textStatus, errorThrown) {
            ... // process errors
        }
    });
}
}

```

Integrate OAuth with Oracle Identity Management (iDM) Server

Oracle iDM servers use a two-legged authorization (Resource Owner Password Credentials Grant). In addition, the iDM servers require that you do the following:

- Keep client credentials on your own proxy server. If you don't have one, you must create one.
- iDM servers use a non standard authorization header and require that `Authorization:access_token` be used instead of `Authorization: Bearer access_token`. To supply the custom header, you must rewrite the OAuth header for specific Authorization using the `getHeader()` method.

The code excerpt below shows an example that adds the `OAuth` object with a modified header to the application's `viewModel`.

```

function viewModel() {
    var self = this;
    self.bearer = {
        access_token: ...,
        token_type: "Bearer",
        expires_in: ...,
        ...
    }
    ...
    self.myOAuth = new OAuth();
    // Rewrite OAuth header for specific Authorization
    self.myOAuth.getHeader = function() {
        var headers = {};
        headers['X-Authorization']=self.bearer.access_token;
        return headers;
    }
    var idmModel = Model.extend({... });
    var myIDM = new idmModel();
    ...
    var idmCollection = Collection.extend ({
        model: myIDM, oauth: self.myOAuth,
        // using embedded feature
        ...
    })
}

```

```
});
self.myIDMCol = new idmCollection();
...
self.myIDMCol.fetch({
    success: function(collection, response, options) {
        ...
    },
    error: function(jqXHR, textStatus, errorThrown) {
        ...
        // process errors or insert new access_token and re-fetch
    }
});
}
```

About Securing Hybrid Mobile Applications

Since hybrid mobile applications are JavaScript HTML5 applications, many of the same security practices that apply to web applications apply to hybrid mobile applications. However, there are additional considerations when you're deploying to mobile devices.

The Cordova documentation includes a [Security Guide](#) that provides some security best practices for Cordova applications. You can use this guide as a starting point to secure your hybrid mobile application. However, as Cordova points out, security is a complicated topic, and its guide is not exhaustive.

JET adopts the AppConfig approach to manage mobile applications in the enterprise. AppConfig is an industry-sponsored community that provides tools and best practices to configure and secure mobile applications. Oracle JET provides the `cordova-plugin-emm-app-config` plugin, described in [Manage App Configuration for JET Hybrid Mobile Apps](#), to facilitate this process.

Manage Authentication in JET Hybrid Mobile Apps

Authenticating end users of a hybrid mobile app and propagating the identity of the authenticated end users to back-end systems so that the end user can access secured resources is a requirement that you will have to frequently implement.

Oracle JET provides a Cordova plugin, `cordova-plugin-oracle-idm-auth`, to assist you with these tasks. Add this plugin to hybrid mobile apps that you deploy to Android and iOS devices using the Oracle JET CLI as follows:

```
ojet add plugin cordova-plugin-oracle-idm-auth
```

For more information about using this plugin in your hybrid mobile app, see <https://github.com/oracle/cordova-plugin-oracle-idm-auth>. This plugin can be used in any Cordova-based hybrid mobile app, not just JET hybrid mobile apps.

Manage App Configuration for JET Hybrid Mobile Apps

Many enterprises make use of Enterprise Mobility Management (EMM) software to manage mobile devices, applications, wireless networks, and other mobile computing services.

EMM software allows administrators to configure mobile apps via a web console and apply the configuration to the app on every managed device as part of the app installation process. This configuration typically consists of generic name-value pairs

defined by the app developer, such as URI, port numbers, tenant IDs, or CSS skin configurations.

The Android and iOS platforms support this functionality natively, but retrieving app configuration data and being alerted when it has changed is just as important for Cordova-based hybrid mobile apps. To make this possible, Oracle JET provides the `cordova-plugin-emm-app-config` Cordova plugin that can be used to implement app configuration in JET mobile apps on the Android and iOS platforms. Using this plugin, you can retrieve the app configuration data at any point in the app lifecycle, including when the EMM server changes the app configuration.

You can add the `cordova-plugin-emm-app-config` to your hybrid mobile app using the Oracle JET CLI as follows:

```
ojet add plugin cordova-plugin-emm-app-config
```

For more information about this plugin, see <https://github.com/oracle/cordova-plugin-emm-app-config>. This plugin can be used in any Cordova-based hybrid mobile app, not just JET hybrid apps.

Vendors and other stakeholders in the application configuration software sector formed the AppConfig community to provide tools and best practices. For more information, see <http://appconfig.org/>.

About Cross-Origin Resource Sharing (CORS)

CORS is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. The same-origin security policy of JavaScript forbids certain cross-domain requests, notably Ajax requests, by default.

Rejected resource requests due to CORS can affect web or hybrid mobile applications. Applications that encounter a rejection receive messages such as the following example in response to resource requests:

```
No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

Server-side administrators can specify the origins allowed to access their resources by modifying the policy used by their remote server to allow cross-site requests from trusted clients. For example, to access a remote service managed by Oracle's Mobile Cloud Service (MCS), an MCS administrator configures MCS's `Security_AllowOrigin` environment policy with a comma-separated list of URL patterns that identify the remote services that serve resources from different domains.

If you serve your web or hybrid mobile application to the local browser for testing, you may encounter CORS rejections. Some browsers provide options to disable CORS, such as Chrome's `--disable-web-security` and Firefox's `security.fileuri.strict_origin_policy` and some browsers support plugins that work around CORS.

Only use these options when testing your application and ensure that you complete further testing in a production-like environment without these options to be sure that your application will not encounter CORS issues in production.

The default web views used by hybrid mobile applications do not implement CORS and therefore hybrid mobile applications will not encounter CORS issues when run on a device in the default web view. However, if you use an alternative web view, such as

WKWebView on iOS, you may encounter CORS issues. To work around this on iOS, consider using the `cordova-plugin-wkwebview-file-xhr` plugin in your hybrid mobile application as an alternative to WKWebView. For additional information, see [Use a Different Web View in Your JET Hybrid Mobile App](#).

Configuring Data Cache and Offline Support

Use the Oracle Offline Persistence Toolkit to enable data caching and offline support within your Oracle JET application.

Topics

- [About the Oracle Offline Persistence Toolkit](#)
- [Install the Offline Persistence Toolkit](#)

About the Oracle Offline Persistence Toolkit

The toolkit is a client-side JavaScript library that Oracle maintains as an open-source project. The toolkit provides caching and offline support at the HTTP request layer.

This support is transparent to the user and is done through the Fetch API and an XHR adapter. HTTP requests made while the client or client device is offline are captured for replay when connection to the server is restored. Additional capabilities include a persistent storage layer, synchronization manager, binary data support and various configuration APIs for customizing the default behavior. This toolkit can be used in both ServiceWorker and non-ServiceWorker contexts within web and hybrid mobile apps.

Using the toolkit, you can configure your application to:

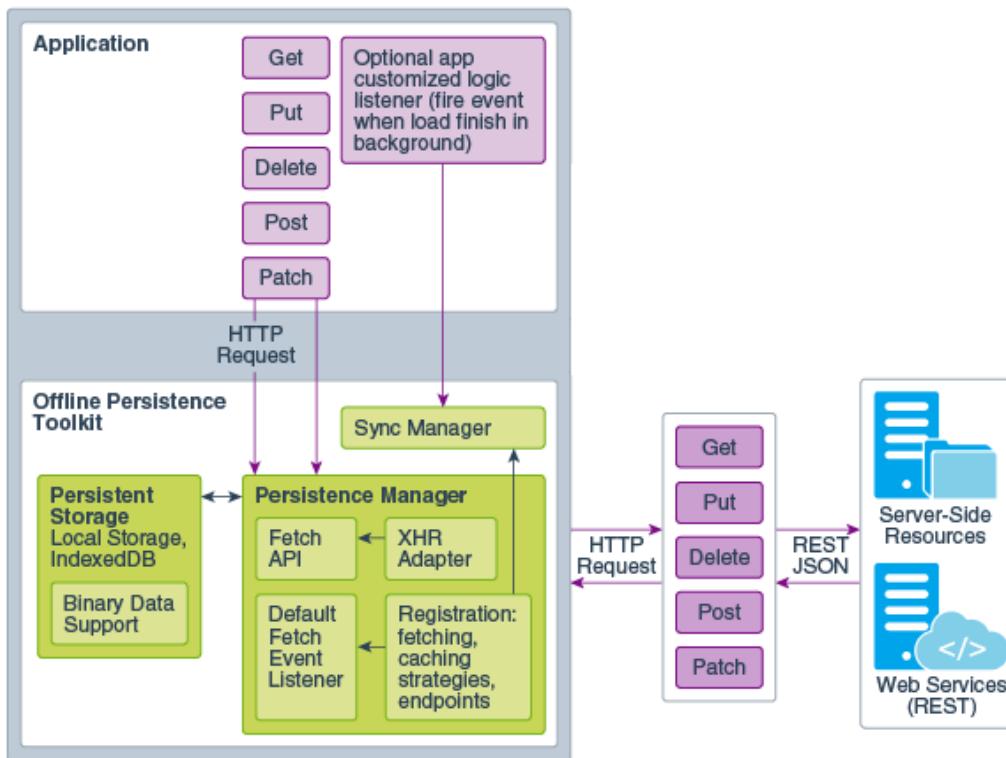
- Download content for offline reading where connectivity isn't available.

For example, an application could include product inventory data that a salesperson could download and read at customer sites where connectivity isn't available.
- Cache content for improved performance.
- Perform transactions on the downloaded content where connectivity isn't available and upload the transactions when connectivity returns.

For example, the salesperson could visit a site with no Internet access and enter an order for some number of product items. When connectivity returns, the application can automatically send the transaction to the server.
- Provide conflict resolution when the offline data can't merge with the server.

If the salesperson's request exceeds the amount of available inventory, the application can configure a message asking the salesperson to cancel the order or place the item on back order.

The architecture diagram illustrates the major components of the toolkit and how an application interacts with it.



Install the Offline Persistence Toolkit

Use npm to install the offline persistence toolkit. After installation, you must update your application's `src/js/path_mapping.json` file to recognize the new package.

1. Change to your application's top level directory and open a terminal window.
2. At the terminal prompt, enter the following command to install the toolkit: `npm install @oracle/offline-persistence-toolkit --save`.
3. Change to your application's `src/js` directory and open `path_mapping.json` for editing.
4. Add the persistence toolkit to the "libs" entry.

The easiest way to add the toolkit is to copy an existing entry that's similar to your library and modify as needed. A sample entry for `offline-persistence-toolkit`, which started with a copy of the "ojs" entry, is shown below.

```
"libs": {

    "offline-persistence-toolkit": {
        "cdn": "",
        "cwd": "node_modules/@oracle/offline-persistence-toolkit/dist",
        "debug": {
            "cwd": "debug",
            "src": [ "**" ],
            "path": "libs/offline-persistence-toolkit/debug",
            "cdn": ""
        }
    }
}
```

```
},
"release": {
  "cwd": "min",
  "src": ["**"],
  "path": "libs/offline-persistence-toolkit/min",
  "cdn": ""
}
},
```

For information about using the toolkit once you have it installed in your Oracle JET application, see the `README.md` and Wiki for the persistence toolkit on Github at <https://github.com/oracle/offline-persistence-toolkit>.

Optimizing Performance

Oracle JET applications are client-side HTML5 applications. Most performance optimization recommendations relating to client-side HTML applications also apply to applications developed using Oracle JET or to Oracle JET components. In addition, some Oracle JET components have performance recommendations that are specific to the component.

Topics:

- [Typical Workflow for Optimizing Performance of Oracle JET Applications](#)
- [About Performance and Oracle JET Applications](#)
- [Add Performance Optimization to an Oracle JET Application](#)
- [About Configuring the Application for Oracle CDN Optimization](#)
- [Understand the Path Mapping Script File and Configuration Options](#)

Typical Workflow for Optimizing Performance of Oracle JET Applications

Understand performance optimization recommendations related to Oracle JET applications and components.

To optimize performance of an Oracle JET application, refer to the typical workflow described in the following table:

Task	Description	More Information
Identify general performance optimization goals	Identify performance optimization goals for client-side HTML5 applications.	About Performance and Oracle JET Applications
Apply performance optimization to your Oracle JET application	Take specific steps to optimize performance of your application.	Add Performance Optimization to an Oracle JET Application

About Performance and Oracle JET Applications

In general, you can optimize an Oracle JET application the same way that you would optimize performance for any client-side HTML5 application.

There are many online resources that provide tips for performance optimization. For example, the [Google Developers](#) website describes their tools for improving the performance of the application.

Most of the recommendations made by the Google tools are up to you to implement, but Oracle JET includes features that can reduce the payload size and the number

of trips to retrieve the Oracle JET application's CSS. In general, strive to follow these guidelines.

1. Always minify and bundle application resources to reduce the number of requests from the server.
2. Configure the application to load resources from Oracle CDN to minimize network usage.
3. Configure the application to use the Oracle JET library on CDN so that the `bundles-config.js` script will load minified bundles and modules by default.
4. Compress the application with gzip to reduce the size (and enable compression on the web server.)
5. Enable HTTP caching on web server so that some requests can be served from the cache instead of from the server. Use ETags on files that should always be served from the server.
6. Take advantage of HTTP/2 to serve page resources faster than is possible with HTTP/1.1.
7. Use a single page application, so that the browser isn't forced to tear down and rebuild the whole application.
8. Avoid putting too many data-centric components into a single page.
9. Optimize graphic images: prefer vector format; choose the appropriate image format based on the best overall compression available.

For more information about these optimization tips and others, see [Add Performance Optimization to an Oracle JET Application](#).

Add Performance Optimization to an Oracle JET Application

Most tips for optimizing performance of web and hybrid mobile applications also apply to Oracle JET applications. However, there are some steps you can take that apply specifically to Oracle JET applications to optimize JavaScript, CSS, Oracle JET components, REST calls, and images.

JavaScript Performance Tips

Performance Tip	Details
Maintain the expected JavaScript folder structure	Use folder organization generated by the Oracle JET tooling and maintain all JavaScript files inside the <code>js</code> folder of the application root.

Performance Tip	Details
Send only the JavaScript code that your application needs.	<p>Oracle JET includes modules that you can load with RequireJS. For additional information, see Using RequireJS for Modular Development. One approach is to preload the JavaScript modules that your application will use. The following sample shows how to modify the require function in the application main.js.</p> <pre data-bbox="612 422 1428 1381"> require(['ojs/objbootstrap', 'knockout', 'ojs/ojknockout'], function (Bootstrap) { Bootstrap.whenDocumentReady().then(function () { function init() { } // If running in a hybrid (e.g. Cordova) environment, we need to wait for the deviceready // event before executing any code that might interact with Cordova APIs or plugins. if (document.body.classList.contains('oj-hybrid')) { document.addEventListener('deviceready', init); } else { init(); } //after main doc is done preload some js require(['ojs/ojbutton', 'ojs/ojdvt-base', 'ojs/ ojtree', 'ojs/ojaccordion', 'ojs/ojtreemap'], //example of what can be preloaded function() { }); }); }); </pre>

Performance Tip	Details
Send minified/obfuscated JavaScript.	<p>Oracle JET provides minified versions of the Oracle JET library as well as third-party libraries when available. By default, the path mappings for the minified versions of these libraries in <code>path_mapping.json</code> will be injected into the Oracle JET RequireJS bootstrap file included with all Oracle JET distributions when you build a release version of the application. The following sample shows a single library from <code>path_mappings.json</code> where the minified library is available for release mode.</p>
	<pre data-bbox="535 487 1181 895"> "jquery": { "cdn": "3rdparty", "cwd": "node_modules/jquery/dist", "debug": { "src": "jquery.js", "path": "libs/jquery/jquery-#{version}.js", "cdnPath": "jquery/jquery-3.x.x" }, "release": { "src": "jquery.min.js", "path": "libs/jquery/jquery-#{version}.min.js", "cdnPath": "jquery/jquery-3.x.x.min" } }, </pre>
	<p>For additional information about using the RequireJS bootstrap file in your Oracle JET application, see About RequireJS in an Oracle JET Application.</p>
Minimize the number of trips to retrieve the JavaScript.	<p>Oracle JET doesn't provide support for minimizing the number of trips, but RequireJS has an optimization tool that you can use to combine modules. For additional detail, see the documentation for the RequireJS optimizer. Alternatively, use a JavaScript code minifier, such as Terser to bundle and minify the JavaScript source.</p>
Use lazy loading for JavaScript not needed on first render.	<p>You can lazy load content that is not needed on first render. For example, you can configure the <code>oj-film-strip</code> component to retrieve child node data only when requested. For an example, see the Lazy Loading (oj-film-strip) Oracle JET Cookbook example.</p>
Compress or zip the payload.	<p>Oracle JET has no control over the server, and this recommendation is up to you to implement. For some additional information and tips, see https://developers.google.com/speed/docs/best-practices/payload#GzipCompression.</p>
Set cache headers.	<p>JET has no control over the server, and this recommendation is up to you to implement. For additional information about cache optimization, see https://developers.google.com/speed/docs/best-practices/caching.</p>

CSS Performance Tips

Performance Tip	Details
Maintain the expected CSS folder structure	<p>Use folder organization generated by the Oracle JET tooling and maintain all CSS files inside the <code>css</code> folder of the application root.</p>
Render pages for all the needed CSS once	<p>Link to style sheets inside the HEAD section of the HTML page and do not use CSS links in the page body to avoid rerendering an already loaded page.</p>

Performance Tip	Details
Send only the CSS that your application needs.	<p>You can control the CSS content that goes into your application. For additional information, see Use Variables to Control CSS Content.</p> <p>If you're using the Oracle JET grid system, you can also control which responsive classes get included in the CSS. For details, see Control the Size and Generation of the CSS.</p>
Send minified/obfuscated CSS.	<p>By default, Oracle JET includes minified CSS. However, if you want to modify the CSS to send only what your application needs, you can use Sass to minimize your output. For additional information, see the :compressed option at: http://sass-lang.com/documentation/file.SASS_REFERENCE.html#output_style.</p>

Oracle JET Application and Component Performance Tips

Performance Tip	Details
Configure your application to load bundled JET modules and libraries using Oracle CDN and the bundle configuration support it provides for JET.	Leveraging the Oracle Content Delivery Network (CDN) and bundle configuration support optimizes the application startup performance of enterprise applications and also ensures that your application builds with the module and library versions required for a particular Oracle JET release. Referring directly to the bundles within the application is not recommended and that includes adding or modifying links that make direct reference to the configuration of the bundles. For additional information, see About Configuring the Application for Oracle CDN Optimization .
Consider using plain HTML components where possible.	For stamping components (like Table or ListView) or declarative components (like a composite component), if you embed a simple Oracle JET component like a read-only input component or button, consider using a plain HTML component instead. The plain HTML component can be lighter weight (less DOM), and if the component is stamped that can add up. However, note that with plain HTML, you also won't have access to any built-in accessibility support, such as converters and validators, which the Oracle JET component provides.
Follow Oracle JET component best practices.	Consult the API documentation for the Oracle JET component. The JavaScript API Reference for Oracle® JavaScript Extension Toolkit (Oracle JET) includes a performance section for a component when applicable. For example, see the ojDataGrid—Performance section.
Limit number of Oracle JET components per page.	The number of components on the page will impact the page load time. If you want to reduce the load time, place fewer data-centric components in the page.
Use the oj-defer element to delay binding execution	When a component contains hidden content to display, the oj-defer element delays the process of applying bindings to the child components until the hidden content becomes visible. For additional information on oj-defer, see the oj-defer API documentation.
Limit the fetch size for collection components	Set the collection component scroll-policy-options.fetch-size attribute equal to the number of items to display in the component viewport. Also set scroll-policy="loadMoreOnScroll" to ensure that the fetch for additional items occurs only when the user scrolls toward the end of the fetched list.

REST Request Performance Tips

Performance Tip	Details
Reduce the number of roundtrips between client and server.	<p>There are a number of techniques that you can use to reduce the number of roundtrips, and here are some examples:</p> <ul style="list-style-type: none"> The number of REST requests on the page will impact the page load time. If you need to reduce the load time, simplify your page and make fewer REST requests. A REST call that needs data from a previous REST call creates a dependency that triggers serialization and increases the data fetch response time. To reduce response time, minimize the number of dependent REST calls by redefining your REST calls to fetch only the data your UI requires. Ideally, the REST endpoint supports fetching of the exact data or can be redesigned as needed. All REST operations should be executed asynchronously. To manage async state, invoke REST endpoints with a method to return a Promise from the start. If the REST endpoint is slow to respond, and the data is essential to the application, consider prefetching the data in a non-blocking REST endpoint invocation. The REST endpoint should be designed to support pagination.

Image Optimization

Performance Tip	Details
Maintain the expected images folder structure	Use folder organization generated by the Oracle JET tooling and maintain all image files inside the <code>images</code> folder of the application root.
Reduce image size.	<p>Reducing the size of the images will result in faster downloads and reduce the time it takes to render the content on the screen. For example, Scalable Vector Graphics (SVG) images are usually smaller than Portable Network Graphics (PNG) images and scale on high resolution devices.</p> <p>There are also a number of third-party tools that you can use to reduce the size of your images. The tool that you select will depend on the image type, for example:</p> <ul style="list-style-type: none"> <code>imagemin</code>: Utility to compress PNG images <code>svgomg</code>: Utility to compress SVG images. You can use this tool online or download <code>svgo</code> to work with the images on your own system.

Performance Tip	Details
Reduce the number of roundtrips between client and server.	<p>There are a number of techniques that you can use to reduce the number of roundtrips, and here are some examples:</p> <ul style="list-style-type: none"> • Icon fonts Icon fonts are useful when your icon uses a single color. Oracle JET uses icon fonts, and you can see examples of them at: Icon Fonts. You can find utilities on the Internet such as IcoMoon that you can use to generate icon fonts. • Image Sprites An image sprite is a collection of images combined into a single image, reducing the number of server requests. You can find examples of them at http://www.w3schools.com/css/css_image_sprites.asp. • Lazy loading You can use lazy loading to defer the loading of images not in the user's viewport. You can find many examples and utilities on the Internet that use this technique. • Base64 Encoding You can use Base64 Encoding to inline image data. They are commonly used in data Uniform Resource Indicators (URIs), and you can find additional information about them at https://developer.mozilla.org/docs/Web/HTTP/data_URIs.

For additional performance tips, see the [Google Developers](#) documentation for improving the quality of web pages, including how to audit for performance.

About Configuring the Application for Oracle CDN Optimization

You can configure the Oracle JET application to minimize the network load at application startup through the use of Oracle Content Delivery Network (CDN) and the Oracle JET distributions that the CDN supports.

The local loading of the required Oracle JET libraries and modules, as configured by default when you create the application, is not recommended for a production application since it does not use Oracle CDN and requires each application running in the browser to load libraries and modules in a standalone manner. This default configuration can be used when you build and serve the application locally until you need to stage the application in a test environment to simulate network access.

To enable Oracle CDN optimization, you configure the `path_mapping.json` file in your application. The choices you make in the path mapping file determine the libraries and modules settings for the entire application. With this file, you will not need to edit the path URL of the required libraries and modules in any other application file. When you configure path mapping for CDN optimization, you will determine how the tooling updates the require block of the `main.js` file and how the application will load modules and libraries as follows:

- If you configure path mappings to use CDN without accessing the bundles configuration, so that modules and libraries will be loaded individually from CDN, the tooling injects the URLs from the path mapping file into the `requirejs` block of `main.js`.

- If you configure the path mappings to use CDN bundle loading, the tooling updates the index.html file to execute the bundles configuration script file (`bundles-config.js`) from the following script reference:

```
<body>
  <script type="text/
javascript" src="https://static.oracle.com/cdn/jet/9.2.0/default/js/
bundles-config.js"></script>
..
</body>
```

Note: Starting in JET release 9.0.0, the convention of using the leading character "v" to identify the release number has changed. As the above sample shows, the release identifier is now a semver value specified as 9.2.0 with no leading character.

The bundles configuration file specifies its own require block that the application executes to load as a set of bundled modules and libraries from CDN. When you configure the application this way, the `main.js` is updated by the tooling to display *only* a URL list composed of third-party libraries. In the bundles configuration scenario, the injected require block in `main.js` becomes a placeholder for any application-specific libraries that you want to add to the list. Where URL duplications may occur between the require block of the bundles configuration file and the application `main.js`, the bundles configuration takes precedence to ensure bundle loading from CDN has the priority.

 **Tip:**

Configuring your application to reference the bundles configuration script file on Oracle CDN is recommended because Oracle maintains the configuration for each release. By pointing your application to the current bundles configuration, you will ensure that your application runs with the latest supported library and module versions.

Configure Bundled Loading of Libraries and Modules

For Oracle CDN optimization, you may use the bundles configuration script that loads a set of bundled libraries and modules from the CDN.

To configure bundle loading of the libraries and modules using the bundles configuration script, perform the following steps.

1. Open the `path_mapping.json` file in the `js` subfolder of your application and change the `use` element to `cdn`:

```
"use": "cdn"
```

2. Leave the `cdns` element unchanged. It should show the following as the default path definitions for Oracle JET and third-party libraries:

```
"cdns": {
  "jet": {
```

```

        "prefix": "https://static.oracle.com/cdn/jet/9.2.0/default/
js",
        "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",
        "config": "bundles-config.js"
    },
    "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty"
},

```

Note: Starting in JET release 9.0.0, the convention of using the leading character "v" to identify the release number has changed. As the above sample shows, the release identifier is now a semver value specified as 9.2.0 with no leading character.

3. Optionally, in the case of bundle loading, for each third-party library, update the cdn element from 3rdparty to jet. For example, this knockout library path definition shows "cdn": "jet" to prevent the knockout URL from being injected into main.js:

```

"libs": {

    "knockout": {
        "cdn": "jet",
        "cwd": "node_modules/knockout/build/output",
        ...
    },
}

```

Setting "cdn": "jet" for each third-party library prevents these libraries' URL from being injected into the require block of main.js. This update is not necessary to ensure bundle loading of third-party libraries since the bundles configuration script overrides duplicate URL paths that appear in main.js.

4. Save the file and either build or serve application to complete the bundle loading configuration.

Configure Individual Loading of Libraries and Modules

You can configure path mappings to use CDN without accessing the bundles configuration script, so that modules and libraries are loaded individually from the CDN.

To configure individual loading of the libraries and modules based on the require block of the main.js (without the use of the bundles configuration script), perform the following steps.

1. Open the path_mapping.json file in the js subfolder of your application and change the use element to cdn.

```
"use": "cdn"
```

2. Edit the cdns element to remove the config element so cdns path definitions for Oracle JET and third-party libraries are formatted as following:

```

"cdns": {
    "jet": "https://static.oracle.com/cdn/jet/9.2.0/default/js",
    "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",
}

```

```

        "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty"
    }

```

Note: Starting in JET release 9.0.0, the convention of using the leading character "v" to identify the release number has changed. As the above sample shows, the release identifier is now a semver value specified as 9.2.0 with no leading character.

3. Save the file and either build or serve application to complete the `main.js` require block configuration.

When you need to make a change to the list of required libraries, for example, to specify a different release version, do not edit the `main.js`; edit instead the `path_mapping.json` file and, in the case of bundle loading, also edit the `bundles-config.js` URL in your application's `index.html` file. You will need to rebuild the application to apply the changes. For details about the `path_mapping.json` file and the configuration updates performed by the tooling, see [Understand the Path Mapping Script File and Configuration Options](#).

Understand the Path Mapping Script File and Configuration Options

When you build the application, the tooling invokes the `path_mapping.json` configuration file and determines the URL path for the Oracle JET modules and libraries based on the settings you configured for the `use` element.

The path mapping `use` element, set to `local` by default, specifies location of these require libraries:

- Core Oracle JET libraries (appear as `ojs`, `ojL10n`, `ojtranslations`)
- Third-party dependency libraries (for example, knockout, jquery, hammerjs, and other)

When you build your application, by default, Oracle JET will load the libraries from the local application as specified in the require block of the application's `main.js`. Each library URL is assembled from the `baseUrl` element and the `path` attribute of the `lib` element as specified by the path mapping file.

For example, the path mapping definition for the knockout library shows the following details.

```

"baseUrl": "js"
"use": "local"

"libs": {
    ...
    "path": "libs/knockout/knockout-3.x.x.debug"
}

```

And, after build or serve, the `main.js` require block contains the following URI:

`/js/libs/knockout/knockout-3.x.x.debug`

When configured for Oracle Content Delivery Network (CDN), the `main.js` require block is determined either entirely by the path mapping file local to the application or, in the case of the bundle loading optimization, partially from the path mapping file and partially from the require block of the `bundles-config.js` file maintained by Oracle on Oracle CDN. Path injector markers in `main.js` indicate where the release specific URLs appear.

CDN Scenario 1: To load libraries and modules as bundles from CDN, by default only the path mappings for third-party libraries will appear in the URL library list in the require block of `main.js`.

For example, the path mapping definition for the knockout library shows the following details. Note that the `config` attribute specifies the name of the bundles configuration script file as `bundles-config.js`.

```
"baseUrl": "js" <==ignored
"use": "cdn"

"cdns": {
    "jet": {
        "prefix": "https://static.oracle.com/cdn/jet/9.2.0/default/js",
        "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",
        "config": "bundles-config.js"
    },
    "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty"
},
"libs": {
    ...
    "cdnPath": "knockout/knockout-3.x.x"
}
```

And, after build or serve, the `main.js` require block contains a list of third-party library URLs as a placeholder, and, as the following code snippet shows, the `index.html` file references the script to load libraries and modules as bundles from CDN.

```
<body>
    <script type="text/javascript" src="https://static.oracle.com/cdn/jet/
9.2.0/default/js/bundles-config.js"></script>
    ..
</body>
```

Note that loading libraries and module as specified in the require block of the `bundles-config.js` file takes precedence over any duplicate libraries that may appear in the `main.js` require block. However, if you prefer, you can configure the third-party library path mapping so their URLs do not appear in the `main.js` require block. To accomplish this, set `"cdn": "3rdparty"` in the `path_mapping.json` file to show `"cdn": "jet"` for each third-party library path definition.

CDN Scenario 2: To load the libraries individually from CDN using the path mapping URLs to specify the location, the list of library URLs will appear entirely in the require block of `main.js`.

For example, the path mapping definition for the knockout library shows the following details after you edit the `cdns` element to remove the bundles configuration script reference.

```
"baseUrl": "js" <==ignored
"use": "cdn"

"cdns": {
    "jet": "https://static.oracle.com/cdn/jet/9.2.0/default/js",
    "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",
    "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty"
}

"libs": {
    ...
    "cdnPath": "knockout/knockout-3.x.x"
}
```

And, after build or serve, the `main.js` require block contains the following URL (along with the URLs for all other base libraries and modules):

```
"knockout": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty/knockout/
knockout-3.x.x"
```

CDN Scenario 3: If your application needs to access libraries that reside on a non-Oracle CDN, you can update the `path-mapping.js` file to specify your own CDN endpoint and library definition.

Depending on whether you use the bundles configuration script, add your CDN name and endpoint URI to the `cdns` definition as follows.

When using the bundles configuration script to load libraries and modules:

```
"cdns": {
    "jet": {
        "prefix": "https://static.oracle.com/cdn/jet/9.2.0/default/js",
        "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",
        "config": "bundles-config.js"
    },
    "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty"
    "yourCDN": "endPoint to your own CDN"
},
...
}
```

Or, when loading libraries and modules individually (not using the bundles configuration script):

```
"cdns": {
    "jet": "https://static.oracle.com/cdn/jet/9.2.0/default/js",
    "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",
    "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty",
    "yourCDN": "endPoint to your own CDN"
},
...
```

Then, in the list of libraries, define your library entry similar to the following sample.

```
"yourLib": {  
    "cdn": "yourCDN",  
    "cwd": "node_modules/yourLib",  
    "debug": {  
        "src": "yourLib.js",  
        "path": "libs/yourLib/yourLib.js",  
        "cdnPath": "yourLib/yourLib.js"  
    },  
    "release": {  
        "src": "yourLib.min.js",  
        "path": "libs/yourLib/yourLib.min.js",  
        "cdnPath": "yourLib/yourLib.min.js"  
    }  
},
```

Auditing Application Files

An Oracle JET audit runs against the application files of your JET project and performs a static analysis of the source code from an Oracle JET perspective. Audit diagnostic messages result from an invocation of the Oracle JET Audit Framework (JAF) command-line utility and are governed by rules that are specific to a JET release version.

Oracle JET Audit Framework (JAF) relies on the configuration file created by the JET tooling when you invoke the JAF initialization command `ojaudit --init` in a Command Prompt window on the JET application.

The `oraclejafconfig.json` file that you create when you initialize Oracle JAF the first time defines the properties that you can use to control many aspects of your JET application audit. For example, by configuring the JAF audit, you can perform the following.

- Specify the JET version when you want to use audit rules that are specific to a JET version. This is configured by default as the JET version of the application to be audited.
- Specify the file set when you want to exclude application directories and file types. This is configured by default to include all files of the application to be audited.
- Invoke custom audit rules that are user-defined and assembled as a JAF rule pack for distribution.
- Prevent specific audit rules from running in the audit or limiting the audit to only rules of a certain severity level.
- Include the metadata of Oracle JET Web Components to audit the HTML files of your application's custom components.
- Control the JavaScript source code to audit based on JAF comments that you embed in your source files.
- Work with the output of the audit to customize the presentation of audit messages or to suppress audit messages.

The properties in the `oraclejafconfig.json` file configuration settings are up to you to specify. By doing so, you can fine-tune the audit to focus audit results on only the source that you intend. Multiple configuration files can be created for specific runtime criteria or projects. The configuration files are JSON format, but JavaScript style comments are permitted for documentation purposes. The configuration file to be used can be specified on the command-line.

Each time you run the audit from a Command Prompt window, Oracle JAF searches the directory in which you initiated the audit for the JAF configuration file `oraclejafconfig.json`. If no configuration file is found there, then JAF processes only HTML files found in the current directory. In that case, the default JAF configuration settings are used for the audit.

If the built-in audit rules provided with the JAF installation do not meet all the diagnostic requirements of your application, you can write custom audit rules to extend JAF. You implement user-defined audit rules as JavaScript files. The JAF API allows

you to register event listeners and handle the audit context created by JAF on the file set of your JET projects. Custom audit rules can be assembled into distributable rule packs and invoked by developers on any Oracle JET application.

For more information about JAF, see *Using and Extending the Oracle JET Audit Framework*.

Testing and Debugging

Test and debug Oracle JET web applications using your favorite testing and debugging tools for client-side JavaScript applications. You can use a similar process to debug hybrid mobile applications when run in the browser, and you can also test and debug hybrid mobile applications on a simulator, emulator, or device.

Topics:

- [Typical Workflow for Testing and Debugging an Oracle JET Application](#)
- [Test Oracle JET Applications](#)
- [Debug Oracle JET Applications](#)

Typical Workflow for Testing and Debugging an Oracle JET Application

Understand testing tools and debugging options for Oracle JET web and hybrid mobile applications.

To test and debug an Oracle JET application, refer to the typical workflow described in the following table:

Task	Description	More Information
Test Oracle JET applications	Identify testing tools for web and hybrid mobile applications.	Test Oracle JET Applications
Debug Oracle JET applications	Identify debugging options for Oracle JET web and hybrid mobile applications.	Debug Oracle JET Applications

Test Oracle JET Applications

Use third-party tools such as QUnit or Selenium WebDriver to test your Oracle JET application. Google, Apple, and Microsoft also provide instructions for testing Android, iOS, and Windows applications.

Topics:

- [Test Applications](#)
- [Test Hybrid Mobile Applications](#)
- [Use BusyContext API in Automated Testing](#)

Test Applications

You can use virtually any testing tool that tests client-side HTML applications written in JavaScript for testing Oracle JET applications.

For internal development, Oracle JET uses the following tools for testing Oracle JET components and toolkit features:

- QUnit: JavaScript unit testing framework capable of testing any generic JavaScript project and used by the jQuery, jQuery UI, and jQuery Mobile projects.

QUnit requires configuration on your test page to include library and CSS references. You must also add the HTML `div` element to your page. In the example below, the highlighted code shows additions required by QUnit. The actual paths will vary, depending upon where you install QUnit.

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="../../../../../../code/css/libs/oj/v9.2.0/alta/oj-alta-min.css"></link>
    <link rel="stylesheet" href="../../../../../css/qunit.css">
    <script type="text/javascript" src="../../../../../js/qunit.js"></script>
    <script>
      QUnit.config.autoload = false;
      QUnit.config.autostart = false;
    </script>
    <script data-main="js/main" src="../../../../../code/js/libs/require/
require.js"></script>
  </head>
  <body>
    <div id="qunit"></div>
    <div id="qunit-fixture">
      <oj-slider id="slider1"></oj-slider>
    </div>
  </body>
</html>
```

For more information about QUnit, see <http://qunitjs.com>.

- Selenium WebDriver: Alternative method of testing applications that you do not want to modify for QUnit or that contain multiple HTML pages.

For additional information, see <http://docs.seleniumhq.org/projects/webdriver>.

Test Hybrid Mobile Applications

You can test your hybrid mobile application in a local browser, using an emulator or simulator, and on an attached device.

- Testing in a local browser

When you invoke `ojet serve` with the `--browser` option to serve your hybrid mobile application in a local browser, you can use the same method for testing Android, iOS, and Windows applications that you would use for testing any web application. However, this method can only approximate the end user experience and is most useful early in the development process.

Depending upon your use case, you may need to add Cordova plugins to add functionality to your mobile hybrid app. Many Cordova plugins provide mock data when deploying to the browser platform. If, however, you add a Cordova plugin that doesn't have browser platform support, you can add objects that represent mock data to `cordovaMock.js` in `src/js`.

- Testing in an emulator or simulator

You can invoke `ojet serve` with the `--emulator` option to test the functionality of your application in the iOS Simulator, Windows emulator, or Android Virtual Devices (AVDs) using the Android emulator. These methods can approximate the look and feel of an actual device, but you won't be able to test performance or responsiveness to touch reliably. For additional information, see

- [iOS Simulator](#)
- [Android Emulator](#)
- [Windows Emulator](#)

- Testing on attached physical devices

You can invoke `ojet serve` with the `destination=device` option to test the functionality on attached physical devices. This provides the most reliable form of testing, but you may not have access to all the devices that your users might use to run your application.

If you want to serve your application to an iOS device, you must take additional steps as described in [Package and Publish Hybrid Mobile Applications](#).

- Working around cross-origin resource sharing (CORS) issues

Hybrid mobile applications that communicate with remote services may encounter issues if they request resources that originate in different domains. An example includes a response such as the following:

```
No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

To work around these types of issue, your server-side administrator may need to modify the policy used by the remote server to allow cross-site requests.

For example, to access a remote service managed by Oracle's Mobile Cloud Service (MCS), an MCS administrator configures MCS's `Security_AllowOrigin` environment policy with a comma separated list of URL patterns that identify the remote services that serve a resource from different domains.

For additional information about testing hybrid mobile applications on specific platforms, see the following resources:

- [Android Testing](#)
- [iOS Testing](#)
- [Windows Debugging, testing, and performance](#)

Use BusyContext API in Automated Testing

Use `BusyContext` to wait for a component or other condition to complete some action before interacting with it.

The purpose of the `BusyContext` API is to accommodate sequential dependencies of asynchronous operations. Typically, you use `BusyContext` in test automation when you

want to wait for an animation to complete, a JET page to load, or a data fetch to complete.

Wait Scenarios

The Busy Context API will block until all the busy states resolve or a timeout period lapses. There are four primary wait scenarios:

- Components that implement animation effects
- Components that fetch data from a REST endpoint
- Pages that load bootstrap files, such as the Oracle JET libraries loaded with RequireJS
- Customer-defined scenarios that are not limited to Oracle JET, such as blocking conditions associated with application domain logic

Determining the Busy Context's Scope

The first step for waiting on a busy context is to determine the wait condition. You can scope the granularity of a busy context for the entirety of the page or limit the scope to a specific DOM element. Busy contexts have hierarchical dependencies mirroring the document's DOM structure with the root being the page context. Depending on your particular scenario, target one of the following busy context scopes:

- Scoped for the page

Choose the page busy context to represent the page as a whole. Automation developers commonly need to wait until the page is fully loaded before starting automation. Also, automation developers are usually interested in testing the functionality of an application having multiple Oracle JET components rather than a single component.

```
var busyContext = Context.getPageContext().getBusyContext();
```

- Scoped for the nearest DOM element

Choose a busy context scoped for a DOM node when your application must wait until a specific component's operation completes. For example, you may want to wait until an `ojPopup` completes an open or close animation before initiating the next task in the application flow. Use the `data-oj-context` marker attribute to define a busy context for a DOM subtree.

```
<div id="mycontext" data-oj-context>
  ...
  <!-- JET content -->
  ...
</div>

var node = document.querySelector("#mycontext");
var busyContext = Context.getContext(node).getBusyContext();
```

Determining the Ready State

After obtaining a busy context, the next step is to inquire the busy state. `BusyContext` has two operations for inquiring the ready state: `isReady()` and `whenReady()`. The `isReady()` method immediately returns the state of the busy context. The `whenReady()`

method returns a Promise that resolves when the busy states resolve or a timeout period lapses.

The following example shows how you can use `isReady()` with WebDriver.

```
public static void waitForJetPageReady(WebDriver webDriver, long
timeoutInMillis)
{
    try
    {
        final WebDriverWait wait = new WebDriverWait(webDriver,
timeoutInMillis / _THOUSAND_MILLIS);
        // Eat any WebDriverException
        // "ExpectedConditions.jsReturnsValue" will continue to be called
if it doesn't return a value.
        // /ExpectedConditions.java#L1519
        wait.ignoring(WebDriverException.class);

        wait.until(ExpectedConditions.jsReturnsValue(_PAGE_WHEN_READY_SCRIPT));
    }

    catch (TimeoutException toe)
    {
        String evalString = "return
Context.getPageContext().getBusyContext().getBusyStates().join('\\\\n');");
        Object busyStatesLog =
((JavascriptExecutor)webDriver).executeScript(evalString);
        String retVal = "";
        if (busyStatesLog != null){
            retVal = busyStatesLog.toString();
            Assert.fail("waitForJetPageReady failed - !
Context.getPageContext().getBusyContext().isReady() - busyStates: " +
retVal);    }
    }

    // The assumption with the page when ready script is that it will
continue to execute until a value is returned or
    // reached the timeout period.
    //
    // There are three areas of concern:
    // 1) Has the application opt'd in on the whenReady wait for bootstrap?
    // 2) If the application has opt'd in on the jet whenReady strategy for
bootstrap "('oj_whenReady' in window)",
    // wait until jet core is loaded and have a ready state.
    // 3) If not opt-ing in on jet whenReady bootstrap, make the is ready
check if jet core has loaded. If jet core is
    // not loaded, we assume it is not a jet page.

    // Check to determine if the page is participating in the jet whenReady
bootstrap wait period.

    static private final String _BOOTSTRAP_WHEN_READY_EXP =
"('oj_whenReady' in window) && window['oj_whenReady']);"

    // Assumption is we must wait until jet core is loaded and the busy
```

```

state is ready.

static private final String _WHEN_READY_WITH_BOOTSTRAP_EXP =
"(window['oj'] && window['oj']['Context'] &&
Context.getPageContext().getBusyContext().isReady() ?" +
" 'ready' : '')";

// Assumption is the jet libraries have already been loaded. If they
have not, it's not a Jet page.
// Return jet missing in action "JetMIA" if jet core is not loaded.
static private final String _WHEN_READY_NO_BOOTSTRAP_EXP =
"(window['oj'] && window['oj']['Context'] ?" +
"(Context.getPageContext().getBusyContext().isReady() ? 'ready' : '') :
'JetMIA')";

// Complete when ready script
static private final String _PAGE_WHEN_READY_SCRIPT =
"return (" + _BOOTSTRAP_WHEN_READY_EXP + " ? " +
_WHEN_READY_WITH_BOOTSTRAP_EXP + " : " +
_WHEN_READY_NO_BOOTSTRAP_EXP + ");";

```

The following example shows how you can use `whenReady()` with QUnit.

```

// Utility function for creating a promise error handler
function getExceptionHandler(assert, done, busyContext)
{
    return function (reason)
    {
        if (reason && reason['busyStates'])
        {
            // whenReady timeout
            assert.ok(false, reason.toString());
        }
        else
        {
            // Unhandled JS Exception
            var msg = reason ? reason.toString() : "Unknown Reason";
            if (busyContext)
                msg += "\n" + busyContext;
            assert.ok(false, msg);
        }
    }

    // invoke done callback
    if (done)
        done();
};

QUnit.test("popup open", function (assert)
{
    // default whenReady timeout used when argument is not provided
    Context.setBusyContextDefaultTimeout(18000);

    var done = assert.async();

```

```
assert.expect(1);

var popup = document.getElementById("popup1");

// busy context scoped for the popup
var busyContext = Context.getContext(popup).getBusyContext();
var errorHandler = getExceptionHandler(assert, done, busyContext);

popup.open("#showPopup1");

busyContext.whenReady().then(function ()
{
    assert.ok(popup.isOpen(), "popup is open");
    popup.close();
    busyContext.whenReady().then(function ()
    {
        done();
    }).catch(errorHandler);
}).catch(errorHandler);
});
```

Creating Wait Conditions

Jet components utilize the busy context to communicate blocking operations. You can add busy states to any scope of the busy context to block operations such as asynchronous data fetch.

The following high level steps describe how to add a busy context:

1. Create a Scoped Busy Context.
2. Add a busy state to the busy context. You must add a description that describes the purpose of the busy state. The busy state returns a resolve function which is called when it's time to remove the busy state.

Busy Context dependency relationships are determined at the point the first busy state is added. If the DOM node is re-parented after a busy context was added, the context will maintain dependencies with any parent DOM contexts.

3. Perform the operation that needs to be guarded with a busy state. These are usually asynchronous operations that some other application flow depends on its completion.
4. Resolve the busy state when the operation completes.

The application is responsible for releasing the busy state. The application must manage a reference to the resolve function associated with a busy state, and it must be called to release the busy state. If the DOM node that the busy context is applied to is removed in the document before the busy state is resolved, the busy state will be orphaned and will never resolve.

Debug Oracle JET Applications

Since Oracle JET web applications are client-side HTML5 applications written in JavaScript, you can use your favorite browser's debugging facilities. The process for

debugging hybrid mobile applications differs, depending on whether you're debugging your application in a web browser, emulator, or device.

Topics:

- [Debug Web Applications](#)
- [Debug Hybrid Mobile Applications](#)

Debug Web Applications

Use your favorite browser's debugging facilities for debugging.

Before you debug your application, you should verify that your application is using the debug version of the Oracle JET libraries. If you used the tooling to build your application in development mode, then your application should be using the debug library. If, however, you are using one of the sample applications, it may be configured to use the minified version of the Oracle JET libraries.

When you are ready to debug the application that you build using the tooling in release mode, then you can use the `--optimize=none` flag to make the bundled output more readable by preserving line breaks and white space.

```
ojet build --release --optimize=none
```

To verify that you are using the debug version of the library, check your RequireJS bootstrap file (typically `app/home/js/main.js`) for the Oracle JET library path, highlighted in **bold** below.

```
requirejs.config(
{
  baseUrl: 'js',

  // Path mappings for the logical module names
  paths:
  //injector:mainReleasePaths
  {
    ... contents omitted

    'ojs': 'libs/oj/v9.2.0/min',
    'ojL10n': 'libs/oj/v9.2.0/ojL10n',
    'ojtranslations': 'libs/oj/v9.2.0/resources',
    'text': 'libs/require/text',
    'signals': 'libs/js-signals/signals'
    ...
  }
  ... contents omitted
};
```

In this example, the path is pointing to the minified version of the Oracle JET libraries. To change to the debug version, edit the bootstrap file and replace `min` with `debug` as shown.

```
'ojs': 'libs/oj/v9.2.0/debug',
```

Debugging facilities vary by browser. For example, with Google Chrome you can:

- do normal debugging, including setting breakpoints and inspecting CSS using Chrome Inspector (<chrome://inspect>) without having to install any 3rd party add-ons.
- add the Knockoutjs Context Debugger extension to examine the Knockout context and data for a DOM node in a sidebar.
- add the Window Resizer extension to resize the browser window to emulate common screen sizes or set your own. You can find the extensions and other developer tools at the [Chrome Web Store](#).

If you're using an Integrated Development Environment (IDE) for development, you can use debugging tools such as the one provided by Microsoft Visual Studio Code to set break points and watches.

Debug Hybrid Mobile Applications

The process for debugging hybrid mobile applications differs, depending on whether you're debugging your application in a web browser, emulator, or device.

There are several ways to debug a hybrid mobile application, depending on the environment where you are running the application:

- Debugging in a local web browser

Since Cordova applications are HTML5 applications written in JavaScript, you can use the same debugging facilities that you would use for web applications as described in [Debug Web Applications](#). With this approach, however, you might find it difficult to debug code that depends on device services and Cordova plugins.

To run an Oracle JET hybrid mobile application in your desktop Safari or Chrome browser, [serve](#) it with the `--browser` option.

```
ojet serve ios|android|windows --browser
```

Once your app loads in the browser, you can use the browser's developer tools to view the source code, debug JavaScript using breakpoints and watches, or change the source code in the browser, which can be especially useful when you want to experiment with CSS changes that you intend to make in your app. For more information about the Chrome browser's developer tools, see the [Chrome DevTools documentation](#) and for the iOS Safari browser's developer tools, see the [Web Inspector Tutorial](#).

- Debugging in the emulator

You can run the app in the emulator which eliminates the need for an actual device. However, emulators can run slowly and include limited support for emulating native device services.

To run an Oracle JET hybrid mobile application in the default emulator, [serve](#) it with the `--emulator` option.

```
ojet serve ios|android|windows --emulator
```

You can also specify a named emulator or simulator on Android or iOS devices:

```
ojet serve ios|android|windows --emulator=emulator-name
```

To determine the emulator or simulator name:

- For Android emulators, run android avd at a command prompt.
- For iOS simulators, at a command prompt in the application's hybrid directory, enter the following: ios-sim showdevicetypes.
- Debugging on the device

Debugging the application on a real device provides the most accurate experience, but the development cycle can take longer since you must package the application, deploy it to the device, and use a desktop browser or other development tool to connect to it running on the device.

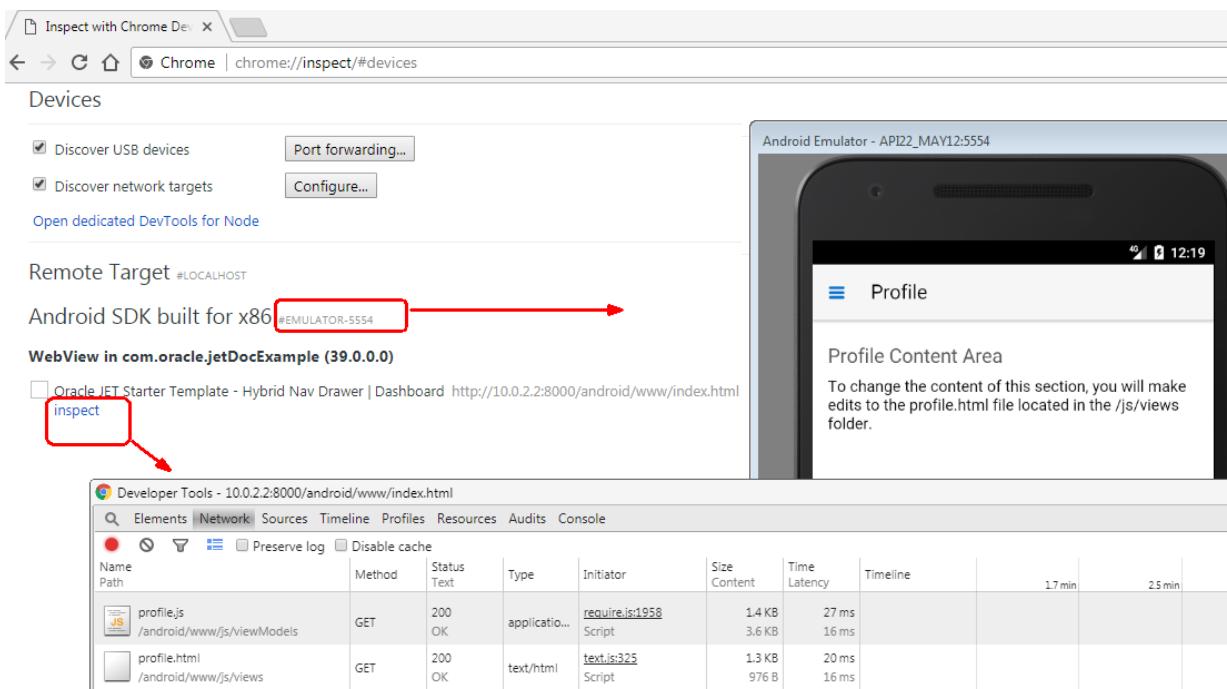
To run an Oracle JET hybrid mobile application on the device, [serve](#) it with the --device option.

```
ojet serve ios|android|windows --device
```

Once the app installs and runs on the device or on the emulator, you can use your browser's developer tools to debug and inspect your application's code as it executes. The browser developer tools that you use depend on the platform where your application runs.

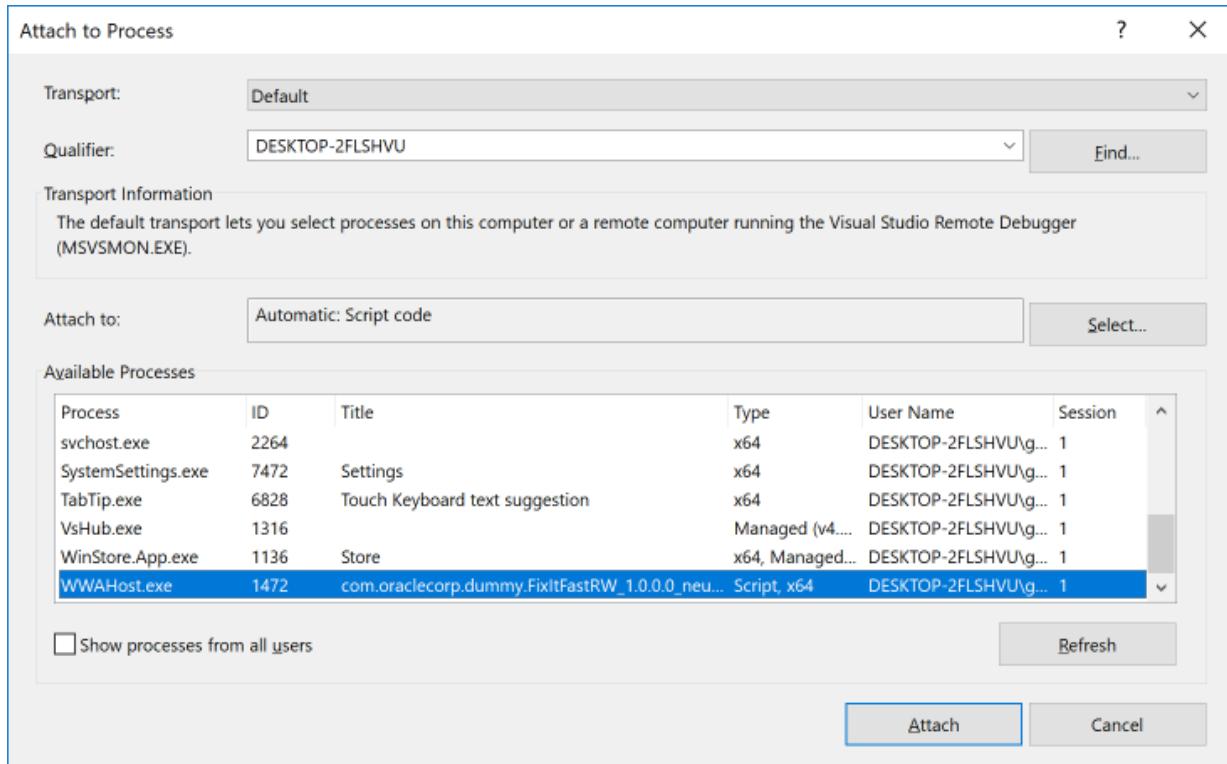
Use the Safari browser's Web Inspector, described in the [Web Inspector Tutorial](#), if you run your app on an iOS device or simulator.

Use the Chrome browser's developer tools if your app runs on an Android emulator or device. For the Android platform, once JET has served the app to the emulator or device, enter chrome://inspect/ in the Chrome browser's address bar to access the developer tools. This renders the Inspect with Chrome Developer Tools page that lists the connected devices and the applications on these devices that you can inspect using the developer tools. Click the **inspect** link beside the app you want to inspect, as illustrated by the following figure which shows the developer tools in the foreground for the application running in the emulator.



For more information about debugging apps on an Android device using Chrome DevTools, see [Get Started with Remote Debugging Android Devices](#) in Google's documentation.

Use Visual Studio to debug and inspect your application's code if your application runs on the Windows platform. Serve your application on your Windows machine and attach the process for the application to the Visual Studio debugger, as described in [Microsoft's documentation](#). JavaScript apps, such as your application, run in an instance of the `wwahost.exe` process on Windows, so multiple instances of the `wwahost.exe` process appear in the Attach to Process dialog if you have more than one application running. Use the Title column of the Attach to Process dialog to select the correct application to attach, as illustrated by the following image where an instance of the FixItFast sample application is selected.



Once you have attached your app, use the Visual Studio debugger, as described in [Microsoft's documentation](#), to debug your application.

21

Packaging and Deploying Applications

If you used Oracle JET tooling to create your Oracle JET application, you can package web applications for deployment to a web or application server and hybrid mobile applications for deployment to the Google Play or Apple App stores.

Topics

- [Typical Workflow for Packaging and Deploying Applications](#)
- [Package and Deploy Web Applications](#)
- [Package and Publish Hybrid Mobile Applications](#)
- [Remove and Restore Non-Source Files from Your JET Application](#)

Typical Workflow for Packaging and Deploying Applications

Understand how to use Oracle JET's tooling to package and deploy your Oracle JET application.

To package and deploy your Oracle JET application, refer to the typical workflow described in the following table:

Task	Description	More Information
Remove Build Output and Extraneous Files from Your JET Application's Source	Use <code>ojet clean</code> or <code>ojet strip</code> to remove extraneous files from the source of your project if you want to package your source code for distribution to colleagues.	Remove and Restore Non-Source Files from Your JET Application
Package and Deploy Web Application	Use the JET tooling to package your application and ready it for deployment.	Package and Deploy Web Applications
Package and Publish Hybrid Mobile Application	Use the JET tooling to package your application and ready it for publishing.	Package and Publish Hybrid Mobile Applications

Package and Deploy Web Applications

Use the Oracle JET tooling and third-party tools to package your Oracle JET web application and deploy it to any type of web or application server.

Topics

- [Package Web Applications](#)
- [Deploy Web Applications](#)

Package Web Applications

If you created your application using the tooling, use the Oracle JET command-line interface (CLI) to create a release version of your application containing your application scripts and applicable Oracle JET code in minified format.

1. From a terminal prompt in your application's root directory, enter the following command: `ojet build --release`.

The command will take some time to complete. When it's successful, you'll see the following message: `Build finished!`.

The command replaces the development version of the libraries and scripts in `web/js/` with minified versions where available.

2. To verify that the application still works as you expect, run `ojet serve` with the `--release` option.

The `ojet serve --release` command takes the same arguments that you used to `serve` your web application in development mode.

```
ojet serve --release [--serverPort=server-port-number --serverOnly]
```



Tip:

For a complete list of options, type `ojet help serve` at the terminal prompt.

Deploy Web Applications

Oracle JET is a collection of HTML, JavaScript, and CSS files that you can deploy to any type of web or application server. There are no unique requirements for deploying Oracle JET applications.

Deployment methods are quite varied and depend upon the type of server environment your application is designed to run in. However, you should be able to use the same method for deploying Oracle JET applications that you would for any other client interface in your specific environment.

For example, if you normally deploy applications as zip files, you can zip the `web` directory and use your normal deployment process.

Package and Publish Hybrid Mobile Applications

You can use the Oracle JET command line tooling to package your hybrid mobile app for publishing to Google Play, Windows Store, or Apple App Store.

Topics

- [About Packaging and Publishing Hybrid Mobile Applications](#)
- [Package a Hybrid Mobile App on Android](#)
- [Package a Hybrid Mobile App on iOS](#)

- [Package a Hybrid Mobile App on Windows](#)

About Packaging and Publishing Hybrid Mobile Applications

The steps to package your app for release are broadly the same, irrespective of the platform where your app is going to be installed.

You create a build configuration file that identifies the location of your digital certificate, and then use the build configuration file as an input parameter to the `ojet build` command in order to sign your app with the digital certificate.

Before you proceed to signing and packaging your app for each platform where it will be used, review the entries in your app's `config.xml` file, as described in [Review Your Application Settings in the config.xml File](#). Once you complete that task, proceed to the platform-specific instructions in the following sections.

Package a Hybrid Mobile App on Android

To prepare your app for distribution on Android devices, you need to sign it using a digital certificate and package it into an `.APK` file.

To sign an app that you want to release and publish through the Google Play Store or some other distribution mechanism, you need a unique key. Your organization may provide you with one, in which case you specify its location in the build configuration file that you use to sign and package your app. If you don't have a key, you can create one that you use to sign your app. Subsequent updates to your app must be signed using the same key that you create. You can create a self-signed key using the JDK's `keytool` utility or using the dialogs that Android Studio provides from its **Build > Generate Signed APK** menu. For more information about the latter option, see the ["Generate a key and keystore"](#) section in the Sign Your App page on the Android developer's site.

Once you have generated your key, specify the location of its store, its alias, and its access credentials in your build configuration file, as shown by the following example.

```
{  
  "android": {  
    "release": {  
      "keystore": "/home/pathTo/keystore/android.jks",  
      "storePassword": "MyKeystorePassword",  
      "alias": "myAndroidKey",  
      "password" : "MyKeyPassword",  
      "keystoreType": ""  
    }  
  }  
}
```

You can now package your app into an `.APK` file (the installation file type for apps on Android devices).

At a terminal prompt, in your app's top-level directory, enter the following command:

```
ojet build android --release --build-config=/pathTo/yourBuildConfig.json
```

On successful completion, the terminal window displays output similar to the following:

...

```
...
Oracle JET CLI
Running "build:release" (build) task
Running "oraclejet-build:release" (oraclejet-build) task
  Oracle JET Tooling
  cleaning staging path.....
...
BUILD SUCCESSFUL

Total time: 4.376 secs

Built the following apk(s):
  /appRootDir/hybrid/platforms/android/build/outputs/apk/android-
release.apk
Cordova compile finished.....

Done.
```

The build command outputs the .APK file that packages your app to the following location:

/appRootDir/hybrid/platforms/android/build/outputs/apk/android-release.apk.

Use this file to release your app through a public app marketplace, such as Google Play, a private app store, or some other means. End users who install your app from a location other than Google Play need to configure their device to opt in to install the app from an unknown source. For information about all these options, see <https://developer.android.com/studio/publish/index.html#publishing-release>.

Package a Hybrid Mobile App on iOS

To prepare your app for distribution on iOS devices, you need to sign it and package it into an .IPA file.

Joining the Apple Developer Program is the first step to submit your iOS app to the Apple App Store, to distribute an in-house app, or to sign an app that you distribute outside the Apple App Store. As a member, you have access to the resources you need to configure app services and to submit new apps and updates. For more information on the Apple Developer Program, see [Apple Developer Program](#) in Apple's documentation.

Code signing your app allows users to trust that your app has been created by a source known to Apple and that it hasn't been tampered with. All apps must be code signed and provisioned to deploy to an iOS device, to use certain Apple services, to be distributed for testing on iOS devices, or to be uploaded to the Apple App Store.

Signing identities are used to sign your app or installer package. A signing identity consists of a public-private key pair that Apple issues. The public-private key pair is stored in your keychain, and is used by cryptographic functions to generate the signature. A *certificate* stored in your developer account contains the associated public key. An *intermediate certificate* is also required to be in your keychain to ensure that your certificate is issued by a certificate authority.

Code signing requires that you have both the signing identity and the intermediate certificate installed in your keychain. When you install Xcode, Apple's intermediate certificates are added to your keychain for you.

There are different types of certificates. A *development* certificate identifies a person on your team and is used to run an app on a single device. A *distribution* certificate identifies the team and is used to submit your app to the Apple App store or to distribute it outside of the store. Within an enterprise (organization) distribution certificates can be shared by team members in order to deploy various apps to a number of different devices. Certificates are issued and authorized by Apple.

If you are a member of an organization membership in the Apple Developer Program, you should request your team agent to provide your required signing identities and certificates. Otherwise, you can create all the types of certificates and signing identities you need using the Apple Developer portal at <https://developer.apple.com/account>.

For more information on managing signing identities and certificates, see [Code Signing](#) in Apple's documentation.

When you code sign an app, a *provisioning profile* is installed into the package that associates one or more certificates and one or more devices with an *application ID*. Provisioning profiles are created using the Apple Developer portal and must be downloaded and installed into Xcode on your development machine.

An application ID is a two-part string used to identify one or more apps from a single development team. The string consists of a *Team ID* and a *bundle ID search string*, with a period (.) separating the two parts. The Team ID is supplied by Apple and is unique to a specific development team, while the bundle ID search string is supplied by you to match either the bundle ID of a single app or a set of bundle IDs for a group of your apps.

Application IDs and devices are registered with your team account using the Apple Developer portal.

For more information on managing application IDs, devices and provisioning profiles, see Apple's documentation.

Once you have created and installed the necessary code signing artifacts, specify the details in your build configuration file, as shown in the following example:

```
{
  "ios": {
    "debug": {
      "provisioningProfile": "My Developer Provisioning Profile ID",
      "developmentTeam": "My Team ID",
      "codeSignIdentity": "iPhone Developer",
      "packageType": "development"
    },
    "release": {
      "provisioningProfile": "My Distribution Provisioning Profile ID",
      "developmentTeam": "My Team ID",
      "codeSignIdentity": "iPhone Distribution",
      "packageType": "enterprise"
    }
  }
}
```

At time of writing, Apache Cordova does not support the new build system that Xcode 10, the latest release of Xcode, uses by default. If you use Xcode 10, you can work around this issue until Apache Cordova supports Xcode 10 by including the

`buildFlag` command-line argument in your build configuration file, as demonstrated by the following example, so that the older build system builds your app.

```
{  
    ...  
    "codeSignIdentity": "iPhone Distribution",  
    "packageType": "enterprise",  
    "buildFlag": [  
        "-UseModernBuildSystem=0"  
    ]  
}
```

You can now package your app into an .IPA file (the installation file type for apps on iOS devices).

At a terminal prompt, in your app's top-level directory, enter the following command:

```
ojet build ios --device --release --build-config=/pathTo/yourBuildConfig.json
```

On successful completion, the terminal window displays output similar to the following:

```
...  
** ARCHIVE SUCCEEDED **  
  
Exported appName.xcarchive to: appDir/hybrid/platforms/ios/build/device  
  
** EXPORT SUCCEEDED **  
  
Cordova compile finished....  
  
Done.
```

The resulting .IPA file will be located at:

```
appDir/hybrid/platforms/ios/build/device/appName.ipa
```

This file can be deployed to an iOS device that matches the installed provisioning profile using iTunes, or you can release your app through the public marketplace, Apple App Store, a private app store, or some other means. For information about submitting your app to the Apple App Store, see [Distribute an app through the App Store](#) in Apple's documentation.

Package a Hybrid Mobile App on Windows

To prepare your app for distribution on Windows devices or submission to the Windows Store, you need to sign it and package it into an .APPX file.

To complete this task, you need access to a digital certificate (a .PFX file) to sign your app. You must also create a build configuration file (`build.json`) where you specify the location of the digital certificate and a number of other parameters. You use the `build.json` file as an input parameter to the `ojet build` command that JET uses to package your app for Windows.

Create the Build Configuration File to Package Your Application on Windows

You need to create a build configuration file in JSON format that passes details, such as the location of your PFX file, to the `ojet build` command that JET uses to package your app for Windows.

The following example build configuration file shows sample entries that you can use to package an app in debug and release mode.

```
{  
  "windows": {  
    "debug": {  
      "packageCertificateKeyFile": "C:\\  
\\AppRootDir\\hybrid\\platforms\\windows\\CordovaApp_TemporaryKey.pfx"  
    },  
    "release": {  
      "packageCertificateKeyFile": "c:\\path-to-key\\keycert.pfx",  
      "packageThumbprint": "ABCABCABC123123123123",  
      "publisherId": "CN=Doc Example,OU=JET,O=Oracle,C=US"  
    }  
  }  
}
```

Create your `build.json` file using the previous example as a reference. For information about creating and installing a PFX file, see [Install a Personal Information Exchange File in Your Computer's Certificate Store](#). Obtain the values for the `packageThumbprint` and the `publisherID` entries by running the following command in a PowerShell command window:

```
Get-PfxCertificate -FilePath "c:\\path-to-key\\keycert.pfx"
```

PowerShell prompts you for the PFX file's password if the file is password protected. Output similar to the following then appears:

Thumbprint	Subject
----- 702F25BA3FED453A3F8ADCC13900A6353703AB54	----- CN=Doc Example, OU=JET, O=Oracle, C=US

For more information about the `Get-PfxCertificate` command, see [Microsoft's documentation](#).

Build Your Application for Windows

Execute the `ojet build` command with the `release` flag from your app's top level directory to build the app.

You must specify the architecture that you want the app to target when you package the app for release, as demonstrated by the following example that targets the `x64` and `x86` architectures.

```
ojet build windows --release --platform-options="--archs=\"x64 x86\" --build-  
config=pathTo\\build.json
```

On successful completion, the following directory contains the application package and other resources for your app:

AppRootDir\hybrid\platforms\windows\AppPackages\

The AppPackages directory contains another directory (CordovaApp.Windows10_*_Test). The actual name of the directory depends on the version number of your app and the architecture that it targets. For example, a version 1 app that targets the x64 architecture will be named CordovaApp.Windows10_1.0.0.0_x64_Test.

Submit the application package (.appx) in the directory to the Windows App Store or distribute the contents to end users with Windows devices who want to install your app. The directory includes a PowerShell script (Add-AppDevPackage.ps1) that end users execute to install the application. In addition to the script, the directory contains dependent packages and the certificate that signed the application. The following example lists the contents:

```
Add-AppDevPackage.ps1  
Add-AppDevPackage.resources  
CordovaApp.Windows10_1.0.0.0_x64.appx  
CordovaApp.Windows10_1.0.0.0_x64.cer
```

For information about how to submit your app to the Windows Store, see <https://developer.microsoft.com/en-us/store/publish-apps>.

Remove and Restore Non-Source Files from Your JET Application

The Oracle JET CLI provides commands (clean, strip, and restore) that manage the source code of your JET application by removing extraneous files, such as the build output for the platforms your JET application supports or npm modules installed into your project.

Consider using these commands when you want to package your source code for distribution to colleagues or others who may work on the source code with you. Use of these commands may not be appropriate in all circumstances. Use of the clean and strip commands will, for example, remove the build output for hybrid mobile applications that includes the installation files to install the application on a user's device.

ojet clean

Use the ojet clean command to clean the build output of your JET application. Specify the appropriate parameter with the ojet clean command to clean the build output on the platform(s) that your JET application supports (android, ios, windows, and web). This command can be useful when developing a hybrid mobile application that makes use of staging files as you can make sure that all staging files are removed between builds. These staging files will be regenerated the next time that you build or serve the hybrid mobile application. Run the following command in your application's top level directory to clean the output of your application that targets the Android platform:

```
ojet clean android
```

Similarly, run ojet clean web to remove the contents of your application's root directory's web directory.

ojet strip

Use `ojet strip` when you want to remove all non-source files from your JET application. In addition to the build output removed by the `ojet clean` command, `ojet strip` removes additional dependencies, such as Cordova plugins, and npm modules installed into your project. A typical usage scenario for the `ojet strip` command is when you want to distribute the source files of your JET application to a colleague and you want to reduce the number of files to transmit to the minimum.

The `ojet strip` command relies on the presence of the `.gitignore` file in the root directory of your application to determine what to remove. The file lists the directories that are installed by the tooling and can therefore be restored by the tooling. Only those directories and files listed will be removed when you run `ojet clean` on the application folder.

If you do not use Git as your SCM tool and you want to run `ojet strip` to make a project easier to transmit, you can create the `.gitignore` file and add it to your application's root folder with a list of the folders and files to remove, like this:

```
#List of web application folders to remove
/node_modules
/bower_components
/themes
/web

#List of hybrid mobile application folders to remove
/hybrid
/node_modules
/hybrid/platforms
/hybrid/www

#List of exceptions not to remove from above folder list
!hybrid/plugins
hybrid/plugins/*
!hybrid/plugins/fetch.json
```

ojet restore

Use the `ojet restore` command to restore the dependencies, plugins, libraries, and Web Components that the `ojet strip` command removes. After the `ojet restore` command completes, use the `ojet build` and/or `ojet serve` commands to build and serve your JET application.

For additional help with CLI commands, enter `ojet help` at a terminal prompt.

A

Troubleshooting

Follow the same procedure for troubleshooting your Oracle JET application that you would follow for any client-side JavaScript application.

If you're having issues troubleshooting a specific Oracle JET component or toolkit feature, see [Oracle JET Support](#). Before requesting support, be sure to check the product [Release Notes](#).

B

Oracle JET Application Migration for Release 9.2.0

If you used Oracle JET tooling to scaffold your application with Oracle JET version 5.x.0 or later, you can migrate your application manually to version 9.2.0.

Before you migrate your application, be sure to check the Oracle JET [Release Notes](#) for any component, framework, or other change that could impact your application.

 **Important:**

This process is not supported for Oracle JET releases prior to version 5.0.0.

Topics:

- [Migrate Application Source to Release 9.2.0](#)
- [Migrate a v9.x.0 Application to v9.2.0](#)
- [Migrate to the Redwood Theme CSS](#)

Migrate Application Source to Release 9.2.0

To migrate your Oracle JET application source from version 5.x.0 through version 8.3.0 to the latest version 9.2.0, you must upgrade npm packages, update theme and library reference paths, and replace the `path_mapping.json` file. Additionally, you must update the `oraclejetconfig.json` file to enable Alta CSS support in JET release 9.2.0. Migration to the Redwood CSS theme, if desired, should be performed only after successful migration of the application source has been verified.

Before You Begin:

- Due to potential incompatibilities with releases prior to JET version 9.0.0, Oracle strongly recommends that you audit your application with Oracle JET Audit Framework (JAF). The built-in audit rules provided with JAF will help you to identify and fix invalid functionality, including deprecated components and APIs. Implement the audit results with some attention to detail to ensure a successful migration to JET release 9.2.0.

As Administrator on Windows or using `sudo` as needed on Macintosh and Linux systems, enter the following command in a terminal window:

```
npm install -g @oracle/oraclejet-audit
```

On your application root, run the following JAF commands to audit your application.

```
ojaf --init  
ojaf
```

For details about JAF, see Initialize Oracle JAF and Run an Audit in *Using and Extending the Oracle JET Audit Framework*.

- A minimum version of 10.0.0 of Node.js is required. Open a Command Prompt window as an administrator and check your Node.js version.

```
node -v
```

If your Node.js version isn't 10.0.0 or later, then go to the [Nodejs.org](#) website. Under **LTS Version (Recommended for Most Users)**, download the installer for your system. In the Download dialog box, choose a location for the file and click **Save**. Run the downloaded installer as an administrator and follow the steps in the installation wizard to install Node.js.

- Starting in JET release 9.0.0, applications configured for TypeScript development require a local installation of the TypeScript library to perform compilation. To ensure that your application has a local installation, run the following command from the root of your application.

```
ojet add typescript
```

To migrate your application:

1. Remove the existing version of the ojet-cli tooling package and install the latest version.

As Administrator on Windows or using sudo as needed on Macintosh and Linux systems, enter the following commands in a terminal window:

```
[sudo] npm uninstall -g ojet-cli  
npm install -g @oracle/ojet-cli@~9.2.0
```

2. Enter the following commands to change to the application's top level directory and upgrade local npm dependencies:

```
cd appDir  
npm uninstall @oracle/oraclejet @oracle/oraclejet-tooling  
npm install @oracle/oraclejet@~9.2.0 @oracle/oraclejet-  
tooling@~9.2.0 --save
```

3. In the application's src directory, replace any hardcoded references to a previous version.

- a. Edit the src/index.html file and replace the existing CSS reference with version v9.2.0.

```
<link rel="stylesheet" href="css/libs/obj/v9.2.0/alta/obj-alta-  
min.css" id="css" />
```

- b. Search for other hardcoded references to a previous release version that may appear in .html and .js files and replace those with the current release version.
4. At a terminal prompt, create a temporary application with the navdrawer template specified to obtain the files that you will copy to your migrating application.

```
ojet create tempApp --template=navdrawer
```

- b. Search for other hardcoded references to a previous release version that may appear in .html and .js files and replace those with the current release version.
5. Update the Oracle JET library configuration paths to reference the 9.2.0 versions of Oracle libraries by copying the path_mappings.json file from the temporary application.
 - a. Rename your migrating application's src/js/path_mapping.json as migrating-path_mapping.json.
 - b. Copy tempApp/src/js/path_mapping.json to your migrating application's src/js directory.
 - c. If you added any third-party libraries to your application, open path_mapping.json for editing and add an entry for each library that appears in migrating-path_mapping.json, copying an existing entry and modifying as needed. The code sample below shows the addition you might make for a third-party library named my-library.

```
"libs": {  
  
    "my-library": {  
        "cdn": "3rdparty",  
        "cwd": "node_modules/my-library/build/output",  
        "debug": {  
            "src": "my-library.debug.js",  
            "path": "libs/my-library/my-library.debug.js",  
            "cdnPath": ""  
        },  
        "release": {  
            "src": "my-library.js",  
            "path": "libs/my-library/my-library.js",  
            "cdnPath": ""  
        }  
    },  
    ...  
}
```

- b. Search for other hardcoded references to a previous release version that may appear in .html and .js files and replace those with the current release version.
6. Update the application script templates by copying from the temporary application.
 - a. Copy any new script template files from the tempApp/scripts/hooks directory to your migrating application's scripts/hooks directory.
 - b. Copy the hooks.json scripting configuration file from the tempApp/scripts/hooks directory to your migrating application's scripts/hooks directory. The updated configuration file associates any new script template files with their corresponding build system hook point and allows the Oracle JET CLI to call your scripts.
7. In your application, open the main.js application bootstrap file and verify that it contains the following code.

- a. Verify that the private function `_ojIsIE11` appears. If the function is missing in `main.js`, add it above the `requirejs.config` definition, as shown.

```
(function () {

    function _ojIsIE11() {
        var nAgt = navigator.userAgent;
        return nAgt.indexOf('MSIE') !== -1 || !nAgt.match(/Trident.*rv:11./);
    };
    var _ojNeedsES5 = _ojIsIE11();

    requirejs.config(
    {
        ...
    });
}());
```

This function detects whether the application is running in the IE11 web browser to enable loading of required polyfills and the transpiled to ES5 version of Oracle JET. If IE11 is not detected at runtime, then Oracle JET enables the default ES6-based browser support, which does not require loading of polyfill libraries.

- b. Verify that the `paths` property of the `requirejs.config` definition includes the opening and closing `//injector` and `//endinjector` comments. If the comments were removed, add them to your `requirejs.config()` definition, as shown.

```
requirejs.config(
{
    baseUrl: 'js',

    paths:
    /* DO NOT MODIFY
     * All paths are dynamically generated from the
     path_mappings.json file.
     * Add any new library dependencies in path_mappings json
     file.
    */
    //injector:mainReleasePaths
    {
        ...no need to revise...
    }
    //endinjector
}
);
```

When you build or serve the application, the tooling relies on the presence of these injector comments to inject release-specific library paths in `main.js`. The updated `path_mapping.json` file (from the previous migration step) ensures that the migrated application has the correct library configuration paths for this release.

- c. Verify that any modifications you made to `app.loadModule()` in your `main.js` file appear.

Starting in JET release 9.0.0, `app.loadModule()` is deprecated and has been removed from `main.js`, but your bootstrap code may continue to use the function, for example to change a path. Because migrating applications rely on `loadModule()`, the function should remain in your migrated `main.js` file, and your migrated `appController.js` file should contain the `loadModule()` definition.

In new applications that you create, starting in JET release 9.0.0, the `loadModule()` observable dependency is no longer used to support the deprecated `ojRouter` for use with the `oj-module` element. Starter application templates now use `CoreRouter` and work with `oj-module` through the `ModuleRouterAdapter`.

8. Open the `oraclejetconfig.json` file in your application root folder and ensure it contains the following properties and settings for `sassVer`, `defaultTheme`, `defaultCssvars`, and `generatorVersion`.

```
{
  "paths": {
    "source": {
      "common": "src",
      "web": "src-web",
      "hybrid": "src-hybrid",
      "javascript": "js",
      "typescript": "ts",
      "styles": "css",
      "themes": "themes"
    },
    "staging": {
      "web": "web",
      "hybrid": "hybrid",
      "themes": "staged-themes"
    }
  },
  "defaultBrowser": "chrome",
  "sassVer": "4.13.0",
  "defaultTheme": "alta",
  "defaultCssvars": "disabled",
  "generatorVersion": "9.2.0"
}
```

The setting "`defaultTheme": "alta`" enables your application to migrate and remain on the Alta theme. This property and the `defaultCssvars` property support migrating to the Redwood theme in a later migration process.

9. Test the migration and verify the look and feel.
- If your application uses a custom theme, run `ojet add sass` to enable Sass processing.
 - Run `ojet build` and `ojet serve` with appropriate options to build and serve the application.

For a list of available options, enter the following command at a terminal prompt in your application's top level directory: `ojet help`.

If your application uses a custom theme, be sure to include the `--theme` option to regenerate the CSS:

```
ojet build [options] --theme=myTheme
```

To specify multiple custom themes, use:

```
ojet build [options] --theme=myTheme --  
themes=myTheme,myTheme1,myTheme2
```

10. Optional: When you are satisfied that your application has migrated successfully, remove the temporary application and delete the renamed `migrating-path_mapping.json` and `migrating-main.js` files from your migrated application. Should you find issues, you can re-run the JAF audit tool for an audit report.

After you migrate your application and ensure it runs with the Alta theme, you can follow an additional, separate process to migrate to the Redwood theme, as described in [Migrate to the Redwood Theme CSS](#).

Migrate a v9.x.0 Application to v9.2.0

When upgrading to a minor release version of Oracle JET, within the same major release version, such as from 9.0.0 and 9.1.0 to 9.2.0, features changes do not introduce backward incompatibility from the previous release. In this case, you need only update the Oracle JET command-line interface (CLI), and you can leave npm unchanged. In the migrating application, you will update the CDN paths in the `path_mappings.json` file. Finally, you can ensure that you have the latest version of the required Oracle JET files installed by running the `ojet strip` and `ojet restore` commands on the application root.

1. Update the Oracle JET CLI, by using this command.

```
npm update -g @oracle/ojet-cli
```

2. Migrate your application to use Oracle JET Tooling for the release, by running these commands on your application root.

```
npm uninstall @oracle/oraclejet @oracle/oraclejet-tooling  
npm install @oracle/oraclejet@~9.2.0 @oracle/oraclejet-  
tooling@~9.2.0 --save
```

3. Edit the migrating application's `src/js/path_mapping.json` file to use the current release version number in these CDN paths near the top of the file.

```
{  
  "baseUrl": "js",  
  "use": "local",  
  
  "cdns": {  
    "jet": "https://static.oracle.com/cdn/jet/9.2.0/default/js",  
    "css": "https://static.oracle.com/cdn/jet/9.2.0/default/css",  
  },  
}
```

```
        "config": "bundles-config.js"
    },
    "3rdparty": "https://static.oracle.com/cdn/jet/9.2.0/3rdparty"
},
...

```

4. On the application root of the migrating application, run the following command to remove non-source files (dependencies, plugins, libraries, and so on) installed from the previous version.

```
ojet strip
```

5. Also on the application root, run the following command to reinstall all required non-source files that are specific to the current version.

```
ojet restore
```

Migrate to the Redwood Theme CSS

Redwood theme is the new Oracle JET standard for application look and feel and is available starting in JET release 9.0.0 when you want to migrate to the Redwood theme.

If you have an existing application that you want to migrate from the Alta theme, you can migrate to JET release 9.2.0 and configure the application to run with the Redwood CSS included with Oracle JET. You obtained the Redwood CSS distribution when you completed the application source migration process.

Migrating your application's Alta theme to Redwood theme requires making a change at the application level. You edit the `oraclejetconfig.json` file to control whether JET Tooling builds with the Redwood or Alta CSS. With the setting configured, you can rebuild your application and all the pages will use the appropriate CSS, as specified by the stylesheet injector in your application's `index.html` file.

After you set the Redwood theme as the new default and run your application, you will find the look and feel of your application changes considerably. To adjust to the Redwood theme, you will need to make manual updates to application layout for new fonts, sizes, and patterns.

Before You Begin:

- Complete migration of your application source files before attempting to migrate to the Redwood theme. First migrate with the Alta theme preserved and then migrate to the Redwood theme. This way you can test your application with the Redwood theme and easily revert back to the Alta theme, if desired. See [Migrate Application Source to Release 9.2.0](#) for details.
- If you use a custom theme, review the [Theme Changes](#) section in the release notes and update your custom theme manually.

Be aware that Sass variables that you may have overridden in an Alta theme will need to be migrated to CSS variables in the Redwood theme. For more information about migrating a custom theme, please see [About CSS Variables and Custom Themes in Oracle JET](#).

- Review application images and consider how you will replace images that belong to the deprecated Oracle JET framework images library with public domain

images, such as those found on [Oracle Apex Universal Theme](#) and on [Font Awesome](#). The Oracle JET framework image classes are no longer shown in JET Cookbook starting in release 9.0.0.

To migrate to the Redwood theme CSS:

1. Configure the application to load the Redwood CSS.

Edit the `<app_root>/oraclejetconfig.json` file and change the property **defaultTheme** to **redwood**.

```
{  
    "paths": {  
        ...  
    },  
    "defaultBrowser": "chrome",  
    "sassVer": "4.13.0",  
    "defaultTheme": "redwood",  
    "generatorVersion": "9.2.0"  
}
```

This configures JET Tooling to inject `oj-redwood-min.css` into the stylesheet link in your application's `index.html` file.

2. Test the migration and verify the look and feel.

Run `ojet build` and `ojet serve` with appropriate options to build and serve the application.

For a list of available options, enter the following command at a terminal prompt in your application's top level directory: `ojet help`.

If your application uses a custom theme, be sure to include the `--theme` option to regenerate the CSS:

```
ojet build [options] --theme=myTheme
```

To specify multiple custom themes, use:

```
ojet build [options] --theme=myTheme --  
themes=myTheme,myTheme1,myTheme2
```

Alta Theme in Oracle JET v9.0.0 and Later

Oracle recommends that you use the Redwood theme when you create new applications. However, you can migrate your existing application to JET 9.0.0 and continue to work with the Alta theme.

Topics:

- [Consider Using the Redwood Theme in Applications](#)
- [CSS Files Included in Alta](#)
- [Understand the Color Palette in Alta](#)
- [Customize Alta Themes Using the Tooling](#)
- [Work with Sass](#)
- [Work with Icon Fonts in Alta](#)
- [Work with Image Files in Alta](#)
- [Use Tag Selectors or Style Classes with Alta](#)
- [Use Normalize with Alta](#)

Consider Using the Redwood Theme in Applications

Starting in JET 9.0.0, when you build a new JET application, Oracle recommends that you use the new Redwood theme to benefit from Oracle Redwood Design System. JET Tooling supports the Redwood theme as the default choice.

Oracle JET is committed to the Oracle standard, Oracle Redwood Design System. For example, starting with JET release 9.0.0, certain new UX features will only be available in the Redwood theme. These features, such as Waterfall Layout and Stream List, illustrate the rich UX supported by Oracle Redwood Design System.

Important:

For details about how you can use the Redwood theme to create new applications or to migrate an existing Alta-themed application, see [Typical Workflow for Working with Themes in Oracle JET Applications](#).

Note that JET Tooling generates different sets of theme definition files to support either Redwood theme or Alta theme. These definition files contain an extensive set of CSS custom properties to enable component style class overrides that safely abstract your customizations away from the styles implemented by components for a particular theme. In Alta, these custom properties are Sass variables, whereas, in Redwood, they are CSS variables. The use of style variables to customize a theme ensures that style changes you make can be migrated to a later JET release without overriding style changes by the component itself.

CSS Files Included in Alta

With the Alta theme, Oracle JET includes CSS files designed for display on web browsers and hybrid mobile applications. Each theme includes minified and readable versions of the CSS and source maps.

The Alta theme is based on Oracle Alta UI, a mobile and browser application design system. There are two types of themes included in Oracle JET, both based on [Oracle Alta UI](#):

- `alta` web theme

Web themes are designed to be used in a browser on all platforms, and the same theme can be used regardless of whether you are looking at a web page on a desktop Firefox, Android Chrome, or iOS Safari browser.

- `android`, `ios`, and `windows` hybrid mobile themes for Android, iOS, and Windows

Hybrid themes are designed to be used with Cordova to create a hybrid mobile application. The colors use Oracle's Alta look and feel, but otherwise these themes try to match the look and feel of a native mobile application.

The Alta CSS included with the Oracle JET distribution are generated by the [Sass](#) preprocessor and include the following files:

- `oj-alta*.css`: Readable version of the CSS
- `oj-alta*-min.css`: Minified version of the CSS
- `oj-alta*.css.map`: CSS source map

In addition, the Alta web theme includes the following generated CSS:

- `oj-alta-notag.css`: Readable version of the CSS generated without tag selectors
For additional details about Oracle JET theming and tag selectors, see [Use Tag Selectors or Style Classes with Alta](#).
- `oj-alta-notag-min.css`: Minified version of the CSS generated without tag selectors.

If the CSS files provided by Oracle JET with your application are sufficient and you only want to add a few application-specific styles, you may find that adding the classes to `app.css` in your application's `src/css` folder will meet your needs.

If, however, you want to use a different theme or add more than a few application-specific classes, then you can use Oracle JET Tooling to generate your own CSS. For instructions, see [Customize Alta Themes Using the Tooling](#).

Use the recommended standards to generate your CSS and Themes. For more information, see [Best Practices for Using CSS and Themes](#).

! Important:

Do not override Oracle JET CSS. Oracle JET's CSS is considered private and can change at any time.

Understand the Color Palette in Alta

Oracle JET defines four sets of colors including branding ramps, neutral ramps, text colors, and accent colors. Each set supports the Alta look and feel, but you can customize them for your needs.

The four color sets include:

- Branding ramp: a progressive set of colors generated by tinting (mixture of a color with white) and shading (mixture of a color with black) a primary brand color (\$brandColor).

In the Oracle look and feel the branding colors are blue.

- Neutral ramp: a progressive set of neutral colors based on the same hue (\$neutralHue) and saturation (\$neutralSaturation), but with varied levels of lightness.

In the Oracle look and feel the neutral colors are gray.

- Text colors: text colors created from a base color of #000 with various opacities.
Text with opacity looks better on colored backgrounds.
- Accent colors: special colors that are used in things like messaging, drag and drop, and data visualization components.

To look at examples of the different color sets and associated variables, see [Alta Theme Builder](#).

Customize Alta Themes Using the Tooling

Use the Oracle JET command-line interface (CLI) to add Sass support and customize your application's Alta theme or specify an alternate theme.

By default, the SCSS files are not included with applications that you scaffold with the Oracle JET CLI. However, you can use tooling commands as described in the procedure below to add node-sass and skeleton theme files to your application. You can then modify the skeleton theme files while the application is running and observe the effect of the change immediately using the live reload feature.

Before you begin:

- Scaffold an application. See [Scaffold a Web Application](#) or [Scaffold a Hybrid Mobile Application](#).
- If you want to modify your directory structure, see [Modify the Web Application's File Structure](#) or [Modify the Hybrid Mobile Application's File Structure](#).
- Learn about Oracle JET and SCSS variables. See [SCSS Variables](#).
- Learn about Oracle JET's use of tag selectors. See [Use Tag Selectors or Style Classes with Alta](#).
- Learn about using variables to include only the CSS you need. See [Use Variables to Control CSS Content](#).

To customize an Oracle JET theme using the tooling:

- 1.** In your application's top level directory, enter the following command at a terminal prompt to add node-sass to your application:

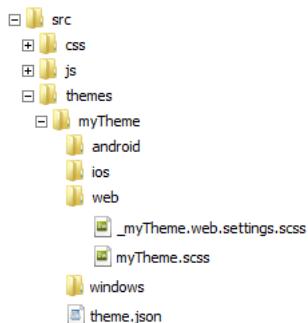
```
ojet add sass
```

- 2.** Add the skeleton theme files to your application.

```
ojet create theme themeName
```

For example, enter the following command to create the skeleton theme files for a custom theme named myTheme: ojet create theme myTheme.

The command creates a new folder with the custom theme name in the application's `src` directory.



In the figure above, the `web` folder is expanded to show the `.scss` files that you can modify. The `myTheme.scss` is the aggregating file for the given platform. The `android`, `ios`, and `windows` folders contain similar files with the name of the platform in the title: `_myTheme.android.settings.scss`, `_myTheme.web.settings.scss`, and so on.

You'll find out more about modifying these files in a later step. If you modified your file structure, the file locations will vary.

The `theme.json` file contains the version number of the theme, starting with `0.0.1`.

- 3.** Use `ojet build` with the `--theme` option to build a development version of your application with the new theme.

```
ojet build --theme=themeName[:android|ios|web|windows] [--platform=android|ios|web|windows]
```

For example, the following command builds a web application with the `myTheme` custom theme.

```
ojet build --theme=myTheme
```

- 4.** To build more than one theme, add the `--themes` option with the names of the desired themes separated by commas:

```
ojet build --theme=myTheme --themes=myTheme,myTheme1,myTheme2
```

5. To run your application with live reload enabled, enter `ojet serve` with the `theme` option and optional platform.

```
ojet serve --theme=themeName[:android|ios|web|windows] [--  
platform=android|ios|web|windows]
```

For example, the following command serves a web application with the `myTheme` custom theme in the default browser.

```
ojet serve --theme=myTheme
```

6. Edit the `themeName.platform.settings.scss` file to set variables as needed for your application.

For example, to turn off tag selectors, remove the comment from the `$allowTagSelectors` variable if needed and set it to `false`.

```
$allowTagSelectors: false;
```

When you use the Oracle JET CLI to serve your application, live reload will reload the page, and any changes you make to your theme files will display in the browser.

7. To optimize performance, consider the use of variables to include only the CSS that your application needs. Use Oracle JET live reload to observe the effect of any changes immediately, if you're not sure whether your application needs a given variable.
8. To exit the application, save your changes, and press `Ctrl+C` at the terminal prompt.

To make working with themes easier, Oracle JET provides [Alta Theme Builder](#), an example application and tutorial. You can work with it online to see the effect of changes before you apply them to your own application. You can also download the sample and create your own themes offline for sharing.

Work with Sass

Oracle JET provides its own Sass mixins and uses Sass tools `postcss-calc`, `postcss-custom-properties`, `autoprefixer` and `node-sass`. When you use the `ojet add theming` command, the JET Tooling adds these to the tooling chain for compilation by the PostCSS tool. Sass is used for bundling and there are a few cases, for example in media queries, where CSS variables aren't supported and SCSS variables are needed.

Topics

- [SCSS Variables](#)
- [SCSS File Organization and Naming Convention](#)
- [Use Variables to Control CSS Content](#)

When working with the Alta theme, Oracle JET provides its own Sass mixins and doesn't use the Sass tools used by the Redwood tooling.

Only when working with the Alta theme, Oracle JET supports Sass source maps to work with a large number of partial files. The toolkit includes source maps for

the default and no tag selector versions of the generated CSS for the Alta theme: `oj-alta.css.map`, and `oj-alta-notag.css.map`.

For additional information about source maps, see <https://developers.google.com/chrome-developer-tools/docs/css-preprocessors>.

SCSS Variables

If you use Sass to generate your own styles and don't rely on any JET Sass variables then you can continue using Sass as before. However if you rely on JET SCSS variables, then most of those SCSS variables will no longer be available starting in JET release 9.0.0.

The SCSS variables that Oracle JET uses to generate the application use the same naming convention and are grouped together according to function. Oracle JET also uses SCSS variables to default some component options, such as the `oj-button chroming` option.

Variable Naming Convention

Oracle JET variables names use a camel case naming structure:
`<widgetName><Element><Property><State>`.

Type	Description	Examples
widgetName	Component or pattern name	dataGrid, button
Element (Optional)	Component subsection	Cell, Header
Property	Property affected	BgColor, BorderColor, TextColor
State (Optional)	Widget's state when applied	Hover, Active, Focus, Selected, Disabled

Using this convention, the following are all valid Oracle JET SCSS variables: `$buttonHeight`, `$sliderThumbWidth`, `$dataGridCellBgColorHover`, `$treeNodeBorderColorSelected`

Variable Documentation

Variable names are typically self-documenting. In cases where additional documentation is needed, the SCSS file will contain comments above the variable definition.

Variable Categories

The following table lists the high level variable categories.

Category	Description
Logistical	Affects logistics, such as the path to an image directory or text direction.
Performance	Set these variables to include only the CSS your application uses.
Palette	Application's color palette. Examples include <code>\$interaction1Color</code> , <code>\$background1Color</code> , and <code>\$border1Color</code> .

Category	Description
Text	Controls text appearance. Widget files use these variables extensively. Examples include \$fontFamily, \$rootFontSize, \$textColor, and \$linkTextColor.
General	Sets general theme properties such as drag and drop colors, border radius defaults, and animation duration defaults.
Widget	Affects specific widgets or widget groups, such as button or form control variables. Some component default options are included here. For example, you can change the <code>ojButton</code> component's chroming option using <code>\$buttonChromingOptionDefault</code> .

SCSS File Organization and Naming Convention

Oracle JET follows the Sass underscore naming convention. Files that start with an underscore are called partials and are not compiled independently with Sass. Instead, the partials are imported into an aggregating file. There is one partial file per module, for example, `_oj.table.scss`, `_oj.formcontrol.inputnumber.scss`, and so on.

The following table lists the directory structure for the files contained within the `alta` directory.

File	Description
<code>widgets/_oj.alta.formcontrol.inputnumber.scss</code>	Alta widget partial files
<code>widgets/_oj.alta.table.scss</code>	
<code>etc.</code>	
<code>_oj.alta.variables.scss</code>	Contains the SCSS variables
<code>_oj.alta.settings.scss</code>	Contains a commented version of the variables file that you can remove the comments from and modify
<code>_oj.alta.widgets.scss</code>	Imports widget partial files
<code>oj-alta.scss</code>	Generates the <code>oj-alta.css</code> and <code>oj-alta-min.css</code> files
<code>oj-alta-notag.scss</code>	Generates the <code>oj-alta-notag.css</code> and <code>oj-alta-notag-min.css</code> files

Use Variables to Control CSS Content

You can use the `$includeAllClasses` variable in your SCSS settings file to control content for the entire application. By default, the variable is set to true as shown below.

```
// by default everything is included, but you can also start by setting
// $includeAllClasses to false and then just add in what you want.
$includeAllClasses: true !default;
```

Some partials also allow you to control their content with a variable. For example, the `ojButton` component has the `$includeButtonClasses` variable near the top of the settings file.

```
$includeButtonClasses: $includeAllClasses !default;
```

To exclude the `ojButton` style classes, you can set `$includeButtonClasses` to `false` as shown below.

```
$includeButtonClasses: false;
```

Oracle JET also uses several higher level groupings that let you control the CSS for a logical group of components. The table lists the groups that are available for your use.

Group	SCSS Variable	File Names
Tags (headers, links, and so on)	<code>\$includeTagClasses: \$includeAllClasses !default;</code>	Files with tag in name, as in: <code>_oj.alta.tags.typography</code>
Data visualizations (charts, gauges, and so on)	<code>\$includeDvtClasses: \$includeAllClasses !default;</code>	Files with dvt in name, as in: <code>_oj.alta.dvt.chart</code>
Form controls (labels, combo box, and so on)	<code>\$includeFormControlClasses: \$includeAllClasses !default;</code>	Files with formcontrol in name, as in: <code>_oj.alta.formcontrol.label</code>

You can include or exclude classes and groups as shown in the following examples.

```
// Example: Exclude the dvt classes
$includeDvtClasses: false;

// Example: Include only the dvt classes, exclude everything else
$includeAllClasses: false;
$includeDvtClasses: true;

// Example: Include the chart and sunburst classes, exclude everything else
$includeAllClasses: false;
$includeChartClasses: true;
$includeSunburstClasses: true;
```

 **Note:**

Some components depend on others. For example, the `ojInputNumber` uses `ojButton` internally, so if you include the `ojInputNumber` classes, you will also get the `ojButton` classes automatically.

Work with Icon Fonts in Alta

Oracle JET uses icon fonts whenever possible because icon fonts have certain advantages over other formats.

- Themable: You can use style classes to change their color instead of having to replace the image, making them very easy to theme.

- High contrast mode: Icon fonts are optimal for high contrast mode as they are considered text. However, keep in mind that you can't rely on color in high contrast mode, and you may need to indicate state (active, hover, and so on) using another visual indicator. For example, you can add a border or change the icon font's size. For additional information about Oracle JET and high contrast mode, see [Configure High Contrast Mode](#).
- High resolution: Icon fonts look good on a high resolution (retina) display without providing alternate icons.
- Performance: You can change icon font colors using CSS so alternate icons are not required to indicate state changes. Alternate images are also not required for high resolution displays.

Icon fonts also have disadvantages. It can be difficult to replace a single image, and they only show one color. You can use text shadows to provide some depth to the icon font.

Oracle JET supports two generic classes for setting the icon font colors in the following Alta

SCSS: \$iconColorDefault, \$iconColorHover, \$iconColorActive, \$iconColorSelected, and \$iconColorDisabled.

- oj-clickable-icon
- oj-clickable-icon-nocontext

These classes, when used in conjunction with an anchor tag and/or marker classes like oj-default, oj-hover, oj-focus, oj-active, oj-selected, and oj-disabled, will use the variables \$iconColor*.

The oj-clickable-icon class is optionally contextual, meaning the anchor or marker style can be on an ancestor, as shown in the example below. The example assumes that a JavaScript method is replacing oj-default as needed with oj-hover, oj-focus, oj-active, oj-selected, and oj-disabled.

```
<div class="oj-default">
  <span class="oj-clickable-icon demo-icon-font demo-icon-gear"></span>
</div>
<a href="http://www.oracle.com">
  <span class="oj-clickable-icon demo-icon-font demo-icon-gear"></span>
</a>
```

The oj-clickable-icon-nocontext class is not contextual and must be placed on the same tag as shown in the example below. oj-clickable-icon would also work in this example.

```
<span class="oj-default oj-clickable-icon-nocontext demo-icon-font demo-icon-gear">
</span>
<a href="http://www.oracle.com"
  class="oj-clickable-icon-nocontext demo-icon-font demo-icon-gear">
</a>
```

For an example that illustrates icon font classes on a link, see [Icon Fonts](#).

Work with Image Files in Alta

Oracle JET uses images when icon fonts can't be used. Informational images must appear in high contrast mode for accessibility, as described in [Configure High Contrast Mode](#).

In the Alta theme, you can replace any of the SVG images in the following application directories. Note that your new image must use the same name as the original image.

```
staged-themes/alta/android/images  
staged-themes/alta/common/images  
staged-themes/alta/ios/images  
staged-themes/alta/web/images  
staged-themes/alta/windows/images
```

To improve performance, the SVG images used by JET components are sprited. JET does not support creating custom image sprites, and you must use a third-party tool to create them. However, if you use Oracle JET tooling, you can overwrite some of the default SVG images and use the tooling to combine them into a single `sprite.svg` file.

When you build your application using `ojet build` or `ojet serve` commands, JET will automatically combine the SVG images into the `sprite.svg` file in the appropriate `themes/alta/*/images/sprites/` directory.

 **Note:**

Starting in release 9.1.0, JET does not install the node module `svg-sprite` by default when you scaffold a new application with the JET CLI. As a result, if your application will contain modified JET SVG image files, the JET CLI will issue a build warning and the application will fail to build. To support working with SVG images in Alta theme-based applications you can manually install the `svg-sprite` module by running this command from the application root directory:

```
npm install svg-sprite
```

You can also use custom images that are not sprited using CSS, but the improved performance gained by using sprited images will be lost. To do so, you must override the image style classes with `{background-position: 0% 0%;}`.

Oracle JET provides Sass mixins that you can use to create CSS for your own icons. For examples, see [CSS Images](#).

Use Tag Selectors or Style Classes with Alta

In Alta themed applications that you create, by default Oracle JET applies styles directly to HTML tag elements such as `a`, `h1`, `h2`, and so on. This feature makes it easier to code a page since you do not have to apply selectors to each occurrence of the element.

The following code shows a portion of a custom SCSS file that defines styles for link and header tags. When the CSS file is generated for the Alta theme, the styles will be applied directly to the link and header tags.

```
/* links */
a {
    color: $linkTextColor;
    line-height: inherit;
    text-decoration: none;
}

a:visited {
    color: $linkTextColorVisited;
}

/* headers */
h1, h2, h3, h4 {
    color: $headerTextColor;
    font-family: inherit;
    font-style: normal;
    font-weight: $headerFontWeight;
    margin: 10px 0;
}
```

If you do not want to apply styles directly to the tags, you can specify that Oracle JET use classes instead of tag selectors in your `custom.scss` file:

```
$allowTagSelectors: false;

// theme import
@import ".../oj.redwood/oj-redwood";
```

The code below shows the classes that Oracle JET will use for the Alta theme's links and headers when you set `$allowTagSelectors` to `false`. To use the class on your page, specify the class in the tag element on your page (``).

```
/* links */
.oj-link {
    color: $linkTextColor;
    line-height: inherit;
    text-decoration: none;
}

.oj-link:visited {
    color: $linkTextColorVisited;
}

/* headers */
.oj-header {
    color: $headerTextColor;
    font-family: inherit;
    font-style: normal;
    font-weight: $headerFontWeight;
    margin: 10px 0;
}
```

The following table lists the HTML tags with default Oracle JET tag styles and the corresponding Oracle JET class.

HTML Tag	Oracle JET Style Class
html	oj-html
body	oj-body
a	oj-link
h1, h2, h3, h4	oj-header
hr	oj-hr
p	oj-p
ul, ol	oj-ul, oj-ol

If you also do not want to use the Oracle JET tag classes, you can set `$includeTagClasses` to `false` in your `custom.scss` file as shown below.

```
$includeTagClasses:false;
```

Use Normalize with Alta

In applications that you created before Oracle JET release 9.0.0, you may use `normalize.css` to promote consistency when rendering an Alta-themed application across browsers. If your Alta-themed application uses `normalize.css`, add the import in your `custom.scss`:

```
@import ".../3rdparty/normalize/normalize";
```

If you do not want to use `normalize`, you can set the following variable to `false` in your `custom.scss`:

```
$includeNormalize: false;
```

Alternatively, you can set the Alta SCSS variable `$allowTagSelectors` to `false` as described in [Use Tag Selectors or Style Classes with Alta](#).

For additional information about `normalize.css`, see <http://necolas.github.io/normalize.css>.

D

Oracle JET References

Oracle JET includes third-party libraries and tools that are referenced throughout the guide. Oracle also provides optional tools and libraries to assist with application development.

Topics:

- [Oracle Libraries and Tools](#)
- [Third-Party Libraries and Tools](#)

Oracle Libraries and Tools

Oracle provides optional tools and libraries to use in conjunction with Oracle JET. Use this reference to locate additional information about the Oracle products referenced throughout the guide.

Name	Description	Additional Information
Voluntary Product Accessibility Template (VPAT)	Documentation of Oracle's degree of conformance with applicable accessibility standards	Voluntary Product Accessibility Template (VPAT)
CSS Variable Theme Builder	An interactive demo application that you can use to try out custom Redwood themes by downloading the sample application.	CSS Variable Theme Builder
SCSS Variable Theme Builder	An interactive demo application that you can use to try out custom Alta themes by downloading the sample application.	Alta Theme Builder

Third-Party Libraries and Tools

Use this reference to locate additional information about the third-party libraries and tools used by Oracle JET and referenced throughout the guide.

Name	Description	Additional Information
Apache Cordova (Mobile only)	Open source mobile development framework that allows you to use HTML5, CSS3, and JavaScript for cross-platform development targeted to multiple platforms with one code base	http://cordova.apache.org/
CSS3 (Cascading Style Sheets)	Used for adding style to Web applications	http://www.w3.org/Style/CSS
Hammer.js	JS library used for multi-touch gestures	http://hammerjs.github.io/getting-started
HTML5 (Hypertext Markup Language 5)	Core language of the World Wide Web	http://www.w3.org/TR/html5

Name	Description	Additional Information
JavaScript	Scripting language used in client-side applications	https://developer.mozilla.org/en/Auto_JavaScript
js-signals	JS library used for custom event/messaging system	http://millermedeiros.github.io/js-signals
jQuery	JS library designed for HTML document traversal and manipulation, event handling, animation, and Ajax	http://jquery.com
jQuery UI	JS library built on top of jQuery for UI development. Oracle JET includes the UI Core download only.	http://www.jqueryui.com
Knockout	JS library used for two-way data binding	http://www.knockoutjs.com
Visual Studio Code	Microsoft Integrated Development Environment (IDE) with available Oracle JET Core Components extension for application development.	https://marketplace.visualstudio.com/items?itemName=Oracle.oracle-jet-core
Node.js	Open source, cross-platform runtime environment for developing server-side web applications, used by Oracle JET for package management.	https://nodejs.org
proj4js	JavaScript library designed for transforming point coordinates from one coordinate system to another, including datum transformations.	http://proj4js.org/
QUnit	JavaScript unit testing framework	http://qunitjs.com
RequireJS	JS file and module loader used for managing library references and lazy loading of resources	http://www.requirejs.org
RequireJS CSS	RequireJS CSS plugin that allows to load CSS files.	http://requirejs.org/docs/faq-advanced.html
SASS (Syntactically Awesome Style Sheets)	Extends CSS3 and enables you to use variables, nested rules, mixins, and inline imports	http://www.sass-lang.com
Selenium WebDriver	Alternative method of testing applications	http://docs.seleniumhq.org/projects/webdriver
