## Contents

### Preface

<table>
<thead>
<tr>
<th>Audience</th>
<th>xiii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation Accessibility</td>
<td>xiii</td>
</tr>
<tr>
<td>Related Resources</td>
<td>xiii</td>
</tr>
<tr>
<td>Conventions</td>
<td>xiv</td>
</tr>
</tbody>
</table>

### 1 Get Started

<table>
<thead>
<tr>
<th>What’s New</th>
<th>1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop Websites, Content Delivery, Content Management, and Collaboration</td>
<td>1-3</td>
</tr>
<tr>
<td>Typical Development Tasks</td>
<td>1-4</td>
</tr>
</tbody>
</table>

### 2 Integrate Oracle Content and Experience Cloud

<table>
<thead>
<tr>
<th>Reusable Web Interface</th>
<th>2-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy Service Settings</td>
<td>2-1</td>
</tr>
<tr>
<td>Oracle Process Cloud Service Integration</td>
<td>2-1</td>
</tr>
<tr>
<td>Configure Oracle Integration Settings in Oracle Content and Experience Cloud</td>
<td>2-2</td>
</tr>
<tr>
<td>Process Cloud Service Integration with Sites</td>
<td>2-3</td>
</tr>
<tr>
<td>Visual Builder Cloud Service Apps</td>
<td>2-3</td>
</tr>
<tr>
<td>Embed a VBCS Visual App in an Oracle Content and Experience Cloud Page</td>
<td>2-5</td>
</tr>
<tr>
<td>Embed a VBCS Page in a Site Page</td>
<td>2-13</td>
</tr>
<tr>
<td>Build an Oracle Content and Experience Cloud VBCS Public Form Component</td>
<td>2-17</td>
</tr>
<tr>
<td>Build an Oracle Content and Experience Cloud VBCS Secure Form Component</td>
<td>2-27</td>
</tr>
<tr>
<td>Build an Oracle Content and Experience Cloud VBCS Public Gated Form Component</td>
<td>2-39</td>
</tr>
<tr>
<td>Build an Oracle Content and Experience Cloud VBCS Data Report Component</td>
<td>2-51</td>
</tr>
<tr>
<td>Build an Oracle Content and Experience Cloud VBCS Multipage Form As a Web App Component</td>
<td>2-55</td>
</tr>
<tr>
<td>Provide a VBCS Endpoint As a URL for Select Menus</td>
<td>2-62</td>
</tr>
<tr>
<td>Custom Actions</td>
<td>2-64</td>
</tr>
<tr>
<td>REST Application Programming Interface</td>
<td>2-64</td>
</tr>
</tbody>
</table>
Develop with Oracle Content and Experience Toolkit

Set Up Oracle Content and Experience Toolkit on Your Local Machine
  Install Dependencies Through npm
  Do a One-Time Gradle Setup
  Use the cec Command-Line Utility
  Test with a Local Test Harness

Develop for Oracle Content and Experience Cloud with Developer Cloud Service
  About Using Developer Cloud Service
  Sign in to the Developer Cloud Service Console for Oracle Content and Experience Cloud
  Create a Project in Developer Cloud Service
    Create a Developer Cloud Service Project with an Oracle Content and Experience Cloud Template
    Create a Project in Developer Cloud Service with a Toolkit Download from Oracle Content and Experience Cloud
  Add Oracle Content and Experience Toolkit to the Project Code in the New Git Repository
  Test Custom Components, Templates, and Content Layouts in a Local Test Harness
  Merge Changes

Index Site Pages with Oracle Content and Experience Toolkit
  Create the Content Type for Site Page Text
  Create Page Index Content Items with Oracle Content and Experience Toolkit
  Add Content Search to a Site in Oracle Content and Experience Cloud
    Add a Search Page to the Site
    Add a Search Field to the Theme

Create a New Site or Asset Translation Job in the Oracle Content and Experience Cloud Server

Translate a Site with a Language Service Provider
  Create a Translation Job with Oracle Content and Experience Toolkit
  List Translation Jobs
  Download a Translation Job
  Create a Translation Connector
  Submit a Translation Job to a Language Service Provider
  Upload a Translation Job to the Server

Index a Multilingual Site with Oracle Content and Experience Toolkit

Generate a Site Map for a Multilingual Site

Create a Simplified Component for Easy Component Development
## 4 Starter Site CLI for React Development

- Install the Starter Site CLI 4-1
- Run CLI Commands 4-2
- Get Content from Oracle Content and Experience Cloud 4-3
- Set Up the Oracle Content and Experience Cloud Server Connection 4-3
- Create a Site 4-3
- Build a Site 4-5
- Run a Site in Development Mode 4-5
- Run a Site with Oracle Content and Experience Cloud Server Content 4-5
- Build a Site for Production 4-6
- Run a Site in Production Mode 4-6
- Structure of the React JS Site Template 4-6
- Generated Components 4-9
- Starter Site Runtime 4-10
- Build a Starter Site Template 4-11

## 5 Develop Templates

- About Templates 5-1
- Basic Template Structure 5-2
- Create a Template 5-4
- Export a Template 5-5
- Import a Template 5-6
- Work with a Starter Template 5-6
- Create a Site Template from Bootstrap or a Website Design Template 5-10
- Develop Templates with Developer Cloud Service 5-16
  - Create Templates in Developer Cloud Service 5-16
  - Copy a Template in Developer Cloud Service 5-17
  - Import a Template into Developer Cloud Service 5-17
  - Export a Template from Developer Cloud Service 5-17

## 6 Develop Themes

- About Themes 6-1
- Basic Theme Structure 6-3
- Site Navigation 6-5
- Create a Theme 6-7
- Associate Components with Themes 6-9
- Render API Reference 6-12
7  Work with Layouts

About Layouts 7-1
Search Engine Optimization (SEO) 7-3
Understand the components.json File and Format 7-5
Customize Toolbar Groups in Site Builder 7-9
Restrict Components in Slots 7-12
Make Layout Content Editable 7-14
Create a Section Layout 7-15
Create a Section Layout That Supports Lazy Load 7-18
Develop Custom Section Layouts with APIs 7-19
Develop Content Layouts 7-23
Create Content Layouts with Oracle Content and Experience Cloud 7-26
Pass a Layout View to a Content Layout 7-31
Generate a Site Details Page URL with an API 7-32
Develop Content Layouts Locally with Developer Cloud Service 7-33
Create a Content Layout with Developer Cloud Service 7-34
Define the RequireJS Module 7-35
Configure the Constructor Function Parameter 7-35
Render the Content Layout 7-36
Edit the Content Layout in the Mustache Template 7-37
Add Dynamic DOM Manipulation 7-37
Define Styles in the design.css File 7-37
Get Reference Items 7-37
Get a Media URL 7-38
Raise Triggers 7-38
Navigate to a Search Page with a Search Query 7-38
Expand Macros and Render Rich Text 7-39
Link to the Details Page 7-40
Expand Macros in Content List Queries 7-40
Create the Sample Blog Template 7-44
Add Content Layout Mappings to Templates 7-44
Test Content Layouts with the Local Test Harness 7-45
Import Templates with Content Layouts into Oracle Content and Experience Cloud 7-45

8  Work with Designs and Styles

About Designs 8-1
Design Files 8-1
Customize Conversation List Styles 8-4
Customize Folder List and File List Styles 8-6
# Work with Backgrounds

- About Background and Themes
- How Backgrounds Are Implemented
- Where Settings Are Stored

# Work with Triggers and Actions

- About Triggers and Actions
- Set Triggers
- Set Actions

# Specify Site Redirects

- Plan for Redirects
  - Simple String-to-String Matching
  - Simplified Wildcard Matching
- Specify Redirect Rules in a JSON File
- Upload a Redirect Rules File to a Site

# Develop Components

- About Components
- About Developing Components
- Create a Component
- Develop Custom Components with Developer Cloud Service
  - Develop a Custom Component for Oracle Content and Experience Cloud
    - Develop Your Custom Component
    - Write and Run Unit Tests
  - Optimize Components (Minify) for Better Performance
  - Run Continuous Integration Jobs
- Develop Translatable Components for Multilingual Sites
- Build an H1 Component with a Settings Panel
  - Create a New Local Component
  - Build the Basic H1 Component
  - Add CSS for Your Component
  - Add a Settings Panel to Change Heading Text
  - Update the Theme for Others to Pick the H1 Component Style
- Create a Simple Component Using HTML Templates
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the HTML Component</td>
<td>12-22</td>
</tr>
<tr>
<td>Add User-Supplied Data to the HTML Component</td>
<td>12-26</td>
</tr>
<tr>
<td>Compare Local Components to Remote Components</td>
<td>12-30</td>
</tr>
<tr>
<td>Render Component Settings</td>
<td>12-31</td>
</tr>
<tr>
<td>Local Component Implementation</td>
<td>12-32</td>
</tr>
<tr>
<td>Style Classes for Components</td>
<td>12-33</td>
</tr>
<tr>
<td>How to Style Built-In Components</td>
<td>12-35</td>
</tr>
<tr>
<td>Component Styling Basics</td>
<td>12-36</td>
</tr>
<tr>
<td>Component-Specific Styling</td>
<td>12-37</td>
</tr>
<tr>
<td>Set Component Properties</td>
<td>12-44</td>
</tr>
<tr>
<td>Components Rendered in Inline Frames</td>
<td>12-45</td>
</tr>
<tr>
<td>About the Instance ID and Structure for Components Rendered in Inline Frames</td>
<td>12-47</td>
</tr>
<tr>
<td>Security for Remote Components</td>
<td>12-48</td>
</tr>
<tr>
<td>Register a Remote Component</td>
<td>12-50</td>
</tr>
<tr>
<td>Delete a Component</td>
<td>12-50</td>
</tr>
<tr>
<td>Sites SDK</td>
<td>12-51</td>
</tr>
<tr>
<td>13 Improve Performance</td>
<td></td>
</tr>
<tr>
<td>Leverage Caching to Improve Performance</td>
<td>13-1</td>
</tr>
<tr>
<td>Runtime Caching</td>
<td>13-2</td>
</tr>
<tr>
<td>Site Builder Caching</td>
<td>13-3</td>
</tr>
<tr>
<td>Above the Fold (ATF) Rendering</td>
<td>13-3</td>
</tr>
<tr>
<td>14 Embed the Web User Interface</td>
<td></td>
</tr>
<tr>
<td>Embed the Web User Interface for Assets</td>
<td>14-2</td>
</tr>
<tr>
<td>Embed the Web User Interface for Documents</td>
<td>14-4</td>
</tr>
<tr>
<td>Authentication and SSO Environments</td>
<td>14-4</td>
</tr>
<tr>
<td>Real-Time Updates for Embedded Conversations</td>
<td>14-5</td>
</tr>
<tr>
<td>Embed Documents Manager on a Site Page</td>
<td>14-6</td>
</tr>
<tr>
<td>Security for Content in Other Domains</td>
<td>14-6</td>
</tr>
<tr>
<td>Embed Content in Other Domains</td>
<td>14-6</td>
</tr>
<tr>
<td>Browser Configuration Parameters</td>
<td>14-7</td>
</tr>
<tr>
<td>Embed Search Results</td>
<td>14-10</td>
</tr>
<tr>
<td>Embed Sites Content in Another Domain</td>
<td>14-11</td>
</tr>
<tr>
<td>Configure the File Picker with the SitesSDK.filePicker Method</td>
<td>14-11</td>
</tr>
<tr>
<td>15 Develop Custom Actions</td>
<td></td>
</tr>
<tr>
<td>Application Integration Framework Overview</td>
<td>15-1</td>
</tr>
<tr>
<td>Configuration File Format</td>
<td>15-4</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Application Properties</td>
<td>15-7</td>
</tr>
<tr>
<td>Action Command</td>
<td>15-7</td>
</tr>
<tr>
<td>Invoke Command</td>
<td>15-9</td>
</tr>
<tr>
<td>Presentation Command</td>
<td>15-9</td>
</tr>
<tr>
<td>Expressions</td>
<td>15-10</td>
</tr>
<tr>
<td>Variables</td>
<td>15-11</td>
</tr>
<tr>
<td>Localization</td>
<td>15-14</td>
</tr>
</tbody>
</table>

16 Oracle Content and Experience Cloud REST APIs

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of the REST API for Content Delivery</td>
<td>16-1</td>
</tr>
<tr>
<td>Overview of the REST API for Content Management</td>
<td>16-2</td>
</tr>
<tr>
<td>Overview of the REST API for Conversations</td>
<td>16-3</td>
</tr>
<tr>
<td>REST API Features for Conversations</td>
<td>16-4</td>
</tr>
<tr>
<td>API Security</td>
<td>16-4</td>
</tr>
<tr>
<td>API Versioning</td>
<td>16-5</td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td>16-5</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>16-5</td>
</tr>
<tr>
<td>Optional Fields and Default Values for JSON Data Representations</td>
<td>16-6</td>
</tr>
<tr>
<td>Return Values</td>
<td>16-6</td>
</tr>
<tr>
<td>Name Patterns</td>
<td>16-7</td>
</tr>
<tr>
<td>URL Encoding</td>
<td>16-7</td>
</tr>
<tr>
<td>Pagination</td>
<td>16-8</td>
</tr>
<tr>
<td>Filters</td>
<td>16-8</td>
</tr>
<tr>
<td>Sort Order</td>
<td>16-8</td>
</tr>
<tr>
<td>Localization</td>
<td>16-8</td>
</tr>
<tr>
<td>Name Field Length Limit</td>
<td>16-10</td>
</tr>
<tr>
<td>HTTP Status Codes and Error Handling</td>
<td>16-10</td>
</tr>
<tr>
<td>Collaboration Resource</td>
<td>16-10</td>
</tr>
<tr>
<td>Configuration Resource for Conversations</td>
<td>16-11</td>
</tr>
<tr>
<td>Security Resource</td>
<td>16-11</td>
</tr>
<tr>
<td>Social Resource</td>
<td>16-11</td>
</tr>
<tr>
<td>Overview of the REST API for Documents</td>
<td>16-12</td>
</tr>
<tr>
<td>Version History</td>
<td>16-12</td>
</tr>
<tr>
<td>REST API Features for Documents</td>
<td>16-14</td>
</tr>
<tr>
<td>REST Overview</td>
<td>16-14</td>
</tr>
<tr>
<td>Service Call URI and Version</td>
<td>16-15</td>
</tr>
<tr>
<td>Service Request</td>
<td>16-15</td>
</tr>
<tr>
<td>Service Response</td>
<td>16-16</td>
</tr>
<tr>
<td>Security</td>
<td>16-17</td>
</tr>
<tr>
<td>Sort Order</td>
<td>16-17</td>
</tr>
</tbody>
</table>
Oracle Content and Experience Cloud SDKs

Content SDK

Sites SDK Reference

SitesSDK.getProperty('componentAssets', callback)
SitesSDK.getProperty(propertyName, callback)
SitesSDK.getSiteProperty(propertyName, callback)
SitesSDK.setProperty('componentAssets', [assets])
SitesSDK.setProperty(propertyName, propertyValue)
SitesSDK.filePicker(options, callback)
SitesSDK.publish(messageType, payload)
SitesSDK.subscribe(messageType, callback)
SitesSDK.Utils.addSiteThemeDesign(cssUrl)
SitesSDK.Utils.Logger
18   Develop Content Connectors

Overview of the Connector Framework 18-1
  Connector REST API Interface 18-2
  Connector SDK 18-2
Build a New Content Connector 18-2
  REST Interfaces for Configuration, Authorization, and Fetching Content 18-4
  REST Interfaces for File System Browsing and Searching 18-9
  Content Picker 18-11
  Authorization 18-15
Content Connector Configuration and Registration 18-19
Content Connector Execution Flow 18-19
Pexels Content Connector Sample Implementation 18-20
  Install the Content Connector 18-20
    Check Prerequisites for Installation 18-20
    Build the Content Connector WAR File 18-21
  Register the Content Connector 18-21
  Test the Content Connector 18-23
  Understand the Content Connector Source Code 18-23
    Custom Picker UI 18-24
    Pexels REST APIs 18-26
    Change and Test the Content Connector Code 18-31
  Switch to a Common UI for the Custom Content Connector 18-31
Download the CEC Content Connector Sample and SDK 18-32

19   Tutorial: Develop Components with Knockout

Introduction and Prerequisites for Component Development with Knockout 19-1
Step 1: Create a Component 19-2
Step 2: Review the Structure of Your Local Component Rendering 19-4
Step 3: Review the Structure of Local Component Settings 19-7
Step 4: Display the New Property in the Component 19-11
Step 5: Register Triggers 19-12
Step 6: Raise Triggers 19-13
Step 7: Register Actions 19-16
Step 8: Execute Actions 19-17
Step 9: Create a Distinct Title for Each Instance of the Component 19-18
Step 10: Use Nested Components with In-Line Editing 19-19
Step 11: Support Different Layouts 19-21
Step 12: Define Custom Styles 19-23
Step 13: Render a Component in an Inline Frame 19-24
Step 14: Use Custom Styles When the Component Is Rendered in an Inline Frame 19-28
20 Troubleshoot

The remote component is not responding 20-1
The remote component settings button is greyed out 20-1
I can't remove a remote component from the catalog 20-1
Preface

Developing for Oracle Content and Experience Cloud provides information and procedures for creating, customizing, and integrating templates, themes, and components for Oracle Content and Experience Cloud, managing content, and collaborating through reusable components and API service calls.

• Audience
• Documentation Accessibility
• Related Resources
• Conventions

Audience

Developing for Oracle Content and Experience Cloud is intended for users who create and customize templates, themes, and components and develop content delivery, content management, and collaboration for websites.

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 Public Cloud:
  http://cloud.oracle.com
• Administering Oracle Content and Experience Cloud
• Creating Experiences with Oracle Content and Experience Cloud
• Getting Started with Oracle Cloud
• Integrating Oracle Content and Experience Cloud
• Known Issues with Oracle Content and Experience Cloud
• Managing Content with Oracle Content and Experience Cloud
• What's New for Oracle Content and Experience Cloud

Conventions

The following text conventions are used in this document.

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

Oracle Content and Experience Cloud enables you to develop themes and templates for websites, customize components, and provide content delivery, content management, and collaboration for your websites.

The Oracle Content and Experience Cloud roles assigned to you by your administrator define what you can do. See User Roles in Administering Oracle Content and Experience Cloud.

- What's New
- Develop Websites, Content Delivery, Content Management, and Collaboration
- Typical Development Tasks

What's New

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Introduced in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site translation with</td>
<td>You can translate a site for multiple languages using Oracle Content and Experience Toolkit and a Language Service Provider. See Translate a Site with a Language Service Provider.</td>
<td>19.2.3 — June, 2019</td>
</tr>
<tr>
<td>a Language Service Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sites Toolkit renamed</td>
<td>The Sites Toolkit has been renamed to Oracle Content and Experience Toolkit. See Develop with Oracle Content and Experience Toolkit.</td>
<td>19.2.3 — June, 2019</td>
</tr>
<tr>
<td>Site index creation</td>
<td>Index an MLS site with an Oracle Content and Experience Toolkit command. See Index a Multilingual Site with Oracle Content and Experience Toolkit.</td>
<td>19.2.2 — May, 2019</td>
</tr>
<tr>
<td>Site map creation</td>
<td>Generate a site map for an MLS site with Oracle Content and Experience Toolkit. See Generate a Site Map for a Multilingual Site.</td>
<td>19.2.2 — May 2019</td>
</tr>
<tr>
<td>Translation support</td>
<td>Create a new site or asset translation job in Content and Experience Cloud with Oracle Content and Experience Toolkit. See Create a New Site or Asset Translation Job in the Oracle Content and Experience Cloud Server.</td>
<td>19.2.2 — May 2019</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Introduced in</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Component creation</td>
<td>Create a simplified component with Oracle Content and Experience Toolkit for easier component development. See Create a Simplified Component for Easy Component Development.</td>
<td>19.2.2 — May 2019</td>
</tr>
<tr>
<td>Custom content connectors</td>
<td>The interface for registering custom content connectors has been updated. See Register the Content Connector.</td>
<td>19.2.1 — April 2019</td>
</tr>
<tr>
<td>Oracle Content and Experience Toolkit</td>
<td>Oracle Content and Experience Toolkit is released on GitHub. See Add Oracle Content and Experience Toolkit to the Project Code in the New Git Repository.</td>
<td>19.2.1 — April 2019</td>
</tr>
<tr>
<td>React starter site</td>
<td>The Starter Site CLI is released on GitHub. See Starter Site CLI for React Development.</td>
<td>19.2.1 — April 2019</td>
</tr>
<tr>
<td>Authentication for REST API for Conversations or REST API for Users and Groups</td>
<td>Social endpoints are no longer supported to work with basic authentication. All social endpoints should be invoked with an OAuth token for the individual user. A random ID is generated when a social endpoint, such as /social/api/v1/connections, is invoked with an OAuth token. See &quot;Using OAuth&quot; under &quot;Authenticate&quot; in REST API for Conversations.</td>
<td>19.1.5 — March 2019</td>
</tr>
<tr>
<td>Content Connectors</td>
<td>You can download a sample content connector and SDK. See Download the CEC Content Connector Sample and SDK.</td>
<td>19.1.5 — March 2019</td>
</tr>
<tr>
<td>Multilingual sites</td>
<td>Enable multilingual translations of custom components for sites. See Develop Translatable Components for Multilingual Sites.</td>
<td>19.1.3 — February 2019</td>
</tr>
<tr>
<td>Custom content connectors</td>
<td>Use the connector framework to implement a custom content connector based on the Pexels connector reference implementation. See Develop Content Connectors.</td>
<td>19.1.3 — February 2019</td>
</tr>
</tbody>
</table>
Develop Websites, Content Delivery, Content Management, and Collaboration

With Oracle Content and Experience Cloud, you can design, build, publish, and manage enterprise-focused websites. These websites include content delivery, content management, and collaboration for users. Published sites are hosted in Oracle Cloud. Sites can be internal to your organization, or sites can be external and visible to the Internet world.

Oracle Content and Experience Cloud provides these features:

• Makes it easy for you to build a website—from concept to launch—using only your web browser.
• Includes templates that have everything you need to get started, including sample pages and content, styling, assets such as images, and even custom components.
• Provides intuitive site building. You simply drag and drop components—image galleries, graphics, text, and more—onto the web page.
• Organizes everything for your sites in one central location.
• Lets folks on your team collaborate and share files related to the design and content of the site.
• Publishes your site for access by web browsers and on mobile devices.

Team web developers can create additional custom themes and apps for use in building sites.

Content, collaboration, and creativity are united in one user interface. You can seamlessly grab and reuse content to build sites, your site content is kept under control, and shared content makes collaboration between and among groups easier than ever.
See Get Started with Sites in *Creating Experiences with Oracle Content and Experience Cloud* and Get Started with Oracle Content and Experience Cloud in *Managing Content with Oracle Content and Experience Cloud*.

## Typical Development Tasks

These are common tasks for developers to support users who build sites, deliver sites, manage content, and collaborate in Oracle Content and Experience Cloud.

When developing for Oracle Content and Experience Cloud, be familiar with the concepts and procedures in *Creating Experiences with Oracle Content and Experience Cloud* and *Managing Content with Oracle Content and Experience Cloud*. These documents describes concepts and tasks you can perform when working with templates, themes, assets, components, documents, and other users.

<table>
<thead>
<tr>
<th>Task</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan a new template to be used by sites</td>
<td>Develop Templates</td>
</tr>
<tr>
<td>Create a template</td>
<td>Work with a Starter Template</td>
</tr>
<tr>
<td>Plan a new theme to be used by sites</td>
<td>Develop Themes</td>
</tr>
<tr>
<td>Create a theme</td>
<td>Create a Theme</td>
</tr>
<tr>
<td>Create or modify theme design and layouts</td>
<td>• Work with Designs and Styles</td>
</tr>
<tr>
<td></td>
<td>• Work with Layouts</td>
</tr>
<tr>
<td>Modify theme backgrounds</td>
<td>Work with Backgrounds</td>
</tr>
<tr>
<td>Set up triggers and actions between components</td>
<td>Work with Triggers and Actions</td>
</tr>
<tr>
<td>Add local and remote components</td>
<td>Develop Components</td>
</tr>
<tr>
<td>Use the Sites SDK to specify functions in the service</td>
<td><a href="#">Oracle Content and Experience Sites SDK Reference</a></td>
</tr>
</tbody>
</table>

Don’t use the following names for templates, themes, components, sites, or site pages: authsite, content, pages, scstemplate_*, _comps, _components, _compsdelivery, _idcservice, _sitescloud, _sitesclouddelivery, _themes, _themesdelivery. Although you can use the following names for site pages, don’t use them for templates, themes, components, or sites: documents, sites.
Integrate Oracle Content and Experience Cloud

As a developer, you can use the functionality provided by Oracle Content and Experience Cloud for integration with web applications and third-party services and applications in a number of ways.

These topics provide an overview of integrating Oracle Content and Experience Cloud:

- Reusable Web Interface
- Proxy Service Settings
- Oracle Process Cloud Service Integration
- Visual Builder Cloud Service Apps
- Custom Actions
- REST Application Programming Interface
- Content SDK and Sites SDK

Reusable Web Interface

Embed the web user interface for Oracle Content and Experience Cloud into your web applications to provide document management features in a ready-to-use interface.

See Embed the Web User Interface.

Proxy Service Settings

Oracle Content and Experience Cloud includes a proxy service so you can use REST services that have Cross-Origin Resource Sharing (CORS) limitations or require service account credentials.

The proxy service is a reverse proxy server. It provides a URL to which web browsers connect. The proxy service then acts as an intermediary between the web browser and a remote REST service (or endpoint). The proxy service explicitly adds CORS support to all endpoints and can optionally insert service account credentials to requests coming from web browsers.

See Configure Proxy Service Settings in Administering Oracle Content and Experience Cloud.

Oracle Process Cloud Service Integration

You can access Oracle Process Cloud Service functionality for documents, folders, and sites through Oracle Content and Experience Cloud, to let users manage business processes, such as document review and approval, or tasks that run on sites.
Before users can take advantage of the Process Cloud Service functionality, you must configure settings in both Oracle Process Cloud Service and Oracle Content and Experience Cloud.

- Configure Oracle Integration Cloud Settings in Oracle Content and Experience Cloud
- Process Cloud Service Integration with Sites

Configure Oracle Integration Settings in Oracle Content and Experience Cloud

Manage workflow for business applications, such as document routing for review or approval, with Oracle Integration integration enabled for documents in Oracle Content and Experience Cloud.

You can allow your users to access Oracle Integration functionality, which lets users manage business processes in the cloud, such as document routing for approval or review. (This feature may not be available depending on the Oracle Content and Experience Cloud subscription type and start date of your service.)

Before users can take advantage of the integrated functionality, you must configure settings in both Oracle Integration and Oracle Content and Experience Cloud. For configuration information, see Integrate with Oracle Process Cloud Service in Administering Oracle Content and Experience Cloud.

When a task step is complete, a user can manage the file according to the defined process. For an incoming document, a user can perform actions based on the assigned role for the document: Contributor, Downloader, or Viewer.

When Oracle Content and Experience Cloud starts a process, the following payload is sent to launch the process:

```json
{
    "operation": "startEvent",
    "processDefId": "testing~LoanApplicationProcessing!1.0~LoanApplicationProcessing",
    "params": {
        "id": "abc123",
        "name": "document name",
        "startedBy": "user id",
        "type": "d",
```
As a developer, you need be aware of the following requirements for the process you develop.

- It needs to be a process that uses an Oracle Content and Experience Cloud Document Start event.
- When you deploy the process, you need to share it with the user specified for enabling the integration so that user has the rights to trigger the process.
- For the user who uploaded the file to appear as the user who started the task, the process must use the value passed in the `startedby` field as the display name for the initiator.

Process Cloud Service Integration with Sites

You can integrate tasks on your sites with Process Cloud Service.

For information about how to set up Process Cloud Service integration with sites, see Integrate with Oracle Process Cloud Service in *Administering Oracle Content and Experience Cloud*.

As a developer, you need be aware of the following requirements for integrating with Process Cloud Service:

- The process author needs to ensure that the sites author is added as an initiator of the process, or the site author will not be able to see the process in the list of processes available when configuring the start form component.
- The process author also needs to ensure that any site visitor is added as an initiator of the process, or the visitor, upon filling in the start form, will not actually be able to initiate the process.

Visual Builder Cloud Service Apps

You can develop apps with Oracle Visual Builder Cloud Service (VBCS), a hosted environment that provides an open-source, standards-based solution to develop, collaborate on, and deploy applications within Oracle Cloud.

A service administrator who has the enterprise user role can configure a connection to VBCS on the Integrations page in the Oracle Content and Experience Cloud administration user interface.
Before using VBCS, do the following setup:

- Create your apps in VBCS and make them available for embedding.
- Configure the apps for use with Oracle Content and Experience Cloud.

See Integrate with Oracle Visual Builder Cloud Service in Administering Oracle Content and Experience Cloud.

You can create a VBCS component for a VBCS app that is configured to allow anonymous access or for a VBCS app that requires visitors to sign in. If a VBCS app allows anonymous access, customers can use the related component on both public and secure sites; for example, to display a public form or public data.

If a VBCS app requires a visitor to sign in, customers can use the related component on secure sites, provided Content and Experience Cloud and VBCS are provisioned on the same identity domain and configured for SSO.

Add a VBCS App As a Remote Component

After the VBCS connection is configured, an administrator can enable it in Content and Experience Cloud. If VBCS integration is enabled, an enterprise user can select an existing VBCS app directly from the Content and Experience Cloud Components catalog and then add it as a remote component:

1. In the Components catalog, register the new VBCS component.
2. In the Register VBCS Component dialog, enter a name and description (optional).
3. Select the VBCS app for a new component.
4. Click Select Application.
5. Select your app from the list of available VBCS apps.
6. Click Create.
7. In the Components catalog, filter the list of components to show VBCS components.

Migrate a Site That Contains a VBCS Component

To migrate a site that contains a VBCS component to another environment, you need to export the VBCS app and site template to the new environment:

1. Export the related VBCS app and then upload the export (ZIP) file into the template’s assets folder together with help documentation.
2. Export the site template and then install it in the target environment.
3. Download the VBCS export file, import the app, configure the app as required, and then make it live on the new VBCS tenant.
4. In Content and Experience Cloud, as an administrator, confirm that the VBCS connection is configured. If not, configure a new connection.

Integration of VBCS Apps with Content and Experience Cloud

VBCS samples templates have been added to Oracle Content and Experience (OCE) Toolkit in VBCSSamplesTemplate.zip. See Develop with Oracle Content and Experience Toolkit.

The following topics provide examples of how to integrate VBCS apps with Content and Experience Cloud:

- Embed a VBCS Visual App in an Oracle Content and Experience Cloud Page
- Embed a VBCS Page in a Site Page
- Build an Oracle Content and Experience Cloud VBCS Public Form Component
- Build an Oracle Content and Experience Cloud VBCS Secure Form Component
- Build an Oracle Content and Experience Cloud VBCS Public Gated Form Component
- Build an Oracle Content and Experience Cloud VBCS Data Report Component
- Build an Oracle Content and Experience Cloud VBCS Multipage Form As a Web App Component

Embed a VBCS Visual App in an Oracle Content and Experience Cloud Page

You can embed a Visual Builder Cloud Service (VBCS) visual app in an Oracle Content and Experience Cloud site page.

To embed a visual app in a site page, you need to create the app in VBCS and then add the app to the page.

Create a Visual App in VBCS

1. Connect to a VBCS Server.
   For example: https://vbcs-server/ic/builder (for a Universal Credits subscription with Oracle Content and Experience Cloud) or https://vbcs-server (for a non-metered subscription).
   If you have a Universal Credits subscription, you must include ic/builder in your Service URL.

   a. From the VBCS menu, choose Settings and then Allowed Origins.
   b. Click New Origin and enter the URL of your Content and Experience Cloud server for Origin Address.
3. Create a new Visual App in VBCS.

4. Create a web application.
   a. Click **Web Applications** in the navigation menu on the left.

5. Allow the web app to be embedded in an iframe:
   a. Select the web app in the navigator.
b. Choose **Settings** (the cog icon), and then click the Security tab. Choose “Allow embedding in any application domain”.

6. Import the Sites SDK:

   a. Right-click the **Resources** node in the navigator.

   b. Locate the Sites SDK (import the `sites.js` or `sites.min.js` file).
The Sites SDK is also available for download from the Oracle Content and Experience Cloud server:

http://{server}/_sitesclouddelivery/renderer/app/sdk/js/sites.min.js

c. Click **Import**. This imports the JS file into the resources directory.

7. Reference the Sites SDK in the page:
   a. With the web app selected, choose the **HTML** tab in Site Builder.
b. Add the following line below the `<link>` tag:

```html
<script type="text/javascript" src="resources/sites.min.js"></script>
```

8. Add a page URL parameter called `id`. Content and Experience Cloud will use this parameter to pass the ID of the component.

   a. Select the page in the web app.

   ![Web Apps](image)

   b. Click the Variables tab.

   ![Variables](image)

   c. Add a variable called `id`, and click Create.
d. In the panel on the right, mark the new variable as a URL input parameter.

9. Add code to automatically set the iframe height when the web app renders:
   a. Click the JS tab on the on the left of the page.
b. Add the following code above the return statement. This will resize the height of the Content and Experience Cloud component when the app renders.

```javascript
setTimeout(function() {
    SitesSDK.setProperty("height", null);
}, 500);
```

10. Stage and Publish the VBCS app. The app must be live for the Content and Experience Cloud site to use it.

Add the VBCS Visual App to a Content and Experience Cloud Site Page

1. In Content and Experience Cloud, configure a VBCS connection:
   a. On the Administration menu, choose Integrations and then Applications.
   b. Select the Visual Builder Cloud Service Integration check box.

   ![Integrations Menu]

   ![Applications Menu]

   ![Visual Builder Cloud Service Integration]

   ![Add VBCS Visual App]

c. Enter the URL of your Content and Experience Cloud instance, and then click Save.
2. Create a new VBCS component:
   a. Choose Developer and then Components.
   b. Choose Create and then Create Visual Builder Component.
   c. Publish the VBCS app you created, copy the URL of the app, and then paste it into the form. Do the same for the web application you created.

3. Add the VBCS component to a site page:
   a. Edit a new or existing site.
   b. In Site Builder, choose Components and then Custom.
c. Drag the VBCS component onto the site page.

Embed a VBCS Page in a Site Page

You can embed a Visual Builder Cloud Service (VBCS) page in a Content and Experience Cloud site page.

To embed a VBCS page in a site page, you need to create an app in VBCS and then add the app to the site page.

Create an App in VBCS

1. Connect to a VBCS Server.

   For example: https://vbcs-server/ic/builder (for a Universal Credits subscription with Oracle Content and Experience Cloud) or https://vbcs-server (for a non-metered subscription).

   Note:

   If you have a Universal Credits subscription, you must include ic/builder in your Service URL.


   a. Choose Administer Visual Builder, then Global Settings, and then Allowed Origins.

   b. Click New Origin and enter URL of your Content and Experience Cloud server for Origin Address.
3. Create a new app in VBCS.

4. Allow the new app to be embedded.
   a. Choose Application Settings, then Security, and then Embedding.
   b. Select Allow embedding in any domain.

5. Use Data Designer and Page Designer to build your app.

6. Add a custom component to the bottom of your page.

7. Select the Custom Component. Enter the following in the Template section:

   `<div data-bind="html: script"></div>`
8. Enter the following code in the **Model** section, substituting your own Sites Cloud server.

```javascript
define([], function () {
  'use strict';

  /** *
   * Inject the SitesSDK and set the Component Height.
   */
  var CustomComponentViewModel = function (params, componentInfo) {
    this.script = ko.observable('' + '(function(d, s, id) {
      var js, fjs = d.getElementsByTagName(s)[0];' + '
      if (d.getElementById(id))' + '
        return;' + '
      js = d.createElement(s);' + '
      js.id = id;' + '
      js.src = ''https://sites-cloud-server'';' + '
      fjs.parentNode.insertBefore(js, fjs);' + '
    })(document, "script", "sites-sdk");' + '
    setTimeout(function() {
      SitesSDK.setProperty("height", null);' + '
    }, 500);' + '
  }' + ']');

  return CustomComponentViewModel;
});
```

9. Enter the following for **Application Style**. This will hide some of the unwanted "chrome" around the component when embedded in an SCS page.

```javascript
/* remove some side padding */
div#abcs-app-content > div {
  max-width: none;
}

/* allow SitesSDK.setProperty("height") to work */
html, body, body.abcs-layout-nonav {
  height: auto;
}
```

10. Stage and publish the VBCS app. The app must be live for Content and Experience Cloud sites to use it.

**Add the VBCS App to a CECS Site Page**

1. **11.** In Content and Experience Cloud, configure a VBCS Connection:
   a. Choose **Administration, Integrations**, and then **Oracle Integrations**.
   b. Select the **Visual Builder Cloud Service Integration** check box.
   c. Enter the Content and Experience Cloud URL (from Step 1 under Create an App in VBCS), and then click **Save**.
2. Create a new VBCS Component:
   a. Choose Experience and then Components,
   b. Choose Create and then Create VBCS Components.
   c. In the drop-down list, choose the VBCS App (created earlier in step 3 under Create an App in VBCS).

3. Add the VBCS component to a site page.
   a. Edit a new or existing site.
   b. In Site Builder, choose Add and then Custom.
c. Drag the VBCS component onto the page.

Build an Oracle Content and Experience Cloud VBCS Public Form Component

You can build a local OCE component that uses REST APIs exposed by business objects in VBCS to deliver a simple, anonymous web form, a public form.

VBCS Configuration

1. Allow Cross-Origin Resource Sharing (CORS):
   a. Choose Visual Builder, then Settings, and then Allowed Origins.
   b. Click New Origin and enter the URL of your Content and Experience Cloud server for Origin Address.
c. Click the check mark to save.

2. Create a new Application:

3. Configure the app to allow anonymous access.
   a. Open Application Settings.
      
      b. On the Settings page, choose **User Roles**.
c. Select **Allow anonymous access**.

4. Create a business object:
   - Add fields.
   - Enable role-based security.
• Grant Anonymous User the Create permission.

Build an OCE Local Component

Assumptions:
• The VBCS app name is "RequestForm".
• The business object name is "requestform" and it contains the following custom fields:
  – name (required)
  – email (required)
  – phone
  – subject
  – message

Modify assets/render.js
1. Define the component template as follows.

```html
<!-- ko if: initialized -->
<div class="form">
    <!-- ko if: requestSuccessMsg -->
    <div class="request-msg green" data-bind="text: requestSuccessMsg"></div>
    <!-- /ko -->
    <!-- ko if: requestFailMsg -->
    <div class="request-msg red" data-bind="text: requestFailMsg"></div>
    <!-- /ko -->
    <label class="required-field" for="name">Name</label>
    <input type="text" id="name" name="name" required placeholder="Your name...." data-bind="value: name"/>
    <label class="required-field" for="email">Email</label>
    <input type="text" id="email" name="email" required placeholder="Your email...." data-bind="value: email"/>
    <label for="phone">Phone</label>
    <input type="text" id="phone" name="phone" data-bind="value: phone"/>
    <label for="subject">Subject</label>
    <input type="text" id="subject" name="subject" data-bind="value: subject"/>
    <label for="message">Message</label>
    <textarea id="message" name="message" rows="6" data-bind="value: message"></textarea>
    <button data-bind="click: sendRequest, , enable: canSubmit">Send Request</button>
</div>
<!-- note that the component has completed rendering into the page -->
</div>
<!-- /ko -->

2. Create the observables for the fields in the Knockout ViewModel.

```javascript
self.initialized = ko.observable(false);
self.requestSuccessMsg = ko.observable();
self.requestFailMsg = ko.observable();
self.VBCSServerUrl = ko.observable();
self.name = ko.observable();
self.email = ko.observable();
self.phone = ko.observable();
self.subject = ko.observable();
self.message = ko.observable();

// Get VBCS server
var serverPromise = getVBCSServerURL();
```
serverPromise.then(function (result) {
  self.VBCSServerUrl(result.url);
  self.initialized(true);
});

self.canSubmit = ko.computed(function () {
  return self.name() && self.email();
}, self);

3. **Handle required fields.**

Enable the Submit button only after all required fields have values.

4. **Obtain the VBCS connection.**

After you configure VBCS connection, there are two ways to get the connection:

- From siteinfo at site runtime
- From Integrations in Site Builder

```javascript
var getVBCSServerURL = function () {
  var serverPromise = new Promise(function (resolve, reject) {
    // First try to get from siteinfo
    var siteConnections = SCSRenderAPI.getSiteProperty('siteConnections);
    var serverUrl = siteConnections && siteConnections.VBCSConnection;
    if (serverUrl) {
      console.log('Get VBCS server from siteinfo: ' + serverUrl);
      resolve({'url': serverUrl});
    } else {
      // Get from integrations
      var configUrl = '/documents/web?
IdcService=AP_GET_APP_INFO_SIMPLE&dAppName=VBCS';
      $.ajax({
        type: 'GET',
        dataType: 'json',
        url: configUrl,
        success: function (data) {
          var appInfo = data.ResultSets.AFApplicationInfo;
          var enabled;
          if (appInfo) {
            for (var i = 0; i < appInfo.fields.length; i++) {
              if (appInfo.fields[i].name === 'dAppEndPoint') {
                serverUrl = appInfo.rows[appInfo.currentRow][i];
              } else if (appInfo.fields[i].name === 'dIsAppEnabled') {
                enabled = appInfo.rows[appInfo.currentRow][i];
              }
            }
            if (serverUrl && enabled) {
              resolve({'url': serverUrl});
            }
          }
        }
      });
    }
  });
};
```
break;
}
}
if (enabled !== '1') {
    serverUrl = '';
}

console.log('Get VBCS server from Idc Service: ' + serverUrl);
resolve({'url': serverUrl});

error: function (xhr, status, err) {
    console.log('Request failed: url:' + configUrl + ' status: ' + status + ' error: ' + err);
    resolve({'url': serverUrl});

});

return serverPromise;

5. Submit the request

self.sendRequest = function (data, event) {
    var vbcsServer = self.VBCSServerUrl();
    var appName = 'requestform',
    appVersion = 'live',
    businessObject = 'Requestform'
    var url = vbcsServer + '/rt/' + appName + '/' + appVersion + '/resources/data/' + businessObject;
    var payload = {
        "name": self.name(),
        "email": self.email(),
        "phone": self.phone(),
        "subject": self.subject(),
        "message": self.message()
    };
    $.ajax({
        type: 'POST',
        url: url,
        beforeSend: function(xhr) {
            xhr.setRequestHeader( "Content-type", "application/vnd.oracle.adf.resourceitem+json" );
        },
        data: JSON.stringify(payload),
        dataType: 'json',
        success: function (data) {
            self.requestFailMsg('');
            self.requestSuccessMsg('Request has been submitted successfully');
            self.name('');
            self.email('');
            self.phone('');
            self.subject('');
Modify styles/design.css

Add the following css to design.css.

```css
.form {
  font-family: "Helvetica Neue", "Segoe UI", sans-serif-regular, Helvetica, Arial, sans-serif;
  font-size: 14px;
}
.form input[type=text] {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
.form textarea {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
.form button {
  width: 100%;
  background-color: #4CAF50;
  color: white;
  padding: 14px 20px;
  margin: 8px 0;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
.form button:hover {
  background-color: #45a049;
}
.form button:disabled {
  background-color: #dddddd;
}
```
Use the Form Component on OCE

1. Configure the VBCS connection:
   - Choose Administration, then Integrations, and then Applications.
   - Click the Visual Builder Cloud Service Integration check box.
   - Enter the URL, and click Save.

2. Import the component:
   - Choose Developer and then Components.
   - Choose Create and then Import Component.
3. Add the component to a page:
   a. Edit a new or existing site.
   b. In Site Builder, choose **Components** and then **Custom**.
   c. Drag the component onto the page.
Build an Oracle Content and Experience Cloud VBCS Secure Form Component

You can build a local OCE component that uses REST APIs exposed by business objects in VBCS to deliver a simple web form that requires user authentication.

**VBCS Configuration**

1. Allow Cross-Origin Resource Sharing (CORS):
   a. Choose Visual Builder, then Settings, and then Allowed Origins.
   b. Click New Origin and enter the URL of your Content and Experience Cloud server for Origin Address.

   **Allowed Origins**

   Type Cross-Origin Resource Sharing (CORS) allows external services to interact with visual builder.

   Origin Address

   Enter allowed origin URL

   c. Click the check mark to save.
2. Create a new Application

3. Configure the app to allow access for authenticated users.
   a. Open Application Settings

   ![Application Settings](image)

   b. On the Settings page, choose **User Roles**.
   c. Add roles to control access to the business object.

   ![User Roles](image)

   The values for **Mapping** are groups from Oracle Identity Cloud Service. To add groups, see Create Groups with Oracle Identity Cloud Service.

4. Create a business object.
• Add fields.

• Enable role-based security
Grant roles to Create permission.

Build an OCE Local Component

Assumptions:

- The VBCS app name is "RequestForm".
- The business object name is "requestform" and it contains the following custom fields:
  - name (required)
  - email (required)
  - phone
  - subject
  - message

Modify assets/render.js

1. Define the component template as follows.

```html
<!-- ko if: initialized -->
<div class="form">
  <!-- ko if: requestSuccessMsg -->
  <div class="request-msg green" data-bind="text: requestSuccessMsg"></div>
  <!-- /ko -->
  <!-- ko if: requestFailMsg -->
  <div class="request-msg red" data-bind="text: requestFailMsg"></div>
  <!-- /ko -->
  <label class="required-field" for="name">Name</label>
  <input type="text" id="name" name="name" required
  placeholder="Your name...." data-bind="value: name"/>

  <label class="required-field" for="email">Email</label>
  <input type="text" id="email" name="email" required
  placeholder="Your email...." data-bind="value: email"/>
```
2. Create the observables for the fields in the Knockout ViewModel.

```javascript
self.initialized = ko.observable(false);
self.requestSuccessMsg = ko.observable();
self.requestFailMsg = ko.observable();
self.VBCSServerUrl = ko.observable();
self.name = ko.observable();
self.email = ko.observable();
self.phone = ko.observable();
self.subject = ko.observable();
self.message = ko.observable();

// Get VBCS server
var serverPromise = getVBCSServerURL();
serverPromise.then(function (result) {
    self.VBCSServerUrl(result.url);
    self.initialized(true);
});
```

```javascript
self.canSubmit = ko.computed(function () {
    return self.name() && self.email();
}, self);
```

3. Handle required fields.
   Enable the Submit button only after all required fields have values.

4. Obtain the VBCS connection
   After configure VBCS connection, there are two ways to get the connection:
   - From siteinfo at site runtime
• From Integrations in Site Builder

```javascript
var getVBCSServerURL = function () {
    var serverPromise = new Promise(function (resolve, reject) {
        // First try to get from siteinfo
        var siteConnections = SCSRenderAPI.getSiteProperty('siteConnections');
        var serverUrl = siteConnections && siteConnections.VBCSConnection;
        if (serverUrl) {
            console.log('Get VBCS server from siteinfo: ' + serverUrl);
            resolve({'url': serverUrl});
        } else {
            // Get from integrations
            var configUrl = '/documents/web?IdcService=AF_GET_APP_INFO_SIMPLE&dAppName=VBCS';
            $.ajax({
                type: 'GET',
                dataType: 'json',
                url: configUrl
                success: function (data) {
                    var appInfo = data.ResultSets.AFApplicationInfo;
                    var enabled;
                    if (appInfo) {
                        for (var i = 0; i < appInfo.fields.length; i += 1) {
                            if (appInfo.fields[i].name === 'dAppEndPoint') {
                                serverUrl = appInfo.rows[appInfo.currentRow][i];
                            } else if (appInfo.fields[i].name === 'dIsAppEnabled') {
                                enabled = appInfo.rows[appInfo.currentRow][i];
                            } else {
                                break;
                            }
                        }
                        console.log('Get VBCS server from Idc Service: ' + serverUrl);
                        resolve({'url': serverUrl});
                    },
                    error: function (xhr, status, err) {
                        console.log('Request failed: url:' + configUrl + ' status: ' + status + ' error: ' + err);
                        resolve({'url': serverUrl});
                    }
                });
            }),
            return serverPromise;
        }
    });
};
```

5. Get an authorization token.
Requirement: OCE and VBCS are deployed in the same identity domain.

```javascript
var getAuthToken = function (args) {
    // dummy function if callbacks not supplied
    var dummyCallback = function () {};

    // extract the args and create the server URL
    var serverURL = (args.serverURL || '/').split('/ic/')[0],
    successCallback = args.successCallback || dummyCallback,
    errorCallback = args.errorCallback || dummyCallback,
    tokenURL = serverURL + '/ic/builder/resources/security/token';

    // For VBCS to get the auth token:
    // - make a POST call to /ic/builder/resources/security/token
    // - include scope=run-time form parameter
    var getToken = function (tokenURL, successCallback, errorCallback) {
        $.ajax({
            'type': 'POST',
            'url': tokenURL,
            data: {
                scope: 'run-time'
            },
            'xhrFields': {
                withCredentials: true
            },
            'success': successCallback
        }).fail(errorCallback);
    };

    // try to get the token normally
    getToken(tokenURL,
    function (resp, status, xhr) {
        var ct = xhr.getResponseHeader("content-type") || "";

        // if the response was an HTML Form....
        if (ct.indexOf('html') > -1) {
            // parse the form and submit it
            var parser = new DOMParser(),
                htmlDoc = parser.parseFromString(resp, "text/html"),
                forms = htmlDoc.getElementsByTagName("form");
            if (forms.length === 1) {
                var f = forms[0];
                $.ajax({
                    'type': 'POST',
                    'url': f.action,
                    'data': $(f).serialize(),
                    'xhrFields': {
                        withCredentials: true
                    },
                    'success': function () {
                        // try to get the token normally
                        getToken(tokenURL, successCallback,
                        errorCallback);
                    }
                }).fail(function () {
        });
```
// even if the form submit failed, retry
getToken(tokenURL, successCallback,
errorCallback);
});
}
} else {
// already logged in return the token
successCallback(resp);
}
},
errorCallback);
);

6. Submit the request.

self.sendRequest = function (data, event) {
var vbcsServer = self.VBCSServerUrl();
var authorization, token;
var appName = 'securerequestform',
mode = 'rt',
appVersion = 'live',
businessObject = 'Requestform';
var url = vbcsServer + '/' + mode + '/' + appName + '/' +
appVersion + '/resources/data/' + businessObject;
var payload = {
"name": self.name(),
"email": self.email(),
"phone": self.phone(),
"subject": self.subject(),
"message": self.message()
};
// get token first
getAuthToken({
'serverURL': self.VBCSServerUrl(),
'successCallback': function (data) {
    token = data;
    authorization = (token.token_type ? token.token_type :
'Bearer') + ' ' + token.access_token;
$.ajax({
    type: 'POST',
    url: url,
    beforeSend: function (xhr) {
        xhr.setRequestHeader('Content-type', 'application/
vnd.oracle.adf.resourceitem+json');
        xhr.setRequestHeader('Authorization',
authorization);
    },
data: JSON.stringify(payload),
dataType: 'json',
success: function (data) {
    self.requestFailMsg('');
    self.requestSuccessMsg('Request has been submitted successfully');

});
self.name('');
self.email('');
self.phone('');
self.subject('');
self.message('');
},
error: function (jqXhr, textStatus, errorThrown) {
  console.log('Error:');
  console.log(jqXhr);
  self.requestSuccessMsg('');
  self.requestFailMsg('Failed to submit the request');
}
});
},
'errorCallback': function (xhr, status, err) {
  if (xhr && xhr.status === 200) {
    token = xhr.responseText;
    console.log('Got token');
  } else {
    console.error('getToken: xhr: ' + JSON.stringify(xhr) +
      ' status: ' + status + ' error: ' + err);
    self.requestSuccessMsg('');
    self.requestFailMsg('Failed to get authorization token');
  }
}
});
);

Modify styles/design.css

Add the following css to design.css.

.form {
  font-family: "Helvetica Neue", "Segoe UI", sans-serif-regular, Helvetica, Arial, sans-serif;
  font-size: 14px;
}
.form input[type=text] {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
.form textarea {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
Use the Form Component on OCE

1. Configure the VBCS connection:
   - Choose Administration, then Integrations, and then Applications.
   - Click the Visual Builder Cloud Service Integration check box.
   - Enter the URL, and click Save.
2. Import the component:
   - Choose **Developer** and then **Components**.
   - Choose **Create** and then **Import Component**.

3. Add the component to a page
   a. Edit a new or existing site.
   b. In Site Builder, choose **Components** and then **Custom**.
c. Drag the component onto the page.

Note:
This VBCS secure form component works only on secure sites.
Build an Oracle Content and Experience Cloud VBCS Public Gated Form Component

You can build a local OCE component that uses REST APIs exposed by business objects in VBCS to deliver a simple, anonymous web form that captures visitor details to allow the visitor to download a document.

**VBCS Configuration**

1. Allow Cross-Origin Resource Sharing (CORS):
   a. Choose **Visual Builder**, then **Settings**, and then **Allowed Origins**.
   b. Click **New Origin** and enter the URL of your Content and Experience Cloud server for Origin Address.
   c. Click the check mark to save.

2. Create a new Application.

3. Configure the app to allow anonymous access.
   a. Open Application Settings.
b. On the Settings page, choose **User Roles**.

c. Select **Allow anonymous access**.

4. Create a business object:
   - Add fields.
- Enable role-based security.
• Grant Anonymous User the Create permission.

**Build an OCE Local Component**

Assumptions:

- The VBCS app name is "RequestForm".
- The business object name is "registration" and it contains the custom fields.
  - firstName (required)
  - lastName (required)
  - email (required)
  - phone
  - company
Modify assets/render.js

1. Define the component template as follows.

```html
<!-- ko if: initialized -->

<div class="form">
  <!-- ko if: !showDownload() -->
  <h1 style="text-align: center;">Fill out the form to access the document</h1>

  <!-- ko if: requestSuccessMsg -->
  <div class="request-msg green" data-bind="text: requestSuccessMsg"></div>
  <!-- /ko -->

  <!-- ko if: requestFailMsg -->
  <div class="request-msg red" data-bind="text: requestFailMsg"></div>
  <!-- /ko -->

  <label class="required-field" for="firstname">First Name</label>
  <input type="text" id="firstname" name="firstname" required placeholder="Your first name...." data-bind="value: firstName"/>

  <label class="required-field" for="lastname">Last Name</label>
  <input type="text" id="lastname" name="lastname" required placeholder="Your last name...." data-bind="value: lastName"/>

  <label for="email">Business E-mail</label>
  <input type="text" id="email" name="email" required placeholder="Your email...." data-bind="value: email"/>

  <label for="phone">Phone</label>
  <input type="text" id="phone" name="phone" data-bind="value: phone"/>

  <label for="company">Company</label>
  <input type="text" id="company" name="company" data-bind="value: company"/>

  <label for="jobtitle">Job Title</label>
  <input type="text" id="jobtitle" name="jobtitle" data-bind="value: jobTitle"/>

  <button data-bind="click: sendRequest, enable: canSubmit">Accept the document</button>
  <!-- /ko -->

  <!-- ko if: showDownload() -->
  <div class="download">
    <h2>Thanks for your registration. Please click the button to download.</h2>
    <button data-bind="click: startDownload">Download</button>
    <button data-bind="click: closeDownload">Close</button>
  </div>
  <!-- /ko -->
</div>
```

<!-- note that the component has completed rendering into the page -->
```
2. Create the observables for the fields in the Knockout ViewModel.

```javascript
self.initialized = ko.observable(false);
self.requestSuccessMsg = ko.observable();
self.requestFailMsg = ko.observable();
self.VBCSServerUrl = ko.observable();
self.showDownload = ko.observable(false);
self.firstName = ko.observable();
self.lastName = ko.observable();
self.email = ko.observable();
self.phone = ko.observable();
self.company = ko.observable();
self.jobTitle = ko.observable();

self.canSubmit = ko.computed(function () {
    return self.firstName() && self.lastName() && self.email();
}, self);
```

3. Handle required fields.

   Enable the Submit button only after all required fields have values.

4. Obtain the VBCS connection.

   After you configure a VBCS connection, there are two ways to get the connection:
   - From siteinfo at site runtime
   - From Integrations in Site Builder

```javascript
var getVBCSServerURL = function () {
    var serverPromise = new Promise(function (resolve, reject) {
        // First try to get from siteinfo
        var siteConnections = SCSRenderAPI.getSiteProperty('siteConnections');
        var serverUrl = siteConnections && siteConnections.VBCSConnection;
        if (serverUrl) {
            console.log('Get VBCS server from siteinfo: ' + serverUrl);
            resolve({'url': serverUrl});
        } else {
            // Get from integrations
            var configUrl = '/documents/web?' +
                'IdcService=AF_GET_APP_INFO_SIMPLE&dAppName=VBCS';
            $.ajax({
                type: 'GET',
                dataType: 'json',
                url: configUrl,
                success: function (data) {
                    var appInfo = data.ResultSets.AFApplicationInfo;
                    var enabled;
                });
```
if (appInfo) {
    for (var i = 0; i < appInfo.fields.length; i += 1) {
        if (appInfo.fields[i].name === 'dAppEndPoint') {
            serverUrl = appInfo.rows[appInfo.currentRow][i];
        } else if (appInfo.fields[i].name === 'dIsAppEnabled') {
            enabled = appInfo.rows[appInfo.currentRow][i];
            if (serverUrl && enabled) {
                break;
            }
        }
        if (enabled !== '1') {
            serverUrl = '';
        }
    }
    console.log('Get VBCS server from Idc Service: ' + serverUrl);
    resolve({'url': serverUrl});
},
error: function (xhr, status, err) {
    console.log('Request failed: url:' + configUrl + ' status: ' + status + ' error: ' + err);
    resolve({'url': serverUrl});
})
});
return serverPromise;
};

5. Submit the request.

self.sendRequest = function (data, event) {
    var vbcsServer = self.VBCSServerUrl();
    var appName = 'requestform',
        appVersion = 'live',
        businessObject = 'Registration';
    var url = vbcsServer + '/rt/' + appName + '/' + appVersion + '/resources/data/' + businessObject;
    var payload = {
        'firstName': self.firstName(),
        'lastName': self.lastName(),
        'email': self.email(),
        'phone': self.phone(),
        'company': self.company(),
        'jobTitle': self.jobTitle()
    };

    $.ajax({
6. Create a trigger to download a document.

```json
{
    "id": "ECVBCS-Gated-Form",
    "settingsData": {
        "settingsHeight": 90,
        "settingsWidth": 300,
        "settingsRenderOption": "dialog",
        "componentLayouts": [],
        "triggers": [{
            "triggerName": "VBCSGatedFormSubmitted",
            "triggerDescription": "VBCS gated form submitted",
            "triggerPayload": [{
                "name": "payloadData",
                "displayName": "Document URL"
            }]
        }],
        "actions": []
    }
}
```
Register the trigger in `appinfo.json`.

```js
self.raiseTrigger = function (triggerName) {
   SitesSDK.publish(SitesSDK.MESSAGE_TYPES.TRIGGER_ACTIONS, {
      'triggerName': triggerName,
      'triggerPayload': {
         'payloadData': 'https://docs.oracle.com/en/cloud/paas/content-cloud/developer/developing-oracle-content-and-experience-cloud.pdf'
      }
   });
};

self.startDownload = function (data, event) {
   console.log('Raise trigger: VBCSGatedFormSubmitted');
   self.raiseTrigger("VBCSGatedFormSubmitted"); // matches appinfo.json
};
```

- Raise the trigger in `render.js`.

\*

**Modify styles/design.css**

Add the following css to `design.css`.

```css
.form {
   font-family: "Helvetica Neue", "Segoe UI", sans-serif-regular, Helvetica, Arial, sans-serif;
   font-size: 14px;
}
.form input[type=text] {
   width: 100%;
   padding: 12px 20px;
   margin: 8px 0;
   display: inline-block;
   border: 1px solid #ccc;
   border-radius: 4px;
   box-sizing: border-box;
}
.form textarea {
   width: 100%;
   padding: 12px 20px;
   margin: 8px 0;
   display: inline-block;
   border: 1px solid #ccc;
   border-radius: 4px;
   box-sizing: border-box;
}
.form button {
   width: 100%;
   background-color: #4CAF50;
   color: white;
   padding: 14px 20px;
   margin: 8px 0;
}
```
Use the Form Component on OCE

1. Configure the VBCS connection:
   - Choose Administration, then Integrations, and then Applications.
   - Click the Visual Builder Cloud Service Integration check box.
   - Enter the URL, and click Save.
2. Import the component:
   - Choose Developer and then Components.
   - Choose Create and then Import Component.

3. Add the component to a page:
   a. Edit a new or existing site.
   b. In Site Builder, choose Components and then Custom.
Drag the component onto the page.

Note:
This VBCS public form component can be used on public or secure sites.
Build an Oracle Content and Experience Cloud VBCS Data Report Component

You can build a local OCE component that uses REST APIs exposed by business objects in VBCS to deliver reports on data collected through forms.

Use data from a public form component to show the number of requests per day in a CSS bar chart. See Build an OCE VBCS Public Form Component.

Build an OCE Local Component

Assumptions:

- The VBCS app name is “RequestForm”.
- The business object name is “requestform”.

Modify assets/render.js

1. Define the component template as follows.

```html
<!-- ko if: initialized -->
<h1 style="text-align: center;">Number of requests per day</h1>
<div class="chartrow">
  <div class="chartbody">
    <div class="chartbody">
      <table id="q-graph">
        <tbody data-bind="foreach: requests">
          <tr class="qtr" data-bind="css: barcss">
            <td class="day bar" data-bind="style: {height: height}"
              >
              <p data-bind="text: value"></p>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
    <div id="ticks" data-bind="foreach: ticks">
      <div class="tick" style="height: 59px;">
        <p data-bind="text: value"></p>
      </div>
    </div>
    <div class="chartlabel">
      <div class="labelrow">
        <div class="colorindex sunday"></div>
        <span>Sunday</span>
      </div>
      <div class="labelrow">
        <div class="colorindex monday"></div>
        <span>Monday</span>
      </div>
      <div class="labelrow">
        <div class="colorindex tuesday"></div>
        <span>Tuesday</span>
      </div>
    </div>
  </div>
</div>
</div>
</div>
```
2. Create the observables for the fields in the Knockout ViewModel.

```javascript
self.initialized = ko.observable(false);
self.VBCSServerUrl = ko.observable();
self.requests = ko.observableArray();
self.ticks = ko.observableArray();

// Get VBCS server
var serverPromise = getVBCSServerURL();
serverPromise.then(function (result) {
    self.VBCSServerUrl(result.url);
    self.initialized(true);
    self.requests().
});
```

3. Obtain the VBCS connection.

See Build an OCE VBCS Public Form Component.


Requirement: OCE and VBCS are deployed in the same identity domain. See Build an OCE VBCS Secure Form Component.

5. Get requests.
Use a business object endpoint:

```
/Requestform

self.getRequests = function () {
  var vbcsServer = self.VBCSServerUrl();
  var authorization, token;
  var appName = 'requestform',
      mode = 'rt',
      appVersion = 'live',
      businessObject = 'Requestform';
  var url = vbcsServer + '/' + mode + '/' + appName + '/' + appVersion + '/resources/data/' + businessObject;

  // get token first
  getAuthToken({
    'serverURL': self.VBCSServerUrl(),
    successCallback: function (data) {
      token = data;
      authorization = (token.token_type ? token.token_type : 'Bearer') + ' ' + token.access_token;
      url = url + '?limit=999&orderBy=creationDate:desc';
      $.ajax({
        type: 'GET',
        url: url,
        beforeSend: function (xhr) {
          xhr.setRequestHeader('Authorization', authorization);
        },
        success: function (data) {
          if (data && data.count > 0) {
            self.showChart(data.items);
          }
        },
        error: function (jqXhr, textStatus, errorThrown) {
          console.log('Error:');
          console.log(jqXhr);
        }
      });
    },
    errorCallback: function (xhr, status, err) {
      if (xhr && xhr.status === 200) {
        token = xhr.responseText;
        console.log('Got token');
      } else {
        console.error('getToken: xhr: ' + JSON.stringify(xhr) + ' status: ' + status + ' error: ' + err);
      }
    }
  });
};
```
6. Set up chart data.

```javascript
self.showChart = function (items) {
    var weekDayCounts = [0, 0, 0, 0, 0, 0, 0];
    var weekDayNames = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
    for(var i = 0; i < items.length; i++) {
        var d = new Date(items[i].creationDate);
        var day = d.getDay();
        if (day >= 0 && day < 7) {
            weekDayCounts[day] = weekDayCounts[day] + 1;
        }
    }
    var max = 0;
    var results = [];
    for(var i = 0; i < weekDayCounts.length; i++) {
        if (max < weekDayCounts[i]) {
            max = weekDayCounts[i];
        }
    }
    var lines = 7,
        buckets = 5;
    var gap = Math.round(max / buckets);
    var top = gap * (lines - 1);
    if (max > top) {
        gap += 1;
    }
    var ticks = [];
    for(var i = 0; i < lines; i++) {
        ticks[i] = {
            value: i * gap
        };
    }
    self.ticks(ticks.reverse());

    for(var i = 0; i < weekDayCounts.length; i++) {
        var height = (weekDayCounts[i] / gap) * 60;
        results.push({
            height: height.toString() + 'px',
            value: weekDayCounts[i],
            barcss: weekDayNames[i].toLowerCase()
        });
    }
    self.requests(results);
};
```

Use the component on OCE

For information about how to add the component on a page, see Build an OCE VBCS Public Form Component.

The component renders as follows.
This VBCS component works only on secure sites.

Build an Oracle Content and Experience Cloud VBCS Multipage Form As a Web App Component

You can build a complex, multipage form for Content and Experience Cloud as a component of a VBCS web application.

To build a VBCS multipage form as a web application component for Content and Experience Cloud, you can remove the header and footer, set the IFrame height, add validation for required form fields, call a REST endpoint to save form data, and, for a public web application, set security to allow anonymous access.

Remove the VBCS Web Application Header and Footer

If you don't want the default VBCS web application's header and footer, you can remove them from the page.
Set the VBCS Component Iframe Height

You need to set the VBCS component iframe height for each page in the flow of the web app. Include a validation group in the page model,

Required Form Fields Validation

When you try to navigate to another page, some of form fields on the current page are marked as “required” and don't have values yet. By default, there is no validation. You can use oj-validation-group to add the validation:

1. Wrap all your form fields inside the component <oj-validation-group>.
2. Include `oj-validation-group` in the page model.
3. Create a Flow module function.

```javascript
// main

define([], function() {
    'use strict';

    var FlowModule = function FlowModule() {

        /**
         * @param {String} arg1
         * @return {String}
         */
        FlowModule.prototype.validateFields = function (arg1) {
            console.log('validate fields!');
            var tracker = document.getElementById('tracker');
            if (!tracker) {
                return;
            }
            tracker.showMessages();
            if (tracker.valid === "invalidShown") {
                tracker.focusOn("@firstInvalidShown");
            }
            setTimeout(function() {
                console.log('validateFields set height');
                SitesSDK.setProperty("height", null);
            }, 500);
        }
        return FlowModule;
    }
```

4. Create an action chain for the “Go to the Next Step” button.
The "if" condition checks the values for the required fields; for example:

```javascript
{{ $flow.variables.enroll_values.field
&& $flow.variables.enroll_values.zip }}
```

If the condition evaluates to true, navigation goes to the second step. Otherwise, it stays on the current page, and the error message is shown.
Call REST Endpoint to Save Form Data

At the last step, after the form fields validation, you can use the "Call REST Endpoint" action to persist form data in the business object.
Public Web Application

If the web application is intended for public, you can set security to allow anonymous access.

Also allow anonymous access for the business object.
Provide a VBCS Endpoint As a URL for Select Menus

A content administrator can provide a Visual Builder Cloud Service (VBCS) endpoint as a URL to get values for select menus.

Instead of typing values for a select menu, users can enter a VBCS URL to get the values for the select menu in the Site Builder dialog. The content administrator can configure what attributes to use for option text and values in a REST response, using a public or secure VBCS URL.

Users can populate VBCS select menus in content item forms based on the URL configuration. When defining a content type and fields for the content type, the content administrator has the option to select a single-value menu that uses a Visual Builder endpoint.

The data for a content item is pooled by the Visual Builder URL. A business object, such as Company, can be defined by many fields in VBCS.

To create a content type with a VBCS endpoint as a URL for select menus:

1. Sign in as a content administrator in your browser and click Assets under Administration in the left navigation menu.
2. Select Content Types from the drop-down menu and click Create.
3. Enter VBCS as the name of your content type.
4. Under Content Type Definition for the VBCS content type, click Edit to edit the settings.
5. In the Text Settings dialog, click Appearance.
7. In the **Visual Builder endpoint URL** field, enter a valid endpoint that is defined in VBCS.

8. For Endpoint security, select the check box if the Visual Builder endpoint requires an authenticated user. If the check box is not selected, anonymous users can use the VBCS endpoint.

9. Enter values for the **REST field to use as label** and **REST Field to use as value** fields, such as `companyName` and `email`.

10. Click **Test Configuration** to test the settings. If the configuration is correct, a “Test successful” message appears.

11. Save your changes and click **OK**.
In the Create Content Item dialog, the UI for a VBCS content type looks identical to a single-select box for static data, except the data for the content item is dynamically pooled from the VBCS side with a run search. The value of REST field to use as a value, such as email, gets stored in the Content and Experience Cloud database.

Custom Actions

Integrate third-party services and applications into the Oracle Content and Experience Cloud interface in a simple and effective way, with custom actions in the Application Integration Framework (AIF).

Using AIF, you can quickly define the actions that are exposed in the interface, respond to user selections, call third-party services, and specify how the results are presented to the user. The framework supports variables and expressions and provides multiple language support. See Develop Custom Actions.

REST Application Programming Interface

Incorporate Oracle Content and Experience Cloud functionality into your web applications using Representational State Transfer (REST) requests and responses.

REST offers a simple, lightweight API for applications that do not require the server to maintain state information or message exchange. The Oracle Cloud REST API for
Content Delivery, REST API for Conversations, REST API for Documents, and REST API for Users and Groups provide the following resources.

For overview information, see these topics:

• Overview of the REST API for Content Delivery
• Overview of the REST API for Content Management
• Overview of the REST API for Conversations
• Overview of the REST API for Documents
• Overview of the REST API for Users and Groups

For API descriptions, examples, and information about getting started, see these documents:

• REST API for Content Delivery
• REST API for Content Management
• REST API for Conversations
• REST API for Documents
• REST API for Users and Groups

Enable Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) allows a web page to make requests such as XMLHttpRequest to another domain. See Understand Cross-Origin Resource Sharing and Enable Cross-Origin Resource Sharing (CORS) in Administering Oracle Content and Experience Cloud.

Content SDK and Sites SDK

Oracle Content and Experience Cloud provides JavaScript software development kit (SDK) functions to support apps, in the Content SDK and the Sites SDK.

Content SDK

The Content SDK for Oracle Content and Experience Cloud is a light-weight JavaScript wrapper that interacts with the Content Delivery REST API. This is a read-only SDK for retrieving structured content, digital assets, and content layouts that are managed in Oracle Content and Experience Cloud.

The Content SDK is available for download from the Oracle Content and Experience Cloud server.

See Content SDK and the Content SDK reference.

Sites SDK

The Oracle Content and Experience Cloud JavaScript SDK (Sites SDK) has a set of functions that enable components to have a more integrated experience with Content and Experience Cloud. The Sites SDK for Oracle Content and Experience Cloud handles all communications between a custom component and a site page, to render the component on the page.
You can use the Sites SDK to build and manage applications. It has functions to list component assets, get and set property values, list selected files, and send and receive messages.

See the Sites SDK Reference.
Develop with Oracle Content and Experience Toolkit

Oracle Content and Experience Toolkit helps you develop site templates, themes, custom components, and content layouts for Oracle Content and Experience Cloud.

With Oracle Content and Experience Toolkit, you work in your own development environment and can use asset repositories, files, and folders in Oracle Content and Experience Cloud. The Oracle Content and Experience Toolkit has tools to create and develop custom components and site templates, including themes and content layouts. It includes a local test harness for quick, iterative development and sample unit tests to start with.

Oracle Content and Experience Toolkit can help you do the following tasks:

• Set up your local development environment to use an Oracle Content and Experience Cloud instance for local development and testing of components, templates, themes, and content layouts
• Create components, site templates, and content layouts from samples, run them in the test harness, explore them, and develop the components, templates, themes, and content layouts in a Developer Cloud Service environment
• Import components and site templates that were created in Oracle Content and Experience Cloud into a Developer Cloud Service project and environment for source management and further development
• Export components, templates, and content layouts from a Developer Cloud Service environment for use in Oracle Content and Experience Cloud
• Copy an existing component, template, or content layout
• Write unit tests
• Optimize components
• Deploy your components and templates to Oracle Content and Experience Cloud

The following topics describe how to set up Oracle Content and Experience Toolkit and develop with it on your local machine or as a Developer Cloud Service project:

• Set Up Oracle Content and Experience Toolkit on Your Local Machine
• Develop for Oracle Content and Experience Cloud with Developer Cloud Service
• Index Site Pages with Oracle Content and Experience Toolkit
• Create a New Site or Asset Translation Job in the Oracle Content and Experience Cloud Server
• Translate a Site with a Language Service Provider
• Index a Multilingual Site with Oracle Content and Experience Toolkit
• Generate a Site Map for a Multilingual Site
• Create a Simplified Component for Easy Component Development
Set Up Oracle Content and Experience Toolkit on Your Local Machine

On your local machine, create a folder to download Oracle Content and Experience Toolkit from your Content Experience Cloud instance.

In a terminal window, enter these commands:

1. `git clone git@github.com:oracle/content-and-experience-toolkit.git`
   Or you can download from https://github.com/oracle/content-and-experience-toolkit/archive/master.zip

2. `cd content-and-experience-toolkit/sites/cec-components`

Complete the setup and prepare to use Oracle Content and Experience Toolkit:

1. Install Dependencies Through npm
2. Do a One-Time Gradle Setup
3. Use the cec Command-Line Utility
4. Test with a Local Test Harness

Install Dependencies Through npm

Use npm (node package manager) to install sites dependencies for your project.

If you are using a proxy to access the Internet, set the proxy for npm with the `npm config` command. See https://docs.npmjs.com/misc/config. To set the proxy for bower, see https://bower.io/docs/config/.

Note:

Ensure that you have Node.js 8.0.0 or later (https://nodejs.org/) installed on your local computer.

In a terminal window, enter these commands:

1. `cd cec-components`
2. `npm install`
3. `npm run install-cec`

   • **Mac:** Run the following command because it adds cec to /usr/local/bin:

     ```
     sudo npm run install-cec
     ```

   • **Windows:**
If the `npm run install-cec` command fails, run the following commands:

```bash
cd bin\cec
npm install

cd ..\..

npm run install-cec
```

**Note:**

In Windows, you can run `start npm start` to start this in a new Command Prompt window.

---

### Do a One-Time Gradle Setup

Set up your local development environment to use a Content and Experience Cloud instance for local testing of components that require services like Process Cloud Service.

From your local machine, set up gradle properties so you can test the components that depend on the Content and Experience Cloud server with the local test harness; for example, `Sample-File-List`. This configuration is also used to deploy components to Content and Experience Cloud.

This gradle setup is required *only* for using Content and Experience Cloud dependent components, like File List, Folder List, Document Manager, Process Cloud Service components, and VBCS components, or for deploying components or templates directly to Content and Experience Cloud.

**Note:**

You need to have Git installed on your local machine before you do the gradle setup.

To do a gradle setup:

1. Enter `cd cec-components` in a terminal window.
2. Enter `mkdir ~/.gradle`
3. Enter `cp gradle.properties ~/.gradle`
4. Edit `~/.gradle/gradle.properties` to configure a Content and Experience Cloud instance and a proxy service:
   - `cec_url=https://<documents-hostname>:443`
   - `cec_username=your.username`
   - `cec_password=your.password`

**Note:**

In Windows, replace `~` with your home directory.

6. To test Sample-Documents-Manager locally, enable embedded content on the Content and Experience Cloud server:
   b. Under Embedded Content, click Enabled.
   c. Enter localhost* in the Allowed domains field.

Use the cec Command-Line Utility

The cec cross-platform command-line utility provides commands for creating and managing templates and components.

Before you use the cec command-line utility, set up gradle properties in your local development environment for commands that require a connection to the Content and Experience Cloud server. See Do a One-Time Gradle Setup.

Integrated help provides information to run the commands, with examples. To view the cec integrated help, you can type cec commands in the command line:

- In a terminal window, go to the cec-components directory.
- Type cec to list the cec commands.

```
cec-components> cec
```

**Templates**
- `cec create-template <name>`                        Creates the template <name>.
  [alias: ct]
- `cec download-template <name>`                      Downloads the template <name> from the Content and Experience Cloud server. [alias: dwt]
  - `cec copy-template <source> [<destination>]`     Copies an existing template named <source> to <destination>. [alias: cpt]
    - `cec import-template <zip>`                     Imports a template from <zip>. [alias: it]
  - `cec export-template <name>`                      Exports the template <name> as a zip file. [alias: et]
    - `cec upload-template <name>`                    Uploads the template <name> to the Content and Experience Cloud server. [alias: ut]
    - `cec delete-template <name>`                    Deletes the template <name> on the Content and Experience Cloud server. [alias: dlt]
```
cec describe-template <name>                      Describes the
   template <name> package.                        [alias: dst]

Themes  
   cec add-component-to-theme <component>         Adds a component to
     a theme.                                   [alias: actt]
   cec remove-component-from-theme <component>   Removes a component
     from a theme.                              [alias: rcft]

Sites   
   cec create-site <name>                       Creates Enterprise
     Site <name>.                               [alias: cs]
   cec update-site <name>                      Update Enterprise
     Site <name>.                               [alias: us]
   cec index-site <site>                       Index the page
     content of site <site> on CEC server.       [alias: is]
   cec create-site-map <site>                  Creates a site map
     for site <site> on CEC server.             [alias: csm]

Content 
   cec create-contentlayout <name>              Creates a content
     layout based on a content type.            [alias: ccl]
   cec add-contentlayout-mapping <contentlayout> Creates content
     type and content layout mapping for local template. [alias: aclm]
   cec remove-contentlayout-mapping <contentlayout> Removes a content
     layout mapping from a local template.       [alias: rclm]
   cec list-server-content-types                Lists all content
     types from server.                         [alias: lsct]
   cec download-content <channel>               Download content in
     channel <channel> from CEC server.          [alias: dwc]

Translation 
   cec list-translation-jobs                    Lists translation
     jobs.                                      [alias: ltj]
   cec create-translation-job <name>            Creates a
     translation job <name> for a site on CEC server. [alias: ctj]
   cec download-translation-job <name>          Downloads
     translation job <name> from CEC server.     [alias: dtj]
   cec submit-translation-job <name>            Submits translation
     job <name> to translation connection <connection>. [alias: stj]
   cec ingest-translation-job <name>            Gets translated job
     from translation connection and ingest.     [alias: itj]
   cec upload-translation-job <name>            Uploads translation
     job <name> to CEC server.                   [alias: utj]
   cec create-translation-connector <name>      Creates translation
     connector <name>.                          [alias: ctc]
   cec start-translation-connector <name>       Starts translation
     connector <name>.                          [alias: stc]
cec register-translation-connector <name> Registers a translation connector.
[alias: rtc]

Local Environment
  cec register-server <name> Registers a CEC server.
  [alias: rs]
  cec list Lists local resources.
  [alias: l]
  cec install Creates source tree.
  [alias: i]
  cec develop Starts a test server.
  [alias: d]

Options:
  --help, -h Show help
  [boolean]
  --version, -v Show version number
  [boolean]

- Type a cec command, such as cec create-template, to get the help information for the command.

Creates the template <name>. By default, it creates a StarterTemplate. Optionally specify -f <source> to create from different source.

Valid values for <source> are:
  CafeSupremoLite
  JETStarterTemplate
  NewProductLaunch
  StarterTemplate
  BlogTemplate

Options:
  --from, -f <source> Source to create from
  --help, -h Show help
  [boolean]

Examples:
  cec create-template Temp1
  cec create-template Temp2 -f CafeSupremoLite

Typing any cec command without parameters or with -h provides help for the command.
Test with a Local Test Harness

Run your custom components, templates, and content layouts in a local test harness before importing them to Content and Experience Cloud.

To start the local test harness:

1. **Enter** `cd cec-components` in a terminal window.
2. **Enter** `npm start &`
3. **Open a browser** at `http://localhost:8085` to see your components, templates, and content layouts running in the local test harness.
4. You can find your components, templates, themes, and so on, in these directories:
   - `cec-components/src/main/components`
   - `cec-components/src/main/templates`
   - `cec-components/src/main/themes`

Develop for Oracle Content and Experience Cloud with Developer Cloud Service

Sites Toolkit, the Developer Cloud Service integration, helps you develop site templates, themes, custom components, and content layouts for Oracle Content and Experience Cloud.

With Sites Toolkit, you can use asset repositories, files, and folders in Content and Experience Cloud. Sites Toolkit has tools to create and develop custom components and site templates, including themes and content layouts. It includes a Git repository as well as a local test harness for quick, iterative development, and sample unit tests to start with.

Sites Toolkit can help you do the following tasks:

- Set up your local development environment to use a Content and Experience Cloud instance for local development and testing of components, templates, themes, and content layouts
- Create components, site templates, and content layouts from samples, run them in the test harness, explore them, and develop the components, templates, themes, and content layouts in a Developer Cloud Service environment
- Import components and site templates that were created in Content and Experience Cloud into a Developer Cloud Service environment for source management and further development
- Export components, templates, and content layouts from a Developer Cloud Service environment for use in Content and Experience Cloud
- Copy an existing component, template, or content layout
- Write unit tests
- Optimize components
- Deploy your components and templates to Content and Experience Cloud
The following topics describe how to set up Developer Cloud Service for developing custom components, site templates and themes, and content layouts:

1. About Using Developer Cloud Service
2. Sign In to the Developer Cloud Service Console for Content and Experience Cloud
3. Create a Project in Developer Cloud Service
4. Add Oracle Content and Experience Toolkit to the Project Code in the New Git Repository
5. Test Custom Components, Templates, and Content Layouts in a Local Test Harness
6. Merge Changes

The following topics provide information about using the Content and Experience Cloud Sites Toolkit:

• Use the cec Command-Line Utility
• Develop Custom Components with Developer Cloud Service
• Develop Templates with Developer Cloud Service
• Develop Content Layouts

About Using Developer Cloud Service

Oracle Developer Cloud Service is a cloud-based software development Platform as a Service (PaaS) and a hosted environment for your application development infrastructure. It provides an open source standards-based integration to develop, collaborate, and deploy applications within Oracle Cloud.

Developer Cloud Service is a collection of software and services hosted on Oracle Cloud to help you manage the application development life cycle effectively through integration with Git, Maven, issues, and wikis. Using Oracle Developer Cloud Service, you can commit your application source code to the Git repository on Oracle Cloud, track assigned issues and defects online, share information using wiki pages, peer review the source code, and monitor project builds. After successful testing, you can deploy the project to Oracle Content and Experience Cloud.

Sign in to the Developer Cloud Service Console for Oracle Content and Experience Cloud

Start developing your custom components for Content and Experience Cloud on the Developer Cloud Service console.

As an administrator for Oracle Cloud services, you can use My Service Administration to create and manage your Cloud services. If you're a service instance administrator for Oracle Content and Experience Cloud and a service administrator for Standard Developer Service, you can set them up and start using them:

1. Sign in to Oracle Cloud, using the information that was provided for your account.
2. Sign in to My Service Administration to create and manage your Oracle Content and Experience Cloud instance and your Standard Developer Service.
3. Verify your Oracle Developer Cloud Service email, as requested.

4. Set up your Oracle Content and Experience Cloud instance, using the subscription details for your service, and go to the Content and Experience Cloud URL for your instance.

5. Go to your URL for the Standard Developer Service.

6. Sign in to your Oracle Developer Cloud Service account.

Access the Developer Cloud Service URL and sign in to the console.

Create a Project in Developer Cloud Service

You can create a project in Developer Cloud Service using the "Content Experience Cloud" project template.

Or you can create a project with an empty Git repository and import the Oracle Content and Experience Toolkit from your Content and Experience Cloud instance.

Create a Developer Cloud Service Project with an Oracle Content and Experience Cloud Template

Create a project for developing custom components, templates, themes, and content layouts in Developer Cloud Service.

To create a project:

1. After you sign in to the Developer Cloud Service console, click **New Project**.
2. In the list of templates, choose **Content and Experience Cloud**, and then click **Next**.

3. In the properties under Project Properties, choose **CONFLUENCE** in the **Wiki Markup** field.

Create a Project in Developer Cloud Service with a Toolkit Download from Oracle Content and Experience Cloud

Create a project for developing custom components, templates, themes, and content layouts in Developer Cloud Service.

To create a project:

1. After you sign in to the Developer Cloud Service console, click **New Project**.
2. Name your project, enter or select other project details you want, and then click Next.
3. In the list of templates, choose **Initial Repository**, and then click **Next**.

4. In the properties under Project Properties, choose **Empty Repository** for the Initial Repository. Click **Finish**.

### Add Oracle Content and Experience Toolkit to the Project Code in the New Git Repository

You can add Oracle Content and Experience Toolkit to the new, empty Git repository for your project.

1. Under **REPOSITORIES** in your new project, copy the HTTP URL of the project Git repository.

2. Open a terminal window and enter this command: `git clone <your-project>.git`
   
   a. When asked, enter your password for Developer Cloud Service.
   
   b. If you see the error "git is not a command", install Git from [https://git-scm.com/downloads](https://git-scm.com/downloads), and then reenter the `git clone` command.

3. `git clone git@github.com:oracle/content-and-experience-toolkit.git`

   Or you can download from here: [https://github.com/oracle/content-and-experience-toolkit/archive/master.zip](https://github.com/oracle/content-and-experience-toolkit/archive/master.zip)

4. `cp -R content-and-experience-toolkit/sites/cec-components <your-project>`

5. `cd <your-project>`

6. `git add cec-components`

7. `git commit -a -m "<your comments>"`

8. `git push`
Test Custom Components, Templates, and Content Layouts in a Local Test Harness

Run your custom components, templates, and content layouts in a local test harness before importing them to Content and Experience Cloud.

To start the local test harness:
1. Enter `cd cec-components` in a terminal window.
   
   Enter `npm start &`

2. Open a browser at `http://localhost:8085` to see your components, templates, and content layouts running in the local test harness.

Merge Changes

After you create a component, template, or content layout or edit source code on your local machine, you need to merge new and changed components and templates into your project's Git repository.

To merge changes into your Git repository, enter the following commands, in order, in a terminal window.

```
cd cec-components
git pull

git add .
git status

git commit -a -m "Your comments"
git pull

git push
```

Index Site Pages with Oracle Content and Experience Toolkit

You can use Oracle Content and Experience Toolkit to create content items for text on site pages and to enable page search for a site.

The following sections describe how to index site pages with Oracle Content and Experience Toolkit:

1. Create the Content Type for Site Page Text
2. Create Page Index Content Items with Oracle Content and Experience Toolkit
3. Add Content Search to a Site in Oracle Content and Experience Cloud

Create the Content Type for Site Page Text

For a content type, you specify a name, required field values, a default content layout for the type.

- Type name
  
  Specify any valid content type name.

- Fields
The following fields are required.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Number of data field values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>site</td>
<td>Text</td>
<td>Single</td>
<td>Site name</td>
</tr>
<tr>
<td>pageid</td>
<td>Text</td>
<td>Single</td>
<td>Page ID</td>
</tr>
<tr>
<td>pagename</td>
<td>Text</td>
<td>Single</td>
<td>Page name</td>
</tr>
<tr>
<td>pageurl</td>
<td>Text</td>
<td>Single</td>
<td>Page URL</td>
</tr>
<tr>
<td>pagedescription</td>
<td>Text</td>
<td>Single</td>
<td>Page description</td>
</tr>
<tr>
<td>keywords</td>
<td>Text</td>
<td>Multiple (no max)</td>
<td>All text on the page and the values from all text fields of content items on the page, obtained by the Oracle Content and Experience Toolkit index-site command</td>
</tr>
</tbody>
</table>

```javascript
{{#fields}}
<div class="indextype"></div>
<div>
  <a href="{{pageFullURL}}" title="{{pagename}}">{{pagename}}</a>
</div>
{{/fields}}
```

content.fields.pageFullURL = SCSRenderAPI.getSitePrefix() + content.fields.pageurl;

- **Create a content layout for the type.**

  The content layout should display the site name and the URL to navigate to the page. For example, in layout.html:

  ```javascript
  {{#fields}}
  <div class="indextype"></div>
  <div>
    <a href="{{pageFullURL}}" title="{{pagename}}">{{pagename}}</a>
  </div>
  {{/fields}}
  ```

- **In render.js, generate the page full URL:**

  ```javascript
  content.fields.pageFullURL = SCSRenderAPI.getSitePrefix() + content.fields.pageurl;
  ```

- **Set the content layout as the default content layout for the type.**

  ```javascript
  content.fields.pageFullURL = SCSRenderAPI.getSitePrefix() + content.fields.pageurl;
  ```
Create Page Index Content Items with Oracle Content and Experience Toolkit

You can use an Oracle Content and Experience Toolkit command to create page index content items.

Prerequisites:

- Oracle Content and Experience Toolkit has been installed and set up on your local machine.
- The gradle properties are configured for the Oracle Content and Experience Cloud instance.
- The site on Oracle Content and Experience Cloud has been published.
- The content items on the site page have been published to the site channel.

In a command-line interface, type the following Oracle Content and Experience Toolkit command:

```bash
cec index-site site name -c content type name -p
```

In the command, `site name` is the name of the site, `content type name` is the content type created for the page text; and the option `-p` indicates to publish the page index content items after creation.

Usage: `cec index-site <site>`

Create content item for each page with all text on the page. If the page index content item already exists for a page, updated it with latest text on the page. Specify `-c <contenttype>` to set the page index content type. Optionally specify `-p` to publish the page index items after creation or update.

Options:

- `--contenttype, -c <contenttype>` page index content type
- `--publish, -p` publish page index items
- `--help, -h` Show help

[boolean]

Examples:

- `cec index-site Site1 -c PageIndex`
- `cec index-site Site1 -c PageIndex -p`

To see the usage, you can type `cec index-site -h`

Add Content Search to a Site in Oracle Content and Experience Cloud

You can add content search to an Oracle Content and experience site with a search page and search field.
To add content search to a site:

1. **Add a Search Page to the Site**
2. **Add a Search Field to the Theme**

### Add a Search Page to the Site

You can add a search page to a site and a Content List component to the search page.

Add the search page:

1. Add a page to the site and set it as a search page.
2. Add a Content List component to the search page.
3. Set **Content Type** to the page index content type created previously.

### Add a Search Field to the Theme

To make a search field show on every page of a site, you can add the search field to the theme’s layout HTML page.

For example:

```html
<div align="center">
<input id="searchonpage" type="text" size="30" placeholder="Search on page..."/>
</div>
```

1. **Add the input field**:

```javascript
<script>
    // Get the search field element
    const node = document.getElementById('searchonpage');
    // Get the search string from the url if it exists
    var params = (new URL(document.location)).searchParams;
    var defaultStr = params && params.get('default');
    if (defaultStr) {
        if (defaultStr.lastIndexOf('*') === defaultStr.length - 1) {
            defaultStr = defaultStr.substring(0, defaultStr.length - 1);
        }
    // Display the search string in the search field
    node.value = defaultStr;

    // When enter from the search field, go to the site search page
    // with the search string
    node.addEventListener('keydown', function onEvent(event) {
        if (event.key === "Enter") {
            var inputElem = event.srcElement || event.target;
            var siteSearchPageUrl = 'search.html';
            var searchUrl = SCSRenderAPI.getSitePrefix() + siteSearchPageUrl + '?' + 'contentType=indextype&default=' + inputElem.value + '*';
            window.location = searchUrl;
    });
</script>
```
Create a New Site or Asset Translation Job in the Oracle Content and Experience Cloud Server

Use Oracle Content and Experience Toolkit to create a translation job for a site or an asset in Oracle Content and Experience Cloud.

Before you can index a multilingual site, you need a translation job. To create a translation job:

1. Click Translate on the top menu of the Sites page.
2. Enter a name for the job in the Create Translation Job dialog, and choose the default source language, the target languages, and the translation job contents.

You can choose to have your translation package include all site content and targeted assets, only site content, or only assets targeted to the site’s publishing channel.

3. Click Create to create the translation job.
4. Use an Oracle Content and Experience Toolkit command to list the available jobs:

```
  cec components> cec list-translation-jobs
  Asset translation jobs:
  Name Language Target Languages Status Source
  demol English (United States) (en-US) Pending Languages

  Site translation jobs:
  Name Language Target Languages Status Source
  demol English (United States) (en-US) READY en-
```
5. Download your translation job:

```
cc components> cec download-translation-job demo1
- translation job downloaded to /Users/<user-name>/Dev/webclient/developers/sites-toolkit/cec-components/demo.zip
- update the translation job status to INPROGRESS.
cc components> cec translate demo1.zip -l all -t demo1-xlate.zip
- target languages: fr-FR,es-ES
- translation finished: /Users/<user-name>/Dev/webclient/developers/sites-toolkit/cec-components/demo1-xlate.zip
```

6. Open the translation bundle and build the folders of resources for languages that you're translating to:

```
Unzip demo1-xlate.zip
_ARCHIVE: demo1-xlate.zip
replace assets/job.json? [n]o, [A]ll, [N]one, [r]ename: A
    inflating assets/job.json
    inflating site/job.json
    inflating assets/es-ES/CORE47653001483240C1AAF180C435F189AB-search_siteSearch202.json
    inflating assets/es-ES/COREA570227E12194356BAA16A80A78A2670-entry1.json
    inflating assets/es-ES/CORED977BC199A3B494596F0D467CAADF7FA-entry2.json
    inflating assets/fr-FR/CORE47653001483240C1AAF18DC435F1B9A8-search_siteSearch202.json
    inflating assets/fr-FR/COREA570227E12194356BAA16A80A78A2670-entry1.json
    inflating assets/fr-FR/CORED977BC199A3B494596F0D467CA4DF7FA-entry2.json
    inflating assets/root/CORE476530014B3240C1AAF18DC435F1B948-search_siteSearch202.json
    inflating assets/root/COREA570227E12194356BAA16A80A78A2870-entry1.json
    inflating assets/root/CORED977BC199A3B494596F0D467CA4DF7FA-entry2.json
    inflating site/es-ES/10.json
    inflating site/es-ES/100.json
    inflating site/es-ES/110.json
    inflating site/es-ES/120.json
    inflating site/es-ES/130.json
    inflating site/es-ES/140.json
    inflating site/es-ES/150.json
    inflating site/es-ES/200.json
    inflating site/es-ES/201.json
    inflating site/es-ES/202.json
    inflating site/es-ES/203.json
    inflating site/es-ES/siteinfo.json
    inflating site/es-ES/structure.json
```
7. Import the translation job:

```
cec-components> cec import-translation-job demo1-xlate.zip
- Logged in to remote server: <server url>
- file demo1-xlate.zip uploaded to home folder, version 1
- importing: percentage 5
- importing: percentage 60
- import demo1 finished
```

**Translate a Site with a Language Service Provider**

You can manage translations of a site for multiple languages with the Oracle Content and Experience Toolkit command-line interface and a Language Service Provider (LSP).

The localization policy for a site specifies a default language, such as English (United States) (en-US), and one or more alternate languages for the site, such as German and French. The text strings for a site can be translated into the specified alternate
languages. If you change the language for the site before translation, the text strings will still appear in the default language.

Oracle Content and Experience Toolkit provides the following translation options in the command-line interface:

Translation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cec list-translation-jobs</td>
<td>Lists translation jobs. [alias: ltj]</td>
</tr>
<tr>
<td>cec create-translation-job &lt;name&gt;</td>
<td>Creates a translation job &lt;name&gt; for a site on CEC server. [alias: ctj]</td>
</tr>
<tr>
<td>cec download-translation-job &lt;name&gt;</td>
<td>Downloads translation job &lt;name&gt; from CEC server. [alias: dtj]</td>
</tr>
<tr>
<td>cec submit-translation-job &lt;name&gt;</td>
<td>Submits translation job &lt;name&gt; to translation connection &lt;connection&gt;. [alias: stj]</td>
</tr>
<tr>
<td>cec ingest-translation-job &lt;name&gt;</td>
<td>Gets translated job &lt;name&gt; from translation connection and ingest. [alias: itj]</td>
</tr>
<tr>
<td>cec upload-translation-job &lt;name&gt;</td>
<td>Uploads translation job &lt;name&gt; to CEC server. [alias: utj]</td>
</tr>
<tr>
<td>cec create-translation-connector &lt;name&gt;</td>
<td>Creates translation connector &lt;name&gt;. [alias: ctc]</td>
</tr>
<tr>
<td>cec start-translation-connector &lt;name&gt;</td>
<td>Starts translation connector &lt;name&gt;. [alias: stc]</td>
</tr>
<tr>
<td>cec register-translation-connector &lt;name&gt;</td>
<td>Registers a translation connector. [alias: rtc]</td>
</tr>
</tbody>
</table>

You can use the cec list-translation-jobs command to list the translation jobs that are already on the server. For example:

```
cec ltj -s
Server: <server-name>
Asset translation jobs:
Name                Status     Source Language
Target Languages    Pending Languages
Name testHash       INPROGRESS en-US     fr-FR,de-DE
Site translation jobs:
Name                Status     Source Language
Target Languages    Pending Languages
demoTest            TRANSLATED en-US     de-DE
```

Typing any cec command without parameters or with -h provides help for the command. See Use the cec Command-Line Utility.

The following sections provide information about translating a site with an LSP:

1. Create a Translation Job with Oracle Content and Experience Toolkit
2. List Translation Jobs
3. Download a Translation Job
4. Create a Translation Connector
5. Submit a Translation Job to a Language Service Provider
6. Upload a Translation Job to the Server

Create a Translation Job with Oracle Content and Experience Toolkit

You can use an Oracle Content and Experience Toolkit command to create a site translation job on your local system.

To create a new translation job for a site, use the `cec create-translation-job` command. This command finds all the assets for the site and creates a zip file for everything that needs to be translated from that site.

```
cec create-translation-job FridayDemo -s Take2 -l all
- Logged in to remote server: <server-name>
- establish user session
- site: Take2, default language: en-US
- query channel
- site localization policy: MyLP
- target languages: de-DE, fr-FR
- create translation job submitted
- creating: percentage 50
- translation job FridyDemo created
```

For translation options, see Create a New Site or Asset Translation Job in the Oracle Content and Experience Cloud Server.

List Translation Jobs

You can list translation jobs on the server to verify that your job was created and is ready to work with.

```
cec list-translation-jobs -s
Server: <server-name>
Asset translation jobs:
<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Source Language</th>
<th>Target Languages</th>
<th>Pending Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>testHash</td>
<td>INPROGRESS</td>
<td>en-US</td>
<td>fr-FR, de-DE</td>
<td>fr-FR, de-DE</td>
</tr>
</tbody>
</table>
Site translation jobs:
<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Source Language</th>
<th>Target Languages</th>
<th>Pending Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>demoTest</td>
<td>TRANSLATED</td>
<td>en-US</td>
<td>de-DE, fr-FR</td>
<td>de-DE, fr-FR</td>
</tr>
<tr>
<td>FridayDemo</td>
<td>READY</td>
<td>en-US</td>
<td>de-DE, fr-FR</td>
<td>de-DE, fr-FR</td>
</tr>
</tbody>
</table>
```

Notice that the `FridayDemo` job is in a READY state.

Download a Translation Job

You can download a translation job from the server so you can work with the job on your local system to translate the site with a Language Service Provider.

To download a translation job:
1. Specify your job in the download-translation-job command.

```
cec download-translation-job FridayDemo
- translation job downloaded to <local toolkit directory>/cec-components/dist/main/FridayDemo.zip
- update the translation job status to INPROGRESS
- validate translation file
- Translation job imported to <local toolkit directory>/cec-components/src/main/translationJobs/FridayDemo
```

2. List the translation jobs on the server again.

```
cec list-translation-jobs -s
Server: <server-name>
Asset translation jobs:
Name                Status        Source Language
Target Languages   Pending Languages
fr-FR,de-DE        fr-FR,de-DE
Site translation jobs:
Name                Status        Source Language
Target Languages   Pending Languages
de-DE,fr-FR         de-DE,fr-FR
FridayDemo          INPROGRESS    en-US
de-DE,fr-FR         de-DE,fr-FR
```

Notice that the status of the FridayDemo job has changed to INPROGRESS. Now you can submit the job to your LSP in a zip file.

**Create a Translation Connector**

A Language Service Provider (LSP) can help you translate a site. With a translation connector to the LSP, you can submit and ingest translation jobs.

Before you submit a translation job, you need to create a translation connector. To translate a site without an LSP, you can create a mock translation connector to run against. Use the `cec create-translation-connector` command to create a translation connector and the `cec start-translation-connector` command to start it:

```
cec create-translation-connector connector1
- translation connector connector1 created at <sites-toolkit folder>/cec-components/src/main/connectors/connector1
- install connector

Start the connector: cec start-translation-connector connector1 [-p <port>]
```

```
cec start-translation-connector connector1 -p 7777
NodeJS running...:
Site page: http://localhost:7777
```

Use the Oracle Content and Experience Toolkit to test the translation connector by running it through the expected APIs:
1. Register the connector with Oracle Content and Experience Toolkit.

   > cec register-translation-connector

2. Open up the toolkit and go to the "Translation Connections" page.


3. Run through the steps on the translation connector validation page. These steps use the translationBundle.zip file in the /data folder in your connector environment to validate your connector.

You can use the Translation Connector SDK to develop a translation connector for Oracle Content and Experience Cloud. This SDK is a sample NodeJS implementation of the Translation connector API. The sample accepts an Oracle Content and Experience translation job zip file, translates all the resource in the file, and returns a new zip file containing all the translations.

The SDK requires the user to have access to an LSP to do the actual string translations. A mock LSP server is included in the SDK to mimic the responses from an LSP by simply prepending the targeted locales to the strings.

The Translation Connector SDK consist of three main modules.

- **Connector**: The translation connector that implements the required Oracle Content and Experience Translation Connector API.
- **Job Manager**: A file system-based sample job manager that maintains the state of the connector jobs while they are translated by the Language Service Provider.
- **Provider**: The implementation of the specific set of APIs required by your LSP to submit documents for translation and retrieve the translated documents.

You can copy the mock translation Provider JS and implement all the methods inside it.

### Submit a Translation Job to a Language Service Provider

Oracle Content and Experience Toolkit provides a zip file that you can send to a Language Service Provider to start work on a translation job.

You can submit the translation job to the LSP through your translation connector. The submission takes awhile because the connector needs to unzip the file and submit all of the individual files to the LSP. Then the LSP can create a project for your translation job. Once the files have been imported into the project, you can start selecting files for translations. Then the LSP starts monitoring the status of the translations.

To check the status, list your translation jobs locally, using the cec list-translation-jobs command with no options. When the status of your job is READY TO INGEST, you can download a zip file from the LSP to ingest the translation job. The translation connector has submitted your zip file to the LSP, the LSP has translated the list of files, and the connector has retrieved the files back from the LSP, in a zip file that you can download and ingest.

```
  cec list-translation-jobs
  Local translation jobs:
  Name                                         Status     Source Language
  Target Languages
```

---

Chapter 3

Translate a Site with a Language Service Provider

---

3-23
Ingesting the zip file pulls the translation job back from the connector, into your Oracle Content and Experience Toolkit.

```bash
cec ingest-translation-job FridayDemo
- use connection <lsp name>
- query translation connection to get job status
- get translation
- translation saved to <sites-toolkit folder>/cec-components/dist/
FridayDemo-translated.zip
- validate translation file
- translation job ingested to <sites-toolkit folder>/cec-components/src/
  main/translationJobs/FridayDemo
```

After you ingest the zip file, when you list translation jobs locally, the status of your translation job is **TRANSLATED**.

```bash
cec list-translation-jobs
Local translation jobs:
Name                                    Status             Source Language
Target Languages
FridayDemo                              TRANSLATED         en-US
  de-DE,fr-FR
demoTest                                READY TO INGEST    en-US
  de-DE,fr-FR
```

You can upload the translated job to the Oracle Content and Experience Cloud server. Normally the job will go through an initial quick translation, which is sent back to you for review. Translation of a site can take a few weeks to finish, with ingestion of a translation job returned by the LSP, corrections to the translations, and resubmissions of the translation job.

### Upload a Translation Job to the Server

After you ingest a translation job, you can upload it to the Oracle Content and Experience Cloud server and then check the translation on your site.

Use the `cec upload-translation-job` command to upload your translation zip file to the server.

```bash
cec upload-translation-job FridayDemo
- created translation job zip file <sites-toolkit folder>/cec-components/dist/FridayDemo.zip
- Logged in to remote server: <server-name>
- file FridayDemo.zip uploaded to home folder, version 1
- importing: percentage 5
- importing: percentage 60
```
- importing: percentage 60
- import FridayDemo finished

After you upload your translation job, the status of the job on the server is INPROGRESS:

```
cec list-translation-jobs -s
Server: <server-name>
Asset translation jobs:
Name                  Status        Source Language
Target Languages
testHash              INPROGRESS    en-US           fr-FR, de-DE
Site translation jobs:
Name                  Status        Source Language
Target Languages
demoTest              TRANSLATED    en-US           de-DE, fr-FR
FridayDemo            INPROGRESS    en-US           de-DE, fr-FR
```

To verify the translation, you can check the text strings in the assets on the site being translated.

## Index a Multilingual Site with Oracle Content and Experience Toolkit

You can use Oracle Content and Experience Toolkit to index multilingual (MLS) sites for translations and for searching pages and content items.

You can build a multilingual site index and test it before publishing the site. Use the Oracle Content and Experience Toolkit `cec index-site` command to index a multilingual site. Go to the `cec-components` directory and issue this command without any options to view the help information for the command:

```
cec-components> cec index-site
Usage: cec index-site <site>
```

Create content item for each page with all text on the page. If the page index content item already exists for a page, update it with latest text on the page. Specify `-c <contenttype>` to set the page index content type. Optionally specify `-p` to publish the page index items after creation or update.

Options:

```
--contenttype, -c <contenttype> page index content type
--publish, -p       publish page index items
--help, -h          Show
```

Help:

```
[boolean]
```

Examples:
Page index items exist per page and per language. The page index content items created for each language are created as translations of the default language page index items. When you do a query in the running site, the search and the content list pick up the language from the site URL. This filters the search automatically.

Before you can publish a multilingual site, you need to index and translate it, for which you will need a translation job. See Create a New Site or Asset Translation Job in the Oracle Content and Experience Cloud Server.

The default language, English, is required. For each language supported (required and optional), execute the index creation and create translations of index items. If you run the index twice, it just does an update.

To index, translate, and publish a multilingual site using Oracle Content and Experience Toolkit commands:

1. Create a content type for the site and make it available in the repository. See Create the Content Type for Site Page Text.

2. Select a validation policy.
   a. Click Assets in the left navigation menu.
   b. Choose Localization Policies in the Assets menu.
   c. Select a localization policy.
   d. Modify the localization policy, if necessary, to include the languages you want to use for indexing and translating the site. For example, if the policy has only English, you can add French and Spanish.

   All translations are done from from English.

3. Download a translation job. Before downloading, you can create You can translate only the assets that are used in the site.

4. Use the cec index-site command to index and translate the site. Specify the -c search_content_type option. This translates

   You can also specify the -p option to publish the site. Then you can validate the indexing and translation before publishing the changes to the live site.

   For example, the following cec index-site command builds a site index for a site that uses English, French, and Spanish. The languages supported by the site are from the assigned L10n policy, including the default language.

cce index-site Demo2 -c search_content_type -p
- Logged in to remote server: server-URL
  - establish user session
  - get CSRF token
  - site: Demo2, default language: en-US, channel token: channel-token
  - site localization policy: search_localization_policy
  - query site repository
  - query content type search_content_type
  - query site structure
  - content types used in the site: search_blog
  - query page data
  - query content on the pages
will create 11 page index items
will update 0 page index items
will remove 1 page index items
create page index item for Blog
create page index item for Privacy Policy
create page index item for Search
create page index item for Components
create page index item for Navigation
create page index item for Detail Page
create page index item for Pages
create page index item for Page Content
create page index item for Developing Templates
create page index item for Themes
add page index items to site channel
remove page index items for page Search from site channel
will create/update translate for fr-FR,es-ES
query site structure with locale fr-FR
query page data (fr-FR)
query content on the pages (fr-FR)
will create 11 page index items (fr-FR)
will update 0 page index items (fr-FR)
will remove 1 page index items (fr-FR)
create page index item for Themes (fr-FR)
create page index item for Navigation (fr-FR)
create page index item for Pages (fr-FR)
create page index item for Detail Page (fr-FR)
create page index item for Search (fr-FR)
create page index item for Page Content (fr-FR)
create page index item for Developing Templates (fr-FR)
create page index item for Blog (fr-FR)
create page index item for Home (fr-FR)
create page index item for Privacy Policy (fr-FR)
add page index items to site channel
set page index items in fr-FR as translated
remove page index items for page Search from site channel
query site structure with locale es-ES
query page data (es-ES)
query content on the pages (es-ES)
will create 11 page index items (es-ES)
will update 0 page index items (es-ES)
will remove 1 page index items (es-ES)
create page index item for Pages (en-ES)
create page index item for Home (en-ES)
create page index item for Themes (en-ES)
create page index item for Components (en-ES)
create page index item for Privacy Policy (en-ES)
create page index item for Detail Page (en-ES)
create page index item for Page Content (en-ES)
create page index item for Navigation (en-ES)
create page index item for Developing Templates (en-ES)
create page index item for Search (en-ES)
create page index item for Blog (en-ES)
add page index items to site channel
set page index items in es-ES as translated
publish job submitted
5. Publish the site to include translations.

Generate a Site Map for a Multilingual Site

Use Oracle Content and Experience Toolkit to generate a site map for a multilingual site and publish the map to the site.

You can use the `cec create-site-map <site>` command to create a site map for a multilingual site on an Oracle Content and Experience Cloud server. For example:

```
cec create-site-map Site1 -u http://www.example.com/site1
```

This command traverses the site structure, produces a site map hierarchy that matches the site page hierarchy, and creates a site map at the specified site URL on the Oracle Content and Experience Cloud server.

Command options follow:

```
--url, -u <url> Site URL [required]
--changefreq, -c How frequently the page is likely to change
--file, -f Name of the generated site map file
--publish, -p Upload the site map to CEC server after creation
--help, -h Show help [boolean]
```

Valid values for the `<changefreq>` option follow:

- always
- hourly
- daily
- weekly
- monthly
- yearly
- never

Examples of the `cec create-site-map` command follow:

```
cec create-site-map Site1 -u http://www.example.com/site1
```

```
cec create-site-map Site1 -u http://www.example.com/site1 -f sitemap.xml
```

```
cec create-site-map Site1 -u http://www.example.com/site1 -p
```

```
cec create-site-map Site1 -u http://www.example.com/site1 -c weekly -p
```
To publish a site map, a site update is created, the site map is updated, and then the update is committed.

Create a Simplified Component for Easy Component Development

Use Oracle Content and Experience Toolkit to create a simplified component for easier development.

SimpleHTML, a simplified component, is available in Oracle Content and Experience Toolkit to give you an easier starting point for custom component development:

```bash
cec create-component -f SimpleHTML
```

A sample for JET component is also available for you to start with:

```bash
cec create-component MyComp -f JET-CCA-Demo-Card
```

See Develop a Custom Component for Oracle Content and Experience Cloud.
Starter Site CLI for React Development

The Starter Site CLI for Oracle Content and Experience Cloud is a quick way to get started with React development, and it requires no build configuration.

In this command-line interface, the `create site` command creates a generic site that is built in React and is an independently runnable application. You can generate a site to run in development mode and in production mode.

Topics:

• Install the Starter Site CLI
• Run CLI Commands
• Get Content from Oracle Content and Experience Cloud
• Set Up the Oracle Content and Experience Cloud Server Connection
• Create a Site
• Build a Site
• Run a Site in Development Mode
• Run a Site with Oracle Content and Experience Cloud Server Content
• Build a Site for Production
• Run a Site in Production Mode
• Structure of the React JS Site Template
• Generated Components
• Starter Site Runtime
• Build a Starter Site Template

Install the Starter Site CLI

To install the Starter Site CLI for React development, you can download a ZIP file, unzip it, and use the `npm install` command.

Follow these steps for installation:

1. Get the files for the Starter Site CLI from here:
   
   ```bash
   git clone git@github.com:oracle/content-and-experience-toolkit.git
   ```

   Or you can download from GitHub: https://github.com/oracle/content-and-experience-toolkit

2. cd content-and-experience-toolkit/react-starter-sites
3. If you are behind a corporate web proxy, configure npm to work with your proxy:

   npm config set proxy http://proxy.company.example.com:8080
   npm config set https-proxy http://proxy.company.example.com:8080

4. Run the npm install command:

   npm install -g

   For a Mac, run the following command instead:

   sudo npm install -g

If you want to reinstall the CLI, uninstall it first:

   npm uninstall -g cecss-cli

   For a Mac, use the following command instead to uninstall the CLI:

   sudo npm uninstall -g cecss-cli

Run CLI Commands

After installation, you can run the command cecss -h to see the usage.

Usage: cecss <command> [options]
Run 'cecss <command> -h' to get the detailed help for the command.
Commands:
   cecss create-site <name>               Creates the site <name> for the
   content from local or from OCE server.
   cecss export-server-content <channel>  Create content template based on
   the channel <channel>, then export and download the archive from OCE
   server.
   cecss list-server-content-types        List all content types from
   server.
   cecss list-server-channels             List all channels from server.
   cecss develop                          Start development server. Watches
   files, rebuilds, and hot reloads if something changes.
   cecss build                            Build an OCE starter site.
   cecss serve                           Serve previously build OCE
   starter site.

Options:
   --help, -h     Show
   help
   [boolean]
   --version, -v  Show version number
Get Content from Oracle Content and Experience Cloud

To create a site to show Oracle Content and Experience Cloud (OCE) content, you need to specify the source of the content.

There are three ways to get OCE content:

- **OCE templates**
  If an OCE template contains content types and content items, you can export it from the OCE server and use the template ZIP file to create a site.

- **Published content from a channel**
  You can use the command `cecss export-server-content` to export all published content items from a channel, and then use the generated ZIP file to create a site.

- **The live content on a OCE server.**
  Example content has been provided in `StarterBlog_export.zip`.

Set Up the Oracle Content and Experience Cloud Server Connection

Some CLI commands require OCE server configuration.

Configure the OCE server before you use the following commands:

```
cecss create-site <name> -s
cecss export-server-content <channel>
cecss list-server-content-types
cecss list-server-channels
```

To configure the OCE server, create the file `.cec_properties` under the user's home directory and configure the server as follows:

```
cec_url=<the oce server url>
cec_username=<user name>
cec_password=<password>
cec_env=pod_ec
```

Create a Site

You can use the `create-site` command and its options to create a site.

Run command `cecss create-site -h` to see the usage and examples:

Usage: cec create-site <name>

Creates the Site `<name>` for the content from local or from OCE server. By default, it creates a StarterSite. Optionally specify `-f <source>` to create from different source.
Options:
--from, -f <source> Source to create from
--content, -c <content> The absolute path of the local OCE template zip file
--server, -s flag to indicate to use the content types from server
--navtypes, -n <navtypes> The comma separated list of content types from server to be used as site navigation
--types, -t <types> The comma separated list of content types on the server, if not specified, all content types will be used
--help, -h Show help

[boolean]

Examples:
  cecss create-site NewSite -c StarterBlog_export.zip
  cecss create-site NewsSite -c ~/Downloads/NewsTemplate.zip
  cecss create-site NewsSite -f ~/Downloads/ReactSiteTemplate.zip -c ~/Downloads/NewsTemplate.zip
  cecss create-site BlogSite -s -n Blog
  cecss create-site BlogSite -s -n Blog -t Blog,Author

Content Type Restriction

If the name of a content type contains character “-” or starts with a digital character, the generated React component for this type won’t compile, and thus the site won’t be runnable. Do not use a content type whose name violates the restrictions.

Create a Site with Local Content

The local content can be the ZIP file of an exported OCE template or a ZIP file of exported channel content. For each content type in the ZIP file, a React component will be generated and the type will also be used for site navigation.

  cecss create-site NewsSite -c ~/Downloads/NewsTemplate.zip

The site created will be placed in a folder with the same name as the site.

Create a Site with OCE Server Content

To create a site with content types from a OCE server, you need to configure the OCE server first. At least one content type should be specified to use for site navigation.

- Create a site for all content types on the OCE server, specifying which types to use in the navigation:

  cecss create-site serverSite -s -n Article
  cecss create-site serverSite -s -n Article,Author
• Create a site with certain content types on the OCE server. The content types used for navigation will be included automatically.

  `cecss create-site serverSite -s -n Article,Author -t Employee`

### Build a Site

Before a site can be run, dependencies need to be fetched. This can be done with the `npm install` command.

If a site is later edited, there is no need to run `npm install` again unless you want to add new dependencies.

After you create a site, run the `npm install` command.

```bash
cd <site name>
npm install
```

### Run a Site in Development Mode

Running a site in development mode starts a NodeJS server and allows changes to the site to be automatically deployed to the running server.

```bash
cd <site name>
cecss develop
```

This command will start a hot-reloading development environment, and the site can be viewed in a browser using the following URL:

```
http://localhost:9090/
```

After JavaScript, html, and css files are updated under src, the saved changes will live reload in the browser.

### Run a Site with Oracle Content and Experience Cloud Server Content

For a site created with content types from an OCE server, it's required to set up the site to use the OCE server to get the content. Also, a site created from a local content ZIP file can be changed to run with content from an OCE server in the same way.

For each created site, an empty `.cec_properties` file is generated in the site directory:

```
#
# To show the content on OCE server, cec_url and cec_channel_token must be set.
# If the channel is secure, cec_username and cec_password are also required.
# Only published items will be displayed.
# If the OCE instance is a development env, set cec_env to dev_ec.
```
Edit the file and set the url and channel token, and username, password if needed. You can get the channel token from OCE server or use CLI:

```bash
cecss list-server-channels
```

# Build a Site for Production

The `build` command will generate an optimized production build of the site.

```bash
cecss build
```

# Run a Site in Production Mode

Running in production mode optimizes a site and starts an optimized server, without hot-update monitoring.

```bash
cd <site name>
cecss serve
```

This command will start a local node server for the site, and the site can be viewed in a browser using the following URL:

```bash
http://localhost:8080/
```

To run the site on a different port, use:

```bash
cecss serve -p <port>
```

# Structure of the React JS Site Template

The `StarterSite.zip` site template is used to create a site by default. This template comes with the CLI.

You can find it here:

- **Windows**:
  ```bash
  C:\Users\<userid>\AppData\Roaming\npm\node_modules\cecss-cli\data
  ```
In the template:

- `index.html` is the page template.
- `index.js` is the JavaScript entry point.
- `Constants.js` is where constants are defined.
- `app/App.js` is a React component that is the main parent component of the Simple Page App. The React router is used to decide which component to show and which to hide.
- `assets/app.css` is the css used by the site.
- `{{types}}/{{name}}.js` is a placeholder. React components will be generated for each content type to render the content item, content list, or search result of this content type, based on parameters.
- `common/ContentItem.js` is a React component that will render an item with passed-in layout, and will also be responsible for kicking off the item query.
• common/ContentList.js is a React component that will render the content list with passed-in layout, and will also be responsible for kicking off the items query.

• common/ItemMultiValues.js is a React component that will render an item’s field with multiple values.

• common/queryItems.js contains JavaScript APIs that create Redux actions to fetch an item or items.

• common/queryReducer.js is the reducer that saves the query result into the Redux store when it receives the fetch success action. It’ll also set a loading flag to true when the fetch begins, and false when it finishes or fails.

• common/Searchbar.js is a React component that will render the search field, and also be responsible for kicking off the search for content items.

After a site is responsible, all the source code is under <site name>/SRC/.

In the SRC directory:

• content is the folder that contains the OCE content.
Starter_Blog_Author/Starter_Blog_Author.js contains React components generated based on the placeholder in the site template for the content type Starter_Blog_Author.

Starter_Blog_Post/Starter_Blog_Post.js contains React components generated based on the placeholder in the site template for the content type Starter_Blog_Post.

Generated Components

For each content type, React components are generated to render the content item, content list, or search result, based on the parameters.

The component can be called as follows:

```jsx
<Starter_Blog_Author />
```

Supported parameters to the component:

- id
- search
- limit
- orderBy (name:asc | name:des | updatedDate:des | updatedDate:asc)

The parameters should be passed in inside the `match.params` object:

```jsx
{...
  match: {
    params: {
    }
  }
}
```

For example:

```jsx
class Starter_Blog_PostDetail extends React.Component {
  render() {
    var item = this.props.item;
    if (!item) {
      return (
        <div />
      );
    }
    var authorId = item.fields['starter-blog-post_author'] ?
      item.fields['starter-blog-post_author']['id'] : '':
    var authorProps = {match: {params: {id: authorId}}};
    return {
      <div>
        <div className="Starter_Blog_Post">
          <span>{item.fields['starter-blog-post_title']}</span>
          <span>{item.fields['starter-blog-post_summary']}</span>
        </div>
      </div>
    }
  }
}
```
Starter Site Runtime

When a site is running in development node, two servers are running. One is a node express server with Webpack middleware that helps with live and hot reloading. The other is a node express server that handles content query.

In the production mode, there is only one server running. All the client-side code will be bundled into static files using Webpack, and it will be served by the node express server.

Details about the starter site runtime follow.

In the starter site runtime:

- `.babelrc` is the Babel config. Babel is used to compile and also to cover JSX to javascript.
- `package.json` contains the site's metadata.
- `server.js` is the entry point to the node express server, which handles content query.
- `webpack-server.js` is the entry point to the node express server with Webpack middleware, which handles live and hot reloading.
- `webpack.config.js` is the Webpack configuration for development mode.
webpack.config-prd.js is the Webpack configuration for production mode.

Build a Starter Site Template

The CLI site generator supports a Mustache template in the site template.

Each js, html, and css file in the site template will be processed and will inject values for the following tags.

<table>
<thead>
<tr>
<th>Tag name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{sitename }}</td>
<td>The site name</td>
</tr>
<tr>
<td>{{types }}</td>
<td>An array of the names of all content types in the site</td>
</tr>
<tr>
<td>{{navtypes }}</td>
<td>An array of the names of all content types in the site used for navigation</td>
</tr>
</tbody>
</table>

If the site template contains a {{types}} directory, the site generator will create a directory for each content type by copying this directory, and a JavaScript file will also created by copying from {{name}}.js. The following tags in the JavaScript file will be injected with values.

<table>
<thead>
<tr>
<th>Tag name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{sitename }}</td>
<td>The site name</td>
</tr>
<tr>
<td>{{type }}</td>
<td>The name of the content type</td>
</tr>
<tr>
<td>{{fields }}</td>
<td>The array of field objects</td>
</tr>
</tbody>
</table>

For each field object, the following tags are supported.

<table>
<thead>
<tr>
<th>Tag name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{name }}</td>
<td>The field name.</td>
</tr>
<tr>
<td>{{&lt;field attribute name&gt;}}</td>
<td>All field attributes returned from the CAAS API /content/management/api/v1.1/types/&lt;type name&gt; are applicable, such as id, description, datatype, required, valuecount, and so on.</td>
</tr>
<tr>
<td>{{__render .single}}</td>
<td>A Boolean flag to indicate the field has a single value.</td>
</tr>
<tr>
<td>{{__render .multiple}}</td>
<td>A Boolean flag to indicate the field has multiple values. You can use the component ItemMultiValues to render: for example, &lt;ItemMultiValues type='direct ' values={{item .fields['{{name }}']}}/&gt;</td>
</tr>
<tr>
<td>{{__render .direct}}</td>
<td>A Boolean flag to indicate to render the item directly on the page: for example, &lt;span&gt;{{item .fields['{{name }}']}}&lt;/span&gt;</td>
</tr>
<tr>
<td>{{__render .image}}</td>
<td>A Boolean flag to indicate the field is an image.</td>
</tr>
<tr>
<td>{{__render.datetime}}</td>
<td>A Boolean flag to indicate the field is a DateTime type, and you can get the value as {{item .fields['{{name }}']}}[value'].</td>
</tr>
<tr>
<td>{{__render.richtext}}</td>
<td>A Boolean flag to indicate the field value is rich text html.</td>
</tr>
<tr>
<td>{{__render.reference}}</td>
<td>A Boolean flag to indicate the field is a reference to another item. You can render this field as a link pointing to the referenced item.</td>
</tr>
</tbody>
</table>
When you create a site template ZIP file, make sure you do not include the top-level folder.
Develop Templates

A template package contains the development version of a site, a theme with page layouts, style and navigation, and associated components used in the site. Oracle Content and Experience Cloud provides a set of templates to be used for creating sites and as starting points for creating custom templates.

- About Templates
- Basic Template Structure
- Create a Template
- Export a Template
- Import a Template
- Work with a Starter Template
- Create a Site Template from Bootstrap or a Website Design Template
- Develop Templates with Developer Cloud Service

About Templates

A template contains all the pieces that users need to start creating a website, including a site with sample pages and content, a theme with styling, navigation, assets such as images, and associated components.

Oracle Content and Experience Cloud provides a number of templates for use in creating sites. These templates are typically installed by your administrator when the service is initialized. See Configure Sites Settings in Administering Oracle Content and Experience Cloud.

Whenever you create a new site, you must select a template. Templates combine themes with sites and components to drive a function or solution, such as a partner portal or a marketing campaign.

For a list of out-of-the-box templates, see Understand Templates in Creating Experiences with Oracle Content and Experience Cloud.

While both developers and users can create new templates and modify and replace existing templates, one of the main tasks for developers is to design new templates. This process basically consists of these steps.

1. Create a new template by copying an existing template, such as the JET Starter Template. This also gets you the theme associated with the template.
   
   For example:

   cec create-template My_JET_Template -f JETStarterTemplate

2. Export the template in a .zip file to your development environment.

3. Open the files in the template package and make your changes.

5. Use the Oracle Content and Experience Cloud interface to import the new template to your instance.

6. Share the template so others can use it.

An alternative is to use the Oracle Content and Experience Cloud interface to modify the template by adding and modifying page layouts and assets, expand the site structure, add components to site pages, and add seed content you want to appear in any sites that use the theme in this template.

See also Manage Templates in Creating Experiences with Oracle Content and Experience Cloud.

Basic Template Structure

The basic structure of a template includes a site (with assets, layouts, pages, and content), an associated theme, and any custom components.

When you create a site, you must choose a template to provide the site structure and initial content, a theme with design and layout specifications, and any custom components.

A template is organized in a specific structure, as illustrated in this example showing basic folders and files:

template_name
  components
    component_name
  assets
    render.js
    settings.html
    _folder.json
    _folder_icon.jpg
    appinfo.json
  template
    assets
    content
    layouts
    pages
      100.json
      200.json
      300.json
      400.json
    variants
      _folder.json
      _folder_icon.png
    componentsused.json
    controller.html
    siteinfo.json
    structure.json
  theme
    assets
      css
        main.css
      js
Component and theme folders and files are described elsewhere in this guide. See About Developing Components and Basic Theme Structure.

**Notes:**

- Generally a theme is shared between templates, unless you use the JET Starter Template or Starter Template, which uses a copy of a theme. See Work with a Starter Template.
- The theme no longer contains the site. The site folders and files are in the `/template` folder.

The `template_name/template` folder contains the folders and files for the site.

- **assets:** Contains images that are displayed in the template details page in the user interface.
- **content:** Contains managed content used in the site.
- **layouts:** Not used at this time.
- **pages:** Contains all the page JSON files with data. Uses the format `nnn.json`, where `nnn` is the page ID.
- **variants:** Contains details of all the updates for the site.
- **_folder.json:** Contains metadata for the template, such as site author, site name, item GUID, short and long site descriptions.
- **_folder_icon.png:** Represents the site in the user interface.
- **componentsused.json:** (Deprecated.) Records the custom components that are used, if any, within the site. Maintained only for backwards compatibility.
- **controller.html:** Contains the key code that displays the site in a browser. If you want to make changes to this file, Oracle recommends that to do this through the site settings in the interface. You can modify the file offline.
• siteinfo.json: Identifies the site name and the name of the associated theme along with other metadata for the site. Don't modify this file.

• structure.json: Defines the hierarchy of the site for pages (parent and child pages). The Render API can be used to draw out the tree structure when setting up navigation for the site. See Site Navigation and Render API Reference.

Create a Template

If you have a site that you want to use as a starting point for other sites, you can create a template from that site. You also can create a new template by copying an existing template and making changes to the copy.

If you create a template from an existing site, the new template uses a copy of the site as its default site. The template references the theme used by the site and any custom components used in the site pages. The theme and custom components are not copied to the template, but are referenced in the same way they are by the site. The template reflects the site used to create it at the time the template is created. Further changes to the site used to create the template are not reflected in the site stored with the template.

If you create a new template by copying an existing template and renaming the copy, you make changes to the copy. Note that when you copy a template, sharing information for the template isn't copied.

Don’t use the following names for templates, themes, components, sites, or site pages: authsite, content, pages, scstemplate_*, _comps, _components, _compsdelivery, _idservice, _sitescloud, _sitesclouddelivery, _themes,...
themesdelivery. Although you can use the following names for site pages, don’t use them for templates, themes, components, or sites: documents, sites.

If you want to create your own custom template (with site and theme), it’s best to use the JET Starter Template or Starter Template provided by Oracle Content and Experience Cloud, which contains basic elements for a site and an associated theme. A starter template includes information and instructions written into the site pages to help you explore how to layout and design a site and theme in a custom template.

See Manage Templates in *Creating Experiences with Oracle Content and Experience Cloud* and *Work with a Starter Template*.

### Export a Template

You can export a template to modify it offline and then import it either as a new template or to replace the existing template. You can also export a template to move it to another Oracle Content and Experience Cloud instance and import it there.

When you export a template, you essentially copy the template to a folder in Oracle Content and Experience Cloud as a single `.zip` file. You can download the template package directly from the folder to unpack and work with the individual files. When you are done working with the template files, create a `.zip` file that contains the template package and import it into Oracle Content and Experience Cloud to overwrite the original template or create a new one.

#### Note:

When you export a template, sharing information for the template isn’t included.

To export a template:

1. On the Oracle Content and Experience Cloud home page, click **Developer**.
   
   The **Developer** page is displayed.
2. Click **Templates**.
   
   A list of existing templates is displayed.
3. Right-click a template and choose **Export**.
4. Navigate to a folder or create new folder by clicking **Create**, providing a name and an optional description, and clicking **Create**.
   
   To open a folder, click the folder icon or the folder name.
5. Select a folder by clicking the checkbox for the associated folder and click **OK**.
   
   A template package file is created in the selected folder with the template name and a `.zip` extension.
Import a Template

You can export a template to modify it offline and then import it either as a new template or to replace the existing template. You can also export a template to move it to another Oracle Content and Experience Cloud instance and import it there.

When you export a template, you essentially copy the template to a folder in the Oracle Content and Experience Cloud as a single .zip file. You can download the template package directly from the folder to unpack and work with the individual files. When you are done working with the template files, create a .zip file that contains the template package and import it into Oracle Content and Experience Cloud and overwrite the original template or create a new one.

To import a template package:

1. On the Oracle Content and Experience Cloud home page, click Developer
   The Developer page is displayed.
2. Click Templates.
   A list of existing templates is displayed.
3. Click Create and choose Import a template package.
4. If you have uploaded the template package, navigate to the folder that contains the template package. To open a folder, click the folder icon or the folder name.
   If you have not yet uploaded the template package:
   a. Navigate to the folder where you want to upload the template package or create a folder by clicking New, providing a name and an optional description, and clicking Create.
   b. Click Upload.
   c. Locate and select the template package, then click Open.
      A progress bar shows the file name and the upload status.
5. Select a template package by clicking the checkbox next to the file name and click OK.
   If there are no conflicts between the contents of the imported template and any existing templates, themes, or custom components, new Oracle Content and Experience Cloud folders are created for the template, its associated theme, and any custom components.
6. If the template, theme, or custom component names or IDs exist, you are prompted to resolve the conflicts.
   Depending on the nature of the conflict, you are given the option to create a new template, theme, or custom component, or in some cases, you can overwrite the existing template, theme or custom component with the imported version.

Work with a Starter Template

You use a copy of a starter template provided by Oracle Content and Experience Cloud to create a new site, with a theme and custom components.
A starter template collects all the pieces you need to build a custom website in one package: the default site, layout, navigation, sample content, theme, associated content items, and so on. Different from other templates provided with Oracle Content and Experience Cloud, a starter template provides you with an easy-to-use basic framework for creating a new site, including a new theme.

**Note:**

A theme is usually shared between templates, but a starter template uses a copy of a theme.

The starter templates, StarterTemplate and JETStarterTemplate, are provided along with other templates in Oracle Content and Experience Cloud when your administrator enables templates during installation and configuration of the service. The Oracle JavaScript Extension Toolkit (JET) starter template includes the latest JET styling for templates, incorporates some page content (as JET components), and provides starter components for building JET-based templates and sites.

**Basic Process**

Here’s the basic process for working with a starter template:

1. Create a new site and select the starter template. A new starter theme is created along with the new site.

**Note:**

Choose the site name carefully. The name you give the site is duplicated as the name of the new theme, and the theme will be visible to users once you publish the site. You can't change the name of the theme after it is created.

The starter site and theme contain a set of folders and files that are required to start site and theme development.

2. Sync the new theme to your desktop. You can work on your desktop to extend and customize the theme with layouts and static assets. Because this is a copy of a theme and uses a name specific to your template, you can make changes to the theme without affecting the source theme.

3. Open the site in Site Builder. You can use Site Builder to add to the site structure, add components and interactions to pages, and add content that will become part of the default site of the new template. You can reuse or remove the site content provided in the starter template.

4. When the site and theme are ready, sync the changes with your Oracle Content and Experience Cloud instance, then create a new template from the selected site.

5. Share the template so others can use it.

**Create a Template Using a Starter Template**

To use a starter template:
1. On the Oracle Content and Experience Cloud home page, click **Sites**.
   A list of existing sites is displayed.

2. Click **Create**.

3. In the Create Site dialog, select the starter template to use as the basis for your site.
   The new site uses the theme provided with the starter template, renamed to match the new site name.

4. In the dialog, enter a name for the site. This name is used in the site URL. You can use letters, numbers, underscores (_), and hyphens (-). If you enter a space, it's automatically replaced with an underscore.
   Don't use the following names for templates, themes, components, sites, or site pages: authsite, content, pages, scstemplate_*, _comps, _components, _compsdelivery, _idservice, _sitescloud, _sitesclouddelivery, _themes, _themesdelivery. Although you can use the following names for site pages, don't use them for templates, themes, components, or sites: documents, sites.

   **Note:**
   The path for a Content and Experience Cloud site URL is case-sensitive.
   Case in the query or fragment strings is managed by developers in their custom code.
   
   ```
   https://[host]:[port]//[/]path[/?query][/#fragment]
   Location                              Data
   ```

5. Optionally, enter a description for the site.

6. When you're ready, click **Create**.
   A progress bar shows the new site name and creation status. When the site is created, the name appears in the list of sites. Its initial status is offline.
   To quickly find your newly created site in the list, sort the list by **Last Updated**.
   The site you just created will appear at the top of the list.

   **Note:**
   You are automatically assigned the role of manager for the site you created.

7. Use the desktop app to sync the theme to your desktop.
   You should now see the folder hierarchies and files for the theme.

8. The starter theme contains a minimal set of folders and files, like these:

   ```
   theme
   assets
   css
   ```
The x-close.png file contains the default close icon for the cookie consent popup. The /layouts folder contains a starter page layout file (index.html) with the following contents:

- A set of HTML tags that allow the file to be used as a page layout.
- A single slot that has seeded text with instructions, such as how to sync the theme to your desktop, how to add a new page layout, how to add components to the page layout, and how to build site hierarchy using the new page layout.
- A simple JavaScript navigation file that provides an example of how to use the renderer API JavaScript functions and objects. The rendered API is needed for traversing the site hierarchy and generating required HTML markup to allow navigation within the site.

See also Basic Theme Structure.

9. When you’re finished modifying the theme, sync the theme folders and files to your Oracle Content and Experience Cloud instance. To see how your changes to the theme look and behave in a site, open the site in Site Builder. This will likely be an iterative process.

10. To view or modify the site, select the site and click Edit.

11. Toggle the editor mode to Edit so you can make changes to the site. You can modify existing pages and add new pages using the page layout available in the starter themes.
Note:

If you are familiar with the page layout structure and usage, you can delete sections provided by the starter template that you don’t want and switch the layout on the sections that you do want to one of the new layouts.

12. When you are finished making changes to the site, Save the site, then click Publish to merge the update to the base site.

13. Select the site and create a template based on the site. This pulls in the assets and theme for the new template.

14. Share the template with members you want to be able to use the template.

Create a Site Template from Bootstrap or a Website Design Template

The open architecture of Oracle Content and Experience Cloud means you can use work done in other coding frameworks such as Foundation or Bootstrap. With a few changes, you can turn a Bootstrap template into a theme and make it part of an Oracle Content and Experience Cloud template.

Basic Process

Here’s an overview of the steps described in detail in the sections that follow:

1. Prerequisites
2. Create a Site
3. Synchronize the Theme Folder
4. Set Up the Basic Theme
5. Update the Site Pages
6. Update Navigation
7. Update Site Layouts
8. Publish the Site
9. Create the New Template

Prerequisites

- Have the Oracle Content and Experience Cloud desktop app for synchronizing folders and files to your local computer set up and running. See Desktop App Overview in Managing Content with Oracle Content and Experience Cloud.
- Download the Bootstrap template theme folders, files, and content to your local computer and have them ready for use.

Create a Site

Create a site from an Oracle Content and Experience Cloud starter template:

1. On the home page, click Sites.
A list of existing sites is displayed.

2. Click **Create**.

3. In the Create Site dialog, select **JET Starter Template** or **Starter Template** to use as the basis for your site.

4. In the dialog, enter a name for the site. This name is used in the site URL. You can use letters, numbers, underscores (_), and hyphens (-). If you enter a space, it's automatically replaced with a hyphen.

Don't use the following names for templates, themes, components, sites, or site pages: authsite, content, pages, scstemplate_*, _comps, _components, _compsdelivery, _idcservice, _sitescloud, _sitesclouddelivery, _themes, _themesdelivery. Although you can use the following names for site pages, don’t use them for templates, themes, components, or sites: documents, sites.

5. Optionally, enter a description for the site.

6. When you’re ready, click **Create**.

A progress bar shows the new site name and creation status. When the site is created, the name appears in the list of sites. Its initial status is offline.

To quickly find your newly created site in the list, sort the list by **Last Updated**. The site you just created will appear at the top of the list.

### Synchronize the Theme Folder

When you create a site from a starter template, a copy of the starter template theme is created and named with the site name followed by the theme name. For example, the theme for **My_New_Site** is **My_New_SiteTheme**.

Use the desktop app to synchronize the theme folder and files for the site to your local computer. See Get Started with sync in *Managing Content with Oracle Content and Experience Cloud*.

You should now see the template theme folder hierarchy and files on your local desktop. Here’s an example:

```plaintext
theme_name
  assets
    css
      main.css
    js
      topnav.js
```

---

**Note:**

The path for a Content and Experience Cloud site URL is case-sensitive. Case in the query or fragment strings is managed by developers in their custom code.

```
https://[//host[:port]]//path/?query[#fragment]
```

<table>
<thead>
<tr>
<th>Location</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Set Up the Basic Theme

1. Copy these Bootstrap files into the synchronized theme folders on your local desktop to overlay the existing files.
   - html files go into the `theme_name/layouts` folder
   - css files go into the `theme_name/assets/css` folder
   - js files go into the `theme_name/assets/js` folder
   - image files go into the `theme_name/assets/images` folder, which may be grouped in subfolders with images for background, footer, people, and so on

2. Modify the html files in the layout folder to update relative paths and add required elements. For a typical Bootstrap theme there will be many relative paths to the `assets` folder, so you must modify them to point to the theme folder.

   Fix the paths for the css, js, and images folders to use:

   ```
   _scs_theme_root_/assets/css/
   _scs_theme_root_/assets/js/
   _scs_theme_root_/assets/images/
   ```

   **Note:**

   Once this step is completed, the `_scs_theme_root_` part will automatically adjust to the environment the theme is being used in.

3. There are three requirements for each Oracle Content and Experience Cloud layout:
   a. Include the following tags in the `<head>` tag of the html file:
      - `<!--$SCS_RENDER_INFO-->`
      - `<!--$SCS_SITE_HEADER-->`
      - `<!--$SCS_PAGE_HEADER-->`
b. Include the renderer script at the end of the layout files, just inside the `<body>` tag. Both of these paths automatically adjust in Site Builder and the runtime environments.

    <script data-main="/sitescloud/renderer/renderer.js" src="/sitescloud/renderer/require.js"></script>

c. Include the following tags after the include of the `renderer.js` file:

- <!--$SCS_PAGE_FOOTER-->
- <!--$SCS_SITE_FOOTER-->

Verify that the modified theme files are synchronized with the site in Oracle Content and Experience Cloud.

**Update the Site Pages**

When you open the site in Site Builder, you'll initially see the pages that are in the default site. You can delete the pages you don't want and switch the layout on the pages you want to keep to one of the new layouts.

1. On the home page, click **Sites**.
   A list of existing sites is displayed.
2. Right-click a site in the list and choose **Edit**.
3. Enter a name for the update and an optional description, then click **Create**.
   For the update name you can use letters, numbers, underscores (_), and hyphens (-). If you enter a space, it's automatically replaced with a hyphen.
   If you already have updates to the site, select an update from the list and click .
4. Site Builder opens in preview mode. To make changes or to use the navigation options in the sidebar, make sure that the Edit switch is set to **Edit**.
5. To edit a particular page, choose the page using the site tree in the sidebar or using the site's own navigation.
6. To remove a page you don't want, select the page and click .
7. To add a new page, click **Add Page**. You can reposition the page in the site tree by dragging and dropping it.
8. To change the layout associated with a page, choose the page in the site tree and click to display the page settings.
   Go to the **Page Layout** field and select a different layout from the menu. The number and type of page layouts depends on the theme associated with your site.
9. **Save** to save your changes to the current update. You can continue working in the current update or create new updates if needed.

**Update Navigation**

When you look at the navigation in the site preview, it doesn't match the current hierarchy because of the hard-wired navigation in the layout from the Bootstrap theme.

Update the navigation to replace the hard-wired code in the Bootstrap theme with dynamically generated code from the site hierarchy.
Edit your local synchronized copies of the theme files.

1. Remove the hard-wired navigation code from the layouts. Here’s an example of the hard-wired navigation that would have to be removed from a typical Bootstrap theme. This is a typical header section, with the logo, the ‘Toggle Navigation’ parts for the ‘Hamburger’ menu when the page is too narrow (the responsive part) and the hard-wired page navigation for the other pages.

```html
<header id="header" class="header navbar-fixed-top">
  <div class="container">
    <h1 class="logo">
      <a href="index.html"><span class="text">Velocity</span></a>
    </h1><!--//logo-->
    <nav class="main-nav navbar-right" role="navigation">
      <div class="navbar-header">
        <button class="navbar-toggle" type="button" data-toggle="collapse" data-target="#navbar-collapse">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button><!--//nav-toggle-->
      </div><!--//navbar-header-->
      <div id="navbar-collapse" class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li class="active nav-item"><a href="index.html">Home</a></li>
          <li class="nav-item"><a href="features.html">Features</a></li>
          <li class="nav-item"><a href="pricing.html">Pricing</a></li>
          <li class="nav-item dropdown">
            <a class="dropdown-toggle" data-toggle="dropdown" data-hover="dropdown" data-delay="0" data-close-others="false" href="#">Pages <i class="fa fa-angle-down"></i></a>
            <ul class="dropdown-menu">
              <li><a href="download.html">Download Apps</a></li>
              <li><a href="blog.html">Blog</a></li>
              <li><a href="blog-single.html">Blog Single</a></li>
              <li><a href="blog-category.html">Blog Category</a></li>
              <li><a href="blog-archive.html">Blog Archive</a></li>
              <li><a href="about.html">About Us</a></li>
              <li><a href="contact.html">Contact</a></li>
            </ul>
          </li><!--//dropdown-->
          <li class="nav-item"><a href="login.html">Log in</a></li>
        </ul>
      </div><!--//nav-collapse-->
    </nav>
  </div><!--//container-->
</header>
```
2. Write JavaScript code to traverse the site structure information and generate the navigation code, then include the JavaScript code on the layouts, for example:

```javascript
<script type="text/javascript" src="_scs_theme_root_/assets/js/navbar.js"></script>
```

3. Modify the exact output of the `topnav.js` file to match the markup expected in your particular CSS.

**Update Site Layouts**

At this point the site is functional but has no editable areas (slots) in the layouts.

1. Locate or add a DIV element in a layout and designate it as a slot.

   **Slots** are DIV elements in the layout that have the value "scs-slot" in the class attribute. Each slot must have a unique `id` attribute. For example:

   ```html
   <div id="slot-content1" class="scs-slot scs-responsive"></div>
   ```

   To make the slot adjust automatically to the size of the browser viewport, include the `class` attribute "scs-responsive".

2. Repeat this step as needed to create additional slots in the layout or in other layouts.

**Publish the Site**

After you’ve completed and saved all your changes to a site, you must **Publish** the site.

Publishing the site takes all of the changes in the current update and merges that into the base, making what was in the update into the new base site.

**Create the New Template**

You now have a functional site with an associated theme, so you can make this into a template to share with others so they can create sites from it.

1. Open the **Templates** tab in the **Experience** user interface.

2. Click **Create** and choose **From existing site**.

3. Select the new site you’ve created using a starter template and the Bootstrap template.

4. Enter a name for the new template and click **Create**.
5. To package the template for use with other Oracle Content and Experience Cloud instances, select the template and choose the Export menu option to create a .zip file that can be downloaded.

Develop Templates with Developer Cloud Service

You can use Developer Cloud Service to develop templates for Content and Experience Cloud.

Take the following steps to develop a template in Developer Cloud Service, test it locally, and then export it into Content and Experience Cloud:

1. **Set Up Oracle Content and Experience Toolkit on Your Local Machine.**
2. **Sign in to the Developer Cloud Service Console for Content and Experience Cloud.**
3. **Create a Project in Developer Cloud Service.**
4. **Add Oracle Content and Experience Toolkit to the Project Code in the New Git Repository.**
5. **Create a Template in Developer Cloud Service.**
   
   You can create a new template to develop, **copy an existing template in Developer Cloud Service**, or **import a template from Content and Experience Cloud**.
6. **Test the Template in a Local Test Harness.**
7. **Merge Changes.**
8. **Export a template from Developer Cloud Service into Content and Experience Cloud.**

Create Templates in Developer Cloud Service

You can use the cec command-line utility to create Content and Experience Cloud templates from the available source templates.

Use the `cec create-template` command to create a template from one of the available source templates. Typing `cec create-template -h` on the command line gives the available source templates.

Here is an example of creating a template:

```
cec create-template CafeSupremoLite_yourname -f CafeSupremoLite
```

**Windows:** This command creates a symlink for themes to render in an external HTML WYSIWYG editor (such as `_scs_theme_root_`) while you are creating templates. To create symlinks in Windows, you normally need to run the command-line utility with administrative privileges. If you are not using a WYSIWYG editor to edit the theme, you don't need to run with administrative privileges, and you can ignore the symlink creation error.

The preceding example creates the `CafeSupremoLite_yournameTheme` template and makes the source code available at `cec-components/src/main/`. The following table shows the locations of the source code after you create a site template in Developer Cloud Service.
The local test harness shows the components too, with the ability to filter them by template and type.

You can edit theme and component files with any text or code editor. See Test with a Local Test Harness. Refresh the browser after editing the theme or component to see your changes.

### Important:

The source code for your templates, themes, and components exists in `src/main/`. You shouldn’t modify any files outside `src/main` because they are needed for the functioning of the Content and Experience Cloud local server.

---

### Copy a Template in Developer Cloud Service

You can copy an existing Content and Experience Cloud template in Developer Cloud Service.

To copy one of your existing template from `src/main/templates`, use the `cec copy-template` command. The following example copies the `Temp1` template to a new template named `Temp2`:

```
cec copy-template Temp1 -n Temp2
```

---

### Import a Template into Developer Cloud Service

You can import templates from Content and Experience Cloud into Developer Cloud Service for further development.

If you have a template ZIP file created from a Content and Experience Cloud server, you can import that file into Developer Cloud Service for further development, such as to edit the theme or components. Use the following command:

```
cec import-template <location of the template zip file>
```

Specify the folder that contains the ZIP file in Content and Experience Cloud. See About Templates and Export a Template.

---

### Export a Template from Developer Cloud Service

You can export a template ZIP file from Developer Cloud Service and use the file to create a site in Content and Experience Cloud.
Once the template development is complete, you can run the following command to export the template. The command response tells you where the ZIP file for the template is created in Content and Experience Cloud.

```
cec export-template CafeSupremoLite_yourname
```

See About Templates and Import a Template.
Develop Themes

A theme defines the general look-and-feel — the overall style — of a site, including color scheme, font size, font type, and page backgrounds. A theme provides visual consistency between the pages in a site. You can create unique themes and variations of themes, specifying the design and sample content, which then can be used to create sites to promote your brand and your vision.

- About Themes
- Basic Theme Structure
- Site Navigation
- Create a Theme
- Associate Components with Themes
- Render API Reference

About Themes

Themes define the general look-and-feel of a site, including content, appearance, and behavior. A theme provides visual consistency between the pages in a site.

Designing a new theme means specifying the layout, style, sample content, navigation, and all the basic information that serves as a starting point for a new site. Theme designers set the expectations for how a site will look and behave. A theme should be designed keeping in mind the way it will be used; for example, most or all users will be expected to access the site with a mobile device. Designing custom themes is useful if you have users who want to create many similar sites. You can design a theme using page layouts for common patterns that can be shared across themes.
A theme contains page layouts that are used to design content, appearance, and behavior for sites. You change the design and settings, and add content, to create a site that sells your style, your brand, and your vision.

A theme includes:

- Assets for background images or other content that are part of page layouts (images, JavaScript files, and so on)
- Style settings for a site (CSS)
- Several page layouts (HTML files)
- Code to construct navigation for the site (JavaScript files)
- A list of basic styles that can be used with the components (specified in design.css and design.json files)

A theme also can include seed data, which is used to populate a new page created from one of the page templates. For example, a user creates a new page for a Products section and chooses the page layout called new_product.html. If the theme
contains a file called `new_product-pageseed.json`, the new page will be populated with the contents of the page seed file when it’s first created. As with sample content, this seed data can be modified and is only there to provide a starting point for you to build out the page.

You can create a theme that uses a subset of components that are intended to work with that theme. When a user chooses that theme for their site, they'll see only the components that are specified for that theme. See [Associate Components with Themes](#).

Each website uses a theme. When you create the site from a template, you inherit the theme from the template. You can change the theme for a site at any time. Oracle Content and Experience Cloud provides a number of templates with themes that you can use to get started.

If a site uses a new, unpublished theme, the theme is automatically published with the site when you place the site online for the first time. If you make changes to a theme and want to update online sites to show the changes, you must explicitly publish the theme. Only the theme owner or a user with manager privileges can explicitly publish a theme.

Note:

If you publish changes to a theme, all online sites that use the theme will reflect the change. For example, if you change the default font specified in the theme and publish the theme, all sites that use the theme will use the new default font.

See also Manage Themes in *Creating Experiences with Oracle Content and Experience Cloud*.

**Basic Theme Structure**

The basic structure of a theme includes the design, navigation, and styles specified in folders stored in Oracle Content and Experience Cloud. A theme is part of the template for a site.

When a user selects a template to create a new site, the associated theme data is automatically loaded. If you use the JET Starter Template or Starter Template, the theme is automatically copied instead of referenced. As a developer, if you’re using the starter template, you want your own copy of the theme.

A theme is organized in a specific folder and file structure, as illustrated in this example showing basic folders and files:

```plaintext
theme
   assets
      css
         main.css
      js
         topnav.js
   designs
      default
         design.css
```
Certain folders contain specific types of information, including the following folders:

- **assets**: JavaScript, Cascading style sheet (CSS), images, and other support files that are referenced by the layouts.
- **designs**: design.css and design.json files, which are used to specify style options for components.
- **layouts**: HTML files for page templates, which are used to display pages of the site.

**Note:**

All HTML files must have a DOCTYPE element at the start of the file that looks like this: `<!DOCTYPE html>

- **publish**: After a theme has been used in a published site, this directory is listed and contains copies of files. This directory is visible if you have sync'd the theme using the desktop app, but it's not included if you've exported the theme as part of a template.
- **responsepages**: Special page for handling errors (404).

**Note:**

If a page in a site is flagged as an Error Page, then the 404 error message from the theme will be ignored and the designated Error Page is used instead.

- **viewport.json**: Specifies Viewport settings for the theme.
- **_folder.json**: Specifies the name and GUID for the theme. For example:

  ```
  {
    "themeName":"MarketingCampaignTheme",
    "itemGUID":"TB79D65F699B022AC4E11F4D4EE870070A1ADD86BBB"
  }
  ```

The GUID is created by Oracle Content and Experience Cloud when the theme is first imported or when it is copied. The theme name is assigned by the theme developer when creating a theme.

- **components.json**: (Deprecated.) Records the custom components used within the theme. Maintained only for backwards compatibility.
There are two key files that you'll work with when creating a new theme. These files set styles for components:

- design.css
- design.json

It is good practice to put navigation information into one JavaScript file; for example, a file named nav.js. The theme's /assets/js/ folder is a good location for such a file.

Site Navigation

The hierarchy of a site is stored in the structure.json file associated with the site. The hierarchy is loaded into memory and made available in the page context as the SCS.structureMap object.

Site Builder reads the structure.json file to draw the site tree in Site Builder. The structure.json file will contain code for the site pages. For example:

```
"pages": [  
  {  
    "id": 100,  
    "name": "Home",  
    "parentId": null,  
    "pageUrl": "index.html",  
    "hideInNavigation": false,  
    "linkUrl": "",  
    "linkTarget": "",  
    "children": [ 200,  
                  300,  
                  400,  
                  500 ],  
    "overrideUrl":false
  }  

  {  
    "id":200,  
    "name": "Products"  
    "parentId":100,  
    "hideInNavigation":false,  
    "linkUrl": "",  
    "linkTarget": "",  
    "children":{ 204, 205},  
    "overrideUrl":false
  }  

  {  
    "id":204,  
    "name": "Hiking Boots",  
    "parentId":200,  
    "pageUrl":"products/hiking_boots.html",  
    "hideInNavigation":false,  
    "linkUrl": "",  
    "linkTarget": "",  
    "children":[]
  }
```
Navigation JavaScript code is necessary within the site pages to also read that structure and draw out the navigation links for the site. Templates provided with Oracle Content and Experience Cloud include sample navigation JavaScript files that illustrate how this works.

The `topnav.js` file used in some of the themes provided with Oracle Content and Experience Cloud is an example of how you can use the `SCS.structureMap` object along with the Render API calls such as `SCSRenderAPI.getPageLinkData` to traverse the site structure and draw out the HTML markup needed to render the navigation menus in the page. Here is code from the sample `topnav.js` file:

```javascript
function renderNode(id, navBar)
{
    if (id >= 0)
    {
        var navNode = SCS.structureMap[id];
        if (navNode &&
            (typeof navNode.hideInNavigation !== "boolean") ||
            (navNode.hideInNavigation === false))
        {
            var navItem = document.createElement("li");
            var navLink = document.createElement("a");
            var navText = document.createTextNode(navNode.name);

            var linkData = SCSRenderAPI.getPageLinkData(navNode.id) || {};
            if (linkData.href)
            {
                navLink.href = linkData.href;
            }
            if (linkData.target)
            {
                navLink.target = linkData.target;
            }

            navLink.appendChild(navText);
            navItem.appendChild(navLink);

            if (navNode.children.length > 0)
            {
                var navSub = document.createElement("ul");

                for (var c = 0; c < navNode.children.length; c++)
                {
                    renderNode(navNode.children[c], navSub);
                }

                navItem.appendChild(navSub);
            }
            navBar.appendChild(navItem);
        }
    }
}
```

Chapter 6
Site Navigation

6-6
function renderNav()
{
    var topnav = document.getElementById("topnav"); // expected to be an empty <div>
    if (topnav)
        {
            var navBar = document.createElement("ul");
            renderNode(SCS.navigationRoot, navBar);
            topnav.appendChild(navBar);
        }
}

// Must wait for all our script to be ready...
if (document.addEventListener)
{
    document.addEventListener('scsrenderstart', renderNav, false);
}
else if (document.attachEvent)
{
    document.documentElement.scsrenderstart = 0;
    document.documentElement.addEventListener("onpropertychange",
        function(event)
        {
            if (event && (event.propertyName == "scsrenderstart"))
                { renderNav(); }
        });
}

You can use Render API calls to generate navigation links that will function in your site Edit and Preview modes and in a published online site. See Render API Reference.

It is good practice to put navigation information into one JavaScript file, such as topnav.js. The JavaScript file is usually stored in the /assets/js/ folder of the theme, as you can see in the sample themes provided with Oracle Content and Experience Cloud.

Create a Theme

You can create a new theme by copying an existing theme and making changes to the copy. You also can import and use Bootstrap content in a new theme.
Copy a Theme

1. On the home page, click Developer.
   The Developer page is displayed.
2. Click Themes.
   A list of existing themes is displayed.
3. Right-click a theme and choose Copy.
   All the folders and files of the theme are copied, including any sample pages and content.

   **Note:**
   When you copy a theme, sharing information for the theme isn't copied.

4. Enter a name for the copied theme. You can't use a name used by another theme.
   You can use letters, numbers, underscores (_), and hyphens (-) in the name. If you enter a space, it's automatically replaced with a hyphen.
   Don't use the following names for templates, themes, components, sites, or site pages: authsite, content, pages, scstemplate_, _comps, _components,
   _compsdelivery, _idservice, _sitescloud, _sitesclouddelivery, _themes,
   _themessdelivery. Although you can use the following names for site pages, don't use them for templates, themes, components, or sites: documents, sites.
5. Optionally, enter a description for the theme.
6. Click Copy.
   A progress bar shows the new theme name and copy status. When the theme is copied, the name appears in the list of themes. You can explore the folders and files that make up the theme by clicking the theme name in the list of themes.
7. Use the Oracle Content and Experience Cloud desktop app to sync the theme folders and files to your local system. This enables you to browse the local folders and work directly with files. Changes you make to the theme are automatically synced. You can make changes using your favorite HTML, code, or text editing tools.

**Verify the Theme with a Site**

After you've completed editing the theme, you need to check that all the pieces work together with the site as planned.
1. Make sure you’ve synchronized your copied folders and files with the Oracle Content and Experience Cloud desktop app and that all the modifications have been saved.

2. Open a site (it can be a test or existing offline site) in Site Builder and switch the site to use the new theme.

3. Test the site with the theme by adding pages and using settings, viewing images, checking the navigation, and anything else you’ve changed in the new theme.

4. Check that everything renders correctly, including all files and links.

5. Publish the theme so the site will implement the theme changes, then check it in a runtime environment.

See Publish Themes in *Creating Experiences with Oracle Content and Experience Cloud*.

**Use a Bootstrap Theme**

Similarities between Oracle Content and Experience Cloud themes and Bootstrap themes makes it possible to convert existing Bootstrap theme pages and content for use in an Oracle Content and Experience Cloud theme.

Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for interface components and JavaScript extensions.

You follow the same instructions for creating a theme, but you work with code (using whichever editor you choose) to import and edit any pages or content you want from the Bootstrap theme.

It’s unlikely that you would use all of a Bootstrap theme with all of its pages and content. Typically you will select parts of the theme to use in an Oracle Content and Experience Cloud theme, maybe only a few of the page templates and only some parts of those. For example, a Bootstrap theme contains several blocks within the templates, including a header, a navigation block, a body block and a footer block.

Because of the way that Bootstrap themes are shipped as a collection of files that can be viewed directly from the file system (without a web server), they contain a lot of duplication within each page (they have to because there is no page assembly engine involved). When you use this information in an Oracle Content and Experience Cloud theme, you need to take these parts and add them to reusable Oracle Content and Experience Cloud page templates, and then use Oracle Content and Experience Cloud to dynamically assemble multiple pages from those templates.

One example of adding information to reusable templates is the navigation section. Navigation in a typical Bootstrap theme is duplicated on all pages, but when you move that into an Oracle Content and Experience Cloud theme, you must use JavaScript code to dynamically walk the hierarchy of the site and generate the navigation structures. Then you just include that script on all the pages and they all get the navigation, and that navigation adapts as pages are added or removed from the site.

See Create a Site Template from Bootstrap or a Website Design Template.

**Associate Components with Themes**

You can associate components, section layouts, and component groups with a theme to use a specific subset of components with the theme.
As a developer, you can create a theme and components for a template that marketers will use to create sites that promote your organization's products. By associating a component with a theme, you make it available when the user selects Theme Components in Site Builder. On the Theme Components tab, the user will see only the components you associated with the theme. Associating a component with a theme ensures that this component will be exported with the site template, even if it is not used on the site.

To associate a component with a theme:

1. In Content and Experience Cloud, click Developer in the left navigation menu.
2. On the Developer page, click View All Themes.
3. On the Themes page, select a theme.
4. Click Properties in the top menu to open the Theme dialog.
5. Click Theme Components on the left to open the Theme Components tab.
6. Select one or more components to associate with the theme and categorize the components:
   a. From the Select a component drop-down list, choose a custom component.
   b. For the first component you choose, enter the name of a category to create a category for the component.
c. For each additional component you choose, select an existing category or create a new category.

d. Click *Add Component*.

7. Click **Save** to associate the components you selected with the theme.

8. On the Themes page, select a theme and click Properties to open the **Theme Components** tab again.

9. Verify that the components you selected were saved.

   To remove an associated component from the theme, click **X** next to the component name on the **Theme Components** tab, and then click **Yes** in the Remove Component dialog.

You can see list of components associated with the theme that a template is using on the Details page for the template. The references to the components associated with the theme are stored in the **components.json** file for the theme.
After configuring the theme association, you can deploy a template to Content and Experience Cloud and share it with the marketing team. When marketers choose that theme for their site, they’ll see only the components you specified for the theme on the Theme Components tab in Site Builder. The lists of Custom and All components also include the associated themed components.

**Associate a Component with a Theme in Oracle Content and Experience Toolkit.**

To associate a component with a theme in Oracle Content and Experience Toolkit, you can use the following cec command. The component will appear on the Theme Components tab in Site Builder, as well as in lists of Custom and All components.

```
cec add-component-to-theme <component>
```

The following cec command removes the association of a component from a theme.

```
cec remove-component-from-theme <component>
```

For information about Oracle Content and Experience Toolkit, see [Develop for Content and Experience Cloud with Developer Cloud Service](#).

**Render API Reference**

The Render API has useful events and functions, several of which are used in the sample navigation JavaScript files.

If you’re working on themes or components as a developer, you may find these events and functions useful for configuring JavaScript files. All Render API commands begin with `SCSRenderAPI` followed by a period followed by the command; for example, `SCSRenderAPI.getPageLinkData()`.

**Events**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scsrenderstart</code></td>
<td>Dispatched after all scripts are ready to let dependent scripts know they can start.</td>
</tr>
<tr>
<td><code>scsrenderend</code></td>
<td>Dispatched after rendering is complete.</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPageLinkData()</td>
<td>Gets the URL, target, and hideInNavigation properties for a link.</td>
</tr>
<tr>
<td>getRenderMode()</td>
<td>Mode is either edit, navigate, or view (runtime).</td>
</tr>
<tr>
<td>getContentUrlPrefix()</td>
<td>Gets the path to the site's content location.</td>
</tr>
<tr>
<td>getContentUrl()</td>
<td>Gets the path to a piece of content used in the site.</td>
</tr>
<tr>
<td>getThemeUrlPrefix()</td>
<td>Gets the location of the theme.</td>
</tr>
</tbody>
</table>

## Global Variables

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window.SCSRenderAPI</td>
<td>Gets the API so you can call things like SCSRenderAPI.getPageLinkData()</td>
</tr>
<tr>
<td>window.SCS.navigationRoot</td>
<td>Gets the ID of the node that is the root of the site.</td>
</tr>
<tr>
<td>window.SCS.navigationCurr</td>
<td>Gets the ID of the current page node.</td>
</tr>
</tbody>
</table>
| window.SCS.structureMap  | All of the nodes of the site hierarchy are listed in the structureMap and accessed by ID. For example, you could access the hideInNavigation property with: window.SCS.structureMap[100].hideInNavigation. Here is an example from a sample site:

```json
"navigationRoot": 100,
"navigationCurr": 100,
"structureMap": {
  "100": {
    "id": 100,
    "name": "Home",
    "parentId": null,
    "pageUrl": "index.html",
    "hideInNavigation": false,
    "linkUrl": "",
    "linkTarget": "",
    "children": [
      200,
      300,
      400,
      500
    ]
  }
},
```
Work with Layouts

A layout defines how content is arranged on a page and is used to produce the HTML for pages used in Oracle Content and Experience Cloud sites.

- About Layouts
- Search Engine Optimization (SEO)
- Understand the components.json File and Format
- Customize Toolbar Groups in Site Builder
- Restrict Components in Slots
- Make Layout Content Editable
- Create a Section Layout
- Create a Section Layout That Supports Lazy Load
- Develop Custom Section Layouts with APIs
- Develop Content Layouts

About Layouts

A layout defines how content is arranged on a site page. Different layouts can contain a different number of named slots. A slot is a region that spans the width of the page and can contain one or more types of content.

Every theme has several page layouts. See About Themes.

When you add a page to a site, you select a layout to use for that page. Each layout has areas on the page—known as slots—where you can drag and drop content. What content goes into these slots is completely up to you. It can be anything from titles, text, and dividers to multimedia, galleries, and social media.

A layout contains valid HTML constructs as well as special markup understood by the Oracle Content and Experience Cloud renderer. A layout must begin with a DOCTYPE statement in order to configure the browser to render the page in a standards-compliant mode; for example: `<!DOCTYPE html>` This statement is required by certain components to achieve best results.

This sample code shows a minimal layout:

```html
1  <!DOCTYPE html>
2  <html>
3  <head>
4          <meta http-equiv="X-UA-Compatible" content="IE=edge">
5  <script src="/_themes/[!--$SCS_THEME_NAME--]/assets/js/topnav.js"></script>
6  <link rel="stylesheet" type="text/css"
7          href="/_themes/[!--$SCS_THEME_NAME--]/assets/css/
```
Various tokens are expanded when a page is rendered in the browser.

- <!--$SCS_THEME_NAME-->
  This expands to the name of the theme currently chosen for the site. Using this token allows the theme to be copied, because URLs that use this token will reference the current theme.

- <!--$SCS_DESIGN_NAME-->
  This expands to the name of the design currently chosen for the site. This allows the layout to be used by multiple designs within the theme.

- <!--$SCS_RENDER_INFO-->
  This expands to a script tag that holds the page hierarchy and component rendering information for the page. This should be placed in the <head> section of the layout.

- <!--$SCS_SITE_HEADER-->
  This expands to the site header value that is specified in the Header field in the Search Engine Optimization (SEO) properties. Site-wide markup that you want placed on all pages can be entered here. See Set Search Engine Properties in Creating Experiences with Oracle Content and Experience Cloud.

- <!--$SCS_SITE_FOOTER-->
  This expands to the site footer value that is found in the Header field in the Search Engine Optimization (SEO) properties. Site-wide markup that you want placed on all pages can be entered here. See Set Search Engine Properties in Creating Experiences with Oracle Content and Experience Cloud.

Note:

The tokens can also use the <!--$ prefix and the --> suffix as delimiters in place of <!-- and -->.
When a site is online, the following tokens in the link are replaced with real values that are aware of the context in which they are being used. This enables the link to function when a site is being edited and in the published site when it is online.

- \/_sitescloud/ is replaced with \/_sitesclouddelivery/
- \/_themes/ is replaced with \/_themesdelivery/

**Slots** are DIV elements in the layout that have the value "scs-slot" in the class attribute. A slot is where users can add components to fill in the site content. Multiple DIV elements can be designated as slots by assigning the "scs-slot" class attribute. Each slot must have a unique id attribute.

---

**Note:**

Slots can't be nested, but you can give the suggestion they are by using CSS to overlay them. If you want to do this, use a component group or section layout. See Create a Section Layout.

Slots that additionally have a class attribute value of "scs-responsive" will refresh their content as the browser viewport changes resolution. This allows slots to render responsively for a large desktop display or a small mobile device.

The final `<script>` tag (line 19 in the example) loads the Oracle Content and Experience Cloud rendering code. This code is responsible for drawing the components on the page, and it also allows custom code to access the Render API. Without this `<script>` tag, pages based on the layout can't be changed by Site Builder.

---

### Search Engine Optimization (SEO)

You can provide keywords to help search engines identify the contents of your site.

**SEO Settings**

Search engine optimization (SEO) settings are defined at the site level and at the page level. The SEO text will be rolled into all out-of-the-box templates, in the footer.

See Setting Search Engine Properties in *Creating Experiences with Oracle Content and Experience Cloud*.

**Cookies for Site Visitors**

Site visitors use cookies for SEO, one cookie for each browser and each site, for billing purposes. Each cookie needs to be renewed hourly or after 24 hours.
Each site must include a popup that notifies visitors about cookies.

The popup should include the following information:

- SEO header and footer text
- Div in footer with a specific ID, where the text will be picked up and linked to
Privacy Policy Page

The text will link to the privacy policy page, which has advisory text.

Inner HTML for Search Engine Optimization

When you save component data in Site Builder, you can save the Inner HTML that would be produced if the component were rendered at runtime. This Inner HTML is stored in the page data so that when the page is rendered, the data can be inserted into the page in place of the component. This can happen very early in the page rendition, allowing a search engine more chance to successfully crawl the page content.

Understand the components.json File and Format

A theme must have a components.json file located at /ThemeName/components.json, which specifies the components used in the theme.

This file must contain valid JSON, and the minimum that the file must contain is an empty JSON array [].

The components.json file syntax lists all local components and fully supports categorization of components. (Remote components are registered in the Component Catalog.)

No matter what components are added to the theme-level components.json file (including none), Oracle Content and Experience Cloud populates a default set of components available to users. This default set is defined in the source code. The following list shows the components and (seeded) components rendered in inline frames. In addition, any remote components registered at the service level and made available to users in your instance will be available in Site Builder.

The following local components are included with Oracle Content and Experience Cloud.
### Name | Type | ID
--- | --- | ---
Title | scs-title | scs-title
Paragraph | scs-paragraph | scs-paragraph
Image | scs-image | scs-image
Gallery | scs-gallery | scs-gallery
Gallery Grid | scs-gallerygrid | scs-gallerygrid
Document | scs-document | scs-document
Button | scs-button | scs-button
Map | scs-map | scs-map
Divider | scs-divider | scs-divider
Spacer | scs-spacer | scs-spacer
YouTube | scs-youtube | scs-youtube
Social Bar | scs-socialbar | scs-socialbar
Video | scs-video | scs-video
Article (custom component) | scs-component | scs-comp-article
Headline (custom component) | scs-component | scs-comp-headline
Image and Text (custom component) | scs-component | scs-comp-image-text

These components, rendered in inline frames, are included with Oracle Content and Experience Cloud. They don’t include registered remote components.

### Name | Type | ID
--- | --- | ---
Conversation | scs-app | Conversation
Documents Manager | scs-app | Documents Manager
Folder List | scs-app | Folder List
File List | scs-app | File List
Facebook Like | scs-app | Facebook Like
Facebook Recommend | scs-app | Facebook Recommend
Twitter Follow | scs-app | Twitter Follow
Twitter Share | scs-app | Twitter Share

### General Format

The general format of the `components.json` file follows:

- Properties for components are specified within each component. Top-level "components" or "apps" properties are deprecated.
- Each component has a "type" property. Components can only have certain values (all possible values are listed in the table for default components).
- Each component has an "id" property, which must be unique. This property is used to distinguish between components with the same "type". Previously, apps
had the "appName" property. While "appName" still works if the "id" property is not available, the "appName" property is deprecated.

- Each component has a "name" property that is the display name in the user interface. If fallback values are not specified, for components the value is the name of the corresponding default component, and for remote components the value is the ID.

Here is an example of a components.json file:

```json
[
  {
    "name": "COMP_CONFIG_TEXT_CATEGORY_NAME",
    "list": [
      {
        "type": "scs-title",
        "id": "my-headline",
        "name": "My Headline",
        ...
      },
      {
        ...
      },...
    ]
  },
  {
    "name": "My own category name",
    "list": [ ... ]
  }
]
```

The general structure is a JSON array of category objects. Each category object has a "name" property and "list" property. The "name" property can be a key that maps to a localized String. If these default categories are not sufficient, you can provide your own category name, which won't be localized. The following table lists available default categories and corresponding keys.

<table>
<thead>
<tr>
<th>Key</th>
<th>Category Name (English)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP_CONFIG_CONTENT_CATEGORY_NAME</td>
<td>Content</td>
</tr>
<tr>
<td>COMP_CONFIG_CUSTOM_CATEGORY_NAME</td>
<td>Custom</td>
</tr>
<tr>
<td>COMP_CONFIG_MEDIA_CATEGORY_NAME</td>
<td>Media</td>
</tr>
<tr>
<td>COMP_CONFIG_SOCIAL_CATEGORY_NAME</td>
<td>Social</td>
</tr>
<tr>
<td>COMP_CONFIG_TEXT_CATEGORY_NAME</td>
<td>Text</td>
</tr>
</tbody>
</table>

The "list" property in each category object contains an array of component objects. Each component or object must have "type" and "id" properties. Other properties are optional.

- The "type" property must be equal to one of the types found in the default components. If the "type" does not already exist, the component won't be displayed.
• The "id" property must be unique across components. If the "id" is found to already exist, the component won't be displayed.

• The "name" property is the display name for the component in the user interface. This replaces the previous "appName" property for apps (now remote components).

• All other properties are treated the same as in previous releases.

Add New Components to components.json

You are not allowed to modify the default components. However, you can create a new component based on an existing default component. For example, you could create a new component based on the "scs-title" component, which sets some default text. The minimum required to add a new component is to specify the "type" and "id" properties.

• The "type" must be equal to one of the types found in the default components. If the "type" does not already exist, the component won't be displayed.

• The "id" must be unique across components. If the "id" is found to already exist, the component won't be displayed.

Here's an example of code to add a new Title component. This component will display along with the default title component.

```
[
  {
    "name": "COMP_CONFIG_TEXT_CATEGORY_NAME",
    "list": [
      {
        "type": "scs-title",
        "id": "my-headline"
      }
    ]
  }
]
```

Here’s an example of code to add a new Title component with a display name and default text.

```
[
  {
    "name": "COMP_CONFIG_TEXT_CATEGORY_NAME",
    "list": [
      {
        "type": "scs-title",
        "id": "my-headline",
        "name": "My Headline",
        "initialData": {
          "userText": "This is a second title component"
        }
      }
    ]
  }
]
```
Note that the title component takes all the properties of the default Title component as the base, and applies theme-level modifications on top of it to create the new component.

**Backwards Compatibility**

The `components.json` files in the previous format can still be read.

- Files with top-level "components" or "apps" properties.
- If the file contains an "apps" property, user-defined remote components under this property are still loaded.
- If the file contains a top-level "apps" property, then assume any remote components listed beneath have type "scs-app".
- If the "appName" property is present, set "id" to the "appName" value. The display name will be the same as "name", if specified, or falls back on the "id" value.

---

**Customize Toolbar Groups in Site Builder**

You can customize toolbar groups displayed in the Site Builder toolbar.

Toolbar groups define what you see in the toolbar when you click the Title or Paragraph component to edit it. You can remove and reorder what is supported by the Title or Paragraph component, but additional plug-ins are not allowed.

To override toolbar groups in Site Builder, you need to use the same syntax as CKEditor uses for its toolbarGroups configuration. See [https://ckeditor.com/latest/samples/toolbarconfigurator/index.html#basic](https://ckeditor.com/latest/samples/toolbarconfigurator/index.html#basic).

Title and Paragraph components support the following groups:

- "styles" - Font Styles [https://docs.ckeditor.com/ckeditor4/latest/guide/dev_styles.html](https://docs.ckeditor.com/ckeditor4/latest/guide/dev_styles.html)
- "colors" - Text and Background colors [https://docs.ckeditor.com/ckeditor4/latest/guide/dev_colorbutton.html](https://docs.ckeditor.com/ckeditor4/latest/guide/dev_colorbutton.html)
- "undo" - Undo/Redo of current instance in the CKEditor
- "links" - Custom Plugin to link dialog
- "paragraph" - bullet/numbered list and indentation support
  - "list"
  - "indent"
- "align" - left/right/center
- "cleanup" - remove any styles for selected text

The Paragraph component also supports the image and table insert plug-ins:

- "insert"
  - "image"
  - "table"
In addition, you can use the row separator entry:

- "/" 

**Note:**

If you set any other value in the toolbar-group configuration, the value will be removed before the Site Builder toolbar is created. You cannot provide "extraPlugins". Only the "name"/"groups" configuration is supported. Any "items" entries will be ignored.

For example, if you want to prevent your users from defining fonts, colors, styles, or sizes, you can update the toolbar configuration as follows:

```json
[
  {
    "name": "<category name>",
    "list": [
      {
        "type": "scs-title",
        "id": "scs-title",
        "config": {
          "toolbarGroups": [
            {
              "name": "basicstyles",
              "groups": ["basicstyles"]
            }, {
              "name": "undo",
              "groups": ["undo"]
            }, {
              "name": "links",
              "groups": ["links"]
            }, {
              "name": "paragraph",
              "groups": ["list", "indent"]
            }, {
              "name": "align",
              "groups": ["align"]
            }, {
              "name": "insert",
              "groups": ["image", "table"]
            }, {
              "name": "cleanup",
              "groups": ["cleanup"]
            }
          ]
        }
      }, {
        "type": "scs-paragraph",
        "id": "scs-paragraph",
        "config": {
          "fontSize_sizes": "16/16px;24/24px;48/48px;"
        }
      }
    ]
  }
]
```
Validation

Replace your `components.json` file with the preceding code and then edit your site (refresh the browser if you are already editing). At this point, when you edit a Title component, it will no longer show the font styles or colors for selection. The Paragraph component will continue to show these, and the list of font sizes available will be limited to 16, 24, and 48.

Default Toolbar Groups

The default toolbar groups for Title and Paragraph follow:

- **Title**

  ```json
  [{
    "name": "basicstyles",
    "groups": ["basicstyles"]
  }, {
    "name": "styles",
    "groups": ["styles"]
  }, {
    "name": "colors",
    "groups": ["colors"]
  }, {
    "name": "undo",
    "groups": ["undo"]
  }, {
    "name": "links",
    "groups": ["links"]
  }, {
    "name": "paragraph",
    "groups": ["list", "indent"]
  }, {
    "name": "align",
    "groups": ["align"]
  }, {
    "name": "cleanup",
    "groups": ["cleanup"]
  }]
  ```

- **Paragraph**

  ```json
  [{
    "name": "basicstyles",
    "groups": ["basicstyles"]
  }, {
    "name": "styles",
    "groups": ["styles"]
  }, {
    "name": "colors",
    "groups": ["colors"]
  }]
  ```
Restrict Components in Slots

For any layout slot, you can specify certain restrictions on the components allowed in the slot.

If you restrict components in a slot, any user dragging a component that is not allowed will see a warning message and will not be able to add or move a component to that slot.

To configure this restriction, you edit the layouts in your theme (for example, a layout file themes\theme_name\layouts\oneslot.htm) and add custom data attributes to the slot div.

This is the format of the custom attributes. The main difference from previous versions is that users must specify only the id of the component. Previously to restrict a component, users had to use the verbose "<type> <id>" syntax (which is still supported).

data-allowed-items='["<id>:\"<type>\"","<type>:\"<id>\",...]'
data-disallowed-items='["<id>:\"<type>\"","<type>:\"<id>\",...]'
The value for `data-allowed-items` and `data-disallowed-items` can use double quote marks (" ") or single quote marks ('). In the following example, `data-allowed-items` uses single quote marks around the JSON array, while `data-disallowed-items` uses double quote marks around the JSON array:

```html
<div id="slot101"
    class="scs-slot"
    data-allowed-items='"["scs-app","scs-title"]"
    data-disallowed-items='"["File List", "scs-map"]"/>
</div>
```

The following table lists components and their respective IDs provided with Oracle Content and Experience Cloud. To prevent any naming conflict, *do not* prefix any customized (local or remote) component ID with `scs-` or use any type or ID listed in this table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents Manager</td>
<td>scs-app</td>
<td>Documents Manager</td>
</tr>
<tr>
<td>Facebook Like</td>
<td>scs-app</td>
<td>Facebook Like</td>
</tr>
<tr>
<td>Facebook Recommend</td>
<td>scs-app</td>
<td>Facebook Recommend</td>
</tr>
<tr>
<td>File List</td>
<td>scs-app</td>
<td>File List</td>
</tr>
<tr>
<td>Folder List</td>
<td>scs-app</td>
<td>Folder List</td>
</tr>
<tr>
<td>Twitter Follow</td>
<td>scs-app</td>
<td>Twitter Follow</td>
</tr>
<tr>
<td>Twitter Share</td>
<td>scs-app</td>
<td>Twitter Share</td>
</tr>
<tr>
<td>Button</td>
<td>scs-button</td>
<td>scs-button</td>
</tr>
<tr>
<td>Article (custom component)</td>
<td>scs-component</td>
<td>scs-comp-article</td>
</tr>
<tr>
<td>Headline (custom component)</td>
<td>scs-component</td>
<td>scs-comp-headline</td>
</tr>
<tr>
<td>Image and Text (custom component)</td>
<td>scs-component</td>
<td>scs-comp-image-text</td>
</tr>
<tr>
<td>Component Group</td>
<td>scs-componentgroup</td>
<td>scs-componentgroup</td>
</tr>
<tr>
<td>Content List</td>
<td>scs-contentlist</td>
<td>scs-contentlist</td>
</tr>
<tr>
<td>Content Search</td>
<td>scs-contentsearch</td>
<td>scs-contentsearch</td>
</tr>
<tr>
<td>Divider</td>
<td>scs-divider</td>
<td>scs-divider</td>
</tr>
<tr>
<td>Document</td>
<td>scs-document</td>
<td>scs-document</td>
</tr>
<tr>
<td>Gallery</td>
<td>scs-gallery</td>
<td>scs-gallery</td>
</tr>
<tr>
<td>Gallery Grid</td>
<td>scs-gallerygrid</td>
<td>scs-gallerygrid</td>
</tr>
<tr>
<td>Image</td>
<td>scs-image</td>
<td>scs-image</td>
</tr>
<tr>
<td>Map</td>
<td>scs-map</td>
<td>scs-map</td>
</tr>
<tr>
<td>Paragraph</td>
<td>scs-paragraph</td>
<td>scs-paragraph</td>
</tr>
</tbody>
</table>
Users can create local or remote components. The name provided in this sample (My_Local_Component) is the ID that can be used to specify this component for restricting inside slots.

## Make Layout Content Editable

You can configure certain text or image content in a layout to make it editable by users working with pages based on the layout.

This functionality can be used in any theme, including bootstrap themes. The modifications automatically assume the style of the original page.

You can add simple markup to the following HTML tags in the layout: `<p>`, `<h1>` through `<h6>`, `<div>`, and `<img>`.

Text and image formatting options include:

- **Text**: Allows users to specify Bold, Italic, Underscore, and Link.
- **Image**: Allows users to specify a link to an image file. Change properties for title (what the user sees when hovering the cursor over an image) and add alternate text for accessibility.

Modifying a text or image element so it can be edited by users requires two things:

1. Add `scs-editable` to the class.
2. Add a unique `id` attribute.

Once a layout has been modified, the functionality will be available for all pages based on that layout (even new pages).

If you copy and paste a page, the modifications will be copied to the new page.

### Modifying a Heading Tag
Here's an example of how to modify a heading tag in a layout so it can be edited by users.

1. Sync the layout file to your local desktop, or edit the HTML source file for the \( <h1> \) header.

2. Add `scs-editable` to the class, and add the attribute `id="test-heading"` in the line of code for the heading, so it allows an editor to be attached to it. For example:

   ```html
   <h1 class="brand-heading scs-editable" id="test-heading">Sample Heading Value</h1>
   ```

3. Save the file.

4. Sync up with Oracle Content and Experience Cloud and reload the browser.
   A black border should appear around the heading when the user hovers the cursor over it, indicating that the content can be edited. When a user clicks the heading, the border becomes green, indicating the user can now edit the heading content.

5. If the theme hasn't been published, do so. If it has been published, the change will appear when you refresh your browser.
   Once the theme is published, site users can click the header and edit it in the Site Builder.
   A pseudo component is created for the `scs-editable` element so that you can change it in Site Builder and store it with the page data. At runtime, before the page is rendered, the controller replaces the `scs-editable` tags with the values you set in Site Builder.

**Modifying an Image Tag**

The procedure to modify an image tag in a layout so it can be edited by users is similar to that for text.

1. Add `scs-editable` to the class.

2. Add a unique image `id`.

Users can click the image, then change properties to use a different image.

Here's sample code for an image that can be edited by users:

```html
<img class="scs-editable" id="test-image" src="_scs_theme_root_/assets/img/downloads-bg-small.jpg"/>
```

**Create a Section Layout**

Create a section layout to arrange content within a slot on a site page.

An enterprise user can arrange content items on a site based on section layouts that you provide as a developer. You can create new section layouts from the default layout.

You can export a section layout to modify it offline and then import it either as a new section layout or to replace the existing section layout. Export the section layout individually or as part of a template package that includes custom components and layouts.
The following out-of-the-box section layouts are available:

- Horizontal
- Two Column
- Three Column
- Vertical
- Tabbed
- Slider

You can use these right away in Site Builder without having to create anything.

The files for these section layouts have comments with more details about the structure of section-layout files. To see the comments, you can create a new section layout based on an out-of-the-box one and then export the new layout for editing, as described in the following procedure.

To create a section layout:

1. On the home page, click **Developer**.
   The **Developer** page is displayed.

2. Click **Components**.

3. From the Create drop-down menu on the right, choose **Create Section Layout**.

4. In the **Create Section Layout** dialog box, provide a name and description for your section layout component.

   ![Create Section Layout dialog box](image)

   The section layout is added to your Component Catalog.
5. To export the section layout for editing, select it, and then click Export in the top menu.
   a. Navigate to an existing folder, or click Create to create a new folder and provide a name and, optionally, a description.
   b. Select the checkbox next to the folder, and click OK.
   c. Click the folder's icon or name to open it.

A layout package file is created in the selected folder with the section layout name and a .zip extension. Download the file to your development environment to edit the files.

You can find information about the Section Layouts API in Develop Custom Section Layouts with APIs.

6. Import your modified files either as a new section layout or to replace the existing section layout.
   a. On the home page, click the Content tab, and then click Documents.
   b. Upload the modified section layout to a folder, in a file with a .zip extension, which includes the same folder and file names that you exported.
   c. On the Developer page, click Components.
   d. From the Create menu, choose Import Component.
   e. Select the checkbox next to the uploaded ZIP file that contains the modified section layout, and click OK.
Your modified section layout is imported to the folder you selected.

You can also export a section layout to copy or move it to another Oracle Content and Experience instance and import it there.

For information about managing section layouts, see Arrange Page Content in Creating Experiences with Oracle Content and Experience Cloud.

Create a Section Layout That Supports Lazy Load

The content list can call the section layout with additional components as they get queried.

When you render a content list, you have the option of selecting a section layout to render all the content items that are returned. This enables you to create various different layouts for the content items, such as a table, a slider, or an eight-column layout. These custom section layouts can also take part in the more advanced pagination features.

Content lists support the following pagination:

• Pagination
• Load on scroll
• Load on click

For the standard pagination feature, the section layout doesn't need to do anything. It will be re-rendered with the next set of items when the user clicks the next page. However, for Load on scroll and Load on click, rather than having the section layout re-render, additional components are added to the section layout. This is used mostly for the infinite scrolling model, where you load the first \( n \) items and, as the user scrolls down the page, you fetch and render the next set of items. To support Load on scroll and Load on click, the custom section layout needs to do

1. **render.js**: Implement the `addComponent()` API. This will be called with each new component that is to be added to the section layout.

   ```javascript
   // dynamic API for adding additional components through "load more" when used in a Content List
   addComponent: function (parentObj, component) {
       // create the component div and add it to the parent object
       $(parentObj).append(this.createComponentDiv(component));
   }
   ``

2. **appInfo.json**: Include the following to let the content list know that the section layout supports the `addComponent()` api.

   ```json
   "contentListData": {
       "addComponent": true
   },
   ``

Once the `appInfo.json` is updated, when the user selects this section layout in the settings panel and goes to the pagination screen, they will see the Load on click and Load on scroll options.
Develop Custom Section Layouts with APIs

You can develop custom section layouts in Oracle Content and Experience Cloud with the Section Layout API, which includes Rendering APIs and Editing APIs.

For starter files to look at, see Create a Section Layout. The starter files for section layouts include comments with details about the structure of section-layout files.

Rendering APIs

The Rendering APIs, loaded from the render.js module, are used in Site Builder and at runtime.

<table>
<thead>
<tr>
<th>Rendering API</th>
<th>Description</th>
<th>Input Parameter(s)</th>
<th>Return Result</th>
</tr>
</thead>
</table>
| (Constructor) | Initializes the Section Layout rendering module. | A JavaScript object that contains the following properties:  
|               |             | • sectionlayoutData (Object) : The section layout data found in the page model.  
|               |             | • componentId (String): The componentId value of the section layout, typically a GUID.  
|               |             | • renderMode (String, optional): The render mode for the rendering operation.  
|               |             | • customSettingsData (Object): A copy of the customSettingsData found in sectionLayoutData.  
| render        | Emits DOM elements appropriate for the section layout to the page, including container DIVs for child components. | container (Element) : The DOM element into which the section layout's markup should be rendered. | After this method returns, child components will be rendered. You can identify child components by finding child div[id] elements. |
| addComponent  | Used with content list components to dynamically add child components to a section layout. This function is optional. | container (Element): The DOM element into which the new component should be rendered. componentId (String): The ID of the new component to add to the section layout. | After this method returns, the element whose ID matches the componentId input will be rendered. |

Editing APIs

The edit.js module is loaded if the hasEditHandlers property is set to true in the appinfo.json file associated with the section layout.

The Editing APIs are used in Site Builder.
All of the functions in this module except the Constructor are optional.

<table>
<thead>
<tr>
<th>Editing API</th>
<th>Description</th>
<th>Input Parameter(s)</th>
<th>Return Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constructor)</td>
<td>Initializes the Section Layout editing module.</td>
<td>A JavaScript object that contains the following property: • <strong>componentId (String)</strong>: The componentId value of the section layout, typically a GUID.</td>
<td>The Section Layout Editing APIs are initialized.</td>
</tr>
<tr>
<td>Editing API</td>
<td>Description</td>
<td>Input Parameter(s)</td>
<td>Return Result</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>getCapabilities</td>
<td>Returns an object describing the editing capabilities of the section layout.</td>
<td>A JavaScript object that describes the editing capabilities of the section layout. Upon input, the default capabilities will be provided to the function. The function can modify the Capabilities object as needed. The Capabilities object can include the following capabilities:</td>
<td>(Object): The capabilities for the section layout.</td>
</tr>
<tr>
<td>- title (String):</td>
<td>The title of the section layout to display to the user.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- settingsTitle (String):</td>
<td>The title to display in the Settings Panel dialog.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- hasSettings (Boolean):</td>
<td>Indicates if the section layout supports a Settings Panel.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- allowMove (Boolean):</td>
<td>Indicates if the section layout allows child items to be moved.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- allowDelete (Boolean):</td>
<td>Indicates if the section layout allows child items to be deleted.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- isHidden (Boolean):</td>
<td>Indicates if the section layout is currently hidden in response to user options.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- dropTarget (Boolean):</td>
<td>Indicates if the section layout is the target for drag and drop operations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- customMenuOptions (Array):</td>
<td>Custom menu options to add to the Section Layout context menu. Each menu-option object has the following properties:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- label (String):</td>
<td>The display text of the menu item.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- action (Function):</td>
<td>The function to invoke when the menu item is clicked.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- disabled (Boolean):</td>
<td>Indicates that the menu item should display in a disabled state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- icon (String):</td>
<td>The URL to display alongside the label in the menu item. (This property is reserved for future use.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- checkbox (Boolean):</td>
<td>Indicates that a checkbox should appear alongside the label in the menu item.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editing API</td>
<td>Description</td>
<td>Input Parameter(s)</td>
<td>Return Result</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getCaptionContent</td>
<td>Returns the section layout display name, which will appear in UI elements.</td>
<td>None.</td>
<td>(String): The display name of the section layout.</td>
</tr>
<tr>
<td>filterCapabilities</td>
<td>Allows the section layout to modify the Capabilities object before menus are displayed to the user. You can use this API to adjust or remove menu options. (See also getCapabilities.)</td>
<td>A JavaScript object that describes the editing capabilities of the section layout. On input, the default capabilities will be provided to the function.</td>
<td>(Object) The capabilities for the section layout.</td>
</tr>
<tr>
<td>onDragOver</td>
<td>Called during a Drag and Drop operation to indicate if the dragged item can be dropped on the section layout.</td>
<td>eventObject (Event Object): An event object that holds information about the drag event. dataTransfer (DataTransfer Object): A DataTransfer object that holds information about the item being dragged over the section layout.</td>
<td>(Boolean) A value indicating if the dragged item can be accepted by the section layout. Returns true if the section layout can accept the dragged item, false otherwise.</td>
</tr>
<tr>
<td>onDrop</td>
<td>Called during the drop portion of a Drag and Drop operation to indicate that the dragged item should be placed inside the section layout.</td>
<td>eventObject (Event Object): An event object that holds information about the drop event. dataTransfer (DataTransfer Object): A DataTransfer object that holds information about the item being dropped on the section layout.</td>
<td>(Boolean) A value indicating if the drop operation was handled by the section layout. Returning true bypasses the default logic.</td>
</tr>
<tr>
<td>onAddComponent</td>
<td>Notifies the Section Layout that a Drag and Drop operation moved an item in the section layout.</td>
<td>eventObject (Event Object): An event object that holds information about the drag event. dataTransfer (DataTransfer Object): A DataTransfer object that holds information about the item being dropped on the section layout. componentId (String): The componentId value of the newly added item.</td>
<td>Section layout notification.</td>
</tr>
<tr>
<td>Editing API</td>
<td>Description</td>
<td>Input Parameter(s)</td>
<td>Return Result</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
</tr>
</tbody>
</table>
| **onMoveComponent** | Notifies the section layout that a Drag and Drop operation moved an item in the section layout. | *eventObject (Event Object)*: An event object that holds information about the drag event.  
*dataTransfer (DataTransfer Object)*: A DataTransfer object that holds information about the item being dragged over the section layout.  
*componentId (String)*: The componentId value of the moved item. | Section layout notification. |
| **getSettingsData** | Allows the section layout to change the Settings data before the Settings Panel is displayed. | *settingsData (Object)*: The default settings data computed for the Section Layout | *(Object)* The settings data for the section layout. |
| **updateSettings** | Allows the section layout to change its settings after the Settings Panel has been closed.  This API is called just before the settings are stored in the page model. | *parameters (Object)*: The raw parameters object returned from the Settings Panel.  
*sectionLayoutData (Object)*: The section layout data that will be stored. Default data will be generated from parameters (Object) and passed to the function in this parameter. | *(Object)* The section layout data to store in the page model. |
| **dispose**     | Allows the editing module to free memory, detach events, and deallocate resources associated with the edit handlers.  This API is called when the section layout needs to completely redraw, as in the case of an Undo/Redo operation. | None. | Redrawing the section layout is enabled. |

**Develop Content Layouts**

Content layouts help users view the data in content items through content list or content placeholder components used in sites pages. You can create multiple content layouts for a content type to create different views or to represent different parts of a content item.
For example, consider the following Blog-Post content type. The site’s home page displays a list of Blog-Post items. When a blog post is clicked on the home page, the Details page shows details about that blog post.

![Blog-Post Content Type](image)

The Home page has a content list configured to list items of the Blog-Post content type, using the Blog-Post-Summary content layout as an item view.

The Details page uses the Blog-Post-Header content layout in a content placeholder to show a header image and title. The two-column section layout encloses two content placeholders, in 70 percent and 30 percent widths, using the Blog-Post-Content and Blog-Post-Author content layouts. As you can see, four different content layouts are used to visualize the same content type.

You can create a content layout in either of two ways:

- In Content and Experience Cloud, choose Developer > Components > Create > Create Content Layout.
- In an Oracle Content and Experience Toolkit project, use the cec create-contentlayout command.

Content and Experience Cloud creates a default content layout for the content type. To modify the default content layout, you can edit the following files:

- assets/layout.html
  Edit this file to change the HTML view.
- assets/design.css
  Edit this file to style the content layout.
- assets/render.js
Edit this file to change the data used in layout.html or to add dynamic behavior to the content layout.

Content layout components render a content item from the Content and Experience Cloud server. Most of the assets are stored in the Content and Experience Cloud server. Sometimes you may want to use a static asset that is locally available in the content layout itself, such as a background image for styling purposes. For example, in the content layout that follows, the absolute URL to images/background.jpg can be generated in render.js and used in layout.html.

The simplest way to generate an absolute URL is to use the built-in requirejs file. Define ‘require’ as the dependency and use require.toURL() to generate the URL, as the following code shows:

```javascript
define([
  'require',
  'jquery',
  'mustache',
  'text!./layout.html',
  'css!./design.css'
], function (require, $, Mustache, templateHtml, css) {
  'use strict';

  var imageURL = req.toUrl('./images/background.jpg');
});
```

The following topics describe how to develop content layouts:

- Create Content Layouts with Content and Experience Cloud
- Pass a Layout View to a Content Layout
- Generate a Site Details Page URL with an API
- Develop Content Layouts Locally with Developer Cloud Service
- Expand Macros in Content List Queries
- Create the Sample Blog Template
- Add Content Layout Mappings to Templates
- Test Content Layouts with the Local Test Harness
- Import Templates with Content Layouts into Content and Experience Cloud

Related Topics

For information about how to manage content layouts, see Manage Custom Components and Layouts in Creating Experiences with Oracle Content and Experience Cloud.

For information about using digital assets and other content items in a site, see Use Assets in Managing Content with Oracle Content and Experience Cloud and Work with Digital Assets in Creating Experiences with Oracle Content and Experience Cloud.
Create Content Layouts with Oracle Content and Experience Cloud

Create a content layout for laying out fields in a content item when displayed on a site page. When a content item is added to a page, it will use the chosen content layout.

An enterprise user can create and use content items based on content types and layouts that you provide as a developer. You can create new content layouts from the default layout. Multiple content layouts associated with the content type make it possible for the site designer to display content items in different contexts without changing the content.

If you use a content layout in a content list component, then the content layout is repeated once per content item. The content layouts are then arranged by the section layout.

You can export a content layout to modify it offline and then import it to replace the existing content layout.

To create a content layout:

1. On the Oracle Content and Experience Cloud home page, click Developer. The Developer page is displayed.
2. Click Components.
3. From the Create drop-down menu on the right, choose Create Content Layout.
4. In the Create Content Layout dialog box, provide a name and description for your content layout component.

The content layout is added to your components.
5. To export the content layout for editing, select it, and then click Export in the top menu.
   a. Navigate to an existing folder, or click Create to create a new folder and provide a name and, optionally, a description.
   b. Select the checkbox next to the folder, and click OK.
   c. Click the folder’s icon or name to open it.

   A layout package file is created in the selected folder with the content layout name and a .zip extension. Download the file to your development environment to edit the files.

   ![Assets screenshot]

   These files control the layout of the fields in content items that use the Employee-Card content layout.
6. Edit the design.css, layout.html, and render.js files to get the content layout you want.

For example, the following files specify the Employee-Card content layout:

a. Edit the design.css file:

```css
.scs-tile-layout {
  font-family: 'Helvetica Neue', 'Segoe UI', sans-serif-regular, Helvetica, Arial;
  font-size: 16px;
  margin: 0px;
  padding: 0px;
  font-style: normal;
  color: #333;
}

.scs-tile-layout li {
  list-style: none;
  font-size: 14px;
  font-style: normal;
  font-variant-caps: normal;
  font-weight: 200;
  margin: 0px;
}
```
b. Edit the layout.html file:

```html
{{#data.employee_profile_pictureURL}}
<div class="scs-tile-layout-img-container">
    <img class="scs-tile-layout-img" src="{{data.employee_profile_pictureURL}}" />
</div>{{/data.employee_profile_pictureURL}}
<ul class="scs-tile-layout">
    <li>
        <p>{{name}}</p>
    </li>
    <li>&nbsp;</li>
    <li>
        <p>{{data.employee_job_title}}</p>
        <p>{{data.employee_location}}</p>
        <p><b>Phone: </b>{{data.employee_phone}}</p>
    </li>
    {{#scsData.detailPageLink}}
    <li>
        <a href="{{scsData.detailPageLink}}" title="Go to detail page">Profile</a>
    </li>{{/scsData.detailPageLink}}
</ul>
```
c. **Edit the** `render.js` **file:**

/* globals define */

define([
  'jquery',
  'mustache',
  'text!./layout.html',
  'css!./design.css'
], function($, Mustache, templateHtml, css) {
  'use strict';

  function ContentLayout(params) {
    this.contentItemData = params.contentItemData || {};
    this.scsData = params.scsData;
    this.contentClient = params.contentClient || (params.scsData ? params.scsData.contentClient : null);
  }

  ContentLayout.prototype = {
    render: function(parentObj) {
      var template,
          content = $.extend({}, this.contentItemData),
          contentClient = this.contentClient,
          contentType,
          secureContent = false;

      if (this.scsData) {
        content = $.extend(content, { 'scsData': this.scsData });
        contentType = content.scsData.showPublishedContent === true ? 'published' : 'draft';
        secureContent = content.scsData.secureContent;
        }

      if (contentClient) {
        var params = {
          'itemGUID': typeof content.data.employee_profile_picture === 'string' ? content.data.employee_profile_picture : content.data.employee_profile_picture.id,
          'contentType': contentType,
          'secureContent': secureContent
        };
        content.data.employee_profile_pictureURL = contentClient.getRenditionURL(params);
      }

      console.log(content);

      try {
        // Mustache
        template = Mustache.render(templateHtml, content);
      }
    }
  }

  module.exports = ContentLayout;

});

Chapter 7
Develop Content Layouts
7-30
if (template) {
    $(parentObj).append(template);
} catch (e) {
    console.error(e.stack);
}

return ContentLayout;
});

For information about editing the render.js and other files, see Develop Components.

7. Import your modified files to replace the existing content layout.
   a. On the home page, click Documents.
   b. Upload the modified content layout to a folder, in a file with a .zip extension, which includes the same folder and file names that you exported.
      If you want to import it as a new content layout, you need to change the GUID of the content layout in _folder.json.
   c. On the home page, click Developer.
      The Developer page is displayed.
   d. Click Components.
   e. From the Create menu, choose Import Component.
   f. Select the checkbox next to the uploaded ZIP file that contains the modified component, and click OK.

Your modified content layout is imported to Components.

You can also export a content layout to copy or move it to another Oracle Content and Experience Cloud instance and import it there.

Pass a Layout View to a Content Layout

When you develop a content layout, you can get at properties for the underlying component if the content layout is being used in sites.

For this specific use case, you can get at the content layout view chosen for the content type in a content list. Then you can alter the way the component renders depending on what the category is. Without access to this property, you would need to create two content layouts that are effectively the same.

The contentLayoutCategory property is available for content layouts rendered for both the content item and the content list components. You can access this property through the Sites SDK, as follows.

```javascript
scsData.SitesSDK.getProperty('contentLayoutCategory', function (layoutCategory) {
    console.log(layoutCategory);
});
```

See the Sites SDK Reference.
This property is available only when the content layout is being rendered from a content item or content list, which are in an Oracle Content and Experience Cloud site. It isn't available when the content layout is rendered from a third-party app.

Generate a Site Details Page URL with an API

If you're rendering a list of content items in a content layout from your own query, you can create a link to a detail page for a content item. You can use the Sites SDK SCSRenderAPI.getPageLinkData API to generate a Site Details page URL.

The detailPageId property is the ID of the detail page selected in the content item or the content list. If the value wasn't set, the value returned is the first page in the SiteStructureMap that has the isDetailPage property set.

To access this property, you can use the Sites SDK. This is available only when the content layout is used for an Oracle Content and Experience Cloud site. It can be accessed only through scsData, which is passed as one of the arguments when the content layout is created. For example:

```javascript
scsData.SitesSDK.getProperty('detailPageId', function (detailPageId) {
    console.log(detailPageId);
});
```

Once you have the detailPageId, you can use it to construct the link to the detail page.

The SCSRenderAPI has a function, getPageLinkData(), that takes in a pageId and additional options and constructs the required URL to the page passing through the options. The signature for this function follows:

```javascript
SCSRenderAPI.getPageLinkData(pageId, options);
```

It has the following parameters:

- **pageId**: The same as the detailPageId returned from the Sites SDK detailPageId property.
- **options**:
  - contentType
  - contentId
  - contentName

The return value is an object with these properties:

- hideInNavigation
- href
- href

The next example puts this all together:

```javascript
scsData.SitesSDK.getProperty('detailPageId', function (detailPageId) {
    var pageDetails = SCSRenderAPI.getPageLinkData(pageId, {
        'contentType': contentType,
```
Develop Content Layouts Locally with Developer Cloud Service

You can use Developer Cloud Service through the Oracle Content and Experience Toolkit to create, edit, configure, and test content layouts locally for Content and Experience Cloud.

Take the following steps to prepare for developing content layouts with Developer Cloud Service:

1. Set Up Oracle Content and Experience Toolkit on Your Local Machine.
2. Sign in to the Developer Cloud Service Console for Content and Experience Cloud.
3. Create a Project in Developer Cloud Service.
5. Create a Content Layout

The following topics describe how to use the Oracle Content and Experience Toolkit to develop a content layout locally with Developer Cloud Service:

- Create a Content Layout with Developer Cloud Service
- Define the RequireJS module
- Configure the Constructor Function Parameter
- Render the Content Layout
- Edit the Content Layout in the Mustache Template
- Add Dynamic DOM Manipulation
- Define Styles in the design.css File
- Get Reference Items
- Get a Media URL
- Raise Triggers
- Navigate to a Search Page with a Search Query
- Expand Macros and Render Rich Text
After you develop your content layout with the Developer Cloud Service template, merge changes with your project’s Git repository.

Create a Content Layout with Developer Cloud Service

You can create a content layout for a content type in your Developer Cloud Service project with the `cec create-contentlayout` command.

To create a content layout in your Developer Cloud Service project:

1. In a terminal window, go to the `cec-components` directory.
2. Enter `cec create-contentlayout` to see options and examples for the command:

   Usage: cec create-contentlayout <name>  
   Creates a content layout based on a content type from a local template or from CEC server.  
   By default, an "overview" content layout is created. Optionally specify -s <style> to create in a different style.  
   Valid values for <style> are:  
   detail overview  
   Options:  
   --contenttype, -c <contenttype> Content layout is based on [required]  
   --template, -t <template> Content type is from  
   --server, -r flag to indicate the content type is from server  
   --style, -s <style> Content layout style  
   --help, -h Show help [boolean]  
   Examples:  
   cec create-contentlayout Blog-Post-Overview-Layout -c Blog-Post -t BlogTemplate  
   cec create-contentlayout Blog-Post-Detail-Layout -c Blog-Post -t BlogTemplate -s detail  
   cec create-contentlayout Blog-Post-Overview-Layout -c Blog-Post -r  
   cec create-contentlayout Blog-Post-Overview-Layout -c Blog-Post -r -s detail  
3. Enter the following command to see what content types are available on your server:

   cec list-server-content-types  
4. As shown in the "Usage", you can either create a content layout for the content type in the Content and Experience Cloud server or for the content type in the
templates under cec-components/src/main/templates. For example, the following command creates the content layout for the type in the server:

```
cec create-contentlayout Blog-Post-Overview-Layout -c Blog-Post -r
```

You can edit the following files to modify the content layout:

- **assets/layout.html**
  
  This file specifies the HTML view. See [Edit the Content Layout in the Mustache Template](#).

- **assets/design.css**
  
  This file specifies the style for the content layout. See [Define Styles in the design.css File](#).

- **assets/render.js**
  
  This file specifies the data used in `layout.html` and lets you add dynamic behavior to the content layout. See [Define the RequireJS Module](#).

### Define the RequireJS Module

You can define a RequireJS module in the `render.js` file. Sites loads the dependencies, such as for JQuery, Mustache, the RequireJS Text Plugin, and the RequireJS CSS plugin.

```javascript
define(['jquery', 'mustache', 'text!./layout.html', 'css!./design.css'], function ($, Mustache, templateHtml, css) {

  // You can use the Mustache template system to render the layout.

  The `assets/render.js` file for a content layout has the following properties:

- It should be a RequireJS module
- It should return a JavaScript Constructor function. Sites invokes the Constructor function by passing a parameter object. The parameter object has the content item data and the APIs required to render the layout.
- This Constructor function should have a `render(parentObj)` method that handles rendering the content layout. It should append the content layout DOM object to the `parentObj` object that is passed to the `render()` method.
- The RequireJS module can use the dependencies, including JQuery, Mustache, the RequireJS Text Plugin, and the RequireJS CSS plugin. These dependencies will be loaded by sites. You can use other libraries too.

### Configure the Constructor Function Parameter

When sites creates a new instance of the constructor function, it passes a parameter that contains `contentItemData`, `scsData`, and `contentClient` to help with content layout development.
Here is example code for the constructor function:

```javascript
function ContentLayout(params) {
    this.contentItemData = params.contentItemData || {};
    this.scsData = params.scsData;
    this.contentClient = params.contentClient;
}

ContentLayout.prototype = {
    render: function (parentObj) {
        var content = {
            blogTitle: this.contentItemData.data['starter-blog-post_title'],
        };

        if (this.scsData) {
            content = $.extend(content, {
                'scsData': this.scsData
            });
        }
    }
};

return ContentLayout;
```

The constructor function parameter includes the following objects:

- **params.contentItemData**: Contains the content item, including its name, description, ID, and data. For example, the field 'blogpost_title' in the content item can be accessed using `params.contentItemData.data['blogpost_title']`.

- **params.scsData**: This object passes in information when the constructor is called from within sites. This object doesn't exist for content layouts rendered in third-party applications. This object contains a Sites SDK object, the method `contentTriggerFunction` to raise a trigger, and the Details page links.

- **params.contentClient**: This is the `contentClient` object created from the Content SDK and used to call the content layout. It is therefore configured with the appropriate parameters for the content server. If you need to make additional calls to the content server, you can use this `contentClient` object rather than creating your own. This object contains client APIs for the content. APIs are available to query, search, and get content items and their content types. Other helper APIs are also available; for example, `expandMacros()` to expand the macros used in rich text.

### Render the Content Layout

The `render(params)` method of `ContentLayout` renders a content layout from a template. The Mustache template is used by default for content layouts, but you can use any template technology you want.

The `render(params)` method of `ContentLayout` can use the following code to render the template with the data:

```javascript
try {
    // Mustache
```
template = Mustache.render(templateHtml, content);
if (template) {
    $(parentObj).append(template);
}

// Dynamic DOM Manipulation can be done here

} catch (e) {
    console.error(e.stack);
}

You can add the required data to the content object created from params.contentItemData. Oracle recommends that you merge the properties from params.scsData into this object, so the template can make use of them too. The rendered template should be appended to the parent object passed to the render() method.

Edit the Content Layout in the Mustache Template

You can edit the default Mustache template in the assets/layout.html file, which contains the default content layout.

A simple template to render a blog title follows:

<h1>
    {{blogTitle}}
</h1>

Add Dynamic DOM Manipulation

You can add Dynamic DOM manipulation to render.js after Mustache.render() is called and the template is appended to the parent object.

For example, you could attach a listener, dynamically changing the style:

// Dynamic DOM Manipulation can be done here
$('h1').click(function (event) {
    alert('Title is : ' + $(this).text());
});

Define Styles in the design.css File

In the design.css file, you can define any style used in the content layout template.

The design.css file is loaded in the module definition using the RequireJS CSS plugin.

Get Reference Items

You can get a reference item for a content type with a reference data field that refers to another content type.
For example, the Author field in the Blog-Post content type is a reference to the Author content type. In the content layout for Blog-Post, `contentClient.getItems()` associates the details of the Author reference item with the current Blog-Post item.

```javascript
var author_id = this.contentItemData.data['starter-blog-post_author'].id;
var ids = [author_id];
// Get the author reference item
contentClient.getItems({
    'ids': ids
}).then(function (items) {
    // Use the item
    }, function (error) {
        // Handle error
});
```

Get a Media URL

You can use `contentClient.getRenditionURL()` to get the default rendition of a digital asset, such as an image.

```javascript
blogHeaderImage = contentClient.getRenditionURL({
    'id': data['blog-post_header_image'].id
}),
```

If you need other renditions, such as thumbnail, you can get the digital asset using `contentClient.getItems()` and refer to `item.data.renditions.default` and `item.data.renditions.thumbnail`.

Raise Triggers

You can use `scsData.contentTriggerFunction(payload)` to raise a trigger from a content layout.

Here is an example of an Author content layout raising a trigger when an author name is clicked:

```javascript
{{#data}}
<div class="author-container">
    <span class="author-name" onclick='{{scsData.contentTriggerFunction}}("field:starter-blog-post_author:equals={{author_id}}")'>{{starter-blog-author_name}}</span>
{{/data}}
```

The payload is a search query for the currently selected author, which other Content List items on the page can listen to.

Navigate to a Search Page with a Search Query

A common use case is to navigate to a search page with a dynamic search query when clicking on a link inside a content layout.

For example, assume that you want to navigate to a search page named "Authors" when clicking on the "More articles from this author" link in your content layout,
passing a search payload. The following code will achieve this. Notice that the global objects SCS and SCSRenderAPI are available to use in the content layout when running inside a sites page.

```javascript
$('.more-from-author').click($.proxy(function () {
    var childrenPages = SCS.structureMap[SCS.navigationRoot].children;

    if (!childrenPages) return; // No pages

    // Find the Authors page
    for (var i = 0; i < childrenPages.length; i++) {
        var page = SCS.structureMap[childrenPages[i]];
        if (page.name === 'Authors') {
            var linkData = SCSRenderAPI.getPageLinkData(page.id);
            if (linkData && linkData.href) {
                var href = linkData.href,
                    searchPayload = content.author_id + '*',
                    contentType = "Starter-Blog-Post";
                // if both the page URL and the search query exists, navigate to the page passing in the query
                if (href && searchPayload) {
                    var queryStart = href.indexOf('?') === -1 ? '?' : '&';

                    // add in the contentType and search parameters
                    // contentType isn't a required URL parameter
                    // Payload contains search string only. No parameter name.
                    href += queryStart + (contentType ? 'contentType=' + contentType + '&' : '') + 'q=' + searchPayload;

                    // navigate to the search results page
                    window.location = href;
                }
            }
        }
    }
}, this));
```

If you expect the same content layout to be used multiple times in the same page, it is better to use the unique ID in the CSS selector rather than the class selector, like `$('.more-from-author').click(...)`. For example:

```html
template.html
<%templateId = this.scsData.id + 'detailTrigger';
$("#${templateId}").click(...)%
```

Expand Macros and Render Rich Text

Rich text in a content item can embed a digital image.
To render this rich text correctly in the content layout, rich text fields use `contentClient.expandMacros()` API. This resolves all the references to digital assets inside the rich text.

```java
data["starter-blog-post_content"] =
contentClient.expandMacros(data["starter-blog-post_content"]);
```

If you use Mustache for rendering, `{{{ }}}` should be used to render a rich text value because rich text has HTML. When `{{{ }}}` is used around the variable, Mustache doesn't escape the HTML.

**Link to the Details Page**

The Details page link is available through `scsData.detailPageLink`.

For example, if you want to navigate to the Details page to display blog details when you click a blog title, you can use the Details page link as follows:

```html
<a href="{{scsData.detailPageLink}}">  
  <h2 class="post-title">  
    {{blogTitle}}  
  </h2>  
</a>
```

**Expand Macros in Content List Queries**

In a content list query, you can define values for properties that are calculated when a page is run, to display content that has been recently updated.

Most properties for components within sites are static. The user selects or enters a fixed string or value for one of the properties of the component, and that doesn't change regardless of when or where the page is run. However, you can define values for properties that are calculated when the page is run. This is useful for displaying content that has been recently updated in content queries. Users can enter dates such as "in the last 3 days".

You can insert a Mustache JS expansion to several properties. The values referenced in these strings are derived from a model that is executed when the page is run. An out-of-the-box model handles dates formatted for Content REST API calls. You can extend this model with additional values to meet any user requirements.

An example of the string you can enter for a property follows:

**Content List component:**

Additional Query String property:

```java
updatedDate gt "{{#content.date}}today - 3 days{{/content.date}}"
```

This Mustache entry for the date will be evaluated at runtime so that the returned value changes depending on when it is run (that is, expands to `updatedDate gt "22201810020600000000"`). In this way, the user can build up any complex date string rather than having to enter a predefined value.
Supported Component Properties

The following properties support Mustache JS template syntax:

- **Content List**
  - Additional Query String
  - For example: updatedDate gt "{{#content.date}}today - 3 days{{/content.date}}"

- **Title/Paragraph/Text**
  - Rich text entered via CKEditor
  - For example: "Content REST API format for date: {{#content.date}}now{{/content.date}}"

**Note:**

Without a custom model for the Mustache template, the expansion in Title/Paragraph/Text isn't that useful. However, it is very useful for validating what you enter in the Additional Query String because it will be evaluated as you switch between edit and view and be immediately visible.

Supported Component Syntax

The `content.date` object is supported out-of-the-box. This takes in two main parameters, today and now.

The **today** value takes the current browser time, converts it to midnight tonight, and then converts that value to UTC time.

- `{{#content.date}}today{{/content.date}}` expands to the browser value for midnight tonight, converted to the UTC value and formatted into the Content REST API date format. For example:

```
2220181008065959999
```

- It can then be augmented with:

```
today +/- [day | week | month | year]
```

- The **today** value also behaves differently as you add or subtract from it. If you subtract from it, it will use the time in the morning. If you add to it, it will use the time at midnight. For example:
  - `{{#content.date}}today - 1 day{{/content.date}}` expands to yesterday at start of day.
  - `{{#content.date}}today + 2 days{{/content.date}}` expands to the day after tomorrow at midnight.

The **now** value takes the current browser time and converts it to UTC time without any adjustment.
• `{{#content.date}}now{{/content.date}}` expands to the current browser time converted to the UTC value and formatted into the Content REST API date format.

• `now` can also be augmented with `hour`. So you have:

  `now +/- [hour | day | week | month | year]`

• For example:
  
  – `{{#content.date}}now + 2 hours{{/content.date}}` - two hours from now converted to UTC time formatted in the Content REST API date format
  
  – `{{#content.date}}now - 1 day{{/content.date}}` - yesterday at this browser time converted to UTC time formatted to Content REST API date format

Using the Supported Component Syntax

To use the macro expansion in the additional query string, suppose you wanted to return everything in the last 3 weeks, you would enter the following:

`updatedDate gt "{{#code.date}}today - 3 weeks{{/code.date}}"`

Only the date is returned, so to work on the Content REST API call, quotes are added when you construct the query string as you would when entering a static value.

MustacheJS

For syntax, reference the Mustache JS template pages at mustache.github.io/mustache.5.html.

One change has been made to the Mustache instance that is run when expanding strings. Mustache provides both a text expansion that uses `{{ }}` and an html expansion that uses `{{{ }}}`). The difference between these two is that the text expansion does an HTML encode on the string; that is, if the value expanded to `a < b`, then the result would be `a &lt; b`. This is not what you would want to construct strings for URLs. You could tell the user to use the HTML expansion, but that is just an overhead and will generate more issues, like explaining why they need to use `{{{ }}}`).

To avoid this, Mustache has been set up so that it doesn't escape values when using `{{ }}`). This means that both `{{ }}` and `{{{ }}])` behave the same. This also leaves the encoding of any result as an exercise for the user if it's required.

The OOTB Mustache Model

Mustache requires a model to be applied to the template for expansion. In the preceding example, `{{#content.date}}` is already defined out-of-the-box, whereas a new property, such as `{{person}}`, can be added by the developer. If the user enters a value in the Mustache template that isn't in the model, the result will be an empty string. So, in the case of `Hello {{person}}`, it would expand to just `Hello` unless the developer adds `person` to the model.

The model object used is a global object called `SCSMacros`. The developer is free to add any additional entries into this object. The object will be passed to Mustache when the template is evaluated.
The out-of-the-box model object currently supports only the `content.date` object:

```javascript
{
    content: {
        date: <lambda implementation>
    }
}
```

### Custom Mustache Model

The supported objects can be enhanced by the developer based on their requirements. So they can introduce a `lastTwoDays` object and simplify the expansion to just `{{lastTwoDays}}`.

To extend the model to support something like `Hello {{person}}` in the preceding example, you would need to add the `person` object to the SCSMacros. You need to do this before the page is run. It can be done within the page layout by adding a script tag to the start of it. For example:

```html
<script type="text/javascript">
window.SCSMacros = window.SCSMacros || {};  // define/get the SCSMacros object
window.SCSMacros.person = "World";
</script>
```

After this change is made, the `Hello {{person}}` template would expand to: `Hello World`.

If you want to pass values to the object (for example, `Hello {{#person}}personId{{/person}}`), then you need to implement a mustache lambda and wrap and expand the value within the implementation.

For example:

```html
<script type="text/javascript">
window.SCSMacros = window.SCSMacros || {};  // define/get the SCSMacros object
window.SCSMacros.person = function () {
    var people = {'111': { firstName: 'Small', lastName: 'World'}, '222': { firstName: 'Big', lastName: 'Universe'});
    return function (text, render) {
        var expandedText = render(text);
        var chosenPerson = people[expandedText] || people['111'];
        return chosenPerson.firstName;
    }
};
</script>
```

After this change is made, the `Hello {{#person}}111{{/person}}` template would expand to `Hello Small`, and the `Hello {{#person}}222{{/person}}` template would expand to `Hello Big`.  

---

**Chapter 7**  
Develop Content Layouts  
7-43
Create the Sample Blog Template

The BlogTemplate sample demonstrates content layout features.

You can create the template in Developer Cloud Service, examine the content layouts in the template, and test the search capabilities:

1. Create a template of the type `BlogTemplate`:
   
   ```
   cec create-template MyBlogTemplate -f BlogTemplate
   
   A new template, MyBlogTemplate, will be created by copying BlogTemplate.
   ```

2. Open `http://localhost:8085/`, and then click `Templates`, then `MyBlogTemplate`.

3. You see a list of Blog posts. Click on one of them. It takes you to the Details page, which uses three different Content Layouts to render.

4. Click `More articles from this author`. It takes you to a search page.

5. Click an author name. That content layout raises a trigger and shows the articles on the right side.

6. Go to the Home page again and test the search.

Add Content Layout Mappings to Templates

After you create the content layout, you can add it to local templates in your Developer Cloud Service project by adding a content layout mapping.

Use the `cec add-contentlayout-mapping` command to add a content layout mapping for a template. The content type that the content layout is based on (`-c`) and the template that the mapping is for (`-t`) are required. For example:

```
cec add-contentlayout-mapping Blog-Post-Detail-Layout -c Blog-Post -t BlogTemplate
```

The default content layout mapping is the Default style for desktop. You can specify the `-s <layoutstyle>` option to specify a different layout style, such as Overview or Details, by name:

```
cec add-contentlayout-mapping Blog-Post-Detail-Layout -c Blog-Post -t BlogTemplate -s Details
```
You can also set the mapping for mobile with the \texttt{-m} option:

\texttt{cec add-contentlayout-mapping Blog-Post-Detail-Layout -c Blog-Post -t BlogTemplate -m}

Test Content Layouts with the Local Test Harness

After you add your content types and content layout mappings to a template, you can test your content layouts in the local test harness.

See \texttt{Test with a Local Test Harness}.

Import Templates with Content Layouts into Oracle Content and Experience Cloud

After you develop and test your content layouts, you can export the template that contains the content layouts from your Developer Cloud Service project and then import the template into Oracle Content and Experience Cloud.

See \texttt{Export a Template}. 
Work with Designs and Styles

Each theme for Oracle Content and Experience Cloud must have a design that specifies the look and feel of pages and style settings for components used in the default site for the theme.

- **About Designs**
- **Design Files**
- **Customize Conversation List Styles**
- **Customize Folder List and File List Styles**
- **Customize Social Bar Icons**
- **Configure Interview Styling Extensions for Oracle Policy Automation**

**About Designs**

Each theme for Oracle Content and Experience Cloud must have a design, which specifies the look-and-feel of pages and style settings for components used in the theme.

When users create a site, they must select a template which includes a theme by default. You can create or choose to use a different theme for a site.

When creating a theme, along with designing page layouts you must specify the available styles that will be displayed in the Settings panel for each component type that will be available to users (Paragraph, Title, Image, and so on). There can be multiple styles within a design to specify the default settings for different components. You make choices for text fonts and font sizes, image framing, and so on, then save them as a named design. The items in the design influence the look of the page layouts when rendered.

See Work with Site Pages in *Creating Experiences with Oracle Content and Experience Cloud*.

**Design Files**

Two files are considered the default design files for a theme: `design.json` and `design.css`.

- `design.json` specifies styles for components
- `design.css` provides definitions for class values (such as color and font)

These files are located in the `/designs/default/` directory in a theme structure.
design.json File

The design.json file has the following structure:

```json
{
    "componentStyles": {
        "scs-image": {
            "styles": []
        },
        "scs-map": {
            "styles": []
        },
        "scs-title": {
            "styles": []
        },
        "scs-paragraph": {
            "styles": []
        },
        "scs-divider": {
            "styles": []
        },
        "scs-button": {
            "styles": []
        },
        "scs-app": {
            "styles": []
        },
        "scs-spacer": {
        },
        "scs-gallery": {
            "styles": []
        },
        "scs-youtube": {
            "styles": []
        },
        "scs-socialbar": {
            "styles": []
        },
        "scs-document": {
            "styles": []
        }
    }
}
```

Each of the "styles":[] entries can contain a list of styles for that particular component. For example, the title component provides these default styles:

```
"styles": [{
    "name": "COMP_STYLE_FLAT",
    "class": "scs-title-default-style"
},
{
    "name": "COMP_STYLE_HIGHLIGHT",
```
Note:

If you create custom styles for a component and map it to styles in design.json, you don’t need to use the full name like those provided with the system. Just specify the string you want to use. For example, instead of "COMP_STYLE_BOX" for the name, simply use "Box". This means only the name "Box" will appear in the list for the Styles tab of the Settings panel for that component, instead of "COMP_STYLE_BOX".

The name values are mapped to the actual words to display in the user interface, like this:

"COMP_STYLE_FLAT": "Flat",
"COMP_STYLE_HIGHLIGHT": "Highlight",
"COMP_STYLE_DIVIDER": "Divider",

design.css File

The design.css file provides the definitions for the class values. Here are a few examples.

.scs-title-default-style {
  color: #333333;
  display: block;
  font-family: "Helvetica Neue", "Helvetica", "Arial", sans-serif;
  font-size: 24px;
  font-weight: normal;
}

.scs-title-style-2 {
  background-color: #DEF300;
  color: #333333;
  font-family: adobe-clean, sans-serif;
  padding-top: 2em;
  padding-bottom: 2em;
}

.scs-button-default-style .scs-button-button:hover {
  background: #f7f8f9;
  border: 1px solid #c4ced7;
  color: #0572ce;
Customize Conversation List Styles

You can customize the style of a Conversation List component by adding selectors in the design.css file.

Use these CSS selectors to customize the style of the Conversation List component.

<table>
<thead>
<tr>
<th>Selector Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-container</td>
<td>Outmost DIV of the component</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-title</td>
<td>Title of a conversation in the list when it is selected</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-line-separator</td>
<td>Separator between list title and the list</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-convo-title</td>
<td>Title of a conversation in the list</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-convo-line-separator</td>
<td>Separator between each conversation</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-active</td>
<td>Title of a conversation in the list when it is selected</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-convo-posts</td>
<td>Number of posts of a conversation</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-convo-unread</td>
<td>Number of unread messages of a conversation</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-convo-updated</td>
<td>Last updated date and of a conversation</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-no-convo-msg</td>
<td>Message when the list is empty</td>
</tr>
<tr>
<td>.scs-convo-list-cust .scs-convo-list-no-auth-msg</td>
<td>Message when the Conversation List is rendered in public site without user authorization</td>
</tr>
</tbody>
</table>

See Using Styles and Formatting in Creating Experiences with Oracle Content and Experience Cloud.

Example

This sample illustrates the use of customized CSS for type font, style, and color changes to a Conversation List.
The following code shows the customized CSS used to create the sample:

```
.scs-convo-list-cust .scs-convo-list-container {
    background-color: azure;
}

.scs-convo-list-cust .scs-convo-list-title {
    color: crimson;
}

.scs-convo-list-cust .scs-convo-list-line-separator {
    border-bottom: 2px dashed #dfe4e7;
}

.scs-convo-list-cust .scs-convo-list-convo-title {
    font-style: italic;
}

.scs-convo-list-cust .scs-convo-list-active {
    text-decoration: underline;
}

.scs-convo-list-cust .scs-convo-list-convo-posts {
    color: cadetblue;
    font-size: 12px;
}

.scs-convo-list-cust .scs-convo-list-convo-unread {
    color: brown;
    font-size: 12px;
    float: left;
}

.scs-convo-list-cust .scs-convo-list-convo-updated {
    color: blueviolet;
    font-size: 12px;
    clear: none;
}

.scs-convo-list-cust .scs-convo-list-no-convo-msg {
    font-size: 18px;
    color: darkorange;
}
```
Customize Folder List and File List Styles

You can customize the styles of Folder List and File List components by adding selectors in the `design.css` file.

You can use a Folder List component to list the folders within a specified folder from your Oracle Content and Experience Cloud account. The folder list automatically communicates with a File List component and document manager on the page to display the files in a folder selected in the folder list.

You can use a File List component to provide a view of files from a specified folder in your Oracle Content and Experience Cloud account. The file list automatically communicates with a Folder List component on the page to display the files in a folder selected in the folder list.

Folder List CSS Selectors

Use these CSS selectors to customize the style of the Folder List component.

<table>
<thead>
<tr>
<th>Selector name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.scs-folder-list-cust .scs-folder-list-container</code></td>
<td>Outmost DIV of the component</td>
</tr>
<tr>
<td><code>.scs-folder-list-cust .scs-folder-list-folder-title</code></td>
<td>Folder name</td>
</tr>
<tr>
<td><code>.scs-folder-list-cust .scs-folder-list-line-separator</code></td>
<td>Separator between the folder name and the list of subfolders</td>
</tr>
<tr>
<td><code>.scs-folder-list-cust .scs-folder-list-sub-folder-title</code></td>
<td>Subfolder name</td>
</tr>
<tr>
<td><code>.scs-folder-list-cust .scs-folder-list-sub-folder-title-active</code></td>
<td>Subfolder name when it's selected</td>
</tr>
<tr>
<td><code>.scs-folder-list-cust .scs-folder-list-no-folder-msg</code></td>
<td>Message when there is no subfolder to display</td>
</tr>
</tbody>
</table>

File List CSS Selectors

Use these CSS selectors to customize the style of the File List component.

<table>
<thead>
<tr>
<th>Selector name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.scs-file-list-cust .scs-file-list-container</code></td>
<td>Outmost DIV of the component</td>
</tr>
<tr>
<td><code>.scs-file-list-cust .scs-file-list-folder-title</code></td>
<td>Folder name</td>
</tr>
<tr>
<td><code>.scs-file-list-cust .scs-file-list-line-separator</code></td>
<td>Separator between the folder name and the list of files</td>
</tr>
<tr>
<td><code>.scs-file-list-cust .scs-file-list-row</code></td>
<td>Row that contains the information for a file</td>
</tr>
</tbody>
</table>
### Customize Social Bar Icons

You can create custom social icons to use in the social bar in a theme’s default site.

Social icons that appear in the social bar in a site are determined by the design of the site’s theme. If you change the theme for a site, the social icons change with the theme. Common social icons are included with Oracle Content and Experience Cloud themes for Facebook, Twitter, LinkedIn, Google+, and YouTube.

You can add custom social icons to the social icon component by editing the `design.json` and `design.css` files.

#### design.json File

In the `design.json` file, you can specify new icons using the name and class structure as shown in this sample code:

```json
"componenticons": {
    "scs-socialbar": {
        "icons": [
        {  
            "name": "COMP_ICON_FACEBOOK",
            "class": "scs-facebook-icon"
        }
        
```

<table>
<thead>
<tr>
<th>Selector name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.scs-file-list-cust .scs-file-list-left-col</td>
<td>File’s thumbnail located in the left section of the component</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-mid-col</td>
<td>Middle section of the component, which contains the name, description, last modification, and size of a file</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-file-title</td>
<td>File name located in the middle section of the app</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-file-desc</td>
<td>File description located in the middle section of the app</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-file-lastModified</td>
<td>Last modification of the file</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-file-size</td>
<td>Size of the file with a vertical separator from the last modification</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-file-size-no-sep</td>
<td>Size of the file without a vertical separator (last modification not shown)</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-right-col</td>
<td>Right section of the app</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-file-download-icon</td>
<td>Download icon located in the right section of the app</td>
</tr>
<tr>
<td>.scs-file-list-cust .scs-file-list-no-file-msg</td>
<td>Message when there is no file to display</td>
</tr>
</tbody>
</table>

See File Lists and Folder Lists in *Creating Experiences with Oracle Content and Experience Cloud*. 

---

**Chapter 8**

**Customize Social Bar Icons**

---

8-7
design.css File

In the design.css file, you can add new icons using the name and url specification as shown in this sample code:

```
.scss-facebook-icon {
    background-image: url("facebook.png");
}
.scss-twitter-icon {
    background-image: url("twitter.png");
}
.scss-linkedin-icon {
    background-image: url("linkedin.png");
}
```

Configure Interview Styling Extensions for Oracle Policy Automation

You can style Oracle Policy Automation (OPA) interviews to unify the appearance with your corporate look and feel.

You can configure the following CSS class selectors in the design.css file. All of these class selectors have the prefix “scss-opainterview-”.

Each class defines all styling for the specified component. The class has full control, and the existing OPA style won’t be used.

<table>
<thead>
<tr>
<th>Selector</th>
<th>Applies to</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interview</td>
<td>interview</td>
<td>The interview region comprises the entire interview content, including the header, footer and navigation area.</td>
</tr>
<tr>
<td>interviewContent</td>
<td>interview content</td>
<td>The interview content region includes the screen title and controls but excludes the header, footer and navigation area.</td>
</tr>
<tr>
<td>Selector</td>
<td>Applies to</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>screenTitleBlock</td>
<td>screen title block</td>
<td>The screen title block composes the region that includes the screen title as well as any other widgets that are contained in that row, such as the screen drop-down list and/or next and back buttons.</td>
</tr>
<tr>
<td>screenTitle</td>
<td>screen title</td>
<td>The screen title region is just the region containing the screen title.</td>
</tr>
<tr>
<td>nextButton</td>
<td>next button</td>
<td>The next button.</td>
</tr>
<tr>
<td>backButton</td>
<td>back button</td>
<td>The back button.</td>
</tr>
<tr>
<td>restartButton</td>
<td>restart button</td>
<td>The restart button.</td>
</tr>
<tr>
<td>exitButton</td>
<td>exit button</td>
<td>The exit button.</td>
</tr>
<tr>
<td>header</td>
<td>header</td>
<td>The header region.</td>
</tr>
<tr>
<td>footer</td>
<td>footer</td>
<td>The footer region.</td>
</tr>
<tr>
<td>question</td>
<td>question text</td>
<td>Styling for question text.</td>
</tr>
<tr>
<td>control</td>
<td>container for controls</td>
<td>Styling for the element that contains controls.</td>
</tr>
<tr>
<td>label</td>
<td>label control</td>
<td>Styling for label controls.</td>
</tr>
<tr>
<td>controlError</td>
<td>all controls</td>
<td>Styling for error text container.</td>
</tr>
<tr>
<td>controlErrorText</td>
<td>all controls</td>
<td>Styling for error text span.</td>
</tr>
<tr>
<td>textInput</td>
<td>single line text box, password and masked</td>
<td>Styling for text input controls.</td>
</tr>
<tr>
<td>textAreaInput</td>
<td>multi-line text box</td>
<td>Styling for multiline text input controls.</td>
</tr>
<tr>
<td>calendarInput</td>
<td>calendar</td>
<td>Styling for calendar input controls. Supports an iconColor field that allows the color of the calendar icon to be changed, and a keepIcon field that indicates whether the calendar icon should be displayed.</td>
</tr>
<tr>
<td>dropDownInput</td>
<td>drop-down list</td>
<td>Styling for drop-down list input controls.</td>
</tr>
<tr>
<td>filterDropDownInput</td>
<td>filtered drop-down list</td>
<td>Styling for filtered drop-down list input controls. Supports an iconColor field that allows the color of the drop-down arrow to be changed.</td>
</tr>
<tr>
<td>listInput</td>
<td>fixed list</td>
<td>Styling for fixed list input controls.</td>
</tr>
<tr>
<td>radioInput</td>
<td>radio buttons</td>
<td>Styling for radio button input controls. They can be styled with borderColor and fillColor options. The iconType property can change the type of icon used. Currently 'tick' and 'fill' are the only supported alternate options.</td>
</tr>
<tr>
<td>checkboxInput</td>
<td>checkbox</td>
<td>Styling for checkbox input controls. They can be styled with borderColor and fillColor options. The iconType property can change the type of icon used. Currently 'square' and 'fill' are the only supported alternate options.</td>
</tr>
<tr>
<td>autoCompleteInput</td>
<td>custom search</td>
<td>Styling for the autocomplete field when a customSearch extension is used.</td>
</tr>
<tr>
<td>captchaInput</td>
<td>CAPTCHA input field</td>
<td>Styling for the input field that the user is entering the CAPTCHA into.</td>
</tr>
</tbody>
</table>
### Selector | Applies to | Description
--- | --- | ---
signatureInput | signature control | Styling for signature controls. Supports an additional inkColor field that allows changing the pen ink for the signature.
explanationHeader | explanation control | Styling for the top level expandable header for explanation controls.
explanationText | explanation control | Styling for the expanded explanation control text.
signatureClearButton | signature control | Styling for the clear button on a signature control.
uploadAddButton | upload control | Styling for the add button on the upload control.

text | entity collect control | The entity collect control.

#### Examples with Style Extensions Defined in design.css

```css
.scs-opainterview-interviewContent {
    background-color: beige;
}
.scs-opainterview-screenTitleBlock {
    background-color: bisque;
}
.scs-opainterview-screenTitle {
    font-style: italic;
    font-size: 20px;
}
.scs-opainterview-nextButton {
    color: darkgreen;
}
.scs-opainterview-backButton {
    color: crimson;
}
.scs-opainterview-question {
    color: green;
}
.scs-opainterview-control {
    background-color: cornflowerblue;
}
.scs-opainterview-label {
    color:aqua;
}
.scs-opainterview-textInput {
    color: red;
    cursor:crosshair;
}
.scs-opainterview-radioInput {
    background-color: pink;
}
.scs-opainterview-checkboxInput {
    cursor: pointer;
}
```
Work with Backgrounds

You can specify the background color and image for site pages and for individual slots on a page. Backgrounds for pages, slots, and components layer on top of one another. For example, if you specify a background for a slot, it is layered above the background specified for the page.

- About Backgrounds and Themes
- How Backgrounds Are Implemented
- Where Settings Are Stored

About Background and Themes

The background feature is primarily intended for use in site pages and slots. Backgrounds set for pages and slots are meant to be configured by users, not developers, at the theme level.

Background effects can be configured in theme slots, however, these effects may override any background effects set for pages when editing a site.

The background feature does not alter themes or constituent theme files. A theme can't be altered by the background settings for a page, and another page based on the same layout in a site won't inherit any of the background settings for the source page. Adding or changing the background through a theme requires an update to the theme.

Users can configure different backgrounds on every page of the site, and also on different pages that use the same layout. This would not be possible within a theme. Although a theme may specify a background for a page, users' background settings can override this specification. (Background settings will only apply overrides for a particular page; it does not in any way modify the theme itself.)

Be careful to avoid having the theme override background styles that will be set up in site pages and slots by users. This can happen in several ways:

- Page background settings in a site can be overridden by a theme when you use element-based "style" attributes on the <body> and the slots.
- Page background settings in a site can be overridden by a theme when you mark theme background styles as "!important" in the theme cascading style sheet (CSS) file.

See Change the Background or Theme in Creating Experiences with Oracle Content and Experience Cloud.

How Backgrounds Are Implemented

To implement the page and slot backgrounds feature, Oracle Content and Experience Cloud dynamically creates a CSS stylesheet in the <head> of a page.
The selector for the styles is a tag-based selector (body) for the page background settings. For the slot background settings and ID-based selector, the slot ID is used.

For example, setting a background color for a page might yield the following CSS markup in the <head> of the page:

```css
body
{
    background-color: #fa7c9d;
}
```

Similarly, setting a background image on a slot might yield the following CSS markup in the <head> of the page:

```css
#PageFooter
{
    background-image: url("footer_image.png");
}
```

This implementation means that styles directly specified in the "style" attribute of the body tag or the slot element can override the settings configured in the stylesheet in the <head> code.

**Important:**
Theme developers should take care not to override background settings with element-based styles.

See Change the Background or Theme in *Creating Experiences with Oracle Content and Experience Cloud*.

**Where Settings Are Stored**

When pages are rendered, background settings are dynamically written to "style" tags in the <head> code for the page.

Background settings are persisted in the page model files (for example, <pageid>.json). In particular, the page background settings are stored in the properties.styles section, and the slot background settings are stored in the slots[slot_id].styles section.

Background settings are stored in the page JSON files, specifically within "styles" as shown in this representative sample.

```json
{
    "properties":
    {
        "pageLayout" : "oneslot.htm",
        "styles": [
            "background-image: url([!--$SCSCONTENTURL--]/background_image.gif)",
```
Chapter 9
Where Settings Are Stored

"background-position: center",
"background-size: auto",
"background-repeat: repeat",
"background-origin: padding-box",
"background-clip: border-box"
],
},
"slots": {
"slot100": {
"components": [
"dedda3a8-615d-44ad-ad71-51f2fa465cef",
"95eb0fd6-bcfc-4e5e-ba67-a5c8c5d9c315"
],
"grid": "<div class="scs-row"><div class="scs-col"
"style="width: 50%;"
><div id="dedda3a8-615d-44ad-ad71-51f2fa465ced">
</div>
</div>
<
</div>
"class="scs-col" style="width: 50%;">
<
</div>
</div>
"style=
<
class="scs-col"
"width: 50%;">
"style=",
"grid": 
"background-image: url(['--$SCS_CONTENT_URL--']/
oracle-cloudworld.jpg),
"background-position: center",
"background-size: cover",
"background-repeat: no-repeat",
"background-origin: padding-box",
"background-clip: border-box",
"background-color: transparent"
]}
},
"componentInstances": {
"dedda3a8-615d-44ad-ad71-51f2fa465cef": {
"type": "scs-title",
"data": {
"alignment": "fill",
"backgroundColor": "",
"borderColor": "#808080",
"borderRadius": 0,
"borderStyle": "none",
"borderWidth": 1,
"foregroundColor": "#333333",
"fontFamily": "'Helvetica Neue', Helvetica, Arial,
sans-serif",
"background-image: url(['--$SCS_CONTENT_URL--']/
oracle-cloudworld.jpg),
"background-position: center",
"background-size: cover",
"background-repeat: no-repeat",
"background-origin: padding-box",
"background-clip: border-box",
"background-color: transparent"
]}
},
"componentInstances": {
"dedda3a8-615d-44ad-ad71-51f2fa465cef": {
"type": "scs-title",
"data": {
"alignment": "fill",
"backgroundColor": "",
"borderColor": "#808080",
"borderRadius": 0,
"borderStyle": "none",
"borderWidth": 1,
"foregroundColor": "#333333",
"fontFamily": "'Helvetica Neue', Helvetica, Arial,
Chapter 9
Where Settings Are Stored

9-4
Communication between components (including components rendered in inline frames) can be configured so a trigger within a component calls an action on another component.

- About Triggers and Actions
- Set Triggers
- Set Actions

**About Triggers and Actions**

Communication between components (including components rendered in inline frames) can be configured so a **trigger** within a component calls an **action** on another component.

Triggers are part of Oracle Content and Experience Cloud intercomponent communication. Any component can raise any number of triggers. The component can provide a payload for a trigger, which then is passed to any action that is executed when the trigger is raised. You can select what actions will be executed for each trigger. Components that are built to work together can automatically raise triggers to execute actions on the other component without user interaction.

The basic process involves:

1. Registering triggers
2. Raising triggers
3. Registering actions
4. Executing actions to verify the setup

For example, you can use the Button component to perform one or more actions such as showing or hiding page components and showing messages. You could have a list of business office locations in one component and when a location in the list is clicked, then details about the location are displayed in another component.

For components you customize, the triggers and actions are part of the component registration data and not part of the component implementation. In the registration data there is a "triggers": [], and "actions":[], entry that contains the list of triggers and actions the component supports. The actual syntax is the same as for local and remote components, only the location and how it’s retrieved is different.

See **Use Triggers and Actions in Creating Experiences with Oracle Content and Experience Cloud**.

**Set Triggers**

A component can include triggers that will execute actions in other components. You must register triggers to be raised by components.
The component provides a payload for a trigger, which is passed to any action that is executed when the trigger is raised. You can select what actions will be executed for each trigger. Components that are built to work together can automatically raise triggers to execute actions without user interaction.

**Register Triggers**

For a custom component, triggers are registered as part of the registration data for the component. To add a trigger, update the "triggers" property array with each trigger the component supports. You also must specify the payload the trigger supports so that the user interface can be created to allow users to map values within the payload to properties supported by the action.

1. **Edit the appinfo.json file and review the "triggers":[], entry:**

```
"triggers": [{
  "triggerName": "helloWorldWhoAreYou",
  "triggerDescription": "Show Who I Am",
  "triggerPayload": [{
    "name": "whoAreYou",
    "displayName": "Who I Am"
  }]
},
```

2. **Sync the file to the sites server.**

In this sample trigger entry, you've defined a `triggerName` ("helloWorldWhoAreYou"). The name value must be unique. You've then given the trigger a description ("Show Who I Am"), which is used by the user interface dialog to display your trigger. Finally, you've defined a single value payload for the trigger; users will be able to select entries in this payload and map them to fields in the action.

Once a trigger is registered, you should be able to see and select the trigger when you go to the Link tab in the Settings panel for your component.

**Raise Triggers**

Triggers can be raised at any point by a component. Typically, a trigger is raised by a user interaction, such as clicking a button or selecting a row in a table. A component can raise the trigger based on any criteria, for example, when data changes because of a REST API call. You can execute any number of actions when a trigger is raised.

Here's an example of how to raise a trigger:

1. **Edit the render.js file and add a JavaScript function in the viewModel object that will call the Sites SDK to raise the trigger.**

```
self.raiseTrigger = function (triggerName) {
  SitesSDK.publish(SitesSDK.MESSAGE_TYGES.TRIGGER_ACTIONS, {
    'triggerName': 'helloWorldWhoAreYou',
    'triggerPayload': { "whoAreYou": "This is " + self.whoAreYou() + "!"}
  });
};
```
2. Add an entry in the user interface to call the function to raise the trigger (`-edit-template.html`) and a button before the `</div>`.

```html
<button data-bind="click raiseTrigger">Who Am I?</button>
```

3. Sync or upload the `render.js` file to your Oracle Content and Experience Cloud instance server.

In the `ViewModel` object, you created a JavaScript function that is called when the button is clicked. This function calls the Sites SDK to tell it to trigger all the actions defined for this trigger "helloWorldWhoAreYou". It also passes through a triggerPayload that has a single field, "whoAreYou". These values "helloWorldWhoAreYou" and "whoAreYou" match those you entered when you registered the trigger in the previous step.

> Note:

There is no predefined order to when an action is executed. Although each action will be called in the order it is listed, there is no wait for it to complete before the next action is called. If an action makes an asynchronous call, it may not complete before the next action is executed.

## Set Actions

You can set a component to leverage action registration so that it can be dropped on a page that will execute actions within your component.

### Register Actions

Actions are called on components when triggers are raised. A component can register any number of actions and also define the payload the action supports. When a user selects an action, they can populate the payload to be passed to the action.

As with registering triggers, you can register actions that your component supports in the `appinfo.json` registration data for your theme.

Here's an example of how to register an action:

1. Edit the `appinfo.json` file for your component and update the "actions": [], entry.

```json
"actions": [{
    "actionName": "helloWorldChangeWhoIAm",
    "actionDescription": "Change Who I Am",
    "actionPayload": [{
        "name": "whoAreYou",
        "description": "Who are you?",
        "type": {
            "ojComponent": {
                "component": "ojInputText"
            }
        },
        "value": ""
    }
},
```

Chapter 10
Set Actions

10-3
2. Once registered, the action will be visible in the action dialog that's invoked when you click a trigger on the Link tab on the Setting panel for your component.

**Execute Actions**

Once an action is registered, you'll be able to drop components onto the page that execute actions within the component. For a component to execute an action, it must listen for the EXECUTE_ACTION message. This message also includes the payload passed to the action from which you must extract the expected values.

As an example, to listen for the EXECUTE_ACTION message, edit the render.js file and update the ViewModel object with these entries:

```javascript
self.executeActionListener = function (args) {
  // get action and payload
  var payload = $.isArray(args.payload) ? args.payload[0] : {},
      action = args.action,
      actionName = action && action.actionName;

  // handle 'helloWorldChangeWhoIAm' actions
  if ((actionName === 'helloWorldChangeWhoIAm') && (payload.name === 'whoAreYou')) {
    self.whoAreYou(payload.value);
  }
};
```

This creates a JavaScript function to execute the action, then uses the Sites SDK to call the function whenever the EXECUTE_ACTION message is raised.

The action will be called whenever an EXECUTE_ACTION message is raised, and it's up to the component to handle only actions it is designed to handle. To do this, you must check the name of the action to ensure it is one you can handle.

The payload for the action is an array of values. In the example, it's assumed that the value is the first entry in the array. Typically, you must find the payload values you care about from the array.

---

**Note:**

Because the action listener is a callback, you should use JavaScript Closure or appropriately bind the function to ensure you have access to your ViewModel when the function is executed.
Specify Site Redirects

When you restructure or move a web site, you can redirect user requests from old URLs to their current ones. Specifying 30x redirects for URLs can maintain bookmarks or published links across site redesigns.

Pages that have high reputation rankings in search engines could move to different URLs when you move to Content and Experience Cloud hosted sites from other infrastructure technologies. Redirects help reorganize the URL structure of a site and preserve the search engine rankings.

- Plan for Redirects
- Specify Redirect Rules in a JSON File
- Upload a Redirect Rules File to a Site

Plan for Redirects

You can specify redirects that send HTTP 30x responses for designated URLs. If a request does not match one of the nominated redirects, then regular processing of the URL happens, and the page is returned in the normal way.

You can create a JSON file specifying redirects and upload that file to the server. The server will use the JSON file as it process incoming request URLs.

Two kinds of redirect rules let you redirect incoming URLs to new locations:

- Simple String-to-String Matching
- Simplified Wildcard Matching

Simple String-to-String Matching

For simple string matching and replacement, you can specify explicit URLs and then redirect each URL by mapping it directly to a target URL.

The following table shows some sample string-to-string matchings.

<table>
<thead>
<tr>
<th>Source URL</th>
<th>Target Location URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>/index.html</td>
<td>/home.htm</td>
</tr>
<tr>
<td>/products/widget</td>
<td>/items/knickknack</td>
</tr>
<tr>
<td>/index?page=widgets</td>
<td>/items/widgets</td>
</tr>
</tbody>
</table>

String-to-string mappings are simple to understand and test. The rules evaluate quickly using simple string matchings and map lookups.

However, there is little flexibility regarding URL query parameters. They would need to match exactly. Extra URL parameters or parameters in a different order would cause a rule to not match.
Simplified Wildcard Matching

Simplified wildcard matching lets a rule match many URLs while also limiting the amount of regular expression backtracking required to obtain a result.

Because regular expressions can be complicated to write, and because poorly constructed ones can evaluate for an undetermined amount of time (ReDoS), a second type of rule allows a simplified matching mechanism. It uses a wildcard character ("*") to match zero (0) or more characters in the incoming URL, and the keyword 'wildcard' with an index value to copy incoming parts of the URL to the redirected URL.

The following table shows some sample simplified wildcard matchings.

<table>
<thead>
<tr>
<th>Source URL</th>
<th>Target Location URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>/old/*</td>
<td>/new/&lt;$wildcard(1)$&gt;</td>
</tr>
<tr>
<td>/dispatch.asp?page=* &amp;facet=Lang*</td>
<td>/page&lt;$wildcard(1)$&gt;/&lt;$wildcard(2)$&gt;</td>
</tr>
</tbody>
</table>

Simplified wildcard matching gives more power to the matching of URLs than simple string-to-string matching, but does so without unbounded regular expression processing. The syntax is simple, and you can use pattern matching on a URL to accommodate a large number of URLs with one pattern.

Because they are based upon regular expressions, wildcard rules would evaluate somewhat slower than simple string matching. A large number of rules could introduce a performance penalty to general page-delivery performance.

Specify Redirect Rules in a JSON File

You can specify redirect rules for URLs in a JSON file.

Use the following format in a JSON file to specify redirect rules for URLs.

```json
{
    "redirectRules": [
        {
            "type": "string",
            "comment": "this rule is applied first",
            "expression": "/index.htm",
            "location": "/home.html"
        },
        {
            "type": "wildcard",
            "expression": "/items/*?page=*",
            "location": "/<$page$>?item=<$wildcard(1)$>",
            "code": 302
        }
    ]
}
```
The outer containing structure in the JSON file is an array. The array contains rule instances.

The "string" rules will be evaluated first, followed by the "wildcard" rules, in order. Once one of the rules matches, the evaluation of subsequent rules is abandoned, and the corresponding redirect is generated.

Each rule has the following properties:

- The "comment" property is an optional string that has no impact upon the evaluation of the rules. It includes notes or commentary.
- The "expression" property is a required string that matches against the incoming candidate URL. In a wildcard rule, the asterisk (*) token matches zero or more characters.
- The "location" property is a required string that indicates the location or destination of the redirect. The redirect can be a full or relative URL.
- The "code" property is an optional integer that provides the HTTP response code to use when issuing the redirect. The value must be one of the following integers:
  - 301: Indicates that the resource moved permanently. This is the default value if the "code" property is omitted.
  - 302: Indicates that the resource moved temporarily.
- The "type" property is an optional string that indicates the type of redirect rule. The value must be one of the following strings:
  - "string" specifies a faster rule whose expression matches the entire input URL exactly.
  - 1. "wildcard" specifies a wildcard rule that can match a number of URLs. This is the default value if the property is omitted.

Location Tokens

You can use location tokens to help manufacture a redirect location. Each of the following location tokens can help specify a redirect:

- <urlPath$>: The path portion of the matching URL.
- <urlQueryString$>: The entire URL query string from the matching URL.
- <urlQueryStringExcept(name1,name2)$>: The entire URL query string from the matching URL minus the named parameters.
- <wildcard(N)$>: The one-based index of the matching wildcard in the matching URL. (This is analogous to \1..\9 in regular expressions.)
- <name$>: The value of the named query string parameter. For example, if you have the query string msmith: ?page=42 on the input, then you can use <page$> in the location to put '42' into the location.

Restrictions

The following restrictions apply to the redirects.json file as a whole and to the rules it contains:

- The maximum overall file size accepted by Content and Experience Cloud is 250 KB.
• The maximum number of rules in the `redirects.json` file is 1,000.
• The maximum "expression" length for a rule is 1,000 characters.
• The maximum "location" length for a rule is 2,000 characters.
• The maximum number of '*' tokens in a wildcard rule expression is 10.

String Matching Example

Rule:

```json
{
    "type": "string",
    "expression": "/old/page.jsp?id=material&type=glass",
    "location": "/new/<$id$>.htm"
}
```

The following URL would match the rule:

`/old/page.jsp?id=material&type=glass`

• The resulting location would be: `/new/material.htm`
• The entire URL matches, including the query string.
• Although `<$id$>` is used in the location, it is not necessary for this example because only one possible query string could match. The location could have been written as `/new/material.htm`.

The following URLs would not match the rule:

• `/old/page.jsp` (The rule’s expression gives a query string that must match.)
• `/old/page.jsp?id=material&type=glass&index=2` (The extra &index=2 in the candidate URL does not exactly match the rule expression.)
• `/old/page.jsp?type=glass&id=material` (The ordering of query string parameters must match in a "string" rule.)

Wildcard Matching Example

Rule:

```json
{
    "type": "wildcard",
    "expression": "/old/*/pages/*?id=*&item=sheet-*",
    "location": "/new/<$id$>/$wildcard(4).html"
}
```

The following URLs would match the rule:

• `/old/phones/android/pages/info.asp?id=XT1045&item=sheet-specs`  
  - The resulting location would be: `/new/XT1045/specs.html`
The path portion of the URL matches, so the query string is also examined for matching conditions.

The parameters in this example happen to match the ordering of the parameters in the rule expression, but this is not required.

- /old/phones/android/pages/info.asp?item=sheet-specs&id=XT1045

  The resulting location would be: /new/XT1045/specs.html

  The path portion of the URL matches the rule expression before the question mark (?), so the parameters are also checked for a match.

  Although the parameters are listed in a different order in the rule expression, the parameters are matched individually.

- /old/phones/android/pages/info.asp?id=XT1045&item=sheet-specs&unrelated=thing

  The resulting location would be: /new/XT1045/specs.html

  The path portion of the URL matches, so the query string is also examined for matching conditions.

  The candidate URL has an extra &unrelated=thing parameter, but since the named query parameters in the rule expression match, the rule is deemed to match.

  The unrelated parameter would be available in the location as a token, as <$unrelated$>, and would have the value thing, even though it did not contribute to the match of the rule.

The following URLs would not match:

- /old/pages/info.jsp

  (The path portion of the URL does not match the path portion of the rule expression.)

- /old/phones/android/pages/info.asp

  (The path portion of the URL matches the path portion of the rule expression, but the query parameters in the rule expression do not match.)

- /old/phones/android/pages/info.asp?id=cellular

  (The path portion of the URL matches the path portion of the rule expression, but not all of the query parameters in the rule expression match.)

### Upload a Redirect Rules File to a Site

You can upload a redirect rules to a site in Content and Experience Cloud.

To upload a redirect.json file to a site:

1. Open the site for editing.

2. Click in the sidebar and then click.

3. Click Select file to upload and navigate to the file you want to use, select it, then click OK.

4. When you publish the update, the changes are published and put into use.
Develop Components

You can develop components for use in Oracle Content and Experience Cloud sites.

- About Components
- About Developing Components
- Create a Component
- Develop Custom Components with Developer Cloud Service
- Build an H1 Component with a Settings Panel
- Create a Simple Component Using HTML Templates
- Compare Local Components to Remote Components
- Render Component Settings
- Local Component Implementation
- Style Classes for Components
- How to Style Built-In Components
- Set Component Properties
- Components Rendered in Inline Frames
- About the Instance ID and Structure for Components Rendered in Inline Frames
- Security for Remote Components
- Register a Remote Component
- Delete a Component
- Sites SDK

About Components

A component is a specific type of content that you can add to a page in a site. Components in Oracle Content and Experience Cloud include items such as paragraph, title, image, divider, and so on.

Oracle Content and Experience Cloud supports these types of components:

- Local component: files are stored in Oracle Content and Experience Cloud
- Remote component: files are stored on a remote server

Local components can be set to render directly within the page or render in an inline frame in the page. Remote components are always rendered in an inline frame.

Oracle Content and Experience Cloud provides a default set of components with each template (which also includes themes and sites). You can create new components and also use these default components within your components. Once you've added a component to your site, you can edit the component's property settings to meet your requirements by specifying page content, fonts and font sizes,
image framing and placement, and other styles. What settings you can edit depends on the component type.

These components are included with Oracle Content and Experience Cloud.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>scs-title</td>
<td>scs-title</td>
</tr>
<tr>
<td>Paragraph</td>
<td>scs-paragraph</td>
<td>scs-paragraph</td>
</tr>
<tr>
<td>Image</td>
<td>scs-image</td>
<td>scs-image</td>
</tr>
<tr>
<td>Gallery</td>
<td>scs-gallery</td>
<td>scs-gallery</td>
</tr>
<tr>
<td>Gallery Grid</td>
<td>scs-gallerygrid</td>
<td>scs-gallerygrid</td>
</tr>
<tr>
<td>Document</td>
<td>scs-document</td>
<td>scs-document</td>
</tr>
<tr>
<td>Button</td>
<td>scs-button</td>
<td>scs-button</td>
</tr>
<tr>
<td>Map</td>
<td>scs-map</td>
<td>scs-map</td>
</tr>
<tr>
<td>Divider</td>
<td>scs-divider</td>
<td>scs-divider</td>
</tr>
<tr>
<td>Spacer</td>
<td>scsspacer</td>
<td>scs-spacer</td>
</tr>
<tr>
<td>YouTube</td>
<td>scs-youtube</td>
<td>scs-youtube</td>
</tr>
<tr>
<td>Social Bar</td>
<td>scs-socialbar</td>
<td>scs-socialbar</td>
</tr>
<tr>
<td>Article (custom component)</td>
<td>scs-component</td>
<td>scs-comp-article</td>
</tr>
<tr>
<td>Headline (custom component)</td>
<td>scs-component</td>
<td>scs-comp-headline</td>
</tr>
<tr>
<td>Folder List</td>
<td>scs-app</td>
<td>Folder List</td>
</tr>
<tr>
<td>File List</td>
<td>scs-app</td>
<td>File List</td>
</tr>
<tr>
<td>Documents Manager</td>
<td>scs-app</td>
<td>Documents Manager</td>
</tr>
<tr>
<td>Facebook Like</td>
<td>scs-app</td>
<td>Facebook Like</td>
</tr>
<tr>
<td>Twitter Follow</td>
<td>scs-app</td>
<td>Twitter Follow</td>
</tr>
<tr>
<td>Twitter Share</td>
<td>scs-app</td>
<td>Twitter Share</td>
</tr>
<tr>
<td>Facebook Recommend</td>
<td>scs-app</td>
<td>Facebook Recommend</td>
</tr>
<tr>
<td>Image and Text (custom component)</td>
<td>scs-component</td>
<td>scs-comp-image-text</td>
</tr>
</tbody>
</table>

See Arrange Page Content in *Creating Experiences with Oracle Content and Experience Cloud.*

**About Developing Components**

Developing your own custom component enables you to develop compound elements that can be embedded within the site page, using any page technology of choice. This effectively enables you to extend the list of components provided with Oracle Content and Experience Cloud.

Entries for all registered components are stored in the Component Catalog, which is a folder in Oracle Content and Experience Cloud that contains the entries for all registered components.

Don't use the following names for templates, themes, components, sites, or site pages: authsite, content, pages, scstemplate_*, _comps, _components, _compsdelivery, _idcservice, _sitescloud, _sitesclouddelivery, _themes, _.
Component Types

The Component Catalog supports these types of components:

- Local component
- Local component rendered in an inline frame
- Remote component

The type of component is stored as an extension attribute, "xScsAppType", of the component folder. Valid values follow.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local component</td>
<td>All dependencies are expected to be local.</td>
</tr>
<tr>
<td>Local component rendered in an inline frame</td>
<td>Component is served from the same domain as Oracle Content and Experience Cloud.</td>
</tr>
<tr>
<td>Remote component</td>
<td>Component is served from remote location.</td>
</tr>
</tbody>
</table>

Component File Structure

Each of the types of components has a different set of files when created in the Component Catalog, based on how they are implemented.

**Local component:**

/Components/component-name
  - appinfo.json
  - _folder_icon.jpg
  - assets
    - settings.html
    - render.js

**Local component using inline frame:**

/Components/component-name
  - appinfo.json
  - _folder_icon.jpg
  - assets
    - settings.html
    - render.js
  - js
    - sites.min.js
    - knockout.min.js
    - jquery.min.js

**Remote component:**

/Components/component-name
  - appinfo.json
Folder Metadata

The component registration data is stored in the folder metadata. The following properties are used to uniquely identify the component and its type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app name</td>
<td>Name of the folder which designates the name of the component.</td>
</tr>
<tr>
<td>app description</td>
<td>Description of the folder.</td>
</tr>
<tr>
<td>app guid</td>
<td>Every component is associated with a GUID and is stored as an extension attribute xScsItemGUID. The GUID is generated by the server when the component is created.</td>
</tr>
<tr>
<td>app type</td>
<td>Property designates the component type. It is stored as an extension attribute xScsApType.</td>
</tr>
</tbody>
</table>

Note:

The property iconUrl, which is stored as extension attribute xScsAppIconUrl, has been deprecated.

appinfo.json File

The appinfo.json registration file for each type of component contains only data that is not available in the folder metadata. Component properties that are defined in the folder metadata are not duplicated over to the appinfo.json file.

Local component:

```
{
    "settingsData":{
        "settingsHeight":80,
        "settingsRenderOption": "dialog",
        "settingsWidth":300,
        "componentLayouts":[],
        "triggers":[],
        "actions":[]
    },
    "initialData":{
        "customSettingsData":[]
    }
}
```

Local component rendered in an inline frame:

```
{
    "endpoints":{
        "settings":{
```
Remote component:

```json
{
    "endpoints": {
        "widget": {
            "url": "http://www.externaldomain.com/app/render.html"
        },
        "settings": {
            "url": "http://www.externaldomain.com/app/settings.html",
            "height": "300",
            "width": "400"
        }
    },
    "initialData": {
        "customSettingsData": {}  
    }  
}
```

There are no registered "url" values for local components. They use certain files which can be edited for content, but you can’t change the location or name.

- Local components use the files `assets/render.js` and `assets/settings.html`.
- Local components rendered in an inline frame use the files `assets/render.html` and `assets/settings.html`.
- Remote components use whatever "url" values are specified.

Create a Component

You can create customized components for use in Oracle Content and Experience Cloud.

While Oracle Content and Experience Cloud provides many predefined components for use in building a website, as a developer, you can create custom components with access to the same features and capabilities. These component types include local, local rendered in an inline frame, and remote rendered in an inline frame.

Local Component

When you create a local component, you are given a fully functional sample component that you use as a basis for creating your own component. See Create Local Components or Layouts in *Creating Experiences with Oracle Content and Experience Cloud*.
Local Component with an Inline Frame

When you create a local component with an inline frame, you are given a fully functional sample component that you use as a basis for creating your own component. You'll use the same instructions as for creating a local component, and also check the Render in iframe box to create an inline frame version of the component that is stored locally. See Components Rendered in Inline Frames.

Remote Component

When you create a remote component, which uses an inline frame, check the Render in iframe box.

Copy the created files to your remote server, and register the remote component. Test the component before including it in your published site.

See Components Rendered in Inline Frames and Register a Remote Component.

Develop Custom Components with Developer Cloud Service

Oracle Developer Cloud Service helps you develop templates, themes, and custom components for Oracle Content and Experience Cloud.

The Developer Cloud Service integration with Content and Experience Cloud provides a template with tools to develop templates and components. It also provides sample unit tests to start with. The integration includes a Git repository and tools, which help develop templates and components, as well as a local test harness for quick, iterative development of templates, themes, and custom components.

Developer Cloud Service can help you do the following tasks:

- Set up your local development environment to use a Content and Experience Cloud instance for local development and testing of templates, themes, and components
- Create templates and components from samples or starters, run them in the test harness, explore them, and develop the templates, themes and components in a Developer Cloud Service environment
- Import templates or components that were created from Content and Experience Cloud into a Developer Cloud Service environment for source management and further development
- Export templates or components from a Developer Cloud Service environment to be imported into Content and Experience Cloud for use in websites
- Copy an existing component
- Write unit tests
- Optimize components
- Deploy your components to Content and Experience Cloud

The following topics describe how to use the integration of Developer Cloud Service to develop for Content and Experience Cloud:

- Develop a Custom Component for Oracle Content and Experience Cloud
- Optimize Components (Minify) for Better Performance
Develop a Custom Component for Oracle Content and Experience Cloud

Use Developer Cloud Service and your local machine to develop a custom component for Content and Experience Cloud.

The following topics describe the steps for developing and testing a Content and Experience Cloud component with Developer Cloud Service:

1. **Set Up Oracle Content and Experience Toolkit on Your Local Machine.**
2. **Sign in to the Developer Cloud Service Console for Content and Experience Cloud.**
3. **Create a Project in Developer Cloud Service.**
4. **Add Oracle Content and Experience Toolkit to the Project Code in the New Git Repository.**
5. **Develop Your Custom Component**
6. **Test with a Local Test Harness**
7. **Write and Run Unit Tests**

Develop Your Custom Component

Use the cec command-line utility to create a new component, develop the component locally, and then export the component to Content and Experience Cloud.

**Create a Component**

To create and develop a custom component locally, use the following command:

```
cec create-component <component-name> -f <source>
```

Choose one of the following values for `source`:

- JET-CCA-Demo-Card
- local
- local-iframe
- Sample-Facebook-Share
- Sample-News-API
- Sample-Stocks-Embedded
- Sample-To-Do
- Sample-Text-With-Image
- Sample-Weather-Embedded
- Sample-Folder-List
- Sample-File-List
• Sample-Documents-Manager
• Sample-Process-Start-Form
• Sample-Process-Task-List
• Sample-Process-Task-Details
• SimpleHTML

Example:

```
cec create-component MyLocalComponent1 -f local
```

The component is created in your Git repository under `cec-components/src/main/components`.

The `src/main/components` directory is seeded with the Sample-To-Do component. All components that you create go in this directory.

Copy a Component

You can copy a component in Developer Cloud Service with the `cec copy-component` command:

```
cec copy-component <source> [destination]
```

This command copies an existing component named `<source>` to `<destination>`.

Export or Deploy the Component to Content and Experience Cloud

Once the component is developed and tested in the local server, you can export the component using the following command. This creates the component ZIP file. You can manually import this component ZIP into Content and Experience Cloud.

```
cec export-component <component name>
```

Alternatively, you can deploy the component directly to Content and Experience Cloud from Developer Cloud Service using the following command.

```
cec deploy <component name>
```

The `cec deployAll` command will deploy all the components in `src/main/components`.

Import Components into Developer Cloud Service

If you have a component ZIP file created from the Content and Experience Cloud server, you can import that into Developer Cloud Service for further development. Use the following command:

```
cec import-component <location of the component zip file>
```
Write and Run Unit Tests

Start with the sample unit test to write unit tests and run them for your custom Content and Experience Cloud components.

Start with the Sample Unit Test

The Developer Cloud Service samples for Content and Experience Cloud include the following files to help you write unit tests:

- **src/test/unit**: Contains the unit test for the Sample-To-Do component. This serves as a sample for writing unit tests for component JavaScript code, which includes RequireJS modules. The Mocha and Chai frameworks for JavaScript unit testing are used.

- **index.html**: Runs the unit test. It loads Mocha, Chai, and the main test file, test-main.js.

- **test-main.js**: Loads the unit test module, Sample-To-Do-Test, and runs the Mocha unit tests.

- **Sample-To-Do-Test.js**:
  - Defines the actual tests. It loads the component source code, components/Sample-To-Do/assets/render.js, using RequireJS.
  - Tests the `add()`, `delete()`, `title()`, and `placeholder()` methods.

Writing Unit Tests for Your Components

To write each of your unit tests for a custom component, follow these steps:

1. Write a unit test similar to Sample-To-Do-Test.js.
2. Load the test you wrote into test-main.js.

Running a Unit Test Locally

To run your unit tests locally, follow these steps:

1. Clone the Git repository locally.
2. In a terminal window on your local machine, enter npm install.
3. Enter npm start.
4. Either enter npm test or open http://localhost:8085/unit/ in a browser.
Optimize Components (Minify) for Better Performance

Improve the performance of your components and reduce download sizes by minifying the JavaScript code with Developer Cloud Service.

Optimize Your Components

An example of optimizing component source code (optional) is provided for the Sample-Text-With-Image component. You can use the RequireJS optimizer to minify and combine source code. Minifying a component removes unwanted space in the JavaScript code, resulting in better performance and reduced download size. Optimizing involves the following actions:

- Minify the JavaScript.
- Compress the CSS.
- Combine the JavaScript, HTML, and CSS into a single file, render.js.

This optimization reduces download size and improves performance.

Enable Optimization for Your Component

In the Sample-Text-With-Image component, optimization will minify and combine all the files that are marked as dependencies in render.js (template.html, data-defaults.js, and design.css) into render.js as a single file. Other components that already support optimization follow:

- Sample-Folder-List
- Sample-File-List
- Sample-Documents-Manager
- Sample-Process-Start-Form
- Sample-Process-Task-List
- Sample-Process-Task-Details

To enable optimization for your local component, take the following steps (required):

1. Copy gulpfile.js from the Sample-Text-With-Image component to your component.
2. Fix the module names in gulpfile.js for your render.js dependencies. Refer to the comments in gulpfile.js.

Note:

If you have an existing component, with a css folder under assets, rename the css folder to styles to avoid errors during optimization.

Run Continuous Integration Jobs

You can run continuous integration jobs to keep your site up to date.
Refer to the “Getting Started with CEC Custom Components Development” wiki in Developer Cloud Service. You can configure a build job for the continuous integration.

Develop Translatable Components for Multilingual Sites

Developers of custom components can designate which strings within a custom component should take part in page translations for multilingual (MLS) sites.

To do this, you need to introduce an nls property at the top level when saving your data to the customSettingsData object.

For example:

```javascript
SitesSDK.setProperty('customSettingsData', {
    'nls': {
        'linkText': 'More...' 
    }
});
```

When a translation job is created, Oracle Content and Experience Cloud (OCE) will check the top-level properties of the customSettingsData object and export the entire nls object for each custom component instance on the page. Translators will translate these values, which can then be imported back into the site.

Once the site translations have been imported, the correct version of the nls object will be returned in the customSettingsData object for the translated locale.

For example, if you translated the site to French and then render the page in the French locale, the value of the nls object in the customSettingsData object passed to the custom component would be updated to:

```javascript
{
    'nls': {
        'linkText': 'Plus...' 
    }
},
```

The format of the nls object in customSettingsData should be limited to name/value pairs. This aids in translation and ensures that translated values can be applied correctly to the base values in the site when the page is rendered in a translated locale.

Build an H1 Component with a Settings Panel

You can create a minimal Oracle Content and Experience Cloud component that has a simple HTML template and CSS. This H1 component has a simple settings panel and an entry for the theme in design.json to allow other Oracle Content and Experience Cloud users to pick from three built-in styles when using the component in an editor.

When you create the new component, you get a set of seeded files that will work out of the box. The seeded files cover most of the functionality of a component within the product. You can change the seeded code to create your own component, which requires only a small subset of seeded code to achieve the end result.
You can build an H1 component with a settings panel in five steps:

1. **Create a New Local Component**
2. **Build the Basic H1 Component**
3. **Add CSS for Your Component**
4. **Add a Settings Panel to Change Heading Text**
5. **Update the Theme for Others to Pick the H1 Component Style**

### Create a New Local Component

Create a local component with Oracle Content and Experience Cloud that you can immediately drop onto the page. This is the starting point for creating any new component.

To create a new local component:

1. Navigate to Sites > Components.
2. Select Create > Create Local Component.
3. Enter a name, for example, BasicTextEditor, and optionally, a description.
4. Click Create to create a new component.

![Create Local Component](image)

Now that you have successfully created a component, you should see it in the Component Catalog as well as in the Add > Custom component palette for any site you create. Use the following steps to validate your component creation (Checkpoint 1):

1. Create a new site using any seeded template; for example, create a site named ComponentTest using the StarterTemplate template.
2. Select the Edit option, and create an update for the site to open it in an editor.
3. Edit a page within the site you created.
4. Click the Add button (+) on the left bar, and select Custom for the list of custom components.
5. Select **H1_Component** from the custom component palette and drop it onto the page.
   You should now see a default rendering for the local component you created.

![Oracle Sites Custom Component Palette](image)

6. Select the context menu for the component.
7. Choose **Settings** from the drop-down menu.
   You can change the settings to see how seeded component rendering will change.
   You can modify seeded files to create a new custom component.

### Build the Basic H1 Component

You can remove most of the contents in seeded files to create an H1 component. It displays the heading text that you seed when you create `viewModel`. Later you can provide settings and styles for the component.

To review the structure of your local component:

1. Using the Oracle Content and Experience Cloud Desktop Sync App, locate your component and sync it with the file system.
   - In a recent version of the Desktop Sync App, choose the **Start Sync** or **Select Folders to Sync** option.
   - If you don't have the Desktop Sync App, you can select the component on the **Components** tab of Oracle Content and Experience Cloud and then drill down to see the files.

2. If you list the files under the component, you can see these files:
   - The component files in the **assets** folder:
     - `render.js`
     - `settings.html`
   - `appinfo.json`: JSON file with the component description.
     See [About Developing Components](#).
   - `folder_icon.jpg`: Icon that is displayed in the Component Catalog.

To build an H1 Component:
1. Open the `appinfo.json` file and replace its contents with the following lines:

```
{
  "id": "h1-component-id",
  "settingsData": {
    "settingsHeight": 90,
    "settingsWidth": 300,
    "settingsRenderOption": "inline",
    "componentLayouts": [],
    "triggers": [],
    "actions": []
  },
  "initialData": {
    "componentId": "h1-component-id",
    "customSettingsData": {
      "headingText": "Heading 1"
    },
    "nestedComponents": []
  }
}
```

2. Open the `render.js` file in the `assets` folder in your favorite text editor.

3. Change the contents of `render.js` to the following lines:

```
/* globals define */
define(['knockout', 'jquery', 'text!./render.html'], function(ko, $, template) {
  'use strict';
  // Define a Knockout ViewModel for your template
  var SampleComponentViewModel = function(args) {
    var SitesSDK = args.SitesSDK;
    // create the observables -- this allows updated settings to automatically update the HTML on the page
    this.headingText = ko.observable();

    // Handle property changes from the Settings panel
    this.updateCustomSettingsData = $.proxy(function(customData) {
      this.headingText(customData && customData.headingText);
    }, this);
    this.updateSettings = function(settings) {
      if (settings.property === 'customSettingsData') {
        this.updateCustomSettingsData(settings.value);
      }
    };
    // Register your updateSettings listener to receive SETTINGS_UPDATED events
    SitesSDK.subscribe(SitesSDK.MESSAGE_TYPES.SETTINGS_UPDATED, $.proxy(this.updateSettings, this));
});
```
// Get the initial settings data for the component and apply it
//
SitesSDK.getProperty('customSettingsData',
this.updateCustomSettingsData);
);

// ----------------------------------------------
// Create a knockout based component implementation
// ----------------------------------------------
var SampleComponentImpl = function(args) {
    // Initialize the custom component
    this.init(args);
};

// initialize all the values within the component from the given argument values
SampleComponentImpl.prototype.init = function(args) {
    // create the viewModel
    this.createViewModel(args);
    this.createTemplate(args);
    this.setupCallbacks();
};

// create the viewModel from the initial values
SampleComponentImpl.prototype.createViewModel = function(args) {
    // create the viewModel
    this.viewModel = new SampleComponentViewModel(args);
};

// create the template based on the initial values
SampleComponentImpl.prototype.createTemplate = function(args) {
    // create a unique ID for the div to add, this will be passed to the callback
    this.contentId = args.id + '_content_' + args.viewMode;
    // create a hidden custom component template that can be added to the DOM
    this.template = '<div id="' + this.contentId + '"><' +
        template +
    '</div>';
};

// SDK Callbacks
// setup the callbacks expected by the SDK API
//
SampleComponentImpl.prototype.setupCallbacks = function() {
    // callback - render: add the component into the page
    this.render = $.proxy(function(container) {
        var $container = $(container);
        // add the custom component template to the DOM
        $container.append(this.template);
        // apply the bindings
        ko.applyBindings(this.viewModel, $('#' + this.contentId)[0]);
    }, this);
};
// callback — dispose: cleanup after component when it is removed from the page
    //
    this.dispose = $.proxy(function() {
        // nothing required for this sample since knockout disposal will automatically clean up the node
        }, this);
    });
// ----------------------------------------------
// Create the factory object for your component
// ----------------------------------------------
var sampleComponentFactory = {
    createComponent: function(args, callback) {
        // return a new instance of the component
        return callback(new SampleComponentImpl(args));
    }
};
return sampleComponentFactory;

4. In the assets folder, create a new file, render.html, to be the simple HTML template of the component.

5. Use the following contents in the render.html file:

   <h1 data-bind="text: headingText()"></h1>

The component assets folder now contains three files.

Add the new H1 component to your page (Checkpoint 2).
Add CSS for Your Component

You can add a CSS that will provide a default style for your component.

To add a CSS:

1. Add a `design.css` file to the `assets` folder of your component, with the following contents:

   ```css
   .h1-component-default-style .scs-component-content {
       font-family: "Helvetica Neue", "Helvetica", "Arial", sans-serif;
       font-size: 24px;
       color:red;
       font-weight: normal;
   }
   ```

2. Add to `appinfo.json` to declare the style class prefix that will be used to style your component. If a `styleClassName` of `h1-component` is added, when your component is dropped onto the page, the default style will be `h1-component-default-style`. The new contents of `appinfo.json` follows:

   ```json
   {
       "id": "h1-component-id",
       "settingsData": {
           "settingsHeight": 90,
           "settingsWidth": 300,
           "settingsRenderOption": "inline",
           "componentLayouts": [ ],
           "triggers": [ ],
           "actions": [ ]
       },
       "initialData": {
           "componentId": "h1-component-id",
           "styleClassName":"h1-component",
           "customSettingsData": {
               "headingText": "Heading 1"
           },
           "nestedComponents": [ ]
       }
   }
   ```
3. **Update** `render.js` to load your CSS file by replacing line line 2

```javascript
define(['knockout', 'jquery', 'text!./render.html'], function(ko, $, template) {})
```

with the following line.

```javascript
define(['knockout', 'jquery', 'text!./render.html', 'css!./design.css'], function(ko, $, template, css) {
```

**Note:**

Because you changed the `appinfo.json` file, you must refresh the browser and add your component again to a page to pick up the changes.

Verify that your component will now pick up its default data from the `appinfo.json` file and look like this (Checkpoint 3):

![Image of component with heading 1](image)

---

**Add a Settings Panel to Change Heading Text**

**Update** the `settings.html` file to provide a settings panel that can be used to set the text of the H1 component.

To add a settings panel to change heading text:

1. **Update the** `settings.html` **file to have the following contents:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- only allow embedding of this iFrame in SCS -->
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>H1 Component</title>
    <!-- include sample apps styling -->
    <link href="/_sitescloud/renderer/app/sdk/css/app-styles.css" rel="stylesheet">
    <!-- include supporting files -->
    <script type="text/javascript" src="/_sitescloud/renderer/app/apps/js/knockout.min.js"></script>
    <script type="text/javascript" src="/_sitescloud/renderer/app/apps/js/jquery.min.js"></script>
```

---
2. Select your component in Site Builder, and click Settings.
The settings panel is small, and is set to be embedded at the top of the component settings panel. Look for the section titled “Heading Text”.

To verify that you can change heading text (Checkpoint 4):

When you change heading text in the settings panel, the component updates to show the new text.

Update the Theme for Others to Pick the H1 Component Style

You can register styles for your component with the theme, so that other users can switch between the styles you provide for your component from the settings panel or Style tab.

To update the theme for other users to pick the component style:

1. Add some more styles to your component design.css file. Prefix each style with the component's registered styleClassName as defined in appinfo.json. For this component, that prefix is h1-component.

Two more styles, h1-component-gothic-style and h1-component-courier-style have been added.

The new contents of design.css will now be as follows:

```css
.h1-component-default-style .scs-component-content {  
  font-family: "Helvetica Neue", "Helvetica", "Arial", sans-serif;  
  font-size: 24px;  
  color:red;  
  font-weight: normal;  
}  
.h1-component-gothic-style .scs-component-content {  
  font-family: "Century Gothic","CenturyGothic","AppleGothic",sans-serif;  
  font-size: 32px;  
  font-weight: bold;  
}  
.h1-component-courier-style .scs-component-content {  
  font-family: "Courier";  
  font-size: 32px;  
  font-weight: bold;  
}  
```

2. Register your styles in the theme's design.json file. You can find this file in the theme the site is using. Drill down to the theme files in the designs folder, then to the defaults folder, and add a section for your component to design.json.
The bold text that follows is an example of what to add:

```json
"news-article": {
"styles": [
{name: "News Article 1",
"class": "news-article-default-style"},
{name: "News Article 2",
"class": "news-article-style-1"}
]
},
"h1-component": {
"styles": [
{name: "Plain",
"class": "h1-component-default-style"},
{name: "Courier",
"class": "h1-component-courier-style"},
{name: "Gothic",
"class": "h1-component-gothic-style"}
]
}
},
"componentIcons": {
"scs-socialbar": {
"icons": [
" Welcome to my Site

The names in the design.json snippet that was added ("Plain", "Courier", "Gothic") will appear in the settings panel for your component on the style tab as below. When selected, they will apply the corresponding styles ("h1-component-default-style", "h1-component-courier-style", "h1-component-gothic-style") respectively, on your component.
Create a Simple Component Using HTML Templates

You can create an Oracle Content and Experience Cloud component that uses HTML and CSS with Mustache JS templates to render its content based on data you enter on a settings panel.

The settings panel that you provide can allow editing of the data that will be rendered in the HTML template.

The examples for this procedure use Mustache JS templates.

You can create a simple component using HTML templates in two steps:

1. Create the HTML Component
2. Add User-Supplied Data to the HTML Component

Create the HTML Component

You can create a new local component in the Oracle Content and Experience Cloud Component Catalog and then build the HTML component by adding and changing files in its assets folder.

To create and build an HTML component:

1. Create a component and name it HTML Component.
   See Create a New Local Component.
2. Add the `mustache.min.js` file to the `assets` folder in HTML Component.
3. Create a new file in the `assets` folder named `render.html`, which contains HTML for the body of your component. For this example, create the `render.html` file with the following contents:

```html
<ul class="wrapper">
  <li class="box">
    <h1 class="title">One</h1>
    <p class="text">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
  </li>
  <li class="box">
    <h1 class="title">Two</h1>
    <p class="text">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
  </li>
  <li class="box">
    <h1 class="title">Three</h1>
    <p class="text">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
  </li>
  <li class="box">
    <h1 class="title">Four</h1>
    <p class="text">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
  </li>
</ul>
```
4. Create a new file in the assets folder named design.css, which is the CSS for your component. For this example, add the following lines as the contents of the design.css file:

```css
.wrapper {
    text-align: center;
}
.box {
    display: inline-block;
    position: relative;
    width: 200px;
    height: 200px;
    padding: 0px 10px 10px 10px;
    background: transparent;
    border-width: 1px;
    border-style: solid;
    border-radius: 5px;
    border-color: #CCCCCC;
    z-index: 0;
    margin: 2px 2px 2px 2px;
    transition: all .15s ease-in-out;
}
.box:hover {
    background: #9CC;
    z-index: 100;
    transform: scale(1.2,1.2);
    box-shadow: 0 5px 10px 0 rgba(0,0,0,.2);
}
.title {
    color: red;
}
.text {
    color: #555555;
}
```

5. Open up the render.js file in the assets folder and change the content to the following. No matter what HTML and CSS you used in the preceding steps, the following render.js file will render your HTML and CSS into the page for you:

```javascript
/* globals define */
define(['jquery', './mustache.min', 'text!./render.html', 'css!./design.css'], function($, Mustache, template, css) {
    'use strict';

    // ----------------------------------------------
    // Create a Mustache-based component implementation
    // ----------------------------------------------
    var SampleComponentImpl = function(args) {
        this.SitesSDK = args.SitesSDK;
```

---

**elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.**
Chapter 12

Create a Simple Component Using HTML Templates

// Initialze the custom component
this.createTemplate(args);
this.setupCallbacks();
};
// create the template based on the initial values
SampleComponentImpl.prototype.createTemplate = function(args) {
// create a unique ID for the div to add, this will be passed to
the callback
this.contentId = args.id + '_content_' + args.viewMode;
// create a hidden custom component template that can be added to
the DOM
this.template = '<div id="' + this.contentid + '">' +
template +
'</div>';
};
SampleComponentImpl.prototype.updateSettings = function(settings) {
if (settings.property === 'customSettingsData') {
this.update(settings.value);
}
};
SampleComponentImpl.prototype.update = function(data) {
this.data = data;
this.container.html(Mustache.to_html(this.template, this.data));
};
//
// SDK Callbacks
// setup the callbacks expected by the SDK API
//
SampleComponentImpl.prototype.setupCallbacks = function() {
//
// callback - render: add the component into the page
//
this.render = $.proxy(function(container) {
this.container = $(container);
this.SitesSDK.getProperty('customSettingsData', $.proxy(this.update,
this));
}, this);
//
// callback - SETTINGS_UPDATED: retrive new custom data and rerender the component
//
this.SitesSDK.subscribe(this.SitesSDK.MESSAGE_TYPES.SETTINGS_UPDATED, $.
proxy(this.updateSettings, this));
//
// callback - dispose: cleanup after component when it is removed
from the page
//
this.dispose = $.proxy(function() {
// nothing required
}, this);
};
// ---------------------------------------------// Create the factory object for your component

12-24


```javascript
// ----------------------------------------------
var sampleComponentFactory = {
    createComponent: function(args, callback) {
        // return a new instance of the component
        return callback(new SampleComponentImpl(args));
    }
};
return sampleComponentFactory;
```

To verify that HTML Component has been built (Checkpoint 1):

1. Check that the `assets` folder in your component has the following five files.

   ![Assets Folder](image)

2. Add the new HTML Component to a page in your test site. In an editor, you should see the component on the page as follows, in Edit and Preview modes.

   **Edit mode**
   ![Edit Mode](image)

   **Preview mode**
   ![Preview Mode](image)
Add User-Supplied Data to the HTML Component

You can add a Custom Settings panel that provides fields for all data elements you define in your HTML Component. A user can then fill in values for the component instance on the page in the Custom Settings panel.

To add user-supplied data to HTML Component:

1. Update the HTML template in your render.html file to include any data elements you want. The following example uses Mustache JS templating language, so the data elements must be added using {{ and }} syntax, as follows:

   ```html
   <ul class="wrapper">
   <li class="box">
      <h1 class="title">{{title1}}</h1>
      <p class="text">{{text1}}</p>
   </li>
   <li class="box">
      <h1 class="title">{{title2}}</h1>
      <p class="text">{{text2}}</p>
   </li>
   <li class="box">
      <h1 class="title">{{title3}}</h1>
      <p class="text">{{text3}}</p>
   </li>
   <li class="box">
      <h1 class="title">{{title4}}</h1>
      <p class="text">{{text4}}</p>
   </li>
   </ul>
   ```

2. Provide default values for the following data fields by changing the content of the appinfo.json file in your HTML Component:

   ```json
   {
      "id": "html-component-id",
      "settingsData": {
         "settingsHeight": 600,
         "settingsWidth": 300,
         "settingsRenderOption": "dialog",
         "componentLayouts": [],
         "triggers": []
   }"
3. **Add a Custom Settings** panel that will look for the data values in your HTML template `render.html` file to create the settings for a user to change in an editor. Change the contents of the `settings.html` file to the following lines:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- only allow embedding of this iFrame in SCS -->
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>H1 Mustache Component</title>

  <!-- include sample apps styling -->
  <link href="/_sitescloud/renderer/app/sdk/css/app-styles.css" rel="stylesheet">

  <!-- include supporting files -->
  <script type="text/javascript" src="/_sitescloud/renderer/app/apps/js/knockout.min.js"></script>
  <script type="text/javascript" src="/_sitescloud/renderer/app/apps/js/jquery.min.js"></script>

  <!-- include the Sites SDK -->
  <script type="text/javascript" src="/_sitescloud/renderer/app/sdk/js/sites.min.js"></script>
</head>
<body data-bind="visible: true" style="display:none; margin:0px; padding:0px; background:transparent; background-image:none;">
  <!-- ko if: initialized() -->
  <div class="scs-component-settings">
  </div>

  <!-- Width -->
```
Create a Simple Component Using HTML Templates

Chapter 12

<html>
<body>

<label id="headingTextLabel" for="headingText" class="settings-heading" data-bind="text: 'Heading Text'"></label>
<input id="headingText" data-bind="value: headingText" placeholder="Heading" class="settings-text-box">
</div>
</div>
</div>
<!-- ko ifnot: initialized() -->
<div data-bind="text: 'waiting for initialization to complete'"></div>
<!-- /ko -->
<script type="text/javascript">
ko.bindingHandlers.scsCompComponentImpl = {
  init: function (element, valueAccessor, allBindings, viewModel, bindingContext) {
    var body = document.body,
    html = document.documentElement;
    SitesSDK.setHeight(Math.max(
      body.scrollHeight,
      body.offsetHeight,
      html.clientHeight,
      html.scrollHeight,
      html.offsetHeight));
  }
};
// define the viewModel object
var SettingsViewModel = function () {
  var self = this;

  // create the observables for passing data
  self.headingText = ko.observable('Heading 1');

  // create rest of viewModel
  self.initialized = ko.observable(false);
  self.saveData = false;

  // Get custom settings
  SitesSDK.getProperty('customSettingsData', function (data) {
    // update observable
    self.headingText(data.headingText);
    // note that viewModel is initialized
    self.initialized(true);
    self.saveData = true;
  });

</script>
</body>
</html>
// save whenever any updates occur
self.save = ko.computed(function () {
    var saveconfig = {
        'headingText':
            self.headingText()
    };

    // save data in page
    if (self.saveData) {
        SitesSDK.setProperty('customSettingsData', saveconfig);
    }
}, self);

// apply the bindings
ko.applyBindings(new SettingsViewModel());

</script>
</body>

---

**Note:**

Because you changed the appinfo.json file, you must refresh the browser and add your component again to a page to pick up the changes.

To verify that your component will now pick up its default data from the appinfo.json file and that you can change the values in the **Custom Settings** panel (Checkpoint 2):

1. After you change the appinfo.json file, remember to refresh the browser and add your component again to a page to pick up the changes.

2. Open the **Custom Settings** panel in HTML Component and enter data.
3. Verify that your Oracle Content and Experience Cloud component uses HTML and CSS with Mustache JS templates to render its content based on the data that you entered on the **Custom Settings** panel.

### Compare Local Components to Remote Components

Local components and remote components are implemented differently.

The following table lists differences in how components can be implemented. It may help you determine whether you want to use a local component or a remote component rendered in an inline frame.

<table>
<thead>
<tr>
<th>Local Component Implementation</th>
<th>Remote Component Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrates complex content-centric user interface into Oracle Content and Experience Cloud</td>
<td>Integrates application logic into Oracle Content and Experience Cloud</td>
</tr>
<tr>
<td>Executes JavaScript to render content in the page</td>
<td>Uses the <code>&lt;iframe&gt;</code> HTML tag to render content in the page</td>
</tr>
<tr>
<td>Uses any JavaScript technology stack</td>
<td>Can use any technology, not just JavaScript</td>
</tr>
<tr>
<td>Leverages the Oracle Content and Experience Cloud JavaScript stack and can re-use Oracle Content and Experience Cloud components</td>
<td>Doesn’t integrate with the Oracle Content and Experience Cloud JavaScript stack</td>
</tr>
<tr>
<td>Adds assets dependencies from <code>/assets</code> documents stored in the Oracle Content and Experience Cloud server</td>
<td>Requires HTTPS URLs and corresponding certificates for all asset access</td>
</tr>
<tr>
<td>Is hosted with a template and published independently of a theme by Oracle Content and Experience Cloud</td>
<td>Requires a hosted middle-tier server for the URL endpoints</td>
</tr>
</tbody>
</table>
Local Component Implementation | Remote Component Implementation
--- | ---
Because code executes within the page, if the component code breaks, it may break the page | The page renders independently to the content of inline frames, so the page won’t break if the component fails
May slow down the rendering of the page if the component is slow to render | The page loads independently to inline frame content, so the page will load as fast as it can and then load the content of any inline frame

**Note:**

While the rendering of local components and remote components differ, the Settings panel implementation in Site Builder is the same. The Settings panel for both types of components is rendered using inline frames, and it uses the same JavaScript SDK to allow for both cross-boundary and cross-domain communication.

## Render Component Settings

To render component settings, you can use a component settings URL and component settings rendering options.

### Component Settings URL

A component settings URL is rendered using an inline frame and called with parameters to allow specific settings of an actual component dropped onto a page. The settings URL has this format:

```
{Component Settings URL}?instance=<app-instance>&width=<width>&currCompId=<id of the app associated with the settings panel>&locale=<locale>
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Settings URL</td>
<td>URL</td>
<td>Component Settings URL of a component</td>
</tr>
<tr>
<td>width</td>
<td>Number</td>
<td>Width of the Settings inline frame in pixels</td>
</tr>
<tr>
<td>currCompId</td>
<td>String</td>
<td>Current component ID of the component edited by the Settings panel</td>
</tr>
<tr>
<td>locale</td>
<td>String</td>
<td>Current locale of the host site (Site Builder). Format is (&lt;language&gt;_&lt;dialect&gt;). Example: En_us.</td>
</tr>
</tbody>
</table>
Local Component Settings Rendering Options

You have three options for rendering the inline frame in the Settings panel for a local component, based on the size and complexity of the inline frame. Each option is specified in the `settingsRenderOption` property.

**Note:**

These options are available only to local components. Remote components Settings panels always render in a dialog.

- **inline**
  - Use this option only if you have a few small properties for the user to enter.
  - The inline frame will be inserted onto the **General** tab in place of the usual button to navigate to the Settings panel.
  - Provides the most integrated solution, requiring the least amount of clicks for a user, but it has limited space.

- **panel**
  - Use this option when you have a longer list of properties, but they can still be reasonably displayed within the 300 pixels of the standard Settings panel.
  - The inline frame will slide into view and a Back button will appear to return the user to the General tab.
  - Provides an integrated solution where you can interact with the inline frame on the page.

- **dialog**
  - The default mode for handling more general settings layouts requiring a complex user interface.
  - Displays the inline frame in a modal dialog on the page.

Local Component Implementation

The component instance is created by the Component Factory object.

The Component Factory object entry in the component registration file is `initData.componentFactory`. This is a reference to a JavaScript file that RequireJS uses to load the component factory. It must return a JavaScript object that implements the `componentFactory` interface.

The Component Factory object must implement the API `customComponentFactory.createComponent(args)`, which creates each instance of the component.

`args` contains these settings:

- **SitesSDK**: The Oracle Content and Experience Sites SDK.
- **id**: The unique ID (GUID) for the component added to the page.
• **viewMode:** The current mode the page is rendering. When a page is being edited, it’s "Edit". When a page is previewed, it’s "Navigate". At runtime, which is when the site is published, the value is undefined. You can provide different implementations based on what functionality should be exposed for each mode. For example, links shouldn’t be active when the page is running in Edit mode.

**Note:**

It isn’t a requirement to use JQuery or Knockout for your component, but if you want to leverage Oracle Content and Experience Cloud features such as nested components, you must use the version of Knockout provided by Oracle Content and Experience Cloud. This version of Knockout has extended component registration and handlers, which wouldn’t be available to you otherwise.

For the component itself, the SDK is passed in when the component is instantiated so that the component can communicate with the page lifecycle. The page lifecycle functions must be implemented by the component and are called by Oracle Content and Experience Cloud to render the component on the page.

Mandatory and optional APIs are provided to implement a component.

**Mandatory APIs**

`customComponent.render(container)`: Asks the component to insert itself into the provided DOM container element.

• `container`: DOM container element for the custom component HTML.

**Optional APIs**

`customComponent.dispose()`: Called when the component is being removed from the page. Provides an opportunity for the component to remove any resources that are no longer required.

**Style Classes for Components**

You can create a defined list of styles that can be applied to your component by users.

Having a predefined list of styles for your component follows the same model as defining style classes for components provided by Oracle Content and Experience Cloud through a theme’s design files. You name your custom style classes in the `appinfo.json` file for the component.

You define additional styles in the `design.css` and `design.json` files. The `json` file provides a mapping from the name that will appear in the user interface to the actual underlying `css` class name, and the `css` file provides the details for each style class.

The `design.json` file has the following structure for components:

```json
{
    "componentStyles": {
        "scs-image": {
```
Each of the "styles":[] entries can contain a list of styles for that particular component. The "name" can either be a reference to a built-in localized string, or a specified value to use. For example, the Title component provides these default styles:

```json
{
    "styles": [{
        "name": "COMP_STYLE_FLAT",
        "class": "scs-title-default-style"
    },
    {
        "name": "COMP_STYLE_HIGHLIGHT",
        "class": "scs-title-style-2"
    },
    {
        "name": "COMP_STYLE_DIVIDER",
        "class": "scs-title-style-3"
    }
}
```
The name values are mapped to the actual words to display in the user interface, like this:

"COMP_STYLE_FLAT": "Flat",
"COMP_STYLE_HIGHLIGHT": "Highlight",
"COMP_STYLE_DIVIDER": "Divider",

The css file provides the definitions for the class values:

.scs-title-default-style {
    color: #333333;
    display: block;
    font-family: "Helvetica Neue", "Helvetica", "Arial", sans-serif;
    font-size: 24px;
    font-weight: normal;
}

As an example, in the theme’s design.json file, you can add entries for your component based on the initialData.componentId value you define in the components.json file:

"componentId": "news-article"

The corresponding entries in the design.json file would be these:

"componentStyles": {
    "news-article": [{
        "styles": [{
            "name": "News Article 1",
            "class": "news-article-default-style"
        },
        {
            "name": "News Article 2",
            "class": "news-article-style-1"
        }
    ]
},
"componentId": "news-article"

The corresponding entries in the design.css file would be these:

.news-article-default-style .scs-image {...}
.news-article-style-1 .scs-image {...}

How to Style Built-In Components

Create your own look and feel to style built-in components in Oracle Content and Experience Cloud by overriding and extending built-in styles.
Built-in components get their visual styling from two places:

- `comp.css`, a built-in CSS file that specifies the base look of each component
- `design.css`, a CSS file that is part of the theme that your site is using

In the `design.css` file, you can override and extend the built-in `comp.css` styles to create your own look and feel. In a theme, the `design.css` file is in the `designs/default` directory.

The following topics describe classes in the `comp.css` file are common to all built-in components and provide an overview of defining a theme:

- Component Styling Basics
- Component-Specific Styling
- Set Component Properties

## Component Styling Basics

All built-in Sites components share a similar CSS class structure.

Each component has the following three CSS classes applied to its outermost `<div>` element:

```
scss-component scs-type design-style
```

The `type` is the component type (such as `image`, `gallery`, or `divider`). The `design-style` is the chosen style class for a component, as defined in the theme's file.

## Theme Styles Basics

A theme's `design.json` file lists all the styles (frame, shadow, highlighted, and so on) that can be applied to each type of component (such as `button` or `image`). Each style has both a display name and a class name. The display name is shown on the `Style` tab of the `Settings` panel. The class name refers to a CSS selector in the theme's `design.css` file. For example, the entry for the `button` component follows:

```
"scs-button": {
  "styles": [{
    "name": "COMP_STYLE_ALTA_SMALL",
    "class": "scs-button-default-style"
  },
  {
    "name": "COMP_STYLE_ALTA_LARGE",
    "class": "scs-button-style-2"
  },
  {
    "name": "COMP_STYLE_SIMPLE",
    "class": "scs-button-style-3"
  }
}
```

Names of built-in components are translated, so you see a key to get style name from the resource bundle. If you add a button component to a page and then choose the Simple style in the Settings > Style panel, the design.json file associates the display name Simple (key COMP_STYLE_SIMPLE) with the class name scs-button-style-3. The button will be rendered with the following classes:

```
scs-component scs-button scs-button-style-3
```

If no style is chosen for a given component, then the default style, scs-type-default-style, is used. In the preceding example, the button will be rendered with the following classes:

```
scs-component scs-button scs-button-default-style
```

The scs-component-content Style

For every built-in component, inside the scs-component <div> mentioned previously, there is a content <div> with the CSS class scs-component-content. In other words:

```
scs-component scs-type design-style
scs-component-content
```

In the design.css file, the scs-component-content class is often used to style the "box" around the component (for example, to apply a border or shadow).

It's worth noting that in the built-in comp.css file, the common scs-component-content class is defined with position:relative and display:inline-block, among other CSS properties.

While scs-component-content is useful for styling the "box" around each component, component-specific classes are needed to fully style a component. See Component-Specific Styling.

Component-Specific Styling

You can apply specific styles to images, buttons, documents, paragraphs, titles, maps, and other components.

Image Component

The Image component has the following CSS class structure below the scs-component-content class:

```
scs-image-container
scs-image-link
scs-image-image
scs-image-caption
```

The scs-image-image class is applied to the <img> tag itself. The scs-image-caption class is used to style the caption, if the caption is present.

The scs-image-link class is present only if the image has a link attached. Neither it nor the scs-image-container class typically requires custom styling.
By default, the image caption is rendered as a semitransparent overlay stretched across bottom of the image.

To place the captions at the top of the image and change colors, add extra style for the Image component in the design.json file and then define CSS for it in the design.css file.
The Button component has the following class structure:

```
scs-button-button
  scs-button-text
```

The `scs-button-button` class is the clickable `<div>`, styled to look like a button. The `scs-button-text` class is used to style the text inside the button.

For example, test changing the look and feel of the Button component by adding extra style for it in the `design.json` file, and then define CSS for it in the `design.css` file.
Chapter 12
How to Style Built-In Components

```
.design-style .scs-button-button {
  background-image: linear-gradient(
    to top, #E3E7E9 0%, #E7EBED 50%, #F1F3F3 100%);
  border: 1px solid #c4ced7;
  color: #000000;
}

.design-style .scs-button-button:hover {
  background: #f7f8f9;
  border: 1px solid #c4ced7;
  color: #0572ce;
}

.design-style .scs-button-button:active {
  background: #0572ce;
  border: 1px solid #0572ce;
  color: #ffffff;
}

.scs-button-style-4 .scs-button-button {
  background-image: radial-gradient(
    red, yellow, green
  );
  border: 1px solid #c4ced7;
  color: #000000;
}

.scs-button-style-4 .scs-button-button:hover {
  background: #f7f8f9;
  border: 1px solid #c4ced7;
  color: #0572ce;
}

.scs-button-style-4 .scs-button-button:active {
  background: #0572ce;
  border: 1px solid #0572ce;
  color: #ffffff;
}
```
Document
The Document component has the following class structure:

- scs-document-container
- scs-document-cap
- scs-document-title
- scs-document-desc

The `scs-document-container` class wraps the document viewer, and is not normally styled.

Gallery
The Gallery component has a single class wrapping the underlying JSSOR slider component:

- scs-gallery-container

The JSSOR slider uses several classes that you can style as well:

- jssorb14 (navigator)
- jssora02l (left arrow)
- jssora02r (right arrow)
- jssort07 (thumbnails)

Gallery Grid
The classes used for the Gallery Grid component depend on the layout and cropping selected in the Settings panel:

- scs-gallerygrid-container
- scs-gallerygrid-layout
- scs-gallerygrid-cell
  - scs-image (multiple)

Depending on the cropping and layout settings selected for Gallery Grid, the value of `layout` will be stretch, crop, fit, or flowing.

The `scs-gallerygrid-cell` class is present only for Column layouts.

Social Bar
The Social Bar component has the following class structure:

- scs-socialbar-container
  - scs-socialbar-icon

The `scs-socialbar-icon` class is applied to each `<img>` tag in the social bar.
Paragraph

The Paragraph component has only a single class wrapping the actual paragraph text:

```
.scs-paragraph-text
```

For example, to make text that you contribute in the Paragraph component to have an engraved-text-on-metal effect often used by Apply, add an extra style class in the `design.json` file and then define CSS for it in the `design.css` file.

```
.scs-paragraph-style-7 {
  font-size: 24px;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: 700;
  padding: .3em;
  color: #000000;
  background: #666666;
  text-shadow: 0px 1px 1px #ffffff;
}
```

Or if you want to get fancy, use something like the next example.

```
.scs-paragraph-style-8 {
  padding: 20px;
  margin: 10px;
  background: #ff0030;
  color: #fff;
  font-size: 21px;
}
```
Title
The Title component also has only a single class wrapping the actual text:

```
scs-title-text
```

Map
The Map component has a single class wrapping the map rendition:

```
scs-map-content
```

This class is not normally styled.

Custom Local Component
The Custom Local Component has only a single class wrapping the actual component:

```
scs-custom-component-wrapper
```

You have full control of the CSS styles that you need to use to render custom view for your custom local component. A local component is rendered inline; that is, you can directly apply CSS styles defined in your theme or in the design.css file.

Custom Remote Component
The Custom Remote Component has only a single class wrapping its iframe:

```
scs-app-iframe-wrapper
```

In addition to applying CSS styles defined in your custom remote component, you can leverage Sites SDK to fetch a design.css file from the host site.

```javascript
// fetch current theme design from host site and then add it to the page
SitesSDK.getSiteProperty('theme',function(data){
    // check if we got a url back
    if ( data.url && typeof data.url === 'string' ) {
        if ( data.url !== '') {
            // theme is loaded, so dynamically inject theme
            SitesSDK.Utils.addSiteThemeDesign(data.url);
        }
    }
});
```
Thus, you can make your component inherit styling from the host style.

**Divider**

Although there are no component-specific classes for the Divider component, the `<hr>` tag itself can be styled.

For example, you can create a dotted divider:

```css
.design-style .scs-divider hr {
  border-top: 1px dotted #333333;
}
```

**Video, YouTube, Spacer**

There are no component-specific classes for Video, YouTube, or Spacer components.

### Set Component Properties

You can configure component properties for use in a site.

Components are the individual parts of a web page, which include text, titles, images, buttons, dividers, maps, galleries, videos, and so on. When you create a design for a theme, you also must specify the default settings for each type of component. Each component has settings — such as size, alignment, spacing, color, and borders — that define how the component looks and behaves. The settings vary based on the component. You also can choose whether a component properties can be changed by users once it is available in a site in a new theme.

As an example, these steps illustrate how you can configure settings for a Paragraph component:

1. With your development site open in Edit mode, select a page that has a Paragraph component in it, or add a Paragraph component.

2. Click the Paragraph component, then click in the corner of the component, and then choose Settings.

   You'll see a Settings panel displayed where you can make selections for the component.

   **Note:**

   The settings options are specific to each type of component. If working with local or remote custom components, you'll see a Custom Settings link.
3. For example, you can change the settings for alignment, width, spacing, fonts, style, color, size, and so on. The component appearance changes to the new settings.

4. When you’re finished, click [X]. Your settings are applied to the page.

Components Rendered in Inline Frames

Components that are rendered in inline frames can be specified in Oracle Content and Experience Cloud pages by registering and adding components from external servers (referred to as remote components), and also by selecting the inline frame option when creating a local component. This type of component can extend functionality for sites, such as adding a social component or a check-out cart component.

For a remote component, you must specify the endpoint URL. For a local component rendered in an inline frame, the URLs are derived from the name of the component in the Component Catalog.

- Endpoint URL: The content of the widget is fetched from this URL and embedded within an inline frame.
- Settings URL: This URL is rendered in an inline frame to configure the component once it is dropped onto a page.

The component can render static or dynamic data visualizations and display a form or other interactive user interface that extends site functionality. See Render Component Settings.

Components Provided by Oracle Content and Experience Cloud

A set of components that are rendered in inline frames is provided with Oracle Content and Experience Cloud.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder List</td>
<td>scs-app</td>
<td>Folder List</td>
</tr>
</tbody>
</table>
Component Registration

Before a remote component rendered in inline frames can be used in a site, the endpoint URL must be registered using HTTPS. This information is stored in the Component Catalog. The endpoint must allow the URL to display in an inline frame; don’t set `X-Frame-Options="sameorigin"` in the header. For a local component, because the files are stored on the Oracle Content and Experience Cloud server, this endpoint criteria is automatically met.

When a component rendered in an inline frames is registered, a new GUID is generated that represents the component. If such a component is registered in an Oracle Content and Experience Cloud instance multiple times, it will get multiple GUIDs because they represent each registration of that component. When the component is registered, the description is all that is stored in Oracle Content and Experience Cloud against the GUID; the component will still run from its remote endpoint. See Register a Remote Component.

Once a component rendered in an inline frame is registered, an instance ID is also generated. This instance ID represents a component registered with a certain Oracle Content and Experience Cloud tenant. The same component, registered more than once within the same Oracle Content and Experience Cloud tenant or with a different Oracle Content and Experience Cloud tenant, will have different instance IDs.

Remote Component Settings Persistence

When the Settings URL for a component rendered in an inline frame is rendered in a Settings panel dialog, the instance ID and component ID are both provided. This allows the component to choose to persist any settings itself in its own server, indexed by instance ID and component ID. Alternatively, the Oracle Content and Experience Sites SDK can be used to allow up to 1.5 KB of JSON data to be stored in the site page’s page model against the component ID.

Using the Sites SDK to persist settings has two benefits for components rendered in inline frames:

- The component can easily participate in page versions, page updates, and the site publishing model.
- The component can comprise HTML endpoints that execute in the browser, as opposed to executing in a back-end system.

See Oracle Content and Experience Sites SDK Reference.
About the Instance ID and Structure for Components Rendered in Inline Frames

The component Instance ID is the unique identifier for a component rendered in an inline frame within a site.

When a user drags and drops a component rendered in an inline frame from the Component Catalog onto a site page, a provisioning call is made to Oracle Sites Cloud Server to generate a new unique component instance ID. This ID is guaranteed to be unique and all such component instances provisioned on that Oracle Content and Experience Cloud will get the same instance ID.

A component instance ID contains additional information that can be used to secure the settings and use of a component rendered in an inline frame, so that the component can be sure that the Settings update is coming from a trusted place.

The instance ID parameter enables developers to identify the site and authenticate the calling party. The caller is authenticated by verifying a digital signature that is generated using the component secret key. The secret key is generated during the component registration process.

The component instance consists of two parts separated by a '.' delimiter: data and structure.

**Component Instance: Data**

The data portion of the instance for a component rendered in an inline frame is a Base64 JSON encoded string. Here's the structure of the JSON string:

```json
{
  "instanceid": "BBDC7614F693B75110D811E6C0B77C935FAEC5112E5B",
  "permissions": ",",
  "entitlements": ",",
  "signdate": "1435426735293",
  "sitedomain": "service1-tenant4.localhost"
}
```

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instanceid</td>
<td>Unique identifier of a component rendered in an inline frame for an Oracle Content and Experience Cloud tenant.</td>
</tr>
<tr>
<td>signdate</td>
<td>Signature generation date.</td>
</tr>
<tr>
<td>sitedomain</td>
<td>Domain name of the Oracle Content and Experience Cloud instance.</td>
</tr>
<tr>
<td>permissions</td>
<td>Set of permissions of the site member. In editing mode, it will have the value &quot;SITE_OWNER&quot;; otherwise, it will have no value.</td>
</tr>
<tr>
<td>entitlements</td>
<td>List of premium features purchased by the site owner.</td>
</tr>
</tbody>
</table>

**Component Instance: Signature**

The data portion of the component instance is serialized before being signed by an APP_SECRET_KEY. This secret key must be generated and shown to the developer while registering the component. The signature is calculated by generating a hash of the
data portion of the component instance (a serialized JSON structure) with the secret key as shown here:

$signature = HMAC (serialized JSON structure, APP_SECRET_KEY)

The hash algorithm used in generating the signature is SHA256. The token is then the concatenation of the serialized JSON structure and the generated signature component as shown here:

$instance = \{base64encoded serialized JSON structure\}.\{base64encoded $signature\}

Example:

\{base64 encoded serialized object\} \{base64 encoded signature\}

Security for Remote Components

Oracle Content and Experience Cloud enables third-party developers to integrate their custom components into the Oracle Content and Experience Cloud platform but have them stored on a remote server.

Each remote component must have registered settings and rendering endpoints with Oracle Content and Experience Cloud. In addition to endpoints, developers also need to provide a secret key unique to the registered component.

Oracle Content and Experience Cloud invokes registered component’s endpoints to realize the content in a site page. Because these endpoints are exposed to public Internet, developers should verify that the endpoints of a registered remote component are being invoked from Oracle Content and Experience Cloud. For verifying the authenticity of the caller, a signed token is delivered to the registered endpoints of an URL. The calling party is authenticated by verifying the digital signature embedded in the signed token with the secret key of the remote component that was provided during the registration process.

The format of the token is:

\{base64 encoded serialized JSON data\}.\{base64 encoded signature\}

A sample token passed to the registered app endpoints follows:

\{base64 encoded serialized object\} \{base64 encoded signature\}

The token consists of two distinct parts: data and signature separated by a ‘.’ delimiter.
As a general guideline, developers should always authenticate the token in Edit or Preview mode before granting access to registered endpoints of a component. In addition, while authenticating the calling party in the settings endpoint, developers should always take care to look for a SITE_OWNER value in the permissions field of the token. The permissions field of the token shows the SITE_OWNER value only in Edit mode. A token generated during an editing session is never persisted back to the page model and is switched out with a runtime token that has a NULL value in the permissions field.

**Data**

The data portion of the instance is a Base64 JSON encoded string. Here’s the structure of the JSON string:

```json
{
    "instanceid": "BBDC7614F693B75110D811E6C0B77C935FAEC5112E5E",
    "permissions": "",
    "entitlements": "",
    "signdate": "1435426735293",
    "sitedomain": "service1-tenant4.localhost"
}
```

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instanceid</td>
<td>Unique identifier of a component for an Oracle Content and Experience Cloud tenant.</td>
</tr>
<tr>
<td>signdate</td>
<td>Signature generation date.</td>
</tr>
<tr>
<td>sitedomain</td>
<td>Domain name of the Oracle Content and Experience Cloud instance.</td>
</tr>
<tr>
<td>permissions</td>
<td>Set of permissions of the site member. In Edit mode, it will have the value &quot;SITE_OWNER&quot;; otherwise, it will have no value.</td>
</tr>
<tr>
<td>entitlements</td>
<td>List of premium features purchased by the site owner.</td>
</tr>
</tbody>
</table>

**Signature**

The data portion of the remote component instance is serialized before being signed by an APP_SECRET_KEY. This secret key must be generated and shown to the developer while registering the component. The signature is calculated by generating a hash of the data portion of the component instance (a serialized JSON structure) with the secret key, as shown here:

```
$signature = HMAC (serialized JSON structure, APP_SECRET_KEY)
```

The hash algorithm used in generating the signature is SHA256. The token is then the concatenation of the serialized JSON structure and the generated signature component as shown here:

```
$instance = {base64encoded serialized JSON structure).
{base64encoded $signature}
```
Register a Remote Component

Before you can use a remote component in a site, it must be registered in Oracle Content and Experience Cloud.

You can register third-party remote components and those that you have developed yourself.

To register a remote component for use in your Oracle Content and Experience Cloud:

1. Click Developer and then click View All Components.
2. Click Create and choose Register Remote Component.
3. In the Register Remote Component window, enter or select information including:
   - **Name**: Name of the component that users will see.
   - **Description**: Description of the component that users will see.
   - **Component URL**: The end point used in an iframe to render component content in a page. It must be HTTPS.
   - **Settings URL**: The end point used in an iframe to render the settings of a remote component added to a page. It must be HTTPS.
   - **Settings Width**: Sets the default width of the component settings panel in pixels.
   - **Settings Height**: Sets the default height of the component settings panel in pixels.
   - **Key**: A 192–bit AES key associated with the remote component and used to create a signed hash token when the component is provisioned. It’s used to encrypt and ensure component settings are read and written securely.
4. Click Register.

When the remote component is created, the name appears in the list of components. You can explore the files used to register the component by clicking the component name in the list of components.

The component registration information is stored in the catalog used by sites created in the same Oracle Content and Experience Cloud instance, but the component remains a remote service.

As the component owner, the component icon is added to the Custom Components panel in Site Builder with the name you assigned to the component. You can share the component with other users and they will see the component in the Custom Components panel in Site Builder.

Delete a Component

If you have the appropriate permissions, you can delete a component from the component manager so it’s no longer available for use. When you delete a
component, the component folder and all its associated folders and files are moved to
the trash.

You can delete a component from the component manager if you created the
component (you’re the component owner) or if someone has shared a component with
you and has given you a manager role.

Note:
You can't delete a component if it’s in use by any site or update, including
sites or updates that are in the trash.

To delete a component:

1. On the home page, click Developer.
The Developer page is displayed.
2. Click Components.
Any currently registered components are displayed.
3. Right-click a component name and choose Delete.
You’re prompted to confirm your action.
4. To confirm the delete action, click Yes. To stop the delete action, click No.
If you confirm the delete, the component and all its associated folders and files are
moved to the trash.

A deleted component folder stays in the trash until:
• You restore the folder.
• You permanently delete the folder.
• Your trash quota is reached.
• The trash is automatically emptied based on the interval set by your service
  administrator. The default value is 90 days.

Sites SDK

Components developed for Oracle Content and Experience Cloud are rendered as a
component in a site and can be dragged and dropped anywhere on a specific page of
a site.

The Sites SDK handles all communication between the component and the page.

• Sites.Settings.getProperty(propertyName, callbackFunction): Provides a
callback to retrieve the requested property for the custom component instance.
• Sites.Settings.setProperty(propertyName, propertyValue): Stores the
  requested property against the custom component instance.

Use Sites.Settings.getProperty or Sites.Settings.setProperty for all custom
component properties, then use SitesSDK.publish and SitesSDK.subscribe for
listening to message events.

See Oracle Content and Experience Sites SDK Reference.
Improve Performance

You can improve the performance of content delivery and rendering in the browser by leveraging the browser cache. Above the fold (ATF) rendering can also improve website rendering.

- Leverage Caching to Improve Performance
- Above the Fold (ATF) Rendering

Leverage Caching to Improve Performance

Delivery of content items, digital assets, and sites should take full advantage of a visitor’s browser cache to improve performance of content delivery and rendering in the browser.

Sites, themes, content items, and digital assets are cached for an amount of time in the visitor’s browser cache. After a site, theme, content item, or digital asset is updated, a cache-buster key in the URL is changed so that the browser has to fetch a different URL and get the new item.

The cache key helps to manage usage of the browser cache by referencing only current resources. Although the cache key is included in the URL, it is a logical element, not a physical location (folder) as is often the case. A change in the cache key does not point to a different physical location to find the resource; it simply notifies the server to fetch the current version of the resource.

Resources can be static, like CSS, JS, and image files, or dynamic, like page data, site data, and content item data. There are five categories of resources for building a website:

- **Product resources** – Resources that are part of the product that gets updated whenever a new version of the product is released or patched.
- **Site Resources** – Resources that are part of the site, like `structure.json`, page data, and images. These are updated when the site is published. The controller is described in the following text.
- **Theme resources** – Resources that are part of the themes, like layouts, CSS, and images. These are updated when the theme is published.
- **Component resources** – Resources that are part of custom components. These include HTML, JS, and CSS, and image files that make up the component. These are updated when a component is published. If one component changes and is republished, then the cache key changes for all components because it’s a single key for all components.
- **CaaS resources** – Resources that serve content items and digital items. These are updated when content items are published or republished or the collection target is changed.
The following topics describe caching for the Content and Experience Cloud runtime and Site Builder:

- **Runtime Caching**
- **Site Builder Caching**

## Runtime Caching

For runtime, the Content and Experience Cloud Cache-Control header is set to 15 days. A cache key is added to the URL for all resources.

As long as the URL is the same, the browser will serve the resource from its local cache if available. When the resource is updated, the cache key is updated in the URL, which forces the browser to make a new request to the server and update the local cache.

The controller, which contains the cache keys, is also cached for 1 minute. Because of this, any updated cache keys will not be seen for up to 1 minute.

At runtime the server returns `controller.html` with the latest cache keys for product, site, theme, components, and CaaS resources. A script with keys is added to `controller.html`; for example:

```html
<script type="text/javascript">
  var SCSCacheKeys = {
    product: '123',
    site: '456',
    theme: '789',
    component: '012',
    caas: '345'
  };
</script>
```

These keys are used by `controller.js` to construct URLs like the ones in the following table.

<table>
<thead>
<tr>
<th>Type of Resources</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Resources</td>
<td>/sitePrefix/productCacheKey/_sitesclouddelivery/...</td>
</tr>
<tr>
<td></td>
<td>/mySite/_cache_947d/_sitesclouddelivery/...</td>
</tr>
<tr>
<td>Theme Resources</td>
<td>/sitePrefix/themeCacheKey/_themesdelivery/themeName/...</td>
</tr>
<tr>
<td>Component Resources</td>
<td>/sitePrefix/compCacheKey/_compdelivery/compName/...</td>
</tr>
<tr>
<td>Type of Resources</td>
<td>Examples</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Site Resources</td>
<td>/sitePrefix/siteCacheKey/content/...&lt;br&gt;/sitePrefix/siteCacheKey/structure.json&lt;br&gt;/sitePrefix/siteCacheKey/pages/100.json</td>
</tr>
<tr>
<td>CaaS Resources</td>
<td>RegularCaaSUrl?cacheKey=caasCacheKey</td>
</tr>
</tbody>
</table>

By inserting the cache key into the URLs like this, Content and Experience Cloud can force the browser to load updated resources by effectively changing the URL so the browser thinks it's actually a new resource.

**Note:**

For secure sites, only the product, theme, and component resources are cached, not the site or CaaS content.

### Site Builder Caching

In Site Builder, static resources are cached for 15 days.

When you use Site Builder, caching happens for product, theme, and component resources. (It does not happen for site and CaaS resources.) Theme and component cache keys are regenerated when Site Builder is launched or refreshed.

If you make a change to a theme or component and want that change to appear in Site Builder, you need to refresh Site Builder (F5).

### Above the Fold (ATF) Rendering

ATF rendering gives the appearance of a website loading faster than it actually does. The goal is to render first all the parts of a page that are visible and then, before the user scrolls down, render the rest of the page that is not initially visible.

A slot can have an "above the fold" designation, which displays an icon on the tab.

For a slot to be rendered in this new way, it must be marked with `scs-atf`, as follows:

```html
<div class="scs-slot scs-atf" id="headline"></div>
```
A component needs to notify the renderer when it is done rendering. Out-of-the-box components do this by default. A custom component needs to make the following API call once it finishes rendering:

```javascript
SCSRenderAPI.onComponentRenderComplete(<componentId>);
```

If not all components in an ATF slot report back that they are done in a timely manner, then the renderer will wait for 2 seconds before continuing with the rest of the page. If you know this will not be long enough, you can extend the time by declaring the following global variable in a page template:

```javascript
var SCSAtfPassTimeout = 3000;
```

**Note:**
The time is in milliseconds so this example set the timeout to 3 seconds.

An API provides diagnostic data for the ATF process. You can call the following method in the debug console, or you can access it from a page if needed:

```javascript
SCSRenderAPI.getRenderMetrics();
```

For example:

```javascript

1. atfComponentCount:13
2. atfPassEndTime:306.535
3. completionCount:23
4. completionRecords:Array(23)
1. 0:{atf: true, componentId: "a7afdd33-3fbb-4329-bc1b-6be60056a995", time: 280.065}
2. 1:{atf: true, componentId: "edfcfcb4-b0d3-422f-aa59-5c925bbbebee", time: 283.54}
3. 2:{atf: true, componentId: "c1c3aec8-e52f-406c-8c29-ab69c05877ed", time: 283.56000000000006}
4. 3:{atf: true, componentId: "b3a31dc6-62a1-44d9-9c80-bdb2c5bedaaa", time: 284.13000000000005}
5. 4:{atf: true, componentId: "c05aa1a2-c11c-4ef5-9051-4799c5bee24a", time: 284.15500000000003}
```
Chapter 13

Above the Fold (ATF) Rendering

6. 5:{atf: true, componentId: "baf4d047-06ec-4739-9b23-9db74f573f30", time: 294.665}

7. 6:{atf: true, componentId: "e7d49528-0357-4b45-801e-b3a2716a086c", time: 297.995}

8. 7:{atf: true, componentId: "a5f33674-4022-4138-8cc5-fe00c02a557", time: 299.78000000000003}

9. 8:{atf: true, componentId: "ccf6dc98-1dbd-440e-b867-5e683cea2ec5", time: 301.19500000000005}

10. 9:{atf: true, componentId: "d691bc44-fed9-474a-9806-2191f465e2e", time: 302.46}

11. 10:{atf: true, componentId: "cf613054-05d8-40dd-83a0-718760d7bc73", time: 303.79}

12. 11:{atf: true, componentId: "b4a6ef98-ffc8-48c7-987c-63346ee97bcc", time: 305.115}

13. 12:{atf: true, componentId: "d1fa2c6e-66ba-419b-b517-2cb4a7601c3", time: 306.535}

14. 13:{atf: false, componentId: "ba3f8ed4-31d4-4347-b6f0-f1019783a57c", time: 318.665}

15. 14:{atf: false, componentId: "ae8af486-76b3-47cd-9989-db4212eefebb", time: 320.45500000000004}

16. 15:{atf: false, componentId: "a48b5abb-49b2-4456-90bd-a3de998150c8", time: 320.48}

17. 16:{atf: false, componentId: "a9650e6d-7e7e-42a2-b758-58f2aeab18a2", time: 322.61500000000007}

18. 17:{atf: false, componentId: "aca9836a-f955-4aa7-8db2-fd3cf189deaa", time: 324.23500000000007}

19. 18:{atf: false, componentId: "e3d7941c-fbc7-4da9-963b-e3810b6467d4", time: 325.85}

20. 19:{atf: false, componentId: "eece809-da54-4066-9326-73f9d9c35fe4", time: 327.315}

21. 20:{atf: false, componentId: "e8f4fb16-4e15-4570-b7de-304e99449a7", time: 328.74}

22. 21:{atf: false, componentId: "a7baa06e-7f30-42c7-94f4-e171ab2edcd6", time: 330.0900000000003}

23. 22:{atf: false, componentId: "fd603b96-2beb-4e87-a54f-12d0e264cd0a", time: 331.38500000000005}

24. length:23
25. __proto__: Array(0)
5. componentCount: 23
6. currentTime: 16243.400000000001
7. mainPassEndTime: 331.38500000000005
8. mainPassStartTime: 316.475
9. renderStartTime: 264.36
10. __proto__: Object
14

Embed the Web User Interface

You can embed the Oracle Content and Experience Cloud web user interface into your own web applications in an HTML inline frame (iframe tag).

The embedded interface removes the default branding and resizes the content to fit the enclosing frame, allowing you to integrate Oracle Content and Experience Cloud into your own web applications.

The following topics describe how to embed the interface and use it to access digital assets, folders, and files in Oracle Content and Experience Cloud from your applications and to display content from other domains.

• Embed the Web User Interface for Assets
• Embed the Web User Interface for Documents
• Authentication and SSO Environments
• Real-Time Updates for Embedded Conversations
• Embed Documents Manager on a Site Page
• Security for Content in Other Domains
• Embed Content in Other Domains
• Browser Configuration Parameters
• Embed Search Results
• Embed Sites Content in Another Domain
• Configure the File Picker with the SitesSDK.filePicker Method

For content, you need to use application links (applinks) to access folders and files. An applink provides access to a folder or file for a user with a particular role for a period of time. You can use the applink on subsequent calls that access the associated folder or file without having to establish user credentials with each call. For more information, see Applinks Resource.

Note:

Before implementing an interface solution that uses inline frames, be sure you understand the possible security risks associated with hosting external sites in inline frames. Security measures vary between different browsers and different browser versions. For more information, see http://www.w3.org/TR/UISecurity/.
Embed the Web User Interface for Assets

Embed the web user interface for assets into your own web applications to get access to digital assets in an Oracle Content and Experience Cloud asset repository.

The next two figures show both versions of the user interface. The first figure shows the user’s assets with an asset repository selected in the standard interface.

The following figure shows an example of the same asset selection using the embedded interface, in the Assets View. Note that the banner and side panels are not displayed in the embedded user interface.
To embed the Oracle Content and Experience Cloud user interface for assets in an inline frame, add `/embed` immediately after the `/assets` element in the URL used to populate the inline frame. For example, the following URL calls the standard user interface for assets:

```plaintext
https://www.example.com/assets
```

**Note:**

The embedded user interface adjusts the content to fit within windows as small as 320 pixels wide. Windows smaller than 320 pixels begin to hide content on the right edge of the window.

You can specify configuration parameters on the URL that control some aspects of the browser display. For information about configuring the appearance of the embedded user interface, see Browser Configuration Parameters.
Embed the Web User Interface for Documents

Embed the web user interface for Oracle Content and Experience Cloud documents into your own web applications to get access to the folder and document management features of the service.

To embed the Oracle Content and Experience Cloud user interface for documents in an inline frame, add `/embed` to any member or public folder link immediately after the `/documents` element in the URL used to populate the inline frame. For example, the following URL calls the standard user interface for documents and shows the home folder for the current user:

https://www.example.com/documents/home/nameasc

To display the home folder in the embedded user interface, use the following form of the URL:

https://www.example.com/documents/embed/home/nameasc

To open a specified folder in the embedded interface, use the `folder` element in the URL and specify the globally unique identifier (GUID) of the folder. For example, the following link opens the specified folder in the embedded user interface.

https://www.example.com/documents/embed/folder/1713A5712BE73C37891915A0127B594F/nameasc

**Note:**

The embedded user interface adjusts the content to fit within windows as small as 320 pixels wide. Windows smaller than 320 pixels begin to hide content on the right edge of the window.

You can also embed member links and public links to folders and specify configuration parameters on the URL that control some aspects of the browser display. For information about configuring the appearance of the embedded user interface, see Browser Configuration Parameters.

Authentication and SSO Environments

If you have signed in to the full web client in another tab, then the embedded web user interface will load without additional authentication, using the already established session in the browser.

If there is no existing browser session, then the behavior of the `/embed` URL is to present a sign in screen in a popup browser window. Once credentials are provided in the popup sign in window, the embedded client will reload without further authentication.
Real-Time Updates for Embedded Conversations

Using `embedEnterHive` enables real-time updates to a conversation from other users who may be posting in the conversation as well as presence indicators for users as displayed in the conversation header, such as Active or Idle.

Embedding a conversation view in the web user interface disables back-channel communication with the server by default. This means the view is a static display of the conversation at the time it was loaded, and the view will not display real-time updates. To get real-time updates, you must add the parameter `embedEnterHive` to the URL used to load the embedded view. For example:

https://www.example.com/documents/embed/conversation/12345?embedEnterHive

The `embedEnterHive` parameter can also be used to display a folder or file view that has a side-by-side conversation in the side bar, to get real-time updates on the conversation.

Using `embedEnterHive` is required in an embedded file view for annotations to function properly. For example:


The web user interface cannot accurately detect when the iframe containing it is destroyed. When the `embedEnterHive` parameter is used, the hosting code must signal to the web user interface in the iframe to close the service. This done through a `postMessage` call as follows:

```javascript
iframewindow.postMessage({message: "closeService"}, ");
```

This ensures the client properly sends an “exit hive” call to the server, discontinuing back-channel communication.
Embed Documents Manager on a Site Page

You can embed documents manager on a site page to provide a view of your home page or files in Oracle Content and Experience Cloud.

See Documents in *Creating Experiences with Oracle Content and Experience Cloud*.

Security for Content in Other Domains

When you embed the Oracle Content and Experience Cloud web user interface, it's enclosed in an inline frame. Security policies are enforced to minimize the security risks of hosting external sites in inline frames.

In general, inline frames can host content if the protocol, domain, and port of the inline frame are identical to those for the content it displays. For example, by default, an inline frame on the page http://www.example.com:12345/home.html can host content only if the content’s protocol is also http, the domain is www.example.com and the port is 12345.

An administrator can explicitly identify domains outside of the host domain and allow content from those domains to display in the embedded web user interface.

**Note:**

Before implementing an interface integration that uses inline frames, be sure you understand the possible security risks associated with hosting external sites in inline frames. Security measures vary between different browsers and different browser versions. For more information, see [http://www.w3.org/TR/UISecurity/](http://www.w3.org/TR/UISecurity/).

Oracle Content and Experience Cloud displays content for authorized users in the embedded interface. Supported content includes folders, member links, and views of individual files.

All unsupported content, such as public links and content from unauthorized domains, opens in a new browser tab or window (depending on the browser settings).

Embed Content in Other Domains

You can display content from Oracle Content and Experience Cloud within other domains. For example, you might embed the Oracle Content and Experience Cloud web user interface into your own web applications to access folder and document management features inside your application.

To allow users to embed content, enable embedded content and add domains:

1. After you sign in to the Oracle Content and Experience Cloud web application as an administrator, click **Settings** in the Administration area of the navigation menu.

2. In the **Settings** menu, click **Security**.

3. Under **Embedded Content**, select **Enabled**.
4. In the **Allowed domains** box, enter a list of permitted domains, separated by commas. Domains must be in the form `www.example.com`.
   - To restrict the domain to a particular port, include the port in the specification. For example, `www.example.com:12345`.
   - If you want to allow a domain that has multiple sub-domains, you can use the * wildcard character. For example, `www.example.*` includes the domains `www.example.com`, `www.example.co.uk`, and so on.

### Browser Configuration Parameters

You can include optional parameters in the URL for a folder or folder link to control some aspects of the browser display. This can be very useful for embedding the user interface in other applications.

#### Layout Configuration

You can append the layout configuration parameter (`lyt`) and a single value to the end of a URL for a folder or folder link using the following format:

```
{URL}/lyt={value}
```

For example, the following URL uses a member link to display the associated folder in the embedded interface with a grid layout:

https://www.example.com/documents/embed/folder/FDB8AC67BE3FCFB4984E1327T0000DEFAULT00000000/_Resources/lyt=grid

**Note:** You can use the layout parameter with folders and folder links in both the standard and embedded interfaces.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Displays the folder content in the standard list layout.</td>
</tr>
<tr>
<td>grid</td>
<td>Displays the folder content in the standard grid layout.</td>
</tr>
</tbody>
</table>

#### Embed Mode Configuration

You can specify an embed mode configuration as a URL query parameter:

- **hide**: Hides elements that are visible by default when embedded.
- **show**: Shows elements that are visible by default when embedded.

Options of the embed mode configuration apply only when you use embed mode URLs, including applink URLs. You can use the following embedded views:

- **Assets View** (`embed/assets`)
- **Folder View** (`embed/folder/{folderId}`) or (`embed/folder/home`)
- **File View** (`embed/fileview/{fileId}`)
• Conversations List (embed/conversations)
• Conversation View (embed/conversations/{conversationId})
• Themes View (embed/themes)

For example:

/documents/embed/folder/{folderId}?hide=header+actions&show=branding&singleSel

In the Assets View, you can use the repository configuration parameter to preselect the Repository filter value:

https://www.example.com/embed/assets/repository/{repositoryID}

You can add individual options directly as parameters. Combine multiple hide and show options with either a plus sign (+) or comma (,). For example:

hide=header+breadcrumbs+sidebar

The following table lists the options you can use for embed mode configuration.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoplay</td>
<td></td>
<td>Auto play video and audio content</td>
</tr>
<tr>
<td>channel</td>
<td>&lt;channelId&gt;</td>
<td>Preselect the channel filter value. Must be valid for the initially loaded (preselected) repository.</td>
</tr>
<tr>
<td>channeloptions</td>
<td>&lt;channelId&gt;++&lt;channelId&gt;+...</td>
<td>Limit the Channel filter control to the provided list</td>
</tr>
<tr>
<td>channeloptions</td>
<td>none</td>
<td>Hide the Channel filter control altogether</td>
</tr>
<tr>
<td>collapsetn</td>
<td></td>
<td>Start with thumbnails area collapsed</td>
</tr>
<tr>
<td>collection</td>
<td>&lt;collectionId&gt;</td>
<td>Preselect the collection filter value. Must be valid for the initially loaded (preselected) repository.</td>
</tr>
<tr>
<td>collectionoptions</td>
<td>&lt;collectionId&gt;++&lt;collectionId&gt;+...</td>
<td>Limit the Collection filter control to the provided list. The list can contain collections from any repository in the repositoryOptions list. Control will display only collection validation for any selected repository.</td>
</tr>
<tr>
<td>collectionoptions</td>
<td>none</td>
<td>Hide the Collection filter control altogether</td>
</tr>
<tr>
<td>contenttype</td>
<td>article+blog+...</td>
<td>Preselect the listed content type filters for the Assets View</td>
</tr>
<tr>
<td>contenttypeoptions</td>
<td>none</td>
<td>Hide the Content Items section in the Assets View</td>
</tr>
<tr>
<td>contenttypeoptions</td>
<td>article+blog...</td>
<td>Display the listed content type filters as available options</td>
</tr>
<tr>
<td>docsonly</td>
<td></td>
<td>Include only documents in list</td>
</tr>
<tr>
<td>fitoriginal</td>
<td></td>
<td>Start document at original size</td>
</tr>
<tr>
<td>fitpage</td>
<td></td>
<td>Start with document fit to page</td>
</tr>
<tr>
<td>fitwidth</td>
<td></td>
<td>Start with document fit to width</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>foldersonly</td>
<td></td>
<td>Include only folders in list</td>
</tr>
<tr>
<td>hide</td>
<td>actions</td>
<td>Hide all document actions</td>
</tr>
<tr>
<td>hide</td>
<td>add</td>
<td>Hide action Add in the Assets view header</td>
</tr>
<tr>
<td>hide</td>
<td>breadcrumbs</td>
<td>Hide breadcrumb area</td>
</tr>
<tr>
<td>hide</td>
<td>collections</td>
<td>Hide action Collections in the Assets view header</td>
</tr>
<tr>
<td>hide</td>
<td>copy</td>
<td>Hide action copy</td>
</tr>
<tr>
<td>hide</td>
<td>delete</td>
<td>Hide action delete</td>
</tr>
<tr>
<td>hide</td>
<td>dnload</td>
<td>Hide action download</td>
</tr>
<tr>
<td>hide</td>
<td>favs</td>
<td>Hide Favorite actions and View menu</td>
</tr>
<tr>
<td>hide</td>
<td>fitpage</td>
<td>Hide fit to page action in file view</td>
</tr>
<tr>
<td>hide</td>
<td>fitwidth</td>
<td>Hide fit to with action in File View</td>
</tr>
<tr>
<td>hide</td>
<td>header</td>
<td>Hide content header</td>
</tr>
<tr>
<td>hide</td>
<td>move</td>
<td>Hide action move</td>
</tr>
<tr>
<td>hide</td>
<td>new</td>
<td>Hide header create action (new folder)</td>
</tr>
<tr>
<td>hide</td>
<td>nextprev</td>
<td>Hide next and previous controls (File View)</td>
</tr>
<tr>
<td>hide</td>
<td>pin</td>
<td>Hide annotation mode pin</td>
</tr>
<tr>
<td>hide</td>
<td>props</td>
<td>Hide action props</td>
</tr>
<tr>
<td>hide</td>
<td>rename</td>
<td>Hide action rename</td>
</tr>
<tr>
<td>hide</td>
<td>reserve</td>
<td>Hide action reserve</td>
</tr>
<tr>
<td>hide</td>
<td>share</td>
<td>Hide action share</td>
</tr>
<tr>
<td>hide</td>
<td>sidebar</td>
<td>Hide sidebar</td>
</tr>
<tr>
<td>hide</td>
<td>thumbs</td>
<td>Hide viewer thumbnail area</td>
</tr>
<tr>
<td>hide</td>
<td>toolbar</td>
<td>Hide toolbar (Folder View)</td>
</tr>
<tr>
<td>hide</td>
<td>trash</td>
<td>Hide Trash View menu</td>
</tr>
<tr>
<td>hide</td>
<td>upload</td>
<td>Hide action upload</td>
</tr>
<tr>
<td>languageoptions</td>
<td>en-US+fr-FR+...</td>
<td>Limit the Language filter control to the provided list</td>
</tr>
<tr>
<td>languageoptions</td>
<td>none</td>
<td>Hide the Language filter control altogether</td>
</tr>
<tr>
<td>launch</td>
<td></td>
<td>Launch documents on view (opens new tab versus inline)</td>
</tr>
<tr>
<td>loop</td>
<td></td>
<td>Loop video and audio content</td>
</tr>
<tr>
<td>mute</td>
<td></td>
<td>Mute video and audio content</td>
</tr>
<tr>
<td>nofocus</td>
<td></td>
<td>Don’t take focus</td>
</tr>
<tr>
<td>noopen</td>
<td></td>
<td>Disable action open for folders in Folder View and conversations in conversation list (conversation list becomes multiselect)</td>
</tr>
<tr>
<td>novidctls</td>
<td></td>
<td>Hide video controls</td>
</tr>
<tr>
<td>noview</td>
<td></td>
<td>Disable action view for files in Folder View</td>
</tr>
<tr>
<td>repository</td>
<td>&lt;repositoryId&gt;</td>
<td>Preselect the Repository filter value in the Assets View</td>
</tr>
<tr>
<td>repositoryoptions</td>
<td>&lt;repositoryId&gt;+&lt;repositoryId&gt;+...</td>
<td>Limit the Repository filter control to the provided list. Must include the repository parameter value, if provided.</td>
</tr>
<tr>
<td>show</td>
<td>backbutton</td>
<td>Add a back arrow</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>show</td>
<td>branding</td>
<td>Show branding header (hidden in embed mode)</td>
</tr>
<tr>
<td>simpleDisplay</td>
<td></td>
<td>Simplified select list</td>
</tr>
<tr>
<td>singlesel</td>
<td></td>
<td>Single select</td>
</tr>
<tr>
<td>status</td>
<td>published+approved +rejected+draft +pending</td>
<td>Preselect the lists status filters</td>
</tr>
<tr>
<td>statusoptions</td>
<td>none</td>
<td>Hide the Status panel altogether</td>
</tr>
<tr>
<td>statusoptions</td>
<td>published+approved +rejected+draft +pending</td>
<td>Display the listed status filters as available options</td>
</tr>
<tr>
<td>type</td>
<td>images+video +documents</td>
<td>Preselect the listed type filters for the Assets View</td>
</tr>
<tr>
<td>typeoptions</td>
<td>images+video +documents</td>
<td>Display the listed type filters as available options</td>
</tr>
<tr>
<td>typeoptions</td>
<td>none</td>
<td>Hide the Digital Assets Type section or Type panel in the Assets View</td>
</tr>
<tr>
<td>wide</td>
<td></td>
<td>Display wide content area</td>
</tr>
</tbody>
</table>

**Browser Configuration**

The `embed` mode configuration has replaced the legacy browser `/cfg` settings. Existing `/cfg` settings are supported in `embed` mode if no legacy query parameters are specified.

**Embed Search Results**

In an embedded web user interface for Oracle Content and Experience Cloud, search results are available through an embed URL.

For example, the embed URL format for displaying search results for document and folder types follows:


The `hide=titlemenu` option parameter must be passed in the URL to hide the list filter drop-down menu. The search type cannot be filtered in an embed URL.

In an embed URL, you can use the same search string that you can use in a URL that is not embedded. For information about search strings, see Search Content and Conversations in *Managing Content with Oracle Content and Experience Cloud*.

You can use these search types in an embed URL:

- documents
- conversations
- folder (scoped folder)
For example, the embed URL format for displaying search results within a scoped folder follows, sorted by last updateds:

https://domain-name/documents/search/search-string/folder/lastupdated/folder_guid?hide=titlemenu

You can use these sort-by types in an embed:

- lastupdated
- filesize
- name

For example, the format for displaying search results for conversation types follows, sorted by last updated:


### Embed Sites Content in Another Domain

You can embed an Oracle Content and Experience Cloud site or a site page into a page served from another domain.

To embed a site or site page into an iframe on an external site page that is served from another domain, set the Embeddable Site property to Yes on the site properties page.

See Embed Content in Other Domains in Administering Oracle Content and Experience Cloud.

### Configure the File Picker with the SitesSDK.filePicker Method

You can use the SitesSDK.filePicker() method to pass configuration parameters in a URL for the embeddable File Picker user interface.

Browser Configuration Parameters in Embed the Web User Interface describes the configuration parameters you can pass.

You can specify a configuration parameter with either of two options:

- Name-value pair

  ```javascript
  SitesSDK.filePicker({
    name: 'value-in-string'
  })
  ```

  For example:

  ```javascript
  SitesSDK.filePicker({
    statusoptions: 'published+draft',
  })
  ```
• Name only

    SitesSDK.filePicker({
        <name>: <value-in-boolean>
    })

For example:

    SitesSDK.filePicker({
        autoplay: true,
    })

Custom options are enabled in an asset URL or a document URL.

The following example uses the hello-there app:

    SitesSDK.filePicker({
        statusoptions: 'published+draft',
        autoplay: true,
        showReset: true,
        hide: 'sidebar',
        docsonly: true,
        singlesel: true,
        'multiSelect': false,
        'supportedFileExtensions': ['jpg', 'png']
    })

This example would generate the following URL for the embedded File Picker:

    http://localhost:8080/documents/embed/folder/home/nameasc?
scheme=light&thinheader=true&layout=grid&docsonly&noview&wide&singlesel&statusoptions=published+draft&autoplay&hide=sidebar
Develop Custom Actions

Integrate third-party services and applications with Oracle Content and Experience Cloud through custom actions you can develop in the Application Integration Framework (AIF).

These topics describe AIF and how to use it for effective integration:

- Application Integration Framework Overview
- Configuration File Format
- Application Properties
- Action Command
- Invoke Command
- Presentation Command
- Expressions
- Variables
- Localization

Application Integration Framework Overview

Application Integration Framework (AIF) provides a simple and effective way to integrate third-party services and applications into the Oracle Content and Experience Cloud interface.

Using AIF, you can quickly define the actions that are exposed in the interface, respond to user selections, call third-party services, and specify how the results are presented to the user. The framework supports variables and expressions and provides multiple language support.

Custom AIF applications are not applied when you access them through an applink or public link.

The definition for one or more integrations is stored in a single file in JSON format. As an administrator, you can upload the configuration file and add it to a list of available applications. Also as an administrator, you can edit and validate the configuration file directly in the interface, enable or disable the app for general use, set administration-level preferences, or delete the app.

The definition for one or more integrations is stored in a single configuration file in JSON format. The configuration file defines and manages the interactions between the app, native objects and interface elements. The configuration file includes:

- App properties including tenant and user preferences
- Actions that are exposed in the interface and the service calls they make
- How the results are presented to the user
- Interface strings with support for multiple languages
An administrator adds the configuration file, enables the integration, and provides tenant and account information to get started. Also as an administrator, you can edit and validate the configuration file directly in the interface, download the configuration file, or delete the app.

To manage apps created with Application Integration Framework, sign in as an administrator, open your user menu, choose **Administration**, and then choose **Integrations**. Under Custom Actions, click **Add**.
From the **Applications** page, an administrator can use the following options.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="On Off" /></td>
<td>Enable or disable the application for users. When you enable the application, you can specify preferences for the application from the user menu, by choosing <strong>Preferences</strong> and then <strong>Applications Settings</strong>. You specify the user preferences resource in the <code>userPrefs</code> element in the configuration file.</td>
</tr>
<tr>
<td><img src="image" alt="Add" /></td>
<td>Browse local folders and files to locate and upload an application configuration file.</td>
</tr>
<tr>
<td><img src="image" alt="i" /></td>
<td>Display the information defined for the application and specified in the <code>info</code> element of the configuration file.</td>
</tr>
<tr>
<td><img src="image" alt="i" /></td>
<td>Display the preferences resource for administrators defined in the <code>tenantPrefs</code> element of the configuration file.</td>
</tr>
</tbody>
</table>
Open the configuration file in the integrated JSON editor. The editor validates the syntax of the file to ensure that the file contains valid JSON code. Changes you make to the configuration file are immediately available in the enabled application.

Changes you make to the configuration file are stored only in the server copy of the file. To back up your changes, use the Download icon to save the file locally.

Download the file from the server to a local destination.

Delete the configuration permanently.

When you delete a configuration file, the deletion is permanent. The file can’t be restored from the trash.

## Configuration File Format

The definition for one or more integrations is stored in a single configuration file in JSON format.

The extension of an AIF configuration file must be `.conf`.

As an administrator, you can upload the configuration file and make it available to users. See Application Integration Framework Overview.

A sample configuration file for one application follows. All keys and values in the configuration are case-sensitive.

```json
{
  "id": "ExampleCoPublishing",
  "name": "APP_SUBMIT_TO_PUBLISH_DOCS_NAME",
  "description": "APP_SUBMIT_TO_PUBLISH_DOCS_DESC",
  "category": "CUSTOM",
  "supportEmail": "support@example.com",
  "baseUrl": "http://www.example.com/",
  "info": {
    "documentation": "Opens a URL with a description of the app in a popup window",
  },
  "info": {
    "presentation": {
      "view": "POPUP"
    },
    "invoke": {
      "method": "GET",
      "url": "http://www.example.com/ExampleCoDescr.jsp",
      "data": ""
    }
  },
  "tenantPrefs": {
    "presentation": {
      "view": "POPUP"
    },
    "invoke": {
      "method": "GET",
      "url": "http://www.example.com/ExampleCoAdmin.jsp",
      "data": ""
    }
  }
}
```


"userPrefs": {
  "presentation": {
    "view": "POPUP"
  },
  "invoke": {
    "method": "GET",
    "url": "http://www.example.com/ExampleCoUser.jsp?user={user.id}",
    "data": ""
  }
},

"actions": [
  {
    "id": "submitToPublish",
    "name": "ACTION_SUBMIT_TO_PUBLISH_NAME",
    "description": "ACTION_SUBMIT_TO_PUBLISH_DESC",
    "type": "UI",
    "trigger": "MENU",
    "presentation": {
      "view": "POPUP",
      "popupWidth": 700,
      "popupHeight": 400
    },
    "evaluate": "type=='folder' && user.isMember && user.role=='owner' && !
    isReservedByAnotherUser",
    "invoke": {
      "method": "GET",
      "url": "http://www.example.com/ExampleCoSubmit.jsp?
    user={user.id}&ids=[{id},]&names=[{name},]"
    },
    "multi": false
  },
  {
    "id": "showPublishStatus",
    "name": "ACTION_SHOW_PUBLISH_STATUS_NAME",
    "description": "ACTION_SHOW_PUBLISH_STATUS_DESC",
    "type": "UI",
    "trigger": "SELECT",
    "presentation": {
      "view": "POPUP",
      "popupWidth": 300,
      "popupHeight": 200
    },
    "evaluate": "type=='file' && user.isMember && user.role=='owner' && !
    _.contains(['doc','docx','xls','xlsx','ppt','pptx','tif','png'], extension) &&
    (_.isEmpty(reservedById) || reservedById === user.id)",
    "invoke": {
      "method": "GET",
      "url": "http://www.example.com/ExampleCoSubmit.jsp?
    user={user.id}&ids=[{id},]&names=[{name},]"
    },
    "multi": false
  },
  {
    "id": "publishDirect",
    "name": "ACTION_PUBLISH_DIRECT_NAME",
    "description": "ACTION_PUBLISH_DIRECT_DESC",
    "type": "DIRECT",
    "trigger": "MENU",
    "presentation": {
      "view": "CLIENT",
The Application Integration Framework supports simple use of the JavaScript underscore library (http://underscorejs.org); however, AIF does not allow defining functions as parameters of underscore functions.

The sections that follow describe the functional areas of the Application Integration Framework.

<table>
<thead>
<tr>
<th>Functional Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Properties</td>
<td>Describes the application properties.</td>
</tr>
<tr>
<td>Action Command</td>
<td>Describes the general elements of the action command. Use the action command to specify how to trigger and start a task. The invoke and presentation components of the action command are presented in their own sections.</td>
</tr>
<tr>
<td>Invoke Command</td>
<td>Describes the elements of the invoke command. Use the invoke command to call a third-party service, page, or script.</td>
</tr>
<tr>
<td>Presentation Command</td>
<td>Describes the elements of the presentation command. Use the presentation command to specify how the third-party content retrieved by the invoke command is presented to the user.</td>
</tr>
<tr>
<td>Expressions</td>
<td>Provides examples and general information about expressions used with the invoke and evaluate commands.</td>
</tr>
<tr>
<td>Variables</td>
<td>Provides descriptions of the available item and user variables, the permissions and status objects, and AIF functions.</td>
</tr>
<tr>
<td>Localization</td>
<td>Provides an example and general information about how to localize an app with translated labels and descriptions.</td>
</tr>
</tbody>
</table>
# Application Properties

The following table shows the properties of an Application Integration Framework application.

<table>
<thead>
<tr>
<th>Property</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Yes</td>
<td>A non-empty string that uniquely identifies the application across Oracle Content and Experience Cloud.</td>
</tr>
<tr>
<td>name</td>
<td>Yes</td>
<td>A non-empty string that is displayed for the application’s user interface. It can also be a key that can be translated using translation maps defined in stringsMap.</td>
</tr>
<tr>
<td>description</td>
<td>No</td>
<td>A string that is displayed with the application name in the user interface. It can also be a key that can be translated using translation maps defined in stringsMap.</td>
</tr>
<tr>
<td>category</td>
<td>No</td>
<td>Category of the application. CUSTOM is the only allowed value. If you do not explicitly include category in the configuration file, it is dynamically added during application configuration and is given the default value of CUSTOM.</td>
</tr>
<tr>
<td>supportEmail</td>
<td>No</td>
<td>Email address of application support owner.</td>
</tr>
<tr>
<td>baseURL</td>
<td>No</td>
<td>Base URL used as a prefix to any relative URLs specified in the application.</td>
</tr>
<tr>
<td>stringsMap</td>
<td>No</td>
<td>Used to map languages to translation objects, so that keys in the application definition can be translated to the corresponding language.</td>
</tr>
<tr>
<td>info</td>
<td>No</td>
<td>Allows an integrator to show a dialog to display information about the application. It contains presentation and invoke properties.</td>
</tr>
<tr>
<td>documentation</td>
<td>No</td>
<td>Lets you add comments in the file, such as a comment to identify the purpose of an action. You might add a documentation property at the start of the code sample to describe the application itself.</td>
</tr>
<tr>
<td>tenantPrefs</td>
<td>No</td>
<td>Allows an integrator to expose application-level options in the preference settings for administrators. It contains presentation and invoke properties.</td>
</tr>
<tr>
<td>userPrefs</td>
<td>No</td>
<td>Allows an integrator to expose application-level options in the preference settings for users. It contains presentation and invoke properties.</td>
</tr>
<tr>
<td>actions</td>
<td>Yes</td>
<td>A collection of actions contained in the application. It must contain at least one action.</td>
</tr>
</tbody>
</table>

## Action Command

Use the action command to specify how to trigger and start a specified task and how to present the results.
The following table shows the properties of an action command.

<table>
<thead>
<tr>
<th>Property</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Yes</td>
<td>A non-empty string that uniquely identifies the action in the application.</td>
</tr>
<tr>
<td>name</td>
<td>Yes</td>
<td>A non-empty string that is displayed for the action in the user interface. It can also be a key that can be translated using translation maps defined in stringsMap.</td>
</tr>
<tr>
<td>description</td>
<td>No</td>
<td>A string that is displayed for the action in a user interface tooltip. It can also be a key that can be translated using translation maps defined in stringsMap.</td>
</tr>
<tr>
<td>trigger</td>
<td>No</td>
<td>Specifies if the action is triggered through a context menu taskbar (MENU), or when the item is opened (SELECT). If you do not explicitly include trigger in the configuration file, it is dynamically added during application configuration and is given the default value of MENU. In the Application Integration Framework, the SELECT trigger occurs when an item is opened, not when simply checked in a folder listing.</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Specifies how action results are displayed to the end user. Possible values are UI and DIRECT. If the value is DIRECT, a call is made to the URL specified with invoke, and the response displays as a notification in the Oracle Content and Experience Cloud interface, if possible. If the value is UI, the presentation property specifies the type of interface to use.</td>
</tr>
<tr>
<td>presentation</td>
<td>No</td>
<td>Specifies the presentation to use if type is set to UI. Possible values are CLIENT to embed the action results in the integrations side panel on the right of the display, POPUP to display action results in a separate browser dialog, and WINDOW to display action results in a new browser window or tab. If type is set to UI and presentation is omitted, it is added to the action with the default value of CLIENT. See Presentation Command.</td>
</tr>
<tr>
<td>multi</td>
<td>No</td>
<td>Specifies if the action applies when multiple items are selected (true) or not (false). If you do not explicitly include multi in the configuration file, it is dynamically added during application configuration and is given the default value of false.</td>
</tr>
<tr>
<td>evaluate</td>
<td>Yes</td>
<td>Must be valid JavaScript Boolean value that evaluates to true or false for the selected item. For more information about expression evaluation, see Expressions.</td>
</tr>
<tr>
<td>invoke</td>
<td>Yes</td>
<td>Specifies the external service in an invoke command to use when action is triggered. See Invoke Command.</td>
</tr>
</tbody>
</table>
Invoke Command

Use the `invoke` command to call an external service, page, or script.

The following table shows the properties of the `invoke` command.

<table>
<thead>
<tr>
<th>Property</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Yes</td>
<td>Specifies the HTTP method to use for invocation. Valid values are GET, POST, PUT, DELETE, and HEAD.</td>
</tr>
<tr>
<td>url</td>
<td>Yes</td>
<td>A URL expression to specify the URL for the third-party service. It can be an absolute or relative URL. If a relative URL is specified, it will be added to the baseUrl of the application, if specified.</td>
</tr>
<tr>
<td>data</td>
<td>No</td>
<td>An expression that specifies data to be sent with the URL. See Expressions. A JSON payload can added to the request body as a string object.</td>
</tr>
<tr>
<td>appLinkRole</td>
<td>No</td>
<td>If the action is a request to create an application link (applink), use appLinkRole to specify the role to use for the applink. Valid values are contributor, downloader, and viewer. See Applinks Resource.</td>
</tr>
<tr>
<td>header</td>
<td>No</td>
<td>A custom header to include when making POST calls.</td>
</tr>
<tr>
<td>token</td>
<td>No</td>
<td>Specifies if any special security token needs to be passed. Valid values are PCS (PCS OAUTH token) and NONE.</td>
</tr>
</tbody>
</table>

Presentation Command

Use the `presentation` command to specify how the third-party content retrieved by the `invoke` command is presented to the user.

The following table shows the properties of the `presentation` command.
<table>
<thead>
<tr>
<th>Property</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view</td>
<td>No</td>
<td>Specifies how action response content is displayed to the user. Use CLIENT to embed results in the application, POPUP to display results in a separate browser dialog, or WINDOW to display results in a new browser window or tab. If you use CLIENT with the SELECT trigger, action results are displayed in the integrations side panel on the right of the display. The integrations panel must be open for the results to display. The SELECT trigger occurs when an item is opened, not when simply checked in a folder listing. If you use CLIENT with the MENU trigger, action results are displayed in a pop-up dialog. If you do not explicitly include view in the configuration file, it is dynamically added during application configuration and is given the default value of CLIENT.</td>
</tr>
<tr>
<td>popupWidth</td>
<td>No</td>
<td>A number specifying the width of the container. If none is specified, a default value is used, depending on the context.</td>
</tr>
<tr>
<td>popupHeight</td>
<td>No</td>
<td>A number specifying the height of the container. If none is specified, a default value is used, depending on the context.</td>
</tr>
</tbody>
</table>

**Expressions**

The `invoke` and `evaluate` commands accept expressions, which can include variables and operators that are evaluated at runtime.

**Invoke Expressions**

An *invoke expression* is a template used to generate URL and data values at runtime by replacing the variables specified in the template with their runtime values. Variables can reference one or more selected items, information about the current user, and other system information.

- Anything enclosed in a brace `{ }` is considered to be a key value that is mapped to the corresponding attribute of a file, folder, or the currently signed-in user. User attributes are prefixed by `user`. For example, if John Smith is the current user, `userid={user.name}` resolves to this:
  
  ```
  userid=John Smith
  ```

- For multiselection, use the [repeated template separator] syntax. The repeated template is resolved for each selected item, and resolved strings for items are separated by the specified delimiter. For example, for items with GUIDs `x189` and `y234`, `item=\{id\},` resolves to this:
  
  ```
  item=x189,y234
  ```
Evaluate Expressions

An evaluate expression determines if the item or items selected by the user qualify for a particular action. The expression can contain zero or more variables that are replaced with their runtime values before the expression is evaluated. This expression must be a valid JavaScript expression after all the values of the variables are replaced with their values at runtime.

The expression must evaluate to true or false. If the result is true, the action is made available as specified by the trigger property. If the result is false, the action is not made available to the user.

For example, the following expression returns a true value only if all three conditions evaluate to true: the item selected in the interface is a file, the user is signed in (not accessing the file using a public link or applink), and the user is the owner of the file.

"evaluate": "type=='file' && user.isMember && user.role=='owner'"

Another example is item.meta.<custom property group name>.<custom property group field name>.

Some commonly used JavaScript comparison and logical operators follow.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>logical not</td>
</tr>
</tbody>
</table>

Variables

You can use variables in applications to dynamically provide information about the current user, the currently selected item or items, and other types of information. Variables are evaluated at runtime when the application is used.

You can use a category prefix (item or user) with variable names to make the relationship explicit, especially in cases where variable names are the same, such as name. If a variable is not prefixed by any category, item is used as the category.

For example, to specify the folder or file name, use item.name. To specify the user name, use user.name. If you specify name without the category, then item.name (the folder or file name) is assumed.
Folder and File Item Variables

The following variables are specific to a folder or file item that is currently selected by the user. Folder and file variables belong to the item category. The following table lists variables that apply to both folder and file items.

<table>
<thead>
<tr>
<th>Folder or File Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Item GUID</td>
</tr>
<tr>
<td>name</td>
<td>Item name</td>
</tr>
<tr>
<td>description</td>
<td>Item description</td>
</tr>
<tr>
<td>type</td>
<td>Item type</td>
</tr>
<tr>
<td>parentid</td>
<td>Item parent ID</td>
</tr>
<tr>
<td>ownerid</td>
<td>Item owner GUID</td>
</tr>
<tr>
<td>ownername</td>
<td>Item owner name</td>
</tr>
<tr>
<td>creatorname</td>
<td>Item creator name</td>
</tr>
<tr>
<td>createdat</td>
<td>Date and time item was created</td>
</tr>
<tr>
<td>lastmodifierid</td>
<td>GUID of the user who most recently modified the item</td>
</tr>
<tr>
<td>lastmodifiername</td>
<td>Name of the user who most recently modified the item</td>
</tr>
<tr>
<td>modifiedat</td>
<td>Date and time the item was last modified</td>
</tr>
<tr>
<td>size</td>
<td>Size of the item</td>
</tr>
<tr>
<td>permissions</td>
<td>Actions the user is allowed to perform on the item</td>
</tr>
<tr>
<td>state</td>
<td>State of the item</td>
</tr>
<tr>
<td>applink</td>
<td>Applink created by AIF for this user to access the item, if appLinkRole was specified in the invoke command</td>
</tr>
</tbody>
</table>

The applink object has id, url, and accessToken properties that can be passed in the URL or data as {applink.id}, {applink.url}, and {applink.accessToken}, respectively.

| favorite | Whether the item is a favorite of the user or not |

File-Item Specific Variables

The following variables are specific to a file item that is currently selected by the user. File variables belong to the item category. The table lists variables that apply to file items only.

<table>
<thead>
<tr>
<th>File Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originalname</td>
<td>Original name of the item</td>
</tr>
<tr>
<td>extension</td>
<td>Extension of the item</td>
</tr>
<tr>
<td>revision</td>
<td>Item's revision</td>
</tr>
<tr>
<td>mimetype</td>
<td>Mime type of the item</td>
</tr>
<tr>
<td>reservedbyid</td>
<td>GUID of the user who reserved the item</td>
</tr>
<tr>
<td>reservedbyname</td>
<td>Name of the user who reserved the item</td>
</tr>
<tr>
<td>reservedat</td>
<td>Date and time when the item was reserved</td>
</tr>
</tbody>
</table>
File Variable | Description
---|---
isReservedByAnotherUser | Whether the item is reserved by a user other than the current user

**Item Permissions**

Permissions is a JSON object that you can query to determine the permissions associated with an item. The properties return a Boolean value.

<table>
<thead>
<tr>
<th>Permission Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotationDelete</td>
<td>User has delete permission for annotations on the item.</td>
</tr>
<tr>
<td>annotationRead</td>
<td>User has read permission for annotations on the item.</td>
</tr>
<tr>
<td>annotationUpdate</td>
<td>User has update permission for annotations on the item.</td>
</tr>
<tr>
<td>annotationWrite</td>
<td>User has write permission for annotations on the item.</td>
</tr>
<tr>
<td>directShareDelete</td>
<td>User has delete permission for the item shared with the user.</td>
</tr>
<tr>
<td>directShareRead</td>
<td>User has read permission for the item shared with the user.</td>
</tr>
<tr>
<td>directShareUpdate</td>
<td>User has update permission for the item shared with the user.</td>
</tr>
<tr>
<td>directShareWrite</td>
<td>User has write permission for the item shared with user.</td>
</tr>
<tr>
<td>fileDelete</td>
<td>User has delete permission for the file.</td>
</tr>
<tr>
<td>filePreview</td>
<td>User has preview permission for the file.</td>
</tr>
<tr>
<td>fileRead</td>
<td>User has read permission for the file.</td>
</tr>
<tr>
<td>fileUpdate</td>
<td>User has update permission for the file.</td>
</tr>
<tr>
<td>fileWrite</td>
<td>User has write permission for the file.</td>
</tr>
<tr>
<td>folderDelete</td>
<td>User has delete permission for the folder.</td>
</tr>
<tr>
<td>folderPreview</td>
<td>User has preview permission for the folder.</td>
</tr>
<tr>
<td>folderRead</td>
<td>User has read permission for the folder.</td>
</tr>
<tr>
<td>folderUpdate</td>
<td>User has update permission for the folder.</td>
</tr>
<tr>
<td>folderWrite</td>
<td>User has write permission for the folder.</td>
</tr>
<tr>
<td>linkShareDelete</td>
<td>User has delete permission for the item shared with the user through a link.</td>
</tr>
<tr>
<td>linkShareRead</td>
<td>User has read permission for the item shared with the user through a link.</td>
</tr>
<tr>
<td>linkShareUpdate</td>
<td>User has update permission for the item shared with the user through a link.</td>
</tr>
<tr>
<td>linkShareWrite</td>
<td>User has write permission for the item shared with the user through a link.</td>
</tr>
</tbody>
</table>

**Item State**

State is a JSON object that you can query to determine certain states of an item. The properties return a Boolean value.

<table>
<thead>
<tr>
<th>State Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAnnotated</td>
<td>The item has annotations.</td>
</tr>
</tbody>
</table>
### State Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAnnotatedLatest</td>
<td>The item has the most recent annotations.</td>
</tr>
<tr>
<td>isLinked</td>
<td>The item has named sharing links defined for it.</td>
</tr>
<tr>
<td>isShared</td>
<td>The item is directly shared with specified users.</td>
</tr>
<tr>
<td>isSyncd</td>
<td>The item is included in the content synced through the desktop client.</td>
</tr>
<tr>
<td>isInTrash</td>
<td>The item is currently in the trash.</td>
</tr>
</tbody>
</table>

### User Variables

The following variables are relative to the current user or session, or both.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>User’s GUID</td>
</tr>
<tr>
<td>name</td>
<td>User’s name</td>
</tr>
<tr>
<td>loginname</td>
<td>User’s sign in name</td>
</tr>
<tr>
<td>email</td>
<td>User’s email address</td>
</tr>
<tr>
<td>timezone</td>
<td>User’s time zone</td>
</tr>
<tr>
<td>ismember</td>
<td>Returns <code>true</code> if the user is not accessing an item through public link or app link and <code>false</code> otherwise</td>
</tr>
<tr>
<td>ispubliclink</td>
<td>Returns <code>true</code> if the user is accessing the item through a public link and <code>false</code> otherwise</td>
</tr>
<tr>
<td>isapplink</td>
<td>Returns <code>true</code> if the user is accessing the item through an applink and <code>false</code> otherwise</td>
</tr>
<tr>
<td>role</td>
<td>Current user’s role for an item</td>
</tr>
<tr>
<td>hasprivilegesas</td>
<td>Specified with a user role as a string argument to determine if the user has the specified role</td>
</tr>
</tbody>
</table>

### API Functions

Use the following utility functions with `evaluate` or `invoke` expressions to return information about a specified item.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isReservedByAnotherUser(item)</td>
<td>Returns <code>true</code> if the item is reserved by another user, or returns <code>false</code> otherwise</td>
</tr>
<tr>
<td>hasPrivilegesAs(item, role)</td>
<td>Returns <code>true</code> if the user has role privileges for the item, or returns <code>false</code> otherwise</td>
</tr>
<tr>
<td>getUserRole(item)</td>
<td>Returns the user’s role for the item</td>
</tr>
</tbody>
</table>

### Localization

Use the `stringsMap` element to provide localized versions of labels and descriptions used in the integration.
The following example shows sample application and action names and descriptions in English and German.

"stringsMap": {
    "en": {
        "APPLICATION_NAME": "Localization Test",
        "APPLICATION_DESCRIPTION": "This example provides translated content.",
        "ACTION1": "Action1",
        "ACTION2": "Action2",
        "ACTION1DESC": "Action1 description",
        "ACTION2DESC": "Action2 description"
    },
    "de": {
        "APPLICATION_NAME": "Lokalisierung Beispiel",
        "APPLICATION_DESCRIPTION": "Dieses Beispiel liefert übersetzten Inhalte.",
        "ACTION1": "Aktion1",
        "ACTION2": "Aktion2",
        "ACTION1DESC": "Aktion1 Beschreibung",
        "ACTION2DESC": "Aktion2 Beschreibung"
    }
}
REST application programming interfaces (APIs) are available in Oracle Content and Experience Cloud for content delivery and for management of content, conversations, documents, and users and groups.

The following topics provide overview information, links to reference information, and descriptions of how to perform some tasks with REST APIs:

- Overview of the REST API for Content Delivery
- Overview of the REST API for Content Management
- Overview of the REST API for Conversations
- Overview of the REST API for Documents
- Overview of the REST API for Users and Groups
- Download the Swagger File for a REST API
- Upload a REST API Swagger File into Mobile Cloud Service
- Search with the Querytext Parameter
- Set Up Searches on Metadata Fields
- Create and Use Applinks for File and Folder Access
- Provide Access to Files and Folders with Public Links

Overview of the REST API for Content Delivery

You can use the Oracle Cloud REST API for Content Delivery to interact with the contents stored in Content and Experience Cloud sites.

The REST API for Content Delivery has the following resources:

- Item Resource
  Use the Item resource to get published items, previews of items, or item metadata.

- Items Resource
  Use the Items resource to search published items or get the metadata catalog of published items.

- Renditions Resource
  Use the Renditions resource to get digital assets, published renditions of digital assets, metadata for digital assets or renditions, or information about published assets or renditions.

- Version Catalog Resource
  Use the Version Catalog resource to get information about APIs, API versions, or API metadata.
For API descriptions, examples, and information about getting started, see REST API for Content Delivery.

Overview of the REST API for Content Management

You can use the Oracle Cloud REST API for Content Management to manage assets in Oracle Content and Experience Cloud. Assets include content items as well as digital assets and their renditions.

The REST API for Content Management has the following resources to manage your content:

• Channels
  Use the Channels resource to create, delete, read, or update a channel or to list all channels.

• Collections
  Use the Collections resource to create, delete, read, or update a collection or to list all collections in a repository.

• Digital Item Renditions
  Use the Digital Item Renditions resource to get a digital item native file with or without a file name or a rendition of a digital item with or without a file name.

• Item Variations
  Use the Item Variations resource to list all item variations of a variation type, read an item variation of a variation type value, read item variations by variation type, or update the master item of an item variations set.

• Items
  Use the Items resource to create, delete, read, or update an item; to list channels, collections, relationships, variations, or all time zones of an item; or to read publish information for an item.

• Items Bulk Operations
  Use the Items Bulk Operations resource to perform bulk items operations or read items bulk operations status.

• Items Search
  Use the Items Search resource to manage items search queries.

• Languages
  Use the Languages resource to list the names of all known language codes.

• Localization Policies
  Use the Localization Policies resource to create, delete, read, or update a localization policy or to list all localization policies.

• Repositories
  Use the Repositories resource to create, delete, read, or update a repository or to list all repositories.

• Taxonomies
With the Taxonomies resource, you can take the following actions to manage your content:

– Create, update, or delete a taxonomy.
– Create, update, or delete a category in a taxonomy
– List all taxonomies or list all categories in a taxonomy
– Promote a taxonomy
– Create a new draft taxonomy

After taxonomies are promoted, they can be assigned to repositories. After they are assigned to repositories, users can apply categories to assets.

• Tokens
Use the Tokens resource to read a Cross-Site Request Forgery (CSRF) valid token.

• Types
Use the Types resource to create, delete, read, or update a type or to list all types or data types.

For API descriptions, examples, and information about getting started, see REST API for Content Management.

Overview of the REST API for Conversations

You can use the Oracle Cloud REST API for Conversations to create and manage conversations in your cloud resources that enable real-time collaboration between individuals and teams and connect your business processes, enterprise applications, and content.

This API uses the JSON data format and relies on a REST-based architectural style that provides a convenient and consistent approach to requesting and modifying data. The client specifies an action using an HTTP verb such as POST, GET, PUT, or DELETE. It specifies a resource using a unique URI in this format:

https://service-tenant.host[:port]/instance/social/api/version/resourcePath

The following topics provide overview information about developing with the REST API for Conversations:

• REST API Features for Conversations
• Collaboration Resource
• Configuration Resource for Conversations
• Security Resource
• Social Resource

For API endpoint descriptions, examples, and information about getting started, see REST API for Conversations.
REST API Features for Conversations

Features of the REST API for Conversations help you develop custom integrations with conversation objects in Oracle Content and Experience Cloud.

- API Security
- API Versioning
- Case Sensitivity
- HTTP Methods
- Optional Fields and Default Values for JSON Data Representations
- Return Values
- Name Patterns
- URL Encoding
- Pagination
- Filters
- Sort Order
- Localization
- Name Field Length Limit
- HTTP Status Codes and Error Handling

API Security

All API invocations happen in the security context of the signed-in user.

Agent users can do actions on the system, either independently or on behalf of a particular user. In specific situations you might need to use one of the following properties. Among agent user techniques, as a best practice you should use Impersonate. Use On Behalf Of or Bypass only on the advice of Oracle support or the Oracle integration team.

- **Impersonate** (in XV1ConnectionUpdateInfo)
  
  During the connection to the server, the impersonation sign in is recorded. The server also tracks when the impersonation ends. The audit log tracks the actions as being done by the user being impersonated; it does not track the impersonating agent for each action.

  The permissions for actions are based on the permission level of the user being impersonated. **Impersonate** is intended for use only in automated processes, such as for conversation object updates where you want to show that the update was made by a particular user. Keep in mind that using **Impersonate** can degrade analytics accuracy because it associates actions with the impersonated user.

- **On Behalf Of** (in XV1ConnectionUpdateInfo) - Allows a user to do a limited number of end-user functions on behalf of another user, such as posting a message to a Conversation or uploading a document. When this privilege is used, the audit log indicates both the user who performed the action and the user on whose behalf it was performed; in clients, the action is labeled as having been performed by the user the action was performed for.
• Bypass (in XV1SocialObjectRole, XV1ConversationRole, and XV1SocialObjectRole) - Allows the user to do any action without failing due to inadequate permissions (although a request might fail for another reason, such as not being a member of the Conversation being acted on). The Bypass privilege is intended for use only in certain automated processes in a highly secure environment.

API Versioning

The current version of the REST API for Conversations is v1. API versions are not tied to releases of Oracle Content and Experience Cloud.

The version number is placed in the URL directly following /social/api. For example:

https://<host>[:<port>]/<instance>/social/api/v1

A GET request to the API endpoint returns a list of the top-level URIs that the server supports. For example, this request:

https://<host>[:<port>]/<instance>/social/api/

Returns a list of supported versions, including the

VCurrent

identifier, which identifies the most recent release of the server API. The current release is "V1". The result follows:

{"V1" : "https://<instance>/api/v1",
}

Case Sensitivity

Case is important throughout the REST API because URLs, methods, names, and object IDs are all case sensitive.

Case is significant in the REST API URLs.

HTTP Methods

The REST API for Conversations supports standard HTTP methods (verbs).

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve or query resources. If you send a GET request for a specific field's value but that field does not exist, the API returns the field that you passed, with a null value and a status of 200.</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new instance of a resource, such as a conversation. In addition, you can use POST to create a relationship between two objects. For example, you can use POST to create a Like (Starred) relationship between a user and a conversation.</td>
</tr>
</tbody>
</table>
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td>Update the state of an existing instance of a resource. For example, you can use PUT to change the state of a conversation from Open to Closed. PUT requests use the XUpdater class.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete an existing instance of a resource or delete a relationship between two objects.</td>
</tr>
</tbody>
</table>

#### Subset of HTTP Methods Supported by Client

If the client supports only a subset of these HTTP methods, you can tunnel the other methods using the POST method by specifying the desired method in the method query parameter. For example, the DELETE operation can be achieved by doing a POST on a resource with "method=delete" in the query string portion of the URL.

#### PUT Requests with XUpdater

When updating an object using a PUT request, such as with `XConversationUpdater`, you can use `XUpdater` to specify the value that you want to change. `XUpdater` is a class that represents the DTO for updating objects; the `updater` field actually represents a map of field values. For example, `XUpdater` can be represented as follows:

```json
{
    "updater": "XUpdater"
}
```

For example, you can use a PUT request to `.../social/api/v1/conversations/ObjectID` with the request payload `{"updater":{"SHORT_NAME":"String for short name"}}` to change the short name.

`XUpdater` is used with Collections, Conversations, Groups, People, and Collaboration objects.

#### Optional Fields and Default Values for JSON Data Representations

The API reference documentation uses the term optional in describing JSON data representation fields.

Optional has a slightly different meaning depending on the data type:

- String: default is null if not specified.
- int: default is zero (0) if not specified.
- boolean: default is false if not specified.

#### Return Values

You can control the response payload's content type by specifying the JSON or HTML MIME type in the "Accept" HTTP header.

The content type defaults to 'application/json'. Examples are presented in JSON.

You can specify the value 'text/html' to get an HTML version of the JSON that is returned.
Name Patterns

All external class names begin with the letter X, followed by the version number, followed by a noun (for example, Conversation, Message, or People), followed by one or more modifiers, depending on the function of the class.

The following table describes the class name conventions and the associated functions, using the Conversation resource type as an example.

<table>
<thead>
<tr>
<th>X1Conversation...</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>Get information about a Conversation (GET).</td>
</tr>
<tr>
<td>CreateInfo</td>
<td>Create a Conversation and set properties (POST).</td>
</tr>
<tr>
<td>FilterInfo</td>
<td>Get Conversations that meet the filter criteria (GET).</td>
</tr>
<tr>
<td>MemberCreateInfo</td>
<td>Add a member to a Conversation and set properties (POST).</td>
</tr>
<tr>
<td>MemberInfo</td>
<td>Get information about a member of a Conversation (GET).</td>
</tr>
<tr>
<td>MessageCreateInfo</td>
<td>Add a message in a Conversation (POST).</td>
</tr>
<tr>
<td>PropertyCreateInfo</td>
<td>Set properties for a Conversation (POST).</td>
</tr>
<tr>
<td>PropertyInfo</td>
<td>Get a property for an existing Conversation (GET).</td>
</tr>
<tr>
<td>PropertyUpdateInfo</td>
<td>Set a property for a Conversation (PUT).</td>
</tr>
<tr>
<td>UpdateInfo</td>
<td>Update attributes for a Conversation (PUT).</td>
</tr>
<tr>
<td>ListInfo</td>
<td>Get a list of all Conversations (GET).</td>
</tr>
<tr>
<td>MemberListInfo</td>
<td>Get a list of all direct members of the Conversation (GET).</td>
</tr>
<tr>
<td>PropertyListInfo</td>
<td>Get a list of all properties for the Conversation (GET).</td>
</tr>
<tr>
<td>StarListInfo</td>
<td>Get a list of starred Conversations and their messages (GET).</td>
</tr>
</tbody>
</table>

XObjectID

Each object in the REST API for Conversations has a unique ID. From that ID, you can load the object, if you have sufficient permissions. Many data transfer objects (DTOs) specify IDs of other objects related to the current object. Some examples include ParentID, ModifiedByUserID, CreatedByUserID, and ConversationID. These are all links or references to other objects in the system.

Note that the ID parameter represents a name for fields, an ObjectID or an External ID for Conversation objects, and an Object ID or sign in name for users.

URL Encoding

For readability, examples might be shown in literal text, but your requests should always be URL-encoded (percent-encoded).

For example:

```
/conversations?filter={"LimitToStarred":true,"ExcludeMuted":true}
```

This example should be URL-encoded as follows:

```
/conversations?filter=%7B%22LimitToStarred%22%3Atrue%2C%22ExcludeMuted%22%3Atrue%7D
```
Pagination

Pagination capabilities are supported by default on all GET operations that have a response payload of XV1*ListInfo (for example, XV1ConversationListInfo), unless otherwise specified.

You can use offset and count in the query argument, where the default values are 0 and 20 respectively. For example:


This request gets the first 25 messages, returning them with the newest message first. You can send additional requests to get incrementally older posts in the Conversation.

Filters

Filter capabilities are supported where specified.

To use a filter, specify the filter query argument in JSON format, using URL-encoding as appropriate. For example, this request returns starred Conversations but no muted Conversations:

/conversations?filter={"LimitToStarred":true,"ExcludeMuted":true}

Sort Order

The currently assigned sort order determines the order in which results are returned.

You can change the sort order using the SortField field, which is available in XV1ConversationFilterInfo, XV1SocialObjectFilterInfo, and others. For example, this request returns results sorted by Conversation age, with the newest Conversation first:

/conversations?filter={"LimitToStarred":true,"ExcludeMuted":true,"SortField":CONVERSATION_IS_NEW}

Localization

This background information about localization might be useful as you are implementing or troubleshooting an Oracle Content and Experience Cloud integration.

Some of the concepts described in this section go beyond the REST API.

Localization is performed according to the following steps:

1. Choose a language locale.
   a. If an HTTP request exists and has an X-Waggle-ForceLanguage header, then use the X-Waggle-ForceLanguage header value as the language locale and go to step 2.
b. If a particular user is signed in or otherwise identified, check to see if the user has set a preferred language locale in their user profile. If set, use it and go to Step 2.

A language locale from LDAP will be copied to the user profile in Oracle Content and Experience Cloud. If the LDAP realm has sync-on-authentication enabled, then for each sign in, the LDAP values will overwrite the user profile settings. However, if the user changes the value in their Oracle Content and Experience Cloud profile, it is not written back to LDAP; Oracle Content and Experience Cloud does not write any attributes to LDAP.

If the LDAP Preferred Language is not supported in Oracle Content and Experience Cloud (for example, ar), English is displayed regardless of the application default language setting.

c. If an HTTP request exists and has an Accept-Language header, use the Accept-Language header to select the language locale.

Using the Accept-Language header, perform a lookup as described in RFC-4647 section 3.4. This will result in a language locale being chosen based on an exact or partial match of the header, the available translation bundles, and the bundle of last resort.

d. If the server has a language locale set, use it.

e. Select the host machine default language locale.

2. Choose a translation bundle with the exact or longest possible partial match of the chosen language locale. If no exact or partial match exists, then choose the bundle for English.

Supported Locale Codes

The currently supported locale codes are:

- cs (Czech)
- da (Danish)
- de (German)
- en (English)
- es (Spanish)
- fi (Finnish)
- fr (French)
- fr-CA (French-Canadian)
- hu (Hungarian)
- it (Italian)
- ja (Japanese)
- ko (Korean)
- nl (Dutch)
- no (Norwegian)
- pl (Polish)
- pt-br or pt_BR (Portuguese-Brazilian)
- ro (Romanian)
- ru (Russian)
- sv (Swedish)
- tr (Turkish)
- zh-CN or zh_CN (Chinese-Simplified)
Name Field Length Limit

The Name field for objects, such as Conversation names or User names, has a length limit of 256 characters.

HTTP Status Codes and Error Handling

HTTP status codes for the Collaboration REST API follow.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Request Successful.</td>
</tr>
<tr>
<td>4xx</td>
<td>Bad requests. Invalid URI, invalid body format.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized. Should sign in or get a security token.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden. Access denied.</td>
</tr>
<tr>
<td>5xx</td>
<td>Internal server error.</td>
</tr>
</tbody>
</table>

For all responses, the status code is also available in the actual payload in the StatusCode field. For all non-HTTP Status 200 responses, a message explaining the cause of the error is available in the return value. For example:

```
{
  "StatusCode" : "401",
  "StatusCode" : "Unauthorized",
  "ResourceID" : "waggle.modules.realm.InvalidLogin",
  "ResourceMessage" : "Login required."
}
```

Exception Response Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statusCode</td>
<td>The HTTP status code.</td>
</tr>
<tr>
<td>statusMessage</td>
<td>The HTTP status message.</td>
</tr>
<tr>
<td>exceptionID</td>
<td>The exception ID.</td>
</tr>
<tr>
<td>exceptionClass</td>
<td>The exception class.</td>
</tr>
<tr>
<td>exceptionMessage</td>
<td>The exception message.</td>
</tr>
<tr>
<td>exceptionDate</td>
<td>The exception date.</td>
</tr>
</tbody>
</table>

Collaboration Resource

Use the Collaboration resource API endpoints to interact with your cloud resources that enable real-time collaboration between individuals and teams and connect your business processes, enterprise applications, and content.

With the Collaboration resource, you can manage these objects:

- Conversations
- Members of conversations (direct and indirect)
• Documents associated with conversations
• Messages and replies to messages in conversations
• Followups on messages for users
• Applinks and hybrid links for conversations
• Inbound mail tokens
• Properties and user properties of conversations
• Likes for conversations and messages
• Stars for conversations and messages

Also, you can list folder IDs that have associated conversations.

For endpoint descriptions and examples, see "Collaboration" in REST API for Conversations.

Configuration Resource for Conversations

Use the Configuration resource API endpoints to configure resources and services.

With the Configuration resource, you can manage these configuration properties:
• hasMore
• items
• nextURL
• previousURL
• total (number of available items)

For endpoint descriptions and examples, see "Configuration" in REST API for Conversations.

Security Resource

Use the Security resource API endpoints to manage connections and access.

With the Security resource, you can manage connections for users and conversations.

For endpoint descriptions and examples, see "Security" in REST API for Conversations.

Social Resource

Use the API endpoints of the Social resource to help users mark favorite conversations and documents and collaborate with each other.

With the Social resource, you can manage followups and stars for users.

To manage collaboration for other users, you need to have a dedicated "Integration User" role. See Functionality Provided Through User Roles in Administering Oracle Content and Experience Cloud.

For endpoint descriptions and examples, see "Social" in REST API for Conversations.
Overview of the REST API for Documents

You can use the Oracle Cloud REST API for Documents to create client applications that interact with folders and files stored on an Oracle Content and Experience Cloud server.

The REST API for Documents is based on the legacy REST API for Content Management 1.1. The following topics provide overview information about developing with the REST API for Documents:

- Version History
- REST API Features for Documents
- Applinks Resource
- Catalog Resource
- Collections Resource
- Configuration Resource for Documents
- Files Resource
- Folders Resource
- Metadata Collection Resource
- Publiclinks Resource
- Shares Resource
- Sites Resource
- Templates Resource
- Users Resource

For API descriptions, examples, and information about getting started, see REST API for Documents.

Version History

The REST API for Documents identifies its version by including a version number in the URI in the form of /x.y/ folders, such as /1.2/.

To use a particular implementation of the API, you must reference that version in the URI. For more information about referencing a particular version of the API, see Service Call URI and Version.

Version 1.0

Version 1.0 provides basic folder, file, resource, and version identification services and their methods.

Version 1.1

Version 1.1 adds the following major services and their methods:

- Metadata Collection Resource
A metadata collection is a simple way of managing custom metadata fields that you can then use to store and retrieve metadata values for individual folders and files.

- **Users Resource**
  Service requests and responses that referenced the simple user string in version 1.0 now reference the user object.
  These search requests were added:
  - User email search
  - Fulltext search for a folder in the user’s home directory
  - Fulltext search for a file in the user’s home directory

- **Applinks Resource**
- **Publiclinks Resource**
- **Shares Resource**
- **Collections Resource**

The Files Resource service includes these additional calls:
- Reserve File (POST)
- Unreserve File (POST)
- Get File Thumbnail (GET)
- Get File Rendition Page Count (GET)
- Generate File Renditions (POST)
- Get File Page Rendition (GET)

The Files Resource also supports additional `mimeType`, `reservedBy`, and `reservationTime` elements.

Requests were added to the Metadata Collection Resource for these tasks:
- Enable or disable a metadata collection
- Enable or disable fields in a metadata collection
- Get a metadata collection definition
- Get user metadata
- Get share metadata
- Get application links metadata
- Get public links metadata
- Get assigned metadata collections to a file showing enabled or disabled field status
- Get assigned metadata collections to a folder showing enabled or disabled field status
- Delete a metadata collection

The Folders Resource includes additional `createdby` and `modifiedby` elements in responses.

The Applinks Resource supports an additional `userLocale` element.
The Upload File Version (POST) method no longer requires the use of parentID in the multipart request.

An errorKey field was added to responses.

The catalog category was renamed metadata-catalog. The catalog name will continue to be supported as an alias for metadata-catalog.

Version 1.1 of the REST API for Content Management provides a Web Application Definition Language (WADL) definition file. Oracle JDeveloper uses this file to interface with the API. You can download the WADL file for Oracle Documents Cloud Service REST APIs from the Oracle Documents Cloud Service - Sample Code page on Oracle Technology Network.

The REST API for Content Management 1.1 is the legacy version of the REST API for Documents.

Version 1.2

Version 1.2 includes all the functionality of Version 1.1 and adds the following major services and their methods:

- Sites Resource, for creating a site from an existing site.
- Templates Resource, for creating a site from a template.

The REST API for Documents, Version 1.2, is the ongoing version of the REST API for Documents.

REST API Features for Documents

Features of the REST API for Documents apply to all services.

The following topics describe these common features:

- REST Overview
- Service Call URI and Version
- Service Request
- Service Response
- Security
- Sort Order
- Pagination
- Links Identification
- HTTP Methods
- HTTP Status Codes
- Error Codes

REST Overview

REST, or Representational State Transfer, uses basic methods such as GET, POST, PUT, and DELETE over a standard protocol, such as HTTPS, to submit requests from a client application to operate on objects stored on a server.
A request is in the form of a uniform resource identifier (URI) that identifies the resource, such as a file or folder, on which to operate and includes any parameters necessary for the operation. The resource itself is represented by a globally unique identifier (GUID).

Each request from the client to a server contains all of the information necessary to understand the request and does not rely on the server to store information about the individual request or about any relationship between requests. Session state is stored (cached) entirely on the client.

You can use the REST API for Documents to create client applications to interact with folders and files stored on an Oracle Content and Experience Cloud server.

For information about the hierarchy of the REST API objects and methods, see REST API for Documents.

Service Call URI and Version

A service call includes an HTTPS method, such as POST, GET, PUT, or DELETE, and a resource identified by a Uniform Resource Identifier (URI).

The URI is in this format:

https://{host}[:port]/serviceid/api/version/resourcePath

Note:

HTTPS is supported. HTTP is not supported.

The REST API identifies its version by including a version number in the URI in the form of /x.y/ folders, where x is incremented for subtractive changes or modifications to existing functionality and y is incremented when new features are added. For example:

https://www.example.com/documents/api/1.1

The Catalog Resource lists the currently supported API versions for the instance.

Service Request

A REST API supports a direct URL and JSON data format for service requests.

REST API for Documents includes examples of service requests.

URL Parameters

Unless otherwise noted, you can include service request parameters directly in the URL. The following example of the Query User (GET) service call includes the required info parameter in the URL itself:

https://www.example.com/documents/api/1.1/users/items?info=smith
JSON Parameters

Unless otherwise noted, you can submit service request parameters in JSON format. The following example shows a simple service call and the info parameter in a JSON-formatted request:

https://www.example.com/documents/api/1.1/users/items

```json
{
    "info": "smith"
}
```

Multipart Requests

Some service calls require a multipart request. For these types of requests, you must include the parameters in a JSON-formatted request. For an example of a multipart request, see the Examples tab of "Upload File Version" under "Files" in REST API for Documents.

Header Parameters

Some types of information, such as authentication information, are passed in the request header rather than in the body of the request. This is also true for service requests that use an application link (applink). An applink grants access to a particular file or folder resource for a particular user. The applink enables you to programmatically carry out other operations on that resource on behalf of the associated user for a period of time. A service call that uses an applink includes the applink identifier and its access token in the header.

For an example, see the applink header parameters on the Examples tab of "Refresh Applink Token" under "Applinks" in REST API for Documents.

For more information about applinks, see Applinks Resource.

Service Response

The REST API supports the JSON and XML data formats for service responses. The Download File (GET) service call returns the contents of a specified file.

You can specify the data format for the response using the Accept HTTP header. If you don't specify a data format, JSON is used by default for the response. For example:

```
Accept: application/xml
```

The service response includes a general status code and, in some cases, a fault response body to help diagnose the issue. This information is returned in the specified JSON or XML format.

Requests that do not reach a valid API endpoint, such as improperly formed requests, may return a response in a format other than JSON or XML. These types of errors typically occur during code development, not with properly tested and deployed applications.

Examples of service responses are included with the API descriptions in REST API for Documents.
Security

The REST API for Documents requires OAuth or Basic authentication.

Sort Order

For services that return items in a folder, you can use the orderby parameter to order
the results based on a specified field name and sort order (ascending or descending).

For an example of the orderby parameter in use, see the Examples tab in the
description of "Get Home Folder Contents" under "Folders" in REST API for
Documents.

Pagination

For services that return items in a folder, you can create multiple pages of results by
specifying the offset and limit parameters. By default, the response shows the first
50 items.

Services that return items in a folder also return the count field, which lists the total
number of items in the response, and the totalResults field, which lists the total
number of child items in the folder.

Links Identification

All service calls support the links=yes parameter, which returns link data that
identifies the call itself and its relationship to other services.

The links parameter provides compliance with the HATEOAS (HTML as the Engine
of Application State) standard, which uses valid HTML links returned by the server to
identify the current resource and to navigate from one resource to another.

Relationships

A service call typically returns a number of links, each with a relationship type that
identifies how the link is related to the associated call. The following table describes
the most common relationship types.

<table>
<thead>
<tr>
<th>Link Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self</td>
<td>Identifies the current service call.</td>
</tr>
<tr>
<td>Canonical</td>
<td>Identifies the preferred version of a resource call.</td>
</tr>
<tr>
<td>Child</td>
<td>For hierarchical resources, such as folders, this link identifies the call to access the child item or items of the resource identified in the current service call.</td>
</tr>
<tr>
<td>Parent</td>
<td>For hierarchical resources, such as folders, this link identifies the call to access the parent item of the resource identified in the current service call.</td>
</tr>
<tr>
<td>Prev</td>
<td>For service calls that include multiple pages of response data, this link identifies the call to access the previous page.</td>
</tr>
</tbody>
</table>
The link information returned for resource metadata includes other types of relationships, including ones that identify the type of service, such as copy, for each of the services supported for the resource. For more information, see Metadata Collection Resource.

Example

This example shows the links portion (only) for the home folder in the response to the following service call:

GET https://www.example.com/documents/api/1.1/folders/self/items?links=yes&limit=2&offset=1

The call requests a list of all the items in the home folder. Note that the call includes the links parameter and pagination parameters to help illustrate the available link relationships. Also note that, because the folder is the home folder, there is no link to represent the parent folder in the response.

"links": [
  {
    "rel": "self",
    "href": "https://www.example.com/documents/api/1.1/folders/self/items?links=yes&limit=2&offset=1"
  },
  {
    "rel": "canonical",
    "href": "https://www.example.com/documents/api/1.1/folders/self"
  },
  {
    "rel": "children",
    "href": "https://www.example.com/documents/api/1.1/folders/self/items"
  },
  {
    "rel": "prev",
    "href": "https://www.example.com/documents/api/1.1/folders/self/items?offset=0&limit=2"
  },
  {
    "rel": "next",
    "href": "https://www.example.com/documents/api/1.1/folders/self/items?offset=3&limit=2"
  }
],

HTTP Methods

The REST API for Documents supports the following standard HTTP methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request information from the specified resource.</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new instance of a resource.</td>
</tr>
<tr>
<td>PUT</td>
<td>Update an existing resource.</td>
</tr>
</tbody>
</table>
### HTTP Status Codes

A REST action returns a general status code.

For descriptions of the HTTP status codes that the REST API for Documents provides, see “Status Codes” under “Get Started” in REST API for Documents.

### Error Codes

Some status codes are accompanied by a fault response body to help you diagnose the issue.

The following table describes general error codes returned by the `errorCode` field in the fault response body for an action. The `errorMessage` field in the response provides a more detailed description of the error.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The action was successful.</td>
</tr>
<tr>
<td>-1</td>
<td>General failure. This code is returned when a more specific status code is not available.</td>
</tr>
<tr>
<td>-16</td>
<td>A folder or file with the specified name or ID does not exist in the specified location. This is typically returned as an HTTP 404 response code.</td>
</tr>
<tr>
<td>-17</td>
<td>A folder or file with the same name already exists in the target location.</td>
</tr>
<tr>
<td>-20</td>
<td>The user does not have sufficient privileges to perform the requested operation. This is typically returned as an HTTP 403 response code.</td>
</tr>
<tr>
<td>-26</td>
<td>The link is invalid or expired. This is typically returned as an HTTP 400 response code.</td>
</tr>
</tbody>
</table>

The following example shows the error code and message portion of a response:

```javascript
{
    "errorCode": "-16",
    "errorMessage": "Security validation failed. 'FABCEFGHIJKLMNOPQRSTUVWXYZ000000000010000002' does not exist. The error was caused by an internally generated issue. The error has been logged."
}
```

### Applinks Resource

Use the Applinks resource (`applinks`) to request access to a folder or file for a specified user and role.

To create an applink, the requester must have admin privileges for the folder or file. That is, the requester must be the owner or have the manager role.

You can grant the specified user any standard role except the manager or owner role.

- **Viewer**: Viewers can look at files and folders but can't change things.
- **Downloader**: Downloaders can also download files and save them to their own computers.
- **Contributor**: Contributors can also modify files, update files, upload new files, and delete files.

For an example of an applink definition, see the **Examples** tab in the description of "Create File Applink" under "Applinks" in REST API for Documents.

### Applink Encryption

All necessary resource object, user, and role information is encrypted and stored in the elements identified in the following table. These elements are returned with the applink response and are used to programmatically carry out other operations on behalf of the associated user for a defined period of time.

Service calls using an applink must include the associated `applinkID` and `accessToken` values in the request header.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>applinkID</code></td>
<td>This element uniquely identifies the resource.</td>
</tr>
<tr>
<td><code>accessToken</code></td>
<td>This element provides access to the resource and expires after 15 minutes. You can refresh the access token any number of times within the time period defined by the refresh token (24 hours).</td>
</tr>
<tr>
<td><code>refreshToken</code></td>
<td>This element enables you to request a new access token when the current access token expires. The refresh token expires after 24 hours.</td>
</tr>
</tbody>
</table>

### Locales for the `userLocale` Applink Parameter

The following table lists locale codes and the locales they represent for the `userLocale` applink parameter.

<table>
<thead>
<tr>
<th>Locale Code</th>
<th>Locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs</td>
<td>Czech</td>
</tr>
<tr>
<td>da</td>
<td>Danish</td>
</tr>
<tr>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>el</td>
<td>Greek</td>
</tr>
<tr>
<td>en-CA</td>
<td>English-CA</td>
</tr>
<tr>
<td>en-GB</td>
<td>English-UK</td>
</tr>
<tr>
<td>en-US</td>
<td>English-US</td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
</tr>
<tr>
<td>fi</td>
<td>Finnish</td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
</tr>
<tr>
<td>fr-CA</td>
<td>French-Canadian</td>
</tr>
<tr>
<td>hu</td>
<td>Hungarian</td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
</tr>
</tbody>
</table>
The Applinks resource supports the **AppLinkReady** event.

### Event Description

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppLinkReady</td>
<td>Rather than send the required token data in a URL, which would make the tokens publicly visible and store them in various caches, this message is sent from an HTML inline frame (iframe tag) that contains the embedded user interface, to notify the parent window that the inline frame is ready to receive data from the Content and Experience Cloud server.</td>
</tr>
</tbody>
</table>

### Message Handler Example

To pass the necessary keys (the short-term `dAppLinkAccessToken` and the long-term `dAppLinkRefreshToken`) into the inline frame without exposing them in a URL, use the **AppLinkReady** event. The basic steps follow:

1. The inline frame that contains the embedded user interface uses a URL without keys to make the initial call to the Documents Cloud server.
2. The server downloads JavaScript to the inline frame, which posts the **AppLinkReady** message to the parent window.
3. The parent window posts data to the inline frame that contains the keys.
4. The inline frame sends another URL to the Documents Cloud server with the short-term key in the header (not in the URL).

The following example illustrates a simple message handler. The `createAppLink` function referenced in the example is not shown but is assumed to return the necessary applink resource data:
$(document).ready(function() {
    var id = "<Folder or File ID>";
    var role = "contributor";
    var assignedUser = "<User ID>";
    // createAppLink function not shown but assumed to return appLink resource data obtained
    // from Create Folder Applink or Create File Applink service
    var appLink = createAppLink(id, role, assignedUser);
    function OnMessage (evt) {
        if (evt.data.message === 'appLinkReady') {
            var iframe= $('#applink_iframe')[0];
            var iframewindow= iframe.contentWindow ? iframe.contentWindow : iframe.contentDocument.defaultView;
            var msg = {
                message: 'setAppLinkTokens',
                appLinkRefreshToken:appLink.refreshToken,
                appLinkAccessToken:appLink.accessToken,
                appLinkRoleName:role,
                embedPreview: true
            }
            iframewindow.postMessage(msg, '*');
        }
    }
    window.addEventListener && window.addEventListener('message', OnMessage, false);
});

Applinks APIs
For REST endpoints and API descriptions, see "Applinks" in REST API for Documents.

Catalog Resource

Use the Catalog resource (metadata-catalog) to get information about what resources are available for a particular version of the REST API.

This resource queries the API Catalog to retrieve information about API versions, application links, files, folders, collections, metadata, public links, shares, users, and configuration.

Catalog APIs
For REST endpoints and API descriptions, see "Catalog" in REST API for Documents.

Collections Resource

Use the Collections API for basic collection operations.

You can create and delete collections, add assets to collections, remove assets, and retrieve a list of collections.

Collections Endpoints
For Collections REST endpoints and API descriptions, see "Collections" in REST API for Documents.
Configuration Resource for Documents

The Configuration resource for documents provides information about the Collaboration and Content Delivery Network (CDN) configurations.

The Get Collaboration Configuration endpoint retrieves the client URL, cList URL, OAuth client URL, OAuth cList URL, REST URL, and Service URL.

The Get CDN Configuration endpoint retrieves the CDN URL and description.

For REST endpoints and API descriptions, see "Configuration" in REST API for Documents.

Files Resource

Use the Files resource (files) to manage files in a file system. A file is a uniquely identified item associated with a folder (or parent). The parameters of the resource definition describe a physical file object.

Most operations, such as copy, move, and delete, are performed using only the information in the file resource definition. Actions that upload a new file or a new file revision require a multipart request that also identifies the content of the physical file object.

For an example of a file definition, see the Examples tab in the description of "Get File Information" under "Files" in REST API for Documents.

Files Endpoints

For Files REST endpoints and API descriptions, see "Files" in REST API for Documents.

Folders Resource

Use the Folders resource (folders) to manage folders in a file system. A folder is a uniquely identified item associated with a "parent" folder and optionally with folder and file "child" items. These parent and child relationships create a hierarchy similar to the familiar folder structure in a conventional file system.

For an example of a folder definition, see the "Examples" tab in the description of "Get Folder" under "Folders" in REST API for Documents.

Folders APIs

For REST endpoints and API descriptions, see "Folders" in REST API for Documents.

Metadata Collection Resource

Use the Metadata Collection resource (metadata) to manage folder and file metadata. A metadata collection is a simple way of managing custom metadata fields that you can then use to store and retrieve metadata values for individual folders and files.

Within a collection, you can create one or more fields. To assign values to the fields in a metadata collection, you must first assign the collection to a particular folder or file.
You can assign a collection to one or more folders or files, and any folder or file can have one or more collections assigned to it.

With a collection, you can add or remove fields and manage the group of fields as a whole. When you assign a collection to a folder or file, you effectively add the fields to the metadata for the folder or file. The values you assign to the fields are stored and retrieved as part of the metadata for the folder or file itself.

**Metadata Inheritance**

Custom metadata fields that you create and manage in a collection follow the same rules of inheritance that standard metadata fields follow. Metadata fields assigned to a folder are available to all folders and files in the hierarchy beneath that folder. Similarly, the values you assign to those fields are inherited by all folders and files nested beneath that folder unless explicitly defined for a nested folder or file. Metadata values specified for a folder or file replace the inherited value for that folder and, by the same rules of inheritance, for any folders or files in the hierarchy nested beneath that folder.

**Field References**

When setting or removing field values within a collection, there are two ways to reference collections and fields.

You can reference the collection and field names independently. Field references on the same service call use the specified collection implicitly:

```
/metadata?collection={collection name}&{field name}={field value}[&{field name}={field value}]
```

Alternatively, you can specify the collection and field name with each field reference:

```
/metadata?{collection name}.{field name}={field value}[&{collection name}.{field name}={field value}]
```

**Metadata Collection APIs**

For REST endpoints and API descriptions, see "Metadata Collection" in REST API for Documents.

**Publiclinks Resource**

Use the Publiclinks resource (publiclinks) to request access to a folder or file for any user, whether the user has an account or not.

You can have multiple, named links for the same resource with different access roles, expiration dates, and so on. You can also edit some of the information for an existing public link.

For an example of a publiclinks definition, see the Examples tab in the definition of "Get Public Link" under "Publiclinks" in REST API for Documents.

**Publiclinks APIs**

For REST endpoints and API descriptions, see "Publiclinks” in REST API for Documents.
 Shares Resource

Use the Shares resource (shares) to share a folder and its contents with a specified user.

For information about finding users, see “Get Users” or “Get User with Email Address” under "Users" in REST API for Documents.

When you share folders, you control the permissions each person has for the folder and its files by assigning a role to the person. The different roles determine what a person can do with a shared file.

- **Viewer**: Viewers can look at files and folders but can't change things.
- **Downloader**: Downloaders can also download files and save them to their own computers.
- **Contributor**: Contributors can also modify files, update files, upload new files, and delete files.
- **Manager**: Managers have all the privileges of the other roles and can add or remove other people as members.

Shares APIs

For REST endpoints and API descriptions, see "Shares" in REST API for Documents.

 Sites Resource

Use the Sites resource (sites) to create a Content and Experience Cloud site from another site.

The Sites API provides basic site operations.

Sites APIs

For REST endpoints and API descriptions, see "Sites" in REST API for Documents.

 Templates Resource

Use the Templates resource (templates) to create a site from a template.

The Templates API provides basic template operations.

Templates APIs

For REST endpoints and API descriptions, see "Templates" in REST API for Documents.

 Users Resource

The Users resource (users) provides basic information about users to identify them for folder sharing and file sharing.

Users APIs

For REST endpoints and API descriptions, see "Users" in REST API for Documents.
Overview of the REST API for Users and Groups

Use the REST API for Users and Groups to manage users, groups, one-on-one conversations, and pictures.

With the REST API for Users and Groups, you can manage these objects:

- Users
- Followers of users
- Properties for users
- People statistics
- Groups
- Group members
- One-on-one conversations
- Pictures

For endpoint descriptions and examples, see “People and Groups” in REST API for Users and Groups.

Download the Swagger File for a REST API

Download a REST API Swagger file for use in your development project.

The Swagger file for each REST API is part of the published REST API document. You can download it from the left Navigation tree.

For example, to download and copy the Swagger file for the REST API for Documents:

1. On docs.oracle.com, open the REST API document at REST API for Documents.
2. On the left, click the download symbol:

3. Click the Swagger button:

The text from the Swagger file for the REST API is displayed.
4. From the Edit menu, choose Select All, and then choose Copy.

5. Paste the copied text into a text file.

Upload a REST API Swagger File into Mobile Cloud Service

In Oracle Mobile Cloud Service (MCS), you can create a REST connector API to connect to the REST API for Documents. If you provide a Swagger descriptor URL, the REST Connector API wizard can examine the descriptive metadata and obtain resources and fields from it.

Only Swagger metadata in JSON format is currently supported.

A REST connector API is a configuration for communicating with a specific external service to send and receive data. REST connector APIs give you a standard way to connect a mobile app to existing REST services and at the same time benefit from the Oracle Mobile Cloud Service's built-in security, diagnostics, and analytics features.

The connector API communicates and passes information between the client and the server using the HTTPS protocol. The information passed can be in the form of XML or JSON (Javascript Object Notation). REST doesn't contain a messaging layer. It uses a set of rules to create a stateless service.

The REST Connector API wizard walks you through creating REST connector APIs, from specifying a remote service and setting security policies to testing your endpoints. Additionally, the wizard supports Swagger. When you provide a Swagger descriptor URL, the wizard introspects the endpoints available from that file. The available resources are identified and displayed. You simply select the resources of the external service.
Only standard internet access ports 80 and 443 are supported. Connection to a service can't be made using a custom port.

Use the REST Connector API wizard to quickly configure your connector API by providing a name and description, specifying timeout settings, adding rules, setting a security policy, and testing it.

Creating a connection to an existing REST service can be a simple two-step operation:

1. Name your REST connector API.
2. Provide the URL to the external service.

In the REST Connector API wizard, on the page where you would provide the URL to the Swagger file, there's a link to the internal Oracle API catalog. You can get the Swagger URL from there and paste it in the field provided for the descriptor.

You have the option of applying rules to form specific requests or responses for the data that you want to access. In addition, you have the ability to configure client-side security policies for the service that you're accessing and test and check the results of your connection.

As soon as it's created, your REST connector API is listed on the Connectors page. When at least one REST connector API exists, you'll be taken directly to the Connectors page when you click Connectors from the side menu. From the catalog, you can select the REST connector API and edit it, publish it, create a new version or update an existing version, deploy it if it has a Published state, or move it to the trash.

For more information, see "REST Connector APIs" in Developing Applications with Oracle Mobile Cloud, Enterprise.

### Search with the Querytext Parameter

You can use the `querytext` parameter of the Search Folders or Files APIs to take advantage of string search, tag search, and custom metadata field search at the same time.

The `querytext` search string is available in the Search Folders or Files and the Search Folders or Files Under Specific Folder ID endpoints, to match folder or file names and allow for tag search or custom metadata field search as well. You can use `querytext` to search an entire directory tree in your home (`self`) directory as well as shared folders.

To set up `querytext` searches with the REST APIs:

1. Create files and folders, and add tags to them for string searches.
   Tags currently support only `CONTAINS`.
   a. Plan where you will place each tag because tags are inherited from parent folders.
   b. Set tags, add tags, or remove tags with the following APIs: Set Folder Tags, Edit Folder Tags, Set File Tags, and Edit File Tags

2. Add metadata collections.
   a. As an administrator, create global collections (personal collections are not supported as indexed collections).
b. Determine which fields you will need to support the search, and call APIs in the metadata resource to index those fields.

There is a limit of 100 fields to be indexed. You cannot remove fields from the index. The search is done on Favorite shared folders first and then other shared folders, up to 100. You may want to designate some folders as Favorites before searching, to ensure better search results.

c. Only a text or string type search is currently supported. Metadata fields of a date or integer type cannot be searched as date or number.

3. Build your query.

a. Use a search strings in the `querytext` parameter of the Search Folders or Files and the Search Folders or Files Under Specific Folder ID endpoints to search your folder and file names, tags, and indexed metadata fields.

For examples of tags and custom metadata searches, see the descriptions of the endpoints in the REST API for Documents.

b. Search queries require URL encoding of the single-quotemation-mark character ('`) into %60. For example, `Collection1.field1<CONTAINS>'myValue'` turns into `Collection1.field1<CONTAINS>%60myValue%60`.

c. Start with simple queries to validate that your conditions do indeed find results.

d. You can build more complex queries by combining parentheses, `<AND>` clauses, and `<OR>` clauses.

Set Up Searches on Metadata Fields

Set up searchable metadata fields with the Metadata Collection resource of the REST API for Documents. Then you can run text searches with custom fields.

To search metadata fields:

1. Sign in to Oracle Content and Experience Cloud as an administrator and create a metadata collection.

See Metadata Collection Resource.

2. As an administrator, check which metadata fields are already searchable with the Get Searchable Metadata Fields endpoint, so you can determine how many fields you can add to the search index.

This REST API call retrieves all metadata fields currently available for searching content, The result list includes all metadata fields prefixed with their respective global metadata collection for the tenant. Each tenant is limited to 100 searchable fields.

3. Use the Set Searchable Metadata Fields endpoint of the REST API for Documents to specify metadata fields that are searchable.

After you reach 100 fields, you cannot index new fields to become searchable. Fields cannot be deleted from the search index.

For custom metadata searches, the REST API for Documents supports only text searches using `CONTAINS`. It does not support numeric or date searches. For example, custom metadata fields created through the web user interface are not searchable because they are numeric or date type fields.
For more information about these endpoints and examples of using them, see "Set Searchable Metadata Fields" and "Get Searchable Metadata Fields" under "Metadata Collection" in REST API for Documents.

Create and Use Applinks for File and Folder Access

You can control access to files and folders in Content and Experience Cloud with applink endpoints of the REST API for Documents.

The following endpoints let you control user access to files and folders:

- Create File Applink
- Create Folder Applink
- Refresh Applink Token

To create an applink for a user to access a file or folder, you must be the owner or have the manager role. Either of these roles gives you admin privileges for the file or folder.

When you create a file or folder applink for a user, you can grant the user any of these roles:

- **Viewer**: Viewers can look at files and folders, but can't change things.
- **Downloader**: Downloaders can also download files and save them to their own computers.
- **Contributor**: Contributors can also modify files, update files, upload new files, and delete files.

The response from the Create File Applink or Create Folder Applink REST API includes an applink ID, access token, and refresh token. For example:

```
{
    "appLinkID": "LDhsn4VPTsnDnKpKLF2TCKjaPkybMC6-3taYSdJAazckhezJ2H1Sjs2THOou6cCAvxcRnw5gpXcU7pIRkCMw8EToJHFww2-ptWwPGhJaiir19bal9mka14Wnwpl6auO040-gfMPvPv230tnj2W3A==",
    "accessToken": "GYrSN5zu0kOT4k60bKvdxx2-ARA546A277toE0gP2GgKk1260eCn1w-Ij",
    "appLinkUrl": "http://www.example.com/documents/embed/link/app/LDhsn4VPTsnDnKpKLF2TCKjaPkybMC6-3taYSdJAazckhezJ2H1Sjs2THOou6cCAvxcRnw5gpXcU7pIRkCMw8EToJHFww2-ptWwPGhJaiir19bal9mka14Wnwpl6auO040-gfMPvPv230tnj2W3A==/fileview/DFD11F62E911327CB1F60F6T0000000000100000001",
    "refreshToken": "Yc_b_dE8V03eDTcmcmC1gi_y3LVJTPiZOSQDhUS_VWim9E_FRpLQGtEhgxCNbKtG",
    "role": "manager",
    "id": "DFD11F62E911327CB1F60F6T0000000000100000001",
    "type": "applink",
    "errorCode": "0"
}
```

The user needs to pass the applink ID and access token parameters in the request header for each REST API operating on the folder or file or on any child folder or file.
The response does not provide a link to the file or folder because headers cannot be represented in a link.

The access token expires after 15 minutes. You can refresh the access token any number of times within the time period defined by a refreshed token (24 hours). Use the refresh token to request a new access token when the current access token expires. The refreshed token expires after 24 hours.

For more information about the applink REST APIs, see Applinks Resource and "Applinks" in REST API for Documents.

Provide Access to Files and Folders with Public Links

To give users access to files and folders, you can use public links to share the files and folders with assigned users.

You can create multiple, named links for the same file or folder resource with different access roles, expiration dates, and so on. You can also edit some of the information for an existing public link.

To provide access to a file and folder with a public link:

1. Use the Create File Public Link and Create Folder Public Link REST APIs to create the public links you need.
2. If you want to edit the available public link parameters, use Edit Public Link REST API.
3. Use the Get File Public Link or Get Folder Public Link REST API to get information about the public links available for a file or folder.

For example, the following Get Folder Public Link request specifies the folder ID F4E111D0D645CD368453C2BT0000000100000001:

```
GET .../publiclinks/folder/F4E111D0D645CD368453C2BT0000000100000001
```

If the HTTP Status Code is 200 and one or more public links are available for the specified folder, the response returns information about public links defined for the folder:

```
{
  "count": 2,
  "errorCode": "0",
  "id": "F4E111D0D645CD368453C2BT0000000100000001",
  "type": "folder",
  "items": [
    {
      "type": "publiclink",
      "linkID": "LF31C09DE51854DBBDA37A90T0000000100000001",
      "linkName": "hasSecondLink",
      "ownedBy": {
        "displayName": "User AA",
        "loginName": "userAALoginName",
        "id": "U0EAA20910FAF3052ACB79E4T00000001",
        "type": "user"
      },
      "role": "viewer",
```

Chapter 16
Provide Access to Files and Folders with Public Links

16-31
This response shows that two public links are available, one for users who are assigned the viewer role for the folder and one for users who are assigned the contributor role. You use information from the response in API headers to share the folder with the assigned users (assignedUsers value) specified in the public link.

In the example, the second link grants contributor-level access to all account holders for the Content and Experience instance that hosts the specified folder. For example:

```
@serviceinstance/documents/link/
LF5E5F73A444FFB8924EF8ACT0000000000100000001/folder/
F4E111D0D645CD368453C2BT0000000000100000001
```

4. To share a file or folder, use the information from the response in the headers of other APIs, like Share Folder or Collection.

A web client can use a URL in one of the following formats to allow file or folder visualization through a public link:

```
domain URL/documents/link/linkID/fileview/file ID

domain URL/documents/link/linkID/folder/folder ID
```

For more information about public links, see Publiclinks Resource or "Publiclinks" in REST API for Documents.
Oracle Content and Experience Cloud SDKs

JavaScript software development kit (SDK) functions to support apps are provided by Oracle Content and Experience Cloud in the Content SDK and Sites SDK.

- **Content SDK**
- **Sites SDK Reference**

**Content SDK**

The Content SDK for Oracle Content and Experience Cloud is a light-weight JavaScript wrapper that interacts with the Content REST APIs. This read-only SDK retrieves structured content, digital assets, and content layouts that are managed in Content and Experience Cloud.

The Content SDK consists of three main modules:

- **ContentSDK**: The main entry-point object. The ContentSDK object lets you create client objects to access content based on your requirements.
- **ContentDeliveryClient**: A client object that is set up to access published content items and digital assets.
- **ContentPreviewClient**: A client object that is set up to access content types, draft content items, and draft digital assets.

The Content SDK is available for download from the Oracle Content and Experience Cloud server. There is an AMD module for use via RequireJS in the Browser and a CommonJS module for use in NodeJS server-side applications:

- **Browser**: load via RequireJS - https://{server}/_sitesclouddelivery/renderer/app/sdk/js/content.min.js
- **Node**: npm install - https://{server}/_sitesclouddelivery/renderer/app/sdk/npm/contentsdk-1.1.0.tgz

See Use the REST APIs, Sites SDK, and Content Delivery SDK.

Detailed descriptions of the ContentSDK, ContentDeliveryClient, and ContentPreviewClient classes are available at Content SDK. For an example of using the Content SDK, see Develop Content Layouts Locally with Developer Cloud Service.

**Sites SDK Reference**

The Oracle Content and Experience Cloud JavaScript SDK (Sites SDK) has a set of functions that enable components to have a more integrated experience with Content and Experience Cloud.
The Sites SDK is available for download from the Oracle Content and Experience Cloud server:

http://{server}/_sitesclouddelivery/renderer/app/sdk/js/sites.min.js

The Sites Application Programming Interface (API) definitions cover all defined namespaces of the global object SitesSDK:

- SitesSDK.getProperty('componentAssets', callback)
- SitesSDK.getProperty(propertyName, callback)
- SitesSDK.getSiteProperty(propertyName, callback)
- SitesSDK.setProperty('componentAssets', [assets])
- SitesSDK.setProperty(propertyName, propertyValue)
- SitesSDK.filePicker(options, callback)
- SitesSDK.publish(messageType, payload)
- SitesSDK.subscribe(messageType, callback)
- SitesSDK.Utils.addSiteThemeDesign(cssUrl)
- SitesSDK.Utils.Logger

The main purpose of the Sites SDK is to enable users to build and manage applications. It provides several features:

- Communication across inline frame and domain boundaries
  - Enables a component to communicate with a page regardless of where the component is hosted
  - Includes participation in the page model life cycle:
    * creating the component
    * rendering the component with stored properties
    * updating the component and saving those changes
    * deleting the component from the page
  - Stores and retrieves custom settings used to configure a component for use in a page
  - Uses intercomponent communication to take part in triggers and actions

- Geometry hints to a page (doesn't apply to custom components)
  - Enables a page to dynamically resize the inline frame when told to do so by a component
  - Results in a component appearing to be part of a page, and the page moving in response to component resize events (no scroll bars)

- Component awareness of when it is called by Oracle Content and Experience Cloud
  - Communication between a component and a page contains a token signed by a private key.
  - Sites SDK enables a component to decode the token and facilitate verification of the caller.
You include Sites SDK functions by providing a known path of the script. For example:

```html
<script type="text/javascript"
src="<sdk_install_dir_path_prefix>/sites.min.js"></script>
```

The Sites SDK has these global objects:

- SitesSDK
- SitesSDK.Utils

**SitesSDK API Definitions**

The SitesSDK global object can be used in render and settings endpoints.

<table>
<thead>
<tr>
<th>Command</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>SitesSDK.getProperty(propertyName, callback)</td>
<td>Gets the value of the named property from the host site.</td>
</tr>
<tr>
<td>SitesSDK.setProperty(propertyName, propertyValue)</td>
<td>Sets the value of the named property to <code>propertyValue</code>.</td>
</tr>
<tr>
<td>SitesSDK.getSiteProperty(propertyName, callback)</td>
<td>Gets a named site property value from the host site. For example, it could fetch the current theme design that is being used by the host site.</td>
</tr>
<tr>
<td>SitesSDK.publish(messageType, payload)</td>
<td>Sends a message to the server, and accepts a <code>messageType</code> and JSON object as a payload.</td>
</tr>
<tr>
<td>SitesSDK.subscribe(messageType, callback)</td>
<td>A message listener for messages dispatched from the host site. The call is asynchronous. These are JavaScript callbacks, so the function should use JavaScript Closure or suitably bind the function to ensure it has access to the appropriate context.</td>
</tr>
</tbody>
</table>

**SitesSDK.Utils Definitions**

The SitesSDK.Utils namespace has utility functions that can be called by all endpoints of an Oracle Content and Experience Cloud component. This namespace is available for the Settings panel for a remote component, but not for the inline local component.

⚠️ **Caution:**

These functions are not officially supported by Oracle; they are intended as sample implementations. Use them at your own risk.
SitesSDK.getProperty(‘componentAssets’, callback)

This function returns a list of assets currently being stored on behalf of the component within the site.

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentAssets</td>
<td>String</td>
<td>Invokes list creator.</td>
</tr>
<tr>
<td>callback</td>
<td>Function</td>
<td>Callback function</td>
</tr>
</tbody>
</table>

### Usage

```javascript
// get/set list of assets
SitesSDK.getProperty(‘componentAssets’, callback);
SitesSDK.setProperty(‘componentAssets’, [assets]);

// invoking list creator
// after calling this, the developer needs to call setProperty against
// ‘componentAssets’ to persist all currently used assets
//
// args:
//  fileTypes - list of supported file types
//  multiSelect - single or multiple file select
//
// returns a list of selected assets. If user cancels out of the picker,
// the callback is never called
SitesSDK.filePicker({options}, callback);

//
// example 'componentAssets' JSON returned:
//
// 'componentAssets': [{
//   'name': <nameOfAssetInContentFolder>,  // this is used to uniquely
//   and persistently identify the asset. It is typically prefixed by the
//   'originalName' followed by a 16 digit string
```
SitesSDK.getProperty(propertyName, callback)

This function gets the value of the named property from the host site.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>String</td>
<td>Name of the property.</td>
</tr>
<tr>
<td>callback</td>
<td>Function</td>
<td>Callback function.</td>
</tr>
</tbody>
</table>

Related Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Settings</th>
<th>Render</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>Height of frame</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>width</td>
<td>Width of frame</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>customSettingsData</td>
<td>Settings data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>styleClass</td>
<td>Selected style class</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Any other style attribute, such as borderColor, borderRadius, alignment, borderStyle, margin, and so on

Usage

SitesSDK.getProperty('customSettingsData', function (propertyData) {
    // store settings data
    self.configuration(
        {
            'id': propertyData.id || self.id,
            'url': propertyData.url || self.params.url,
            'limit': propertyData.limit || self.params.limit
        }
    );
});

Additional Related Properties Fetched From Rendering URL

In addition to using getProperty() to fetch property values, other related properties that can be fetched from the rendering URL include these.
<table>
<thead>
<tr>
<th>PropertyName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Component Id</td>
</tr>
<tr>
<td>instance</td>
<td>Encrypted token</td>
</tr>
<tr>
<td>width</td>
<td>Width of iframe</td>
</tr>
<tr>
<td>height</td>
<td>Height of iframe</td>
</tr>
<tr>
<td>viewMode</td>
<td>Render mode values</td>
</tr>
<tr>
<td>locale</td>
<td>Language set in site or browser</td>
</tr>
<tr>
<td>settingsId</td>
<td>Current settings Id</td>
</tr>
</tbody>
</table>

Additional Properties Fetched from Settings URL

In addition to using `getProperty()` to fetch property values, other properties that can be fetched from the settings URL include these.

<table>
<thead>
<tr>
<th>PropertyName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currCompId</td>
<td>Id of the component being edited</td>
</tr>
<tr>
<td>instance</td>
<td>Encrypted token</td>
</tr>
<tr>
<td>width</td>
<td>Width of the settings iframe</td>
</tr>
<tr>
<td>locale</td>
<td>Language set in site or browser</td>
</tr>
<tr>
<td>OrigSettingsId</td>
<td>Original settings Id of the component</td>
</tr>
<tr>
<td>settingsId</td>
<td>New settings Id</td>
</tr>
</tbody>
</table>

SitesSDK.getSiteProperty(propertyName, callback)

This function gets a named site property value from the host site. An example is fetching the current theme design that is being used by the host site.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>string</td>
<td>Name of the site property</td>
</tr>
<tr>
<td>callback</td>
<td>function</td>
<td>Callback function to read the property value</td>
</tr>
</tbody>
</table>

Site Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Settings</th>
<th>Render</th>
</tr>
</thead>
<tbody>
<tr>
<td>theme</td>
<td>URL of the current theme design</td>
<td>NA</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Usage

SitesSDK.getSiteProperty('theme',function(data){
  console.log('Theme data ' + JSON.stringify(data));
  // check if we got an url back
  if ( data.url && typeof data.url === 'string' ) {
    if ( data.url !== '') {
      // theme is loaded so dynamically inject theme
      SitesSDK.Utils.addSiteThemeDesign(data.url);
    }
  }
});

SitesSDK.setProperty('componentAssets', [assets])

This function updates the list of site assets stored on behalf of the custom component.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentAssets</td>
<td>String</td>
<td>Creates list of component assets.</td>
</tr>
<tr>
<td>assets</td>
<td>JSON Object</td>
<td>Returns list of selected assets.</td>
</tr>
</tbody>
</table>

Arguments

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filetypes</td>
<td>List of supported file types.</td>
</tr>
<tr>
<td>multiSelect</td>
<td>Select single or multiple files.</td>
</tr>
</tbody>
</table>

Usage

// get/set list of assets
SitesSDK.getProperty('componentAssets', callback);
SitesSDK.setProperty('componentAssets', [assets]);

// invoking list creator
// after calling this, the developer needs to call setProperty against
// 'componentAssets' to persist all currently used assets
//
// args:
// fileTypes - list of supported file types
// multiSelect - single or multiple file select
//
// returns a list of selected assets. If user cancels out of the picker,
// the callback is never called
SitesSDK.filePicker({options}, callback);

//
// example 'componentAssets' JSON returned:
//
'componentAssets': [{
   'name': <nameOfAssetInContentFolder>,  // this is used to uniquely and persistently identify the asset. It is typically prefixed by the originalName: followed by a 16 digit string
   'originalName': <originalName>,        // name of the asset as selected from DOCS
   'description': <description>,         // description/other attributes that are available from DOCS
   'url': <fully qualified url to the asset>
}]

SitesSDK.setProperty(propertyName, propertyValue)

This function sets the value of the named property to `propertyValue`.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>String</td>
<td>Name of the property.</td>
</tr>
<tr>
<td>propertyValue</td>
<td>JSON Object</td>
<td>Value of the property.</td>
</tr>
</tbody>
</table>

**Related Properties**

This table lists the set of related properties that can be fetched using `setProperty()`.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Settings</th>
<th>Render</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>Height of frame</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>width</td>
<td>Width of frame</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>customSettingsData</td>
<td>Settings data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>styleClass</td>
<td>Selected style class</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>description</td>
<td>Used in banner for an instance of local component on the page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>layout</td>
<td>Returns the currently selected layout only for local components. Can be updated from the Settings panel based on values in the appinfo.json file.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>renderStatus</td>
<td>Allows a custom component to report that render is done.</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

Any other style attribute, such as `borderColor`, `borderRadius`, `alignment`, `borderStyle`, `margin`, and so on
Usage

// configuration that can be saved
var saveconfig = {
   id': self.appSettingsProperties['settingsId'],
   'url': self.url(),
   'limit': self.limit()
};

// save property 'customSettingsData'
SitesSDK.setProperty('customSettingsData', saveconfig);

SitesSDK.filePicker({options}, callback)

This function returns a list of selected files.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>String</td>
<td>Options to specify selected files to return.</td>
</tr>
<tr>
<td>callback</td>
<td>Function</td>
<td>Callback function. If user cancels out of the picker, the callback is never called.</td>
</tr>
</tbody>
</table>

Usage

// get/set list of assets
SitesSDK.getProperty('componentAssets', callback);
SitesSDK.setProperty('componentAssets', [assets]);

// invoking list creator
// after calling this, the developer needs to call setProperty against
// 'componentAssets' to persist all currently used assets

// args:
// fileTypes - list of supported file types
// multiSelect - single or multiple file select

// returns a list of selected assets. If user cancels out of the picker,
// the callback is never called
SitesSDK.filePicker({options}, callback);

//
// example 'componentAssets' JSON returned:
//
'componentAssets': [{
   'name': <nameOfAssetInContentFolder>, // this is used to uniquely
   and persistently identify the asset. It is typically prefixed by the
   'originalName': <originalName>, // name of the asset as
   followed by a 16 digit string
   selected from DOCS
SitesSDK.publish(messageType, payload)

This function sends a message to the server. It accepts a `messageType` and JSON object as a payload. The message type is recognized and handled in the page. If there is no handler, then the message type passed across is ignored.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageType</td>
<td>string</td>
<td>Type of message</td>
</tr>
<tr>
<td>payload</td>
<td>JSON Object</td>
<td>Message payload</td>
</tr>
</tbody>
</table>

**Message Types**

This table lists the types of messages that are known to a local (using inline frames) or remote component and can be sent using the `SitesSDK.publish()` function.

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETTINGS_UPDATED</td>
<td>Use in the Settings panel to send a Settings updated message to the rendering endpoint.</td>
</tr>
<tr>
<td>TRIGGER_ACTIONS</td>
<td>Use in the component rendering endpoint to trigger actions in the host site.</td>
</tr>
</tbody>
</table>

**Usage**

```javascript
// raise trigger
SitesSDK.publish(SitesSDK.MESSAGE_TYPES.TRIGGER_ACTIONS,{
    'triggerName': 'scsChangeSettings',
    'triggerPayload': { 'url':
        this.configuration()]['url'],
    'feedcount': this.configuration()]['limit']
});
```

SitesSDK.subscribe(messageType, callback)

This function is effectively a message listener for messages dispatched from the host site. The call is asynchronous.

The registered callback is called when a message of a specific type is received by the SDK. If the callback returns a value, then that value is returned back to the page. If no callback is passed to this function, then any registered listener for that specific message type is deleted.

These are JavaScript callbacks, so the function should use JavaScript Closure or suitably bind the function to ensure it has access to the appropriate context.
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageType</td>
<td>string</td>
<td>Type of message:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SETTINGS_UPDATED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TRIGGER_ACTIONS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EXECUTE_ACTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GET_ACTIONS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GET_TRIGGERS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• COPY_CUSTOM_DATA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PASTE_CUSTOM_DATA</td>
</tr>
<tr>
<td>callback</td>
<td>function</td>
<td>The function when a message is received from a host site.</td>
</tr>
</tbody>
</table>

Message Types

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETTINGS_UPDATED</td>
<td>Subscribe to this message to be notified when any value is changed in the Settings Panel. Typically the component will respond by updating its rendition in the user interface.</td>
</tr>
<tr>
<td>TRIGGER_ACTIONS</td>
<td>Publishing this message raises a trigger and passes payload.</td>
</tr>
<tr>
<td>EXECUTE_ACTION</td>
<td>Subscribe to this message to handle custom actions. The listener typically handles this message by performing the given action.</td>
</tr>
<tr>
<td>GET_ACTIONS</td>
<td>Subscribe to this message to declare an array of actions that the component can execute. Returns array of actions.</td>
</tr>
<tr>
<td>GET_TRIGGERS</td>
<td>Subscribe to this message to declare an array of triggers that the component can raise. Returns array of triggers.</td>
</tr>
<tr>
<td>COPY_CUSTOM_DATA</td>
<td>Subscribe to this message to handle copying Custom Settings Data to the clipboard. Returns an object representing data to be put on the clipboard.</td>
</tr>
<tr>
<td>PASTE_CUSTOM_DATA</td>
<td>Subscribe to this message to handle pasting Custom Settings Data from the clipboard.</td>
</tr>
</tbody>
</table>

Usage

Some listeners expect data to be passed, and some are expected to return data.

**Example 1:**

Register a message listener for message type `EXECUTE_ACTION`:

```javascript
SitesSDK.subscribe(SitesSDK.MESSAGE_TYPES.EXECUTE_ACTION, self.executeAction, self);
```
The registered callback function should accept an argument as shown here:

```javascript
// typical signature of a callback function registered with a message
// type
function (args) {
    var payload = args.detail.message.payload,
        action = args.detail.message.action,
        actionName = action && action.actionName;

    // do something here with the payload data
}
```

**Example 2:**
The `COPY_CUSTOM_DATA` listener is typically be implemented with code like this:

```javascript
// listen for COPY_CUSTOM_DATA request
SitesSDK.subscribe(SitesSDK.MESSAGE_TYPES.COPY_CUSTOM_DATA,

The method `copyCustomDataListener()` would return an object representing the data to be copied to the clipboard, for example:

```javascript
// Handle Copy Style (save customSettingsData to the clipboard)
self.copyCustomDataListener = function() {
    return {
        prop1: this.prop1(),
        prop2: this.prop2()
    };
};
```

**Example 3:**
The `PASTE_CUSTOM_DATA` request accepts data, but doesn’t have to return anything:

```javascript
// listen for PASTE_CUSTOM_DATA request
SitesSDK.subscribe(SitesSDK.MESSAGE_TYPES.PASTE_CUSTOM_DATA, pasteCustomDataListener);
```

Along with:

```javascript
// Handle Paste Style (apply customSettingsData from the clipboard)
self.pasteCustomDataListener = function(data) {
    ...
};
```

**SitesSDK.Utils.addSiteThemeDesign(cssUrl)**

This function creates a link element in the current page HTML `<head>` tag. The source is set to the path of `cssUrl`. 

Caution:
The functions in the Utils namespace are not officially supported by Oracle. They are intended to serve as sample implementations. Use them at your own risk.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cssUrl</td>
<td>string</td>
<td>URL path of the current theme design. Use only with remote components. Local components (realized in an iframe) must fetch properties from the page instead of the URL.</td>
</tr>
</tbody>
</table>

Usage

This function is typically used in conjunction with fetching the current theme design from the host site, as shown in this sample code:

```javascript
// fetch current theme design from host site and then add it to the page
SitesSDK.getSiteProperty('theme', function(data){
  // check if we got an url back
  if (data.url && typeof data.url === 'string') {
    if (data.url !== '') {
      // theme is loaded so dynamically inject theme
      SitesSDK.Utils.addSiteThemeDesign(data.url);
    }
  }
});
```

SitesSDK.Utils.Logger

This function returns the logger object being used by the Sites SDK.

Caution:
The functions in the Utils namespace are not officially supported by Oracle. They are intended to serve as sample implementations. Use the Utils functions at your own risk.

The log level can be set to:

- debug
- log
- info
- error
• warn
Develop Content Connectors

Oracle Content and Experience Cloud (OCE) provides a connector framework for developers to develop their custom content connectors. The connector framework is extensible, and a developer or partner can build a content connector on top of any content repository.

OCE has prebuilt content connectors for Google Drive, Microsoft OneDrive, and Dropbox. It also provides a WebCenter Content connector out of the box.

A custom content connector needs to implement the following artifacts:

- REST interfaces for defining configuration, authorization, and fetching content.
- A Content Picker UI.
  You can choose from the common UI or implement a custom UI that follows predefined interfaces.
- REST interfaces for browsing a file system and for search (required only for using the common UI).

OCE provides a reference implementation, the Pexels Connector, which uses a custom UI. The same content connector, with a few tweaks, can be converted to use the common UI provided by OCE.

The following topics describe how to develop custom content connectors for OCE to import content and assets from third-party content repositories:

- Overview of the Connector Framework
- Build a New Content Connector
- Content Connector Configuration and Registration
- Content Connector Execution Flow
- Pexels Content Connector Sample Implementation
- Download the CEC Content Connector Sample and SDK

Overview of the Connector Framework

The Oracle Content and Experience Cloud (OCE) connector framework enables you to bring content you have already created elsewhere into OCE, manage it centrally, and use it in new experiences across multiple channels.

Part of OCE, the connector framework integrates with remote content stores to let you bring content into OCE as easily as uploading files from your desktop. You can plug in any type of content store to OCE, making it a content hub.

The connector framework abstracts the functionality that is provided and implemented by end connectors. It does so by providing the REST API interface to configure a content connector as well as to call content-related APIs on the end store, like browsing the remote file system and copying the files from a remote store to OCE.
• Connector REST API Interface
• Connector SDK

Connector REST API Interface

You can deploy and run content connectors anywhere and implement them with any technology stack, as long as they can be called from Oracle Content and Experience Cloud (OCE) through REST APIs.

Content connectors are deployable binaries that use Restful services for the remote stores implementing the REST interfaces (SPIs/SDK) defined by OCE. They internally use cloud store native SDKs to connect to remote systems.

Connector SDK

To help you build a new content connector, OCE provides the Connector SDK

This SDK contains:

• A javadoc
• A JAR file containing DTO and REST interfaces
• The connector.yaml file, which documents the REST APIs and the payloads

The Connector SDK is in the cec-connector-sdk.zip file.

Build a New Content Connector

To build a new content connector as a Restful service, a developer needs to implement the Restful interfaces provided by the connector framework.

The Oracle Content and Experience Cloud (OCE) connector framework has the following interfaces.

• APIResource Interface: This interface is the starting point of the content connector. You simply return the version that's supported.
• ServerResource Interface: When an Administrator or Developer registers a content connector in the OCE server, OCE calls this service to get the basic details about the connector server, such as these:
  – Authorization type:
    * OAUTH: If the remote cloud store supports 3-legged OAuth authentication
    * BASIC_AUTH: If the remote cloud store supports Basic authentication
    * NO_AUTH: If the remote store doesn't require any authentication
  – Picker type: OCE supports the OCE generic picker to list the file system. If the content connector supports the native picker, then the content connector implementer can implement JavaScript APIs to integrate that picker with the OCE UI. The picker type can be one of three values:
    1. COMMON, using the common UI
    2. CUSTOM, using a custom UI
    3. NATIVE, using a native UI like Google Drive or Microsoft OneDrive
– Also use this Interface to define custom fields for the content connector. Custom properties are connector specific, so the connector framework can’t provide such a properties list by itself. Every content connector has its own requirement to connect to a remote store; for example, one content connector might need just ClientID and ClientSecret, while another might require ClientID, ClientSecret, AppID, and so on. Each content connector can provide a custom properties list to the OCE server by ServerResource. If any of these properties need to be filled in by an administrator during configuration, the connector framework will surface them in the administration UI.

This service will be called only once, at the time of registration.

– AuthorizationResource Interface: Used to complete the authorization before content from the remote store can be copied to the OCE repository. This interface has support for both OAuth and Basic authentication protocols.

  – 3-legged OAuth authentication
    * In the OAuth flow, the OCE UI, based on the custom property value provided by the content connector, supports OAuth authentication. It triggers an OAuth flow by calling the connector framework, which in turn calls the /authorization/authorization URLs service on the content connector to get the authorization URL.
    * The OCE UI renders the authorization URL in a browser, where the user authorizes the application and gives OCE permission to access the token for further communication. To complete the authorization and get the OAuth token, the OCE server will call the /authorization/completedAuthorizations service on the content connector. To complete 3-legged OAuth, the OCE server also provides the redirect callback servlet, which the remote store will call to complete the authorization process.

  – Basic authentication: In case the content connector supports Basic authentication, the OCE UI will prompt the user to enter a user name and password, which it will pass to the OCE server (connector framework). The connector framework will call the /authorization/basicAuthorization service to pass the credentials and will expect the content connector to validate with the remote store if the credentials are correct.

  • FileSystemResource Interface: The connector framework supports the generic file picker, which can be used by content connectors to render the file-system information for a user to browse the files. But, the content connector needs to implement the /filesystem service to use this functionality. Highlights of this service follow:
    – The user can browse the file system like any other file picker; for example, the Google Drive or Microsoft OneDrive picker.
    – This supports pagination.
    – The user can also specify the search criteria to get the filtered result.

  • ContentResource Interface: The connector framework uses the service /content to get the actual content (the bytes of the files) stored in the cloud store. This service returns javax.ws.rs.core.Response as a response, and the response entity is the content of the file. The response should include the following headers:
    – Content-Length: Set to the entity length, or -1 if unknown.
Content-Type: Set to the content type, or application/octet-stream if a more accurate type is not possible.

Content-Disposition: Set to the disposition type of attachment, and includes a filename parameter (for example, Content-Disposition: attachment; filename="meeting agenda.doc"). Note that file names with spaces must be in quotation marks. See RFC 6266 for details about the format and encoding of the header.

Each of these interfaces is described in more detail, with examples of REST payloads, in the following sections:

- REST Interfaces for Configuration, Authorization, and Fetching Content
- REST Interfaces for File System Browsing and Searching
- Content Picker
- Authorization

REST Interfaces for Configuration, Authorization, and Fetching Content

A content connector needs to implement the following REST APIs for defining the connector configuration, setting up authorization, and fetching content.

/rest/api

Implements intradoc.connectorcommon.server.APIResource

Here you return the latest version supported by the content connector.

GET http://host:port/connector/rest/api

[   "v1"
]

/rest/api/v1/server

Implements intradoc.connectorcommon.server.ServerResource

This returns information about the content connector configuration, like the authentication type, picker type, and custom fields it exposes.

GET http://host:port/connector/rest/api/v1/server

{   "name": "Pexels Connector",
    "nameLocalizations": [
        {
            "locale": "en",
            "localizedString": "Pexels Connector"
        }
    ],
    "version": " (, , )",
    "about": "Pexels Connector.<br>Copyright (c) 2019, Oracle and/or its affiliates. All rights reserved.",
    "aboutLocalizations": [}
Chapter 18
Build a New Content Connector

{ "locale": "en",
 "localizedString": "Pexels Connector.<br>Copyright (c) 2019, Oracle and/or its affiliates. All rights reserved." }
"authenticationType": "NO_AUTH",
"pickerType": "CUSTOM",
"enableMultiUserCopyBack": false,
"maxUploadSize": 1073741824,
"fields": [
{
 "ID": "ProxyHost",
 "datatype": "STRING",
 "siteSettable": true,
 "userSettable": false,
 "connectorSettable": false,
 "authorizationURLParameter": false,
 "label": "HTTP Proxy Hostname",
 "labelLocalizations": [
 { "locale": "en",
 "localizedString": "HTTP Proxy Hostname"
 }
 ],
 "description": "The HTTP proxy hostname, leave blank to disable.",
 "descriptionLocalizations": [
 { "locale": "en",
 "localizedString": "The HTTP proxy hostname, leave blank to disable." }]
 },
 "required": false
 },
{ "ID": "ProxyPort",
 "datatype": "STRING",
 "siteSettable": true,
 "userSettable": false,
 "connectorSettable": false,
 "authorizationURLParameter": false,
 "label": "HTTP Proxy Port",
 "labelLocalizations": [
 { "locale": "en",
 "localizedString": "HTTP Proxy Port"
 }
 ],
 "description": "The HTTP proxy port number, leave blank to default to port 80.",
 "descriptionLocalizations": [
 { "locale": "en",
 "localizedString": "HTTP Proxy Port"
 }
 ]}
"localizedString": "The HTTP proxy port number, leave blank to default to port 80."
},
"required": false
},
{
"ID": "ProxyScheme",
"datatype": "STRING",
"siteSettable": true,
"userSettable": false,
"connectorSettable": false,
"authorizationURLParameter": false,
"label": "HTTP Proxy Scheme",
"labelLocalizations": [
{
   "locale": "en",
   "localizedString": "HTTP Proxy Scheme"
}
],
"description": "The HTTP proxy scheme, leave blank to default to http.",
"descriptionLocalizations": [
{
   "locale": "en",
   "localizedString": "The HTTP proxy scheme, leave blank to default to http."
}
],
"required": false
},
{
"ID": "ClientID",
"datatype": "STRING",
"siteSettable": true,
"userSettable": false,
"connectorSettable": false,
"authorizationURLParameter": false,
"label": "Client ID",
"labelLocalizations": [
{
   "locale": "en",
   "localizedString": "Client ID"
}
],
"description": null,
"descriptionLocalizations": [],
"required": true
},
"supportedConnectorTypes": [
   "COPY"
],
"proprietorName": "",
"serviceProviderName": "Pexels"
"nativeAppInfos": null
}

/rest/api/v1/authorization/authorizationURLs
Implements intradoc.connectorcommon.server.AuthorizationResource
This is required only if the content connector supports OAuth.
It returns the authorization URL to which the browser will redirect to invoke the OAuth flow where the user provides credentials and authorizes the access. The redirect URL passed in the payload is what the OAuth provider will redirect to with temporary code.

POST http://host:port/connector/rest/api/v1/authorization/authorizationURLs

Headers
Content-Type:application/json
X-CEC-ClientID: client-id
X-CEC-ClientSecret: client-secret
X-CEC-ProxyHost: proxy-host
X-CEC-ProxyPort: 80
X-CEC-ProxyScheme: http

Payload
{"redirectURL":"http://host:port/documents/web/AR_COMPLETE_AUTHORIZATION"}

Response
{
   "fieldValueMap": null
}

/rest/api/v1/authorization/completedAuthorizations
Implements intradoc.connectorcommon.server.AuthorizationResource
This is required only if the content connector supports OAuth.
This is called to complete the second part of the OAuth flow, where the code obtained from the OAuth provider in the previous step is passed along with the client ID and secret to obtain the access token and refresh token along with expiry times. This information is then returned to OCE, which stores it securely against the signed-in user.

POST http://host:port/connector/rest/api/v1/authorization/completedAuthorizations

Headers
Content-Type:application/json
X-CEC-ClientID: client-id
X-CEC-ClientSecret: client-secret
X-CEC-code: code
X-CEC-ProxyHost: proxy-host
X-CEC-ProxyPort: 80
Payload
{"redirectURL":"http://host:port/documents/web/AR_COMPLETE_AUTHORIZATION"}

Response
{
   "authorized": true,
   "authorizedUserDisplayName": null,
   "authorizedUserEmailAddress": null,
   "authorizedUserPictureURL": null,
   "fieldValueMap": {
      "RefreshToken": "refresh-token",
      "AccessToken": "access-token"
   }
}

/rest/api/v1/authorization/basicAuthorization

Implements intradoc.connectorcommon.server.AuthorizationResource

This is required only if the content connector supports BASIC authorization. Here the sign in credentials are passed in the headers where the password field is base64 encoded. It is always recommended that a content connector be deployed on an SSL endpoint.

POST http://host:port/connector/rest/api/v1/authorization/basicAuthorization

Headers
Content-Type: application/json
X-CEC-UserName: user
X-CEC-UserPwd: password
X-CEC-ProxyHost: proxy-host
X-CEC-ProxyPort: 80
X-CEC-ProxyScheme: http

Response
true

/rest/api/v1/content

Implements intradoc.connectorcommon.server.ContentResource

Given a file ID, this return the input stream for a file.

GET http://host:port/connector/rest/api/v1/content?uri=fFileGUID:xxxx

Request Headers
X-CEC-ClientID: client-id
X-CEC-ProxyHost: proxy-host
X-CEC-ProxyPort: 80
X-CEC-ProxyScheme: http
REST Interfaces for File System Browsing and Searching

If a content connector uses the common picker available out of the box with OCE, then the content connector needs to implement the following REST API.

This REST API is required for the Common UI.

/api/v1/filesystem

Implements intradoc.connectorcommon.server.FilesystemResource

This interface supports search as well as where queryText is passed.

POST http://host:port/connector/rest/api/v1/filesystem

Request Headers
X-CEC-ClientID: client-id
X-CEC-ProxyHost: proxy
X-CEC-ProxyPort: 80
X-CEC-ProxyScheme: http
Content-Type: application/json

Request Body
{
  "itemCount": "50",
  "itemStartRow": "0",
  "itemsSortField": "ASC",
  "queryText": "animals"
}

Response
{
  "numItems": 50,
  "hasMoreItems": true,
  "totalItemsCount": 7190,
  "fileSystemInfo": {
    "uri": "fFolderGUID: null",
    "parentUri": "fFolderGUID: null",
    "name": "/",
    "description": null,
    "isDirectory": true,
    "size": null,
    "mimeType": null,
    "extension": null,
    "creator": null,
    "createdTimeStamp": null,
    "lastModifiedBy": null,
    "lastModifiedTimeStamp": null,
    }
Chapter 18
Build a New Content Connector
Content Picker

Oracle Content and Experience Cloud (OCE) supports two mechanisms for integration so users can browse and select content and assets to bring into OCE.

The content picker for a content connector can be either the common content picker or a custom one:

- The common content picker is provided by OCE.
  With the generic common picker, OCE provides a basic framework and capabilities to build content browsing and selection. This works seamlessly in conjunction with the Connector Framework SDK and the APIs defined.
- The custom picker needs to be built by the content connector and can be built using any JS technology.
  The custom picker is typically used in cases where the content connector builds its own experience for accessing content in the remote store. Some stores have their own native pickers, which the content connector could reuse for integrating with the asset repository; for example, Google Drive, Microsoft OneDrive, and Dropbox. Additionally, the content connector can develop its own picker UI following the predefined interfaces and callback mechanisms prescribed by OCE.

Use the Common UI

If the content connector uses the common picker, then it needs to define pickerType in the eserver implementation:

```java
intradoc.connectorcommon.server.ServerResource implementation

serverInfo.pickerType = PickerType.COMMON;
```

The content connector also needs to implement the intradoc.connectorcommon.server.FilesystemResource interface. The implementation needs to adhere to following guidelines in its filesystem REST implementation:

1. The content connector must append "fFolderGUID:" in uri, parentUri for each folder in the response of FileSystem.
2. A few fields are mandatory for every folder and file to be set in FileSystemResource for the common UI to work without any issues: uri, parentUri, and name. If parentUri is not applicable, then it can be set to fFolderGUID: null.
3. You must know the size of the result set being returned up front; that is, totalItemsCount needs to be set.
Use a Custom UI

A content connector can implement a custom UI for its picker. This can be built using any JS technology. However, it needs to invoke some predefined methods in OCE to obtain connector configuration and to pass back selections. The custom picker can be built to expose its own **OK** and **Cancel** buttons, in which case it needs to invoke the appropriate handlers. If the picker doesn’t expose its own **OK** and **Cancel** buttons, OCE will provide this.

The **pickerType** needs to be defined as **CUSTOM** in the server implementation:

```javascript
intradoc.connectorcommon.server.ServerResource implementation

serverInfo.pickerType = PickerType.CUSTOM;
```

The custom implementation needs to include the following JS file in its library:

**oracle-oce-custompicker.js**

```javascript
/*global window:true*/
(function() {
  'use strict';

  var name = window.name; // name set on iframe element to uniquely identify the instance
  var receiver = window.opener || window.parent;
  var origin = "*";
  function _postMessage(msg) {
    msg.name = name;
    receiver.postMessage(msg, origin);
  }

  var namespace = "OCE"; // todo: allow passing in data attribute
  if (!window[namespace]) {
    window[namespace] = {};
  }
  var OCE = window[namespace];

  if (!OCE.CustomPicker) {

    OCE.CustomPicker = {

      selection: [],

      addItem: function(id, name, type, size) {
        var item = {id: id, name: name, type: type, size: size};
        this.selection.push(item);
        this.onChange();
      },

      removeItem: function(id) {
        this.selection = this.selection.filter(function(item) {
          return item.id !== id;
        });
      }
    };

```

Chapter 18
Build a New Content Connector
18-12
this.onChange();
},

onInit: function(cbInit) {
  var msg = {
    message: 'init',
    needAuthToken: true
  };
  _postMessage(msg);
  this.messageListener = this.onPostMessage.bind(this);
  window.addEventListener('message', this.messageListener, false);
  this.cbInit = cbInit;
},

onClose: function() {
  window.removeEventListener('message', this.messageListener, false);
},

onPostMessage: function(event) {
  if (event.data && event.data.message === "init") {
    if (this.cbInit) {
      this.cbInit(event.data);
    }
  }
},

onChange: function() {
  var msg = {
    message: 'change',
    selection: this.selection
  };
  _postMessage(msg);
},

onOk: function(selection) {
  selection = selection || this.selection;
  var msg = {
    message: 'ok',
    selection: selection
  };
  _postMessage(msg);
},

onCancel: function() {
  var msg = {
    message: 'cancel'
  };
  _postMessage(msg);
}

());

This JS file needs to be packaged in a content connector.
This library follows the JS postMessaging paradigm to pass information. The following methods are of interest.

**onInit**

The custom picker needs to invoke this method during its initialization. This is where the custom picker can get connector configuration information, like client ID and access token. The attributes defined in the connector configuration (/rest/api/v1/server) are passed via this method.

```javascript
OCE.CustomPicker.onInit(function(data) {
  if (data != null) {
    self.clientId = data.ClientID;

    // If connector uses OAuth
    self.AccessToken = data.AccessToken;

    // If connector uses BASIC auth
    self.user = data.UserName;
    self.password = Base64.decode(data.UserPwd);
  }

  // Use client id and access token to fetch data from back end.
});
```

**onOk**

If the custom picker exposes its own **OK** button, then it needs to invoke this method to pass back the event to OCE.

```javascript
function pickerCallback(data) {
  if (data[google.picker.Response.ACTION] === google.picker.Action.PICKED) {
    data[google.picker.Response.DOCS].forEach(function(doc) {
      OCE.CustomPicker.addItem(doc.id, doc.name, "file", doc.sizeBytes);
    });
    OCE.CustomPicker.onOk();
  } else if (data[google.picker.Response.ACTION] === google.picker.Action.CANCEL) {
    OCE.CustomPicker.onCancel();
  }
}
```

**onCancel**

If the custom picker exposes its own **Cancel** button, then it needs to invoke this method to pass back the event to OCE, like the **onOk** method does for the **OK** button in the preceding snippet.

**addItem**

This method needs to be invoked to pass back selections from the custom picker to OCE so that it displays the list of selected items in the **Add to repository** dialog. The
IDs passed here are then used to fetch content with `/rest/api/v1/content` while adding an item to the asset repository.

```javascript
addImage(selectedImages: Image[]) {
  selectedImages.forEach(o => {
    var name = "PexelsImages_" + o.uri;
    OCE.CustomPicker.addItem(o.uri, name, o.thumbnail, o.size);
  });
}
```

**removeItem**

This method needs to be invoked when an item is unselected from a custom picker so that the list maintained by OCE is up to date. The final list is shown in the **Add to repository** dialog. The ID passed here is the same as the ID passed in the `addItem` call.

```javascript
removeImage(image: Image) {
  OCE.CustomPicker.removeItem(image);
}
```

Package the custom picker in the connector. While registering the content connector, specify the following values:

1. **Custom Picker URL**: Such as `http://host:port/pexels-picker/web` (This is the custom picker packaged in the content connector.)

2. **Custom Picker uses its own OK/Cancel buttons**: Leave it unchecked. (This indicates that you want OCE to embed your picker in their dialog.)

   If the custom picker implements its own **OK** and **Cancel** buttons, then check this.

**Authorization**

A content connector can support one of the following authorization models.

- No Auth
- OAuth
- Basic

**No Auth**

This is used when the content connector does not require any authorization. In this case, the connector picker is launched directly without invoking any authorization screen.

**ServerResource Implementation**

```javascript
serverInfo.authenticationType = AuthenticationType.NO_AUTH
```
OAuth

This is used when a content connector supports OAuth. In this case, OCE ensures that the OAuth flow is invoked to fetch the access token. For subsequent access, the already fetched token is used until it expires.

serverResource Implementation

serverInfo.authenticationType = AuthenticationType.OAUTH

The content connector also needs to define the required custom fields used in the OAuth flow.

ServerResource Implementation

```java
{
    FieldInfo field = new FieldInfo();
    field.ID = UnsplashAdapter.FIELD_ID_REFRESH_TOKEN;
    field.label = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.refreshtoken.label");
    field.labelLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.refreshtoken.label");
    field.datatype = FieldDatatype.STRING;
    field.userSettable = false;
    field.siteSettable = false;
    field.connectorSettable = true;
    field.authorizationURLParameter = false;

    serverInfo.fields.add(field);
}
```

```java
{
    FieldInfo field = new FieldInfo();
    field.ID = UnsplashAdapter.FIELD_ID_ACCESS_TOKEN;
    field.label = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.accesstoken.label");
    field.labelLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.accesstoken.label");
    field.datatype = FieldDatatype.STRING;
    field.userSettable = false;
    field.siteSettable = false;
    field.connectorSettable = true;
    field.authorizationURLParameter = false;

    serverInfo.fields.add(field);
}
```
FieldInfo field = new FieldInfo();

field.ID = UnsplashAdapter.FIELD_ID_AUTHORIZATION_URL_PARAMETER_CODE;
field.label = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.authurl.code.label");
field.labelLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.authurl.code.label");
field.datatype = FieldDatatype.STRING;
field.userSettable = false;
field.siteSettable = false;
field.connectorSettable = false;
field.authorizationURLParameter = true;

serverInfo.fields.add(field);
}

FieldInfo field = new FieldInfo();

field.ID = UnsplashAdapter.FIELD_ID_AUTHORIZATION_URL_PARAMETER_ERROR;
field.label = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.authurl.error.label");
field.labelLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.authurl.error.label");
field.datatype = FieldDatatype.STRING;
field.userSettable = false;
field.siteSettable = false;
field.connectorSettable = false;
field.authorizationURLParameter = true;

serverInfo.fields.add(field);

And also implement the following interfaces documented in the preceding text:

1. /rest/api/v1/authorization/authorizationURLs
2. /rest/api/v1/authorization/completedAuthorizations
Basic

This is used when content connector requires login credentials to connect to the backend. Here OCE will prompt you to enter sign-in details before launching the picker for the first time. If the credentials are available, then they will be used.

ServerResource Implementation

serverInfo.authenticationType = AuthenticationType.BASIC

You also need to define all the login fields in the server implementation. These fields will display in the login screen.

// custom field for User Name
{
    FieldInfo field = new FieldInfo();
    field.ID = "UserName"
    field.label = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.username.label");
    field.labelLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.username.label");
    field.description = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.username.desc");
    field.descriptionLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.username.desc");
    field.datatype = FieldDatatype.STRING;
    field.userSettable = true;
    field.siteSettable = false;
    field.connectorSettable = false;
    field.authorizationURLParameter = false;
    field.required = true;
    serverInfo.fields.add(field);
}

// custom field for password
{
    FieldInfo field = new FieldInfo();
    field.ID = "UserPwd"
    field.label = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.password.label");
    field.labelLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.password.label");
    field.description = ResourceBundleUtil.getDefaultLocalizedString("cds.unsplash.adapter.field.password.desc");
    field.descriptionLocalizations = ResourceBundleUtil.getLocalizedData("cds.unsplash.adapter.field.password.desc");
    field.datatype = FieldDatatype.STRING;
    field.userSettable = true;
    field.siteSettable = false;
    field.connectorSettable = false;
    field.authorizationURLParameter = false;
    field.required = true;
    serverInfo.fields.add(field);
}
Besides this, the content connector needs to implement the following interface, as described in the preceding text.

/rest/api/v1/authorization/basicAuthorization

Subsequently the user credentials (password base64 encoded) will be passed via headers for other calls, like filesystem and get content.

Content Connector Configuration and Registration

Once you have built your content connector, you need to register it in OCE. You can register a content connector through the OCE administration interface. The minimum properties you are required to add to a content connector follow:

- Content connector name
- Content connector service URL
- User name and password, if the preceding URL access requires it
- Whether or not to accept a self-signed certificate

Once a content connector gets registered successfully and the content connector service URL is reachable, the OCE server will call a server service on the content connector to get the custom properties and show those in the administration UI. An administrator or developer can add values for those properties (for example, clientid and clientsecret for OAuth flows or the user name and password for a Basic authorization central account). Clicking Save saves all these properties with the connector framework. After you enable it, the content connector becomes available in the Add menu item of the OCE UI Assets page.

See Create and Configure a Custom Content Connector in Administering Oracle Content and Experience Cloud and Register the Content Connector, which describes how to register and configure a Pexel content connector.

Content Connector Execution Flow

From the Oracle Content and Experience Cloud (OCE) UI Assets page, the user can click Add and choose a content connector from the menu.

Based on the authorization type defined, the OAuth flow triggers or the user is prompted to add a user name and password for the Basic Auth flow (if a central account has not been configured in the administration UI). For No Auth, the content connector picker is launched directly without invoking any authorization screen.
After successful authorization, the configured file picker (either common or custom) is launched. It lists the files available to the user from the remote store.

The user can select one or more files. The selected files list is passed to the connector framework. The connector framework then makes REST API calls to the remote content connector to fetch the content of the assets being added and copies them into the Asset repository. This happens as a background process; however, the status of the add is displayed in the UI where success and failure can be tracked for each asset added. Refer to the diagnostic logs for any failures.

Pexels Content Connector Sample Implementation

You can use this sample implementation of a Pexels content connector to help build your own custom content connectors. The Pexels content connector lets content creators bring rich, high-quality images from the Pexels library into an Oracle Content and Experience Cloud asset repository.

Pexels is a stock photography site where designers, bloggers, and others find photos to use for free. The Pexels content connector is built using the Connector Framework SDK provided by Oracle Content and Experience Cloud. Two versions of this content connector are available:

1. A content connector that uses a custom picker built using Angular JS. This version is built by default.
2. A content connector that uses the common picker provided by Oracle Content and Experience Cloud. To build this version of the Pexels content connector, see Switch to a Common UI for the Custom Content Connector.

The following sections describe how to develop a Pexels content connector.

- Install the Content Connector
- Register the Content Connector
- Test the Content Connector
- Understand the Content Connector Source Code
- Pexels REST APIs
- Custom Picker UI
- Change and Test the Content Connector Code
- Switch to a Common UI for the Custom Content Connector

Install the Content Connector

To install the Pexels content connector, you need to meet the installation prerequisites and then build the content connector WAR file.

The following sections describe the prerequisites and how to build the WAR file.

Check Prerequisites for Installation

Your system needs to be set up with node, npm, JDK, and Apache Maven before you install the Pexels content connector.
1. Ensure you have node and npm installed on your machine and set in PATH. Configure proxy if required. 
   See Configure Proxy Service Settings in Administering Oracle Content and Experience Cloud.
2. Ensure you have latest JDK installed and set in PATH.
3. Ensure you have Apache Maven installed and set in PATH, with proxy configured if required.

**Build the Content Connector WAR File**

Download the content connector source bundle, unzip the bundle into a location on your machine, and follow the instructions in the Readme file to generate the WAR file.

You can download the OCE content connector sample and SDK from here:

https://your-cec-service/_sitesclouddelivery/renderer/app/sdk/connector/cec-connector-sample.zip

https://your-cec-service/_sitesclouddelivery/renderer/app/sdk/connector/cec-connector-sdk.zip

Briefly, the instructions are as follows:

1. In a command-line interface, go to pexelsPickerWeb\src\main\webapp.
2. Run pexels_setup.bat or pexels_setup.sh (based on your environment).
3. If the run is successful, pexels-picker.war file will be available in the pexelsPickerWeb\target folder.

The WAR file generated is the one that uses the custom picker UI, which was built using Angular JS. If you want to use the content connector with a common UI provided by Oracle Content and Experience Cloud, follow the instructions in Switch to a Common UI for the Custom Content Connector to build the WAR file.

Deploy this WAR file to your server. Run the following URLs to test that it works well:

1. GET http://host:port/pexels-picker/rest/api
2. GET http://host:port/pexels-picker/rest/api/v1/server

For expected output, see Understand the Content Connector Source Code.

**Register the Content Connector**

To use a Pexels content connector, you first need to access to the Pexels API. For this you need to register with Pexels and request access.

Follow the instructions in the Pexels API documentation, https://www.pexels.com/api/documentation/, to register the content connector and request access. After you make the request, you will be provided with an API key that you need to capture. The Pexels API is rate-limited to 200 requests per hour and 20,000 requests per month. If you need higher limits, contact Pexels.

1. Sign in to your Oracle Content and Experience Cloud (OCE) instance as an administrator.
2. Go to Administration > Integration > Content Connectors and click Create.

3. Enter the details to register your Pexels content connector.
   a. Name: Pexels
   b. Description: Connector to bring Pexels images into OCE asset repository
   c. Connector Service URL: http://host:port/pexels-picker/rest/api (a URL tested previously)
   d. Check Accept Self-Signed Certificate (if required).

4. Click Verify Settings and then Save.

5. On the Content Connectors tab, click Configure to go back to the content connector configuration and provide the following values.
      This is the custom picker packaged in a content connector. This setting is not applicable when you use the common UI.
   b. Hide OK/Cancel:
      Leave this setting unchecked. It indicates whether or not you want Oracle Content and Experience Cloud to embed your picker in their dialog, which is not applicable when you use the common UI.

6. Also provide the Client ID in the Custom Fields section:

7. Check Enable for End Users.

If your content connector is deployed on an instance that uses a self-signed certificate, the picker launched from OCE will show an error as it is launched in an iframe. You must launch the picker URL in a new browser tab, accept the certificate explicitly, and then launch the URL from OCE.

The default timeout values for the content connector are set in the following two properties:

- ConnectorConnectionTimeout=20000
- ConnectorReadTimeout=30000
If you want to change the values of these properties, you can add the properties to your `config.cfg` and then modify either or both values.

**Test the Content Connector**

Verify the functionality of your Pexels content connector.

To test the content connector, go to the **Assets** tab and click **Add**. You should see **Import from Pexels** in the drop-down menu. Ensure that you have created an asset repository and selected it before you click **Add**. This will launch the Pexels picker showing curated images up front.

This picker is either the custom UI packaged in the content connector or the common picker, depending on which you installed.

You can search for required images, select the ones you are interested in, and click **OK**. This will open up the **Add to Repository** dialog, which lists the selected images you can add. After you add them, the images should appear in the asset repository.

**Understand the Content Connector Source Code**

The source code of the Pexels content connector contains REST APIs and a custom picker.

The Pexels content connector contains a set of REST APIs implemented using the Oracle Content and Experience Cloud (OCE) content connector interfaces. This is implemented as per the JAX-RS specification.

The Connector SDK and sample implementation are in the following files:

- `cec-connector-sdk.zip`
This file contains the SDK interface JAR, Javadoc, and `connector.yaml` file, which describes the REST APIs and their payloads.

- `oracle-oce-custompicker.js`
  This is the JS file that the content connector custom UI must package to interact with OCE.

- `pexelsPickerWeb.zip`
  This file contains the Pexels content connector reference implementation.

### Custom Picker UI

The Pexels picker packaged in the content connector is built using Angular JS. It provides the Custom Picker UI.

The Pexels picker files are in `pexelsPickerWeb\src\main\webapp\src`.

### src/app/oracle-oce-custompicker.js File

Oracle Content and Experience Cloud (OCE) exposes the library in the `src/app/oracle-oce-custompicker.js` file.

This library contains the contracts the custom picker must implement to interact with the parent OCE window to obtain configuration and pass back selections. The library, which works on the JS postMessaging paradigm, is defined in `angular.json` to be used in the project.

### index.html File

The `index.html` file is the starting HTML file, which includes the angular app components defined.

It contains `app-root`, which in turn contains the `app-gallery` component. This component is the picker that displays a search field and results in grid view.

Pagination is implemented in scroll fashion, which retrieves results in batches of 15.

### app.module.ts File

The `app.module.ts` file has dependency added for `http`, `image.service`, `gallery.component`, `ngx-infinite-scroll`, and other angular libraries.

ImageService is added to list of providers, and `AppComponent` and `GalleryComponent` are added in declarations.

### responses.ts File

The `responses.ts` file has all the interfaces for responses from the Pexel REST APIs.

### App Component

The app component is the main component referenced in `index.html`, which displays the title.
This component in turn adds the `app.gallery` component. The files of interest are `app.component.html`, `app.component.ts`, and `app.component.css`.

### Gallery Component

The gallery component contains the meat of the picker implementation. This component contains `gallery.component.ts`, `gallery.component.html`, and `gallery.component.css`.

This component is the entry point where we invoke the Oracle Content an Experience Cloud `onInit` to fetch the client ID that will be used to access the Pexels APIs. The gallery component also injects an image service to fetch image data from Pexels and convert Pexels response format to UI response format in `getImages()` and `searchImageService()`.

OCE.CustomPicker.onInit obtains the client ID and any other information required to be configured for the content connector from Oracle Content and Experience Cloud.

```javascript
OCE.CustomPicker.onInit(function(data) {
    if (data != null) {
        console.log("OAuth token is " + data.ClientID);
        self.clientId = data.ClientID;
    }

    if (self.clientId == null) {
        console.error("No Client Id Present");
        self.clientId = 'client-id';
    }

    console.log("ClientId is " + self.clientId);
    self.getImages(self.clientId);
    self.appendItems(0, this.sum);
});
```

### src/app/image/shared/image.service.ts File

This file includes the Pexels image service declared as injectable. It handles all the REST API calls made to Pexels APIs and is used by the gallery component to fetch results.

The image service follows the contracts laid out by Oracle Content and Experience Cloud to pass back a selection. The service is invoked by the gallery component on selection events.

You can invoke OCE.CustomPicker.addItem when an item is selected. This will build the list maintained by the Oracle Content and Experience Cloud parent that is used to display the final selection in the Add to Asset Repository UI.

```javascript
addImage(selectedImages: Image[]) {
    selectedImages.forEach(o => {
        var name = "PexelsImages_" + o.uri;
        OCE.CustomPicker.addItem(o.uri, name, o.thumbnail, o.size);
    });
}
```
You can invoke `OCE.CustomPicker.removeItem` when an item is unselected. This will update the list maintained by the Oracle Content and Experience Cloud parent that is used to display the final selection in the Add to Asset Repository UI.

```javascript
removeImage(image: Image) {
    OCE.CustomPicker.removeItem(image);
}
```

### Pexels REST APIs

The Pexels content connector contains a set of REST APIs.

**PexelsAPIResourceImpl**

The `PexelsAPIResourceImpl` API returns the latest version of the REST APIs supported by the Pexels content connector.

GET http://host:port/pexels-picker/rest/api

```
[  
  "v1"
]
```

**PexelsServerResourceImpl**

The `PexelsServerResourceImpl` API returns information on content connector configuration, like authentication type, picker type, and the custom fields it exposes.

GET http://host:port/pexels-picker/rest/api/v1/server

```
{
  "name": "Pexels Connector",
  "nameLocalizations": [  
    {
      "locale": "en",
      "localizedString": "Pexels Connector"
    }
  ],
  "version": " (, , )",
  "about": "Pexels Connector.<br>Copyright (c) 2019, Oracle and/or its affiliates. All rights reserved.",
  "aboutLocalizations": [  
    {
      "locale": "en",
      "localizedString": "Pexels Connector.<br>Copyright (c) 2019, Oracle and/or its affiliates. All rights reserved."
    }
  ],
  "authenticationType": "NO_AUTH",
  "pickerType": "CUSTOM",
  "enableMultiUserCopyBack": false,
  "maxUploadSize": 1073741824,
  "fields": [  
  ]
}
```
"ID": "ProxyHost",
"datatype": "STRING",
"siteSettable": true,
"userSettable": false,
"connectorSettable": false,
"authorizationURLParameter": false,
"label": "HTTP Proxy Hostname",
"labelLocalizations": [
  {
    "locale": "en",
    "localizedString": "HTTP Proxy Hostname"
  }
],
"description": "The HTTP proxy hostname, leave blank to disable.",
"descriptionLocalizations": [
  {
    "locale": "en",
    "localizedString": "The HTTP proxy hostname, leave blank to disable."  
  }
],
"required": false
},
{
  "ID": "ProxyPort",
  "datatype": "STRING",
  "siteSettable": true,
  "userSettable": false,
  "connectorSettable": false,
  "authorizationURLParameter": false,
  "label": "HTTP Proxy Port",
  "labelLocalizations": [
    {
      "locale": "en",
      "localizedString": "HTTP Proxy Port"
    }
  ],
  "description": "The HTTP proxy port number, leave blank to default to port 80.",
  "descriptionLocalizations": [
    {
      "locale": "en",
      "localizedString": "The HTTP proxy port number, leave blank to default to port 80."
    }
  ],
  "required": false
},
{
  "ID": "ProxyScheme",
  "datatype": "STRING",
  "siteSettable": true,
  "userSettable": false,
  "connectorSettable": false,
PexelsFileSystemResourceImpl

**Required only if you use the Oracle Content and Experience Cloud common UI, the PexelsFileSystemResourceImpl API supports search as well.**
POST http://host:port/pexels-picker/rest/api/v1/filesystem

Request Headers
X-CEC-ClientID: client-id
X-CEC-ProxyHost: host-name
X-CEC-ProxyPort: 80
X-CEC-ProxyScheme: http
Content-Type: application/json

Request Body
{
   "itemCount": "50",
   "itemStartRow": "0",
   "itemsSortField": "ASC",
   "queryText": "animals"
}

Response
{
   "numItems": 50,
   "hasMoreItems": true,
   "totalItemsCount": 7190,
   "fileSystemInfo": {
      "uri": "fFolderGUID:null",
      "parentUri": "fFolderGUID:null",
      "name": "/",
      "description": null,
      "isDirectory": true,
      "size": null,
      "mimeType": null,
      "extension": null,
      "creator": null,
      "createdTimeStamp": null,
      "lastModifiedBy": null,
      "lastModifiedTimeStamp": null,
      "browseURL": null,
      "thumbnailURL": null,
      "directoryContents": [
         {
            "uri": "45170",
            "parentUri": "fFolderGUID:null",
            "name": "kittens-cat-cat-puppy-rush-45170.jpeg",
            "description": "kitten cat rush lucky cat",
            "isDirectory": false,
            "size": null,
            "mimeType": "image/jpeg",
            "extension": "jpeg",
            "creator": "Pixabay",
            "createdTimeStamp": null,
            "lastModifiedBy": null,
            "lastModifiedTimeStamp": null,
See Switch to a Common UI for the Custom Content Connector.

PexelsContentResourceImpl

For a given Pexels image ID, the PexelsContentResourceImpl API returns the input stream for the image.

GET http://host:port/pexels-picker/rest/api/v1/content?uri=fFileGUID;xxxx

Request Headers
X-CEC-ClientID:client-id
X-CEC-ProxyHost:host-name
X-CEC-ProxyPort:80
Response Headers
content-disposition attachment; filename=pexels-photo-xxxx.jpg
content-type image/jpeg

Response Body
File Content

PexelsAdapter

The PexelsAdapter API contains utility functions.

Change and Test the Content Connector Code

If you make changes to the content connector code, you can run commands from pexelsPickerWeb\src\main\webapp to test the changes and rebuild the WAR file.

1. If you modify package.json to include additional libraries, run the following command from pexelsPickerWeb\src\main\webapp:

   npm install

2. If you make UI code changes, run the following command from pexelsPickerWeb\src\main\webapp:

   ng build

3. Finally, run mvn from pexelsPickerWeb to rebuild the WAR file:

   mvn clean install

Switch to a Common UI for the Custom Content Connector

You can switch to a common UI to build a content connector that uses the common picker provided by Oracle Content and Experience Cloud.

The Pexels content connector uses a custom UI packaged in the content connector WAR file. However, you can switch over to using a common UI provided by the Oracle Content and Experience Cloud connector framework. To do so, modify PexelsServerResourceImpl.java:

serverInfo.pickerType = PickerType.COMMON;

You can remove the additional field Picker_URL in custom fields.

The common UI requires http://host:port/pexels-picker/rest/api/v1/filesystem to be implemented. Refer to PexelsFileSystemResourceImpl in the source section.

Rebuild the WAR file using the instructions in Change and Test the Content Connector Code, and then register a new content connector.
The old content connector will still go against the custom UI. Test the new content connector in assets UI, where it should now render the common UI.

Download the CEC Content Connector Sample and SDK

A sample content connector and SDK are available for download.

You can download the following ZIP files, which contain the CEC content connector sample and SDK:

- https://your-cec-service/_sitesclouddelivery/renderer/app/sdk/connector/cec-connector-sample.zip
- https://your-cec-service/_sitesclouddelivery/renderer/app/sdk/connector/cec-connector-sdk.zip
Tutorial: Develop Components with Knockout

This tutorial takes you through working with the set of JavaScript objects, which leverage the standard Knockout ViewModel and Template functionality, to create a component which is stored in the Oracle Content and Experience Cloud Component Catalog.

• Introduction and Prerequisites for Component Development with Knockout
• Step 1: Create a Component
• Step 2: Review the Structure of Your Local Component Rendering
• Step 3: Review the Structure of Local Component Settings
• Step 4: Display the New Property in the Component
• Step 5: Register Triggers
• Step 6: Raise Triggers
• Step 7: Register Actions
• Step 8: Execute Actions
• Step 9: Create a Distinct Title for Each Instance of the Component
• Step 10: Use Nested Components with In-Line Editing
• Step 11: Support Different Layouts
• Step 12: Define Custom Styles
• Step 13: Render a Component in an Inline Frame
• Step 14: Use Custom Styles When the Component Is Rendered in an Inline Frame
• Step 15: Integration with the Page Undo and Redo Behavior
• Step 16: Asset Management
• Tutorial Review

Introduction and Prerequisites for Component Development with Knockout

This tutorial presents steps and verification procedures to create a sample component using JavaScript objects, which leverage the standard Knockout JS ViewModel and Template functionality.

You should be able to take the code referenced in these steps (provided in files that are seeded when you create a component) and update only the .html template and JavaScript viewModel with your own code.
Prerequisites

This tutorial only focuses on the implementation of a component. For a more general information about components, see Develop Components.

To complete the steps in this tutorial, you must meet the following requirements:

- You must have access to an Oracle Content and Experience Cloud instance with permissions to create sites and components.
- The Oracle Content and Experience Cloud instance server has been synced to your local computer using the Oracle Content and Experience Cloud desktop or using a custom component. See Develop Custom Components with Developer Cloud Service.

In addition, you should be familiar with these JavaScript concepts and frameworks:

- JavaScript browser debugging
- JavaScript Closure
- JavaScript Asynchronous Module Definition (AMD) development
- RequireJS and KnockoutJS frameworks

Continue to Step 1: Create a Component.

Step 1: Create a Component

This step explains how to create your custom component in Oracle Content and Experience Cloud.

When you create a custom component, it must be registered to be usable by Oracle Content and Experience Cloud. To inform Oracle Content and Experience Cloud about your component, you use the Components page in Site Builder to register the component.

There are two types of components to register.

- **Local component:**
  - This is a component whose files are stored on the Oracle Content and Experience Cloud instance server.
  - The main advantage is that you don’t have to worry about cross-domain or cross-protocol issues because the files are located with your site.
  - The disadvantage is that you can’t execute any middle-tier logic in the Oracle Content and Experience Cloud server, so you must use REST APIs to remote servers that support CORS.
  - This type of component may be embedded into the page directly, or you can choose to use an inline frame to render the component on the page.
• Remote component:
  – A component where the files are stored on a remote server and you only register the URLs to the Render and Settings panel for the component.
  – The advantage is if you have server-side logic that must execute when creating the content for your component.
  – The disadvantage is that you must ensure that any cross-domain and security issues are resolved for accessing those URLs.
  – Remote components always use an inline frame to render on the page.

To create and register a local component:
1. On the Oracle Content and Experience Cloud home page, click Developer. The Developer page is displayed.
2. Click Components.
3. From the menu, choose Create Local Component.
4. Enter a name for the component; for example, A_Local_Component.
5. Enter an optional description.
6. Click OK.

After you have done this you’ll see a component named A_Local_Component in your list of components.

Check the Results for Step 1
Now that you’ve successfully created a component, you should see the component in the Component palette for any site you create. Use these steps to validate your component creation:
1. Create a site named localComponentTest.
2. Create an update for the site and give it a name and, optionally, a description.
3. Place the site into Edit mode and select a page on the site.
4. Click in the side palette and select Custom to display the list of custom components.
5. Select A_Local_Component from the Custom component list and drag and drop it onto the page.

You should now see a default rendering for the local component you created.
6. Select in the banner for the component you just dropped onto the page.
7. Select Settings.
8. Change the alignment and set the style for the component.
9. Close the Settings panel.

The following steps explain how the custom component is built and how to modify it for your own purposes. Continue to Step 2: Review the Structure of Your Local Component.
Step 2: Review the Structure of Your Local Component Rendering

In this step we review the structure of the default files created for a local component.

For a simple Hello World example, four JavaScript objects and the number of lines of code may seem like too much, but this is to provide you with the basis for building more complex components, as well as dealing with interaction with the Oracle Cloud Sites Service page lifecycle.

To review the structure of your local component:

1. Using the Oracle Content and Experience Cloud desktop sync client, locate your component and sync it with the file system.

   If you don't have the desktop client, you can select the component in the Components tab of the Oracle Content and Experience Cloud interface and drill down to see the files.

2. If you list the files under the component, you will see these files:

   assets
   render.js
   settings.html
   appinfo.json
   _folder_icon.jpg

3. Open the render.js file under the /assets directory.

   The main points of the render.js file are:
   • It is structured as a JavaScript AMD module so that it can be “required” into the page.
   • It also includes references to KnockoutJS and JQuery that are already loaded as part of the Oracle Content and Experience Cloud page.

Consider the structure of the render.js file.

In the contents of the render.js file there are two JavaScript objects that implement the required Oracle Content and Experience Cloud component APIs: sampleComponentFactory and SampleComponentImpl. These objects are an example of an implementation for creating any KnockoutJS based components. The implementation of these objects will change based on the technology you use.

- sampleComponentFactory
  - This object is returned by the render.js AMD module.
  - This is a very simple Factory object and implements the single createComponent() interface.
  - More complex implementations may use the args value passed to return different implementations of the component based on the viewMode parameter. This enables you to have a significantly lighter-weight implementation of the component for runtime versus Site Builder.

- SampleComponentImpl
The main function within this object is the `render` function, which is used to render the component onto the page. To render the Knockout component into the page, the `render` function dynamically adds the template to the page, then applies the `viewModel` bindings to the template. The rest of the implementation deals with initialization of the `viewModel` parameter and template, and with handling the messaging between the page and the component.

The last two objects in the `render.js` file, `sampleComponentTemplate` and `SampleComponentViewModel`, provide a custom implementation for the component. The implementation of these will differ based on your requirements.

- **sampleComponentTemplate**
  - This object provides the KnockoutJS template creation. It waits until the component has all the data initialized before attempting to display anything.
- **SampleComponentViewModel**
  - The `viewModel` retrieves the information stored by Oracle Content and Experience Cloud on behalf of the component, then selects how to appropriately lay out the component based on that data.

General Knockout observables used by the template to handle access to the metadata stored on the component’s behalf:

```javascript
self.imageWidth = ko.observable('200px');
self.alignImage = ko.observable();
self.layout = ko.observable();
self.showTopLayout = ko.observable();
self.showStoryLayout = ko.observable();
```

- **Triggers and actions integration:**

  **Trigger**: A function to raise an Oracle Content and Experience Cloud trigger from the component that can be bound to actions of other components on the page.

  ```javascript
  self.imageClicked = function (data, event) {
    self.raiseTrigger("imageClicked"); // matches appinfo.json
  };
  ```

  **Action**: A function to handle the callback for when the component is told to execute an action with a given payload.

  ```javascript
  self.executeActionsListener = function (args) {
    // get action and payload
    var payload = args.payload,
        action = args.action;

    // handle 'setImageWidth' actions
    if (action && action.actionName === 'setImageWidth') {
      $.each(payload, function(index, data) {
        if (data.name === 'imageWidth') {
          self.imageWidth(data.value);
        }
      });
    }
  };
  ```
Callback to execute any registered actions on demand.

```javascript
SitesSDK.subscribe(SitesSDK.MESSAGE_TYPES.EXECUTE_ACTION, $.proxy(self.executeActionsListener, self));
```

Subscriptions to component life cycle:

* Component initialization: Make sure the component doesn't render until all data has been fetched. This is handled through Knockout observables.

```javascript
self.componentLayoutInitialized = ko.observable(false);
self.customSettingsDataInitialized = ko.observable(false);
```

Get the initial values for any required properties. This is handled by callbacks to retrieve the data.

```javascript
SitesSDK.getProperty('componentLayout', self.updateComponentLayout);
SitesSDK.getProperty('customSettingsData', self.updateCustomSettingsData);
```

* Metadata updates: Callback whenever component metadata stored on the component's behalf is changed; for example, when the user invokes the Settings panel and updates the data.

```javascript
SitesSDK.subscribe(SitesSDK.MESSAGE_TYPES.SETTINGS_UPDATED, $.proxy(self.updateSettings, self));
```

---

**Note:**

Because the Oracle Content and Experience Cloud server always sets the mime-type for `.html` files, you cannot upload a `.html` file and use the "text!" require plug-in to load it. Therefore, for templates, you either need to use a different extension to load it using "text!" or in-line the template in the JavaScript directly as shown in the seeded data.

Check the Results for Step 2

You should now have an overview of how the structure of a custom component renderer is created. To validate that it works:

1. Update the `sampleComponentTemplate` object in the `render.js` file to change the following line. Change this code:

   ```javascript
   '<!-- ko if: initialized -->'
   ```
Use this code instead:

```html
<!-- ko if: initialized -->
<div data-bind="text: 'image width is: ' + imageWidth()"></div>
</ko if-->
```

2. Sync or upload the component to the Oracle Content and Experience Cloud instance server.

3. Edit a page within the site and drop in the A_Local_Component custom component onto the page.

   At this point you should see image width is: 260px in the component.

4. Bring up the Settings panel and click the Custom Settings button.

5. Change the image Width field to 300px.

6. At this point two things will happen in the component:
   a. The default image will expand from 260px to 300px in size.
   b. The text you added will update to image width is 300px.

Continue to Step 3: Review the Structure of Local Component Settings.

---

**Step 3: Review the Structure of Local Component Settings**

In this step we review the structure of the settings specified for a local component.

Similar to the render.js file in the /assets directory, there is a pre-created settings.html file in the same directory. The settings.html file renders any custom settings data for your component. In the default implementation, there is a single property imageWidth in the custom settings data.

To review the structure of your local component:

1. Using the Oracle Content and Experience Cloud desktop sync client, locate your component and sync it with the file system.

   If you don't have the desktop sync client, you can select the component on the Components tab of the Oracle Content and Experience Cloud interface and drill down to see the files.

2. If you list the files under the component, you'll see these files:

   ```
   assets
   render.js
   settings.html
   appinfo.json
   _folder_icon.jpg
   ```

Open the settings.html file under the /assets directory and review the content. Unlike the render.js file, the settings.html file uses an inline frame in the Settings panel in Site Builder, which is why it also needs access to the supporting files to render correctly within the inline frame. Site Builder is needed to manage your site so any errors in your JavaScript code can be isolated from Site Builder, which is why the settings.html file uses an inline frame.

These are the main areas of the settings.html file:
• Knockout Template to render the Settings panel.

```html
<!-- ko if: initialized() -->
<div class="scs-component-settings">
  <div>
    <!-- Width -->
    <label id="widthLabel" for="width" class="settings-heading" data-bind="text: 'Image Width'"></label>
    <input id="width" data-bind="value: width" placeholder="example: 200px or 33%" class="settings-text-box">
  </div>
</div>
<div data-bind="setSettingsHeight: true"></div>
<!-- /ko -->
```

• Custom Binding Handler to adjust the height of the inline frame once the Settings panel has been rendered.

```javascript
ko.bindingHandlers.scsCompComponentImpl
```

• A Knockout ViewModel to apply to the Knockout Template.

  ```javascript
  SettingsViewModel
  ```

These are the main elements of SettingsViewModel:

• Subscriptions to component lifecycle.

• Component initialization:
  – Make sure the component doesn't render until all data has been fetched. This is handled through Knockout observables.

    ```javascript
    self.initialized = ko.observable(false);
    ```
  
  – Make sure we don't attempt to update the data until we're ready.

    ```javascript
    self.saveData = false;
    ```
  
  – Get the initial values for any required properties. This is handled by callbacks to retrieve the data.

    ```javascript
    SitesSDK.getProperty('customSettingsData', function (data) {
      // update observable
      self.width(data.width);
      // note that viewModel is initialized and can start saving data
      self.initialized(true);
      self.saveData = true;
    });
    ```
Save any property changes to the custom settings data.

    self.save = ko.computed(function () {
        var saveconfig = {
            'width': isNaN(self.width()) ? self.width() : self.width() + 'px'
        };

        // save data in page
        if (self.saveData) {
            SitesSDK.setProperty('customSettingsData', saveconfig);
        }
    }, self);

To add another property that you want to capture, several steps are required:
1. Update the user interface to display the new value.
2. Initialize the value to the current value stored against the component.
3. Save any changes to the value back to the component.

To add another property to your custom component, make these changes to the settings.html file:

1. Add another observable to handle the new property. Change this code:

        self.width = ko.observable();

    Use this code instead:

        self.width = ko.observable();
        self.imageBannerText = ko.observable();

2. Get any current value for the new property when the settings panel is first displayed. Change this code:

        self.width(data.width);

    Use this code instead:

        self.width(data.width);
        self.imageBannerText(data.imageBannerText);

3. Save any change to this new property. Change this code:

        'width': isNaN(self.width()) ? self.width() : self.width() + 'px'

    Use this code instead:

        'width': isNaN(self.width()) ? self.width() : self.width() + 'px',
        'imageBannerText': self.imageBannerText()
4. Add user interface to display the new field. Change this code:

```html
<label id="widthLabel" for="width" class="settings-heading" data-bind="text: 'Image Width'"></label>
<input id="width" data-bind="value: width" placeholder="example: 200px or 33%" class="settings-text-box">
```

Use this code instead:

```html
<label id="widthLabel" for="width" class="settings-heading" data-bind="text: 'Image Width'"></label>
<input id="width" data-bind="value: width" placeholder="example: 200px or 33%" class="settings-text-box">
<label id="imageBannerTextLabel" for="imageBannerText" class="settings-heading" data-bind="text: 'Image Banner'"></label>
<input id="imageBannerText" data-bind="value: imageBannerText" placeholder="Text to display above an image" class="settings-text-box">
```

5. Sync or upload the settings.html file.

If you were to run this now, the field would display. However, the size of the Settings panel doesn't change automatically. Because you increased the size of the panel, you also must update the components.json registration entry to the new size.

1. Download the appinfo.json file, which is at the same level as the assets/ directory for your component, and update the size of the settings panel. Change this code:

```
"settingsHeight": 90,
```

Use this code instead:

```
"settingsHeight": 160,
```

2. Sync or upload the appinfo.json file.

Check the Results for Step 3

You should now be able to see and enter the new property you added to the Settings panel.

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Click the Custom Settings button.

You will see two fields displayed for each of the properties you have in your settings.html file.

Continue to Step 4: Display the New Property in the Component.
Step 4: Display the New Property in the Component

At the end of this section, you will be able to enter a value for a new property in the Settings panel and see the custom component change to reflect the new value. Updates to the property will also automatically be saved for you with the page.

In the `render.js` file, you must update two JavaScript objects in the component:

- `SampleComponentViewModel`
- `sampleComponentTemplate`

Edit `render.js` and update the `SampleComponentViewModel` component to include the new property. Change this property:

```javascript
self.showStoryLayout = ko.observable();
```

Use this instead:

```javascript
self.showStoryLayout = ko.observable();
self.imageBannerText = ko.observable();
```

Update `SampleComponentViewModel` to get any change in values. Change this property:

```javascript
self.imageWidth(customData && customData.width);
```

Use this instead:

```javascript
self.imageWidth(customData && customData.width);
self.imageBannerText(customData && customData.imageBannerText);
```

Change `sampleComponentTemplate` to display the new property. Change this property:

```html
'<div data-bind="text: \'image width is: \' + imageWidth()"></div>' +
```

Use this instead:

```html
'<div data-bind="text: imageBannerText"></div>' +
```

Sync or upload the component to the Oracle Content and Experience Cloud server.

You've now altered the component to display the new property. Unlike the Settings panel that is embedded in an inline frame on the page, because the component is inserted directly into the page, as it grows in size the area available to it will automatically increase.

Check the Results for Step 4
To see the new property displayed:
1. Refresh your page in your site to Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Click the **Custom Settings** button.
6. Change **Image Banner** to **Workspace**.

You will see the component update on the page to **Workspace** appear above the image.

Continue to Step 5: Register Triggers.

**Step 5: Register Triggers**

In this step, you will review how an Oracle Content and Experience Cloud trigger can been registered, which you can select using the Trigger Actions option under the Link tab in the Settings panel for your component.

Triggers are part of Oracle Content and Experience Cloud intercomponent communication. Any component can raise any number of triggers. The component can provide a payload for a trigger, which is then passed to any action that is executed when the trigger is raised. Users can select what actions should be executed for each trigger. Finally, components that are built to work together can automatically raise triggers to execute actions on the other component without the user needing to define the interaction between the components.

For components you add, triggers are registered as part of the registration data for the component. To add a trigger, update the "triggers" property array with each trigger the component supports. You also must specify the payload the trigger supports so that the user interface can be created to allow the user to map values within the payload to properties supported by the action.

Open the `appinfo.json` file and review the "triggers":[], entry.

```
"triggers": [{
   "triggerName": "imageClicked",
   "triggerDescription": "Image clicked",
   "triggerPayload": [{
      "name": "payloadData",
      "displayName": "Trigger Payload Data"
   }]
}]
```

In this entry, you’ll see the following:

- A **triggerName**, "imageClicked", which should be a unique value, and will be typically namespaced by your custom component ID.
- A **triggerDescription**, "Image clicked", which is used by the user interface dialog to display your trigger.
- A single value **triggerPayload**, "payloadData", for your trigger. Users will be able to select entries in this payload and map them to fields in the action.
Check the Results for Step 5

You can see and select your trigger when you go to the Link tab in the Settings panel for your component:

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Select the Link tab at the top of the Settings panel.
6. Click **Trigger Actions** as the Link Type.
7. Click the **Image clicked** trigger that you registered.
8. In the dialog, drag the **Show Alert** action from within the Page Actions section. (Page Actions are built-in actions supplied by Oracle Content and Experience Cloud.)
9. In the **Message** field, select the **Trigger Payload Data** value, which is the name of the entry in the payload you see when you registered the trigger. You can change this to any name you want.

Now you’re able to register a trigger and map the trigger to a built-in action, passing through a value. In the next step we’ll review how the trigger is raised to execute the action.

Continue to **Step 6: Raise Triggers**.

---

Step 6: Raise Triggers

In this step, we’ll show you how the trigger you saw registered is raised.

Triggers can be raised at any point by a component. Typically it is raised by a user interaction, by clicking a button or selecting a row in a table. However, the component can raise the trigger based on any criteria; for example, when data changes because of a REST call.

For this sample, when you click the image it will raise a trigger passing through the current value of the **whoAreYou** property.

Review the **render.js** file and look at the **SampleComponentViewModel** object.

To raise a trigger:

1. Review the function in the **SampleComponentViewModel** object that calls the Sites SDK to raise the trigger.

```javascript
self.raiseTrigger = function (triggerName) {
  SitesSDK.publish(SitesSDK.MESSAGE_TYPES.TRIGGER_ACTIONS, {
    'triggerName': triggerName,
    'triggerPayload': {
      'payloadData': 'some data here'
    }
  })
```


2. Now you need something in the user interface to call the function to raise the trigger. Review the render.js file and update the sampleComponentTemplate object to have this entry:

```javascript
'<div data-bind="attr: {style: imageStyle, \'data-layout\': alignImage()}, click: imageClicked">'
```

In the SampleComponentViewModel object, you see the JavaScript function that is called when the image is clicked. This function calls the Sites SDK to tell it to trigger all the actions defined for the trigger "imageClicked", which is the value passed in from the click binding in step 2. It also passes through a triggerPayload that has a single field:payloadData and passes through a static value 'some data here'. These values imageClicked and whoAreYou match those in the appinfo.json file where the trigger is registered (in the previous step).

In the sample code, the trigger is raised by a data-bind of the click binding and passes in the trigger name imageClicked. There are currently three renderings of the <scs-image> component based on the layout the user chooses. To ensure that the trigger is raised for each of the layouts, edit the render.js file to make the following changes.

- Raise triggers from different layouts. Find the two entries of this code:

  `<div data-bind="attr: {style: imageStyle, \'data-layout\': alignImage()}">'

  Change the code to this:

  `<div data-bind="attr: {style: imageStyle, \'data-layout\': alignImage()}, click: imageClicked">'

- Specify the payload to pass to the triggers. Change this code:

  ```javascript
  self.raiseTrigger = function (triggerName) {
    SitesSDK.publish(SitesSDK.MESSAGE_TYPES.TRIGGER_ACTIONS, {
      'triggerName': triggerName,
      'triggerPayload': {
        'payloadData': 'some data here'
      }
    });
  };
  ```

  Use this code instead:

  ```javascript
  self.raiseTrigger = function (triggerName) {
    SitesSDK.publish(SitesSDK.MESSAGE_TYPES.TRIGGER_ACTIONS, {
      'triggerName': triggerName,
      'triggerPayload': {
        'payloadData': self.imageBannerText() // pass banner text as payload
      }
    });
  };
  ```
• Sync or upload the render.js file to the Oracle Content and Experience Cloud instance server.

Now that you've reviewed the required code, you can hook up the trigger so that your custom component will raise it when the button is clicked.

Check the Results for Step 6

You should now be able to register an action to execute against your trigger and also have the action execute when the trigger is raised:

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Select the Link tab at the top of the Settings panel.
6. Select Trigger Actions as the Link Type.
7. Click the imageClicked trigger that you saw registered.
8. In the dialog, drag the Show Alert action from with the Page Actions section.
9. In the Message field, select the payloadData value, which is the payload you entered when you registered the trigger.
10. Close the Settings panel and switch Site Builder to Preview mode.
11. Click the image in the component.

   An alert will appear showing no message defined because you haven't specified the imageBannerText value.

12. Take the page into Edit mode and bring up the Settings panel again for the component.
13. Click Custom Settings and enter Workplace.
14. Close the Settings panel and switch the page to Preview mode.
15. Click the image in the component.

   Now it should show the updated payload Workplace, which is invoked from the change you made to the click binding.

You can execute any number of actions when a trigger is raised.

Note:

There is no pre-defined order to when an action is executed. Although each action will be called in the order it is listed, there is no wait for it to complete before the next action is called. If an action makes an asynchronous call, it may not complete before the next action is executed.

Continue to Step 7: Register Actions.
Step 7: Register Actions

Oracle Content and Experience Cloud actions are called on components when triggers are raised.

A component can register any number of actions and also define the payload the action supports. When a user selects an action, they can populate the payload to be passed to the action.

As with registering triggers, you can register actions that your component supports in the `appinfo.json` file registration data. To review the registration of the sample action in your component, open the `appinfo.json` file and find the "actions" code.

```
"actions": [{
  "actionName": "setImageWidth",
  "actionDescription": "Update the image width",
  "actionPayload": [{
    "name": "imageWidth",
    "description": "Image Width in pixels",
    "type": {
      "ojComponent": {
        "component": "ojInputText"
      }
    },
    "value": ""
  }]
}]
```

This registered action will be visible in the Action dialog that's invoked when you click a trigger in the Link tab in the Settings panel for the component.

Check the Results for Step 7

1. Refresh your page in your site so Site Builder can pick up the changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Drop a Button component onto the page.
5. Bring up the Settings panel against the Button component.
6. On the General tab, change the label of the button to **Click me!**
7. Select the Link tab on the Settings panel.
8. Select **Trigger Actions** as the Link Type.
9. Click the **Click on Button** trigger against the Button component.
10. In the dialog, expand the `A_Local_Component` component in the left side palette.
11. Drag and drop the **Update the image width** action from the `A_Local_Component` component onto the page.
12. Enter 300px in the **Image width in pixels** field.

You've now seen how you can register an action and how that action will show up in the user interface. In the next step you will learn how to handle an action within your component when it is called.

Continue to **Step 8: Execute Actions**.

**Step 8: Execute Actions**

At the end of this topic, you'll be able to drop components on the page that execute actions within your component. This leverages the action registration you created in the previous step.

For a component to execute an action, it must listen for the **EXECUTE_ACTION** message. This message also includes the payload passed to the action from which you will extract the expected values.

To listen for the **EXECUTE_ACTION** message, edit the `render.js` file and update the `SampleComponentViewModel` object with the following entry:

```javascript
SitesSDK.subscribe('EXECUTE_ACTION', $.proxy(self.executeActionsListener, self));
```

When the **EXECUTE_ACTION** message is received, the associated callback function is executed:

```javascript
self.executeActionsListener = function (args) {
  // get action and payload
  var payload = args.payload,
      action = args.action;

  // handle 'setImageWidth' actions
  if (action && action.actionName === 'setImageWidth') {
    $.each(payload, function(index, data) {
      if (data.name === 'imageWidth') {
        self.imageWidth(data.value);
      }
    });
  }
}
```

This creates a JavaScript function to execute the action, then uses the Sites SDK to call the function whenever the **EXECUTE_ACTION** message is raised.

The component will be called whenever an **EXECUTE_ACTION** message is raised, and it is up to the component to only handle actions it is designed to handle. To do this, you must check the name of the action to ensure it is one the component can handle.

The payload for the action is an array of values. Typically, you will need to find the payload values you care about from the array.
Check the Results for Step 8

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Drag and drop a Button component onto the page.
5. Bring up the Settings panel against the Button component.
6. On the General tab, change the Label of the button to Click me!
7. Select the Link tab at the top of the Settings panel.
8. Select Trigger Actions as the Link Type.
9. Click the Click on Button trigger against the Button component.
10. In the dialog, expand the A_Local_Component component in the left side.
11. Drag and drop the Update the image width action from the A_Local_Component component onto the right side.
12. Enter 300px in the Image Width in pixels field.
13. Switch the page to Preview mode.
14. Click the Click me! button.

At this point the size of your image will increase to 300px.

Note:
Triggers and actions are designed to support inter-component communication. They are not designed to create or manage state. If you refresh the page, the page will revert to its original state as if no triggers were raised or actions executed.

Continue to Step 9: Create a Distinct Title for Each Instance of the Component.

Step 9: Create a Distinct Title for Each Instance of the Component

This step explains how to create distinct titles for different instances of your component.

When you drop your component onto the page, you will have noticed the banner for your component reads: A_Local_Component. While this is fine if the user only drops
one of your components onto the page, you may want to create distinct titles so the user can distinguish between different instances of your component.

You can use the Sites SDK to update the title for the component. In this step, you will update it based on the "imageBannerText" property.

To update the title, edit the render.js file and add this code to your SampleComponentViewModel object:

```javascript
self.updateDescription = ko.computed(function () {
    SitesSDK.setProperty('description', self.imageBannerText());
});
```

This Knockout computation will update the description for your component whenever the `imageBannerText` observable changes.

**Check the Results for Step 9**

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Click the Custom Settings button.
6. Change the Image Banner to Workplace.
7. Close the Settings panel and hover your cursor over your component to show the banner.

You should now see `A_Local_Component Workplace` displayed.

Continue to Step 10: Use Nested Components with In-Line Editing.

### Step 10: Use Nested Components with In-Line Editing

Oracle Content and Experience Cloud components are implemented using KnockoutJS component architecture. This means that if you are using KnockoutJS to implement your components, you can include Oracle Content and Experience Cloud built-in components directly into your template.

**Note:**

Because Oracle Content and Experience Cloud built-in components can only run in the Oracle Content and Experience Cloud page, you can't use nested components if your component is rendered in an inline frame.

To leverage nested components:

1. Implement your component using KnockoutJS.
2. Use RequireJS to include your component and use the same Knockout "ko" instance variable that is created by Oracle Content and Experience Cloud.
This is required because Oracle Content and Experience Cloud extends Knockout with components and these components won't be available if you use your own instance of KnockoutJS.

In this step you will review how the Oracle Content and Experience Cloud Image, Paragraph, and Title components are rendered in your custom component. A user will be able to edit it directly within the page and access the Settings panel for the nested component.

To see how these components are included in your template, edit the `render.js` file and look at the `sampleComponentTemplate` object. The default section that is rendered is shown here:

```html
<!-- ko if: alignImage() !== 'right' --><!--
<div style="display:flex;">
  <div data-bind="attr: {style: imageStyle, 'data-layout': alignImage()}, click: imageClicked">
    <scs-image params="{ scsComponent: { 'renderMode': mode, 'parentId': id, 'id': 'imageId', 'data': imageData } }"></scs-image>
  </div>
  <div data-bind="attr: {style: paragraphStyle}">
    <scs-title params="{ scsComponent: { 'renderMode': mode, 'parentId': id, 'id': 'titleId', 'data': titleData } }"></scs-title>
    <scs-paragraph params="{ scsComponent: { 'renderMode': mode, 'parentId': id, 'id': 'paragraphId', 'data': paragraphData } }"></scs-paragraph>
  </div>
</div><!-- /ko -->
```

Looking at the `<scs-image>` nested component, you will see the following entry:

```html
'<scs-image params="{ scsComponent: { 'renderMode': mode, 'parentId': id, 'id': 'imageId', 'data': imageData } }"></scs-image>'
```

The `scsComponent` data passed to the `params` template binding includes the following:

- `renderMode`: This refers to the mode Site Builder is in. You can use this to enable and disable features. For example, when used by the `<scs-title>` component, it adds the rich text editor when running in edit mode.
- `parentId`: This is required so the Oracle Content and Experience Cloud component knows that it is rendering as a nested component. All changes to the nested component will be saved in the data for the custom component.
- `id`: A unique ID for the nested component. This will be further namespace by the ID for the custom component.
- `data`: Initial data for the nested component. If the component isn't further modified, it will render with this initial data.
The referenced id and mode values are passed in to your custom component in the SampleComponentViewModel object, so you don't need to modify the object to get these values:

```javascript
// Store the args
self.mode = args.viewMode;
self.id = args.id;
```

The syntax for all the other supported nested components follows the same pattern as for `<scs-paragraph>`; for example: `<scs-image>, `<scs-title>, `<scs-button>.

Check the Results for Step 10

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Click the As a page author, you can edit... text in your component and update the description using the rich text editor.
5. Switch to Preview mode to see your update.
6. Switch back to Edit mode.
7. Bring up the Settings panel against your component.
8. Click the Components link that now appears, because it found your nested component.
9. Click Paragraph, which is the nested component it found.

You can now update the properties against the Paragraph component within your component.

Note:

Until the component has been instantiated, Oracle Content and Experience Cloud does not know about any nested components that may exist in the template. To tell Oracle Content and Experience Cloud about hidden nested components, you can use the SitesSDK.setProperty('visibleNestedComponents', []); API. To have hidden nested components show-up by default, you must update the "nestedComponents": [] array in the component registration.

Continue to Step 11: Support Different Layouts.

Step 11: Support Different Layouts

In this step we will review layouts that allow the user to alter how the component displays.

A custom component can support any number of layouts that you want to allow the user to choose. Each of these layouts will alter how the custom component displays. Layouts are another extension to the registration data.
To review the three layouts supported in the sample code, review the "componentLayouts" entry in the appinfo.json file

```
"componentLayouts": [

    {
        "name": "default",
        "displayName": "IMAGE_LEFT_LAYOUT"
    },

    {
        "name": "right",
        "displayName": "IMAGE_RIGHT_LAYOUT"
    },

    {
        "name": "top",
        "displayName": "IMAGE_TOP_LAYOUT"
    }
],
```

If you bring up the Settings panel against the custom component, you will see an option to switch between layouts. To enable your component to react to the change in selection, the render.js file has code to get the currently selected value and listen for changes to this value.

Edit the render.js file and look at the SampleComponentViewModel object.

- There is a layout observable, which is referenced in the template:

  ```
  self.layout = ko.observable();
  ```

- There is an update function to handle whenever this value changes:

  ```
  self.updateComponentLayout = $.proxy(function (componentLayout) {
      var layout = componentLayout ? componentLayout : 'default';
      self.layout(layout);
      self.alignImage(layout === 'right' ? 'right' : 'left');
      self.showTopLayout(layout === 'top');
      self.showStoryLayout(layout === 'default' || layout === 'right');
      self.componentLayoutInitialized(true);
  }, self);
  ```

- The initialization code gets the original value for the layout and calls the update function:

  ```
  SitesSDK.getProperty('componentLayout', self.updateComponentLayout);
  ```

The property change listener checks for any changes to this property and calls the update function:

```
self.updateSettings = function (settings) {
  if (settings.property === 'componentLayout') {
    self.updateComponentLayout(settings.value);
  }
  else if (settings.property === 'customSettingsData') {
    self.updateCustomSettingsData(settings.value);
  }
};
```
Finally the `sampleComponentTemplate` template object has code to reflect changes in this value:

'<!-- ko if: alignImage() === 'right' -->' +

Together, these changes allow you to select your layout in the Settings panel and have the component update.

**Check the Results for Step 11**

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Select `Image Right` from the Layout property.

At this point the component will update to show the `"<scs-image>"` component.

Continue to **Step 12: Define Custom Styles**.

---

**Step 12: Define Custom Styles**

Components you create are treated like any other component in the `design.json` and `design.css` files in the theme used for your site.

To add your own style for your custom component, confirm the `id` value you used when you registered your component. In the `appinfo.json` file; this was "id": "hello-world".

Using that value, edit the theme’s `design.json` file and add in the new styles you want to support against that `id`. For example, edit the `/designs/default/design.json` file in your theme and add this code:

```
"hello-world": {
    "styles": [{
        "name": "Plain",
        "class": "hello-world-default-style"
    },
    {
        "name": "Gothic",
        "class": "hello-world-gothic-style"
    }
]}
```

If you bring up the Settings panel against your component, you should now see **Plain (default)** and **Gothic** as the two options listed in the Style tab. However, switching
between these options will not do anything until you actually define the style classes listed in the design.css file.

Edit the theme’s design.css file and add in the cascading style sheet (CSS) classes of your style. For example, edit the /designs/default/design.css file in your theme and add this code:

```css
.hello-world-default-style .scs-component-content {
    font-family: "Helvetica Neue", "Helvetica", "Arial", sans-serif;
    font-size: 24px;
    font-weight: normal;
}

.hello-world-gothic-style .scs-component-content {
    font-family: "Century Gothic","CenturyGothic","AppleGothic",sans-serif;
    font-size: 32px;
    font-weight: bold;
}
```

Save and sync your files to the Oracle Content and Experience Cloud instance server.

Check the Results for Step 12

1. Refresh your page in your site so Site Builder can pick up the changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Go to the Style tab.
6. Switch between the Gothic and Plain styles that were defined in your design.json file.

You’ll notice that the font size within your component adjusts to reflect the changes as it switches between the applied CSS class for each selection.

Continue to Step 13: Render a Component in an Inline Frame.

Step 13: Render a Component in an Inline Frame

The sample so far has shown a local component rendered in-line in the page. You can also choose to render a component in an inline frame.

For example, you may choose to render a component in an inline frame if your component does non-omnipotent updates to the page, which requires you to re-create the page whenever properties change. In addition, remote components are always rendered in an inline frame.

The samples in this section are taken from the files created for you when you choose the Create a component that renders in an iframe option when creating a local component. You can, however, take these set of files and host them on your remote server so they apply equally to remote components.

Similarities Between Inline Frame and Non-Inline Frame Components
Because the Settings panel is always placed into the page in an inline frame, the code for the Settings panel doesn't change regardless of whether the component uses an inline frame or not. You'll create the same Settings panel code for both use cases.

**Sites SDK API**

The SDK API is the same for both use cases. You'll use the same code to raise triggers, listen for actions, and get and set property values. While certain properties may not be applicable in both cases (for example, you can't set the "height" property for a component that doesn't use an inline frame) the API remains the same. Therefore, you can copy the code between both of these types of components and the example code discussed in this tutorial will work for both cases.

**Differences Between Inline Frame and Non-Inline Frame Components**

**File Structure and Dependencies**

When you select **Create a component that renders in an iframe** when creating a local component, you will see the following files created for you:

```markdown
<component name>
  assets
  css
    app-styles.css
  js
    jquery.mn.js
    knockout.mn.js
    sites.min.js
  render.html
  settings.html
  appinfo.json
  _folder_icon.jpg
```

These files are created to allow you to immediately run your component in an inline frame on the page. The main differences between this structure and that of a standard local component are:

- **JavaScript dependencies:**
  - You are getting a complete copy of these files so your component will run. These files are required for the sample inline frame component to run. You can add and remove the content of this directory based on your requirements.
  - Because everything under the `assets` directory for your component is pushed to a public site when the component is published, everything in the `js` directory will be available both in Site Builder and at runtime.
  - Note: These files are created for ease of use. You should look at consolidating these files in the theme or in another public location rather than create separate versions of these files for each of your inline frame components.

- **render.html:**
  - This is a full HTML document as opposed to the `render.js` file for standard components, which is an AMD module.

**Component "Height" Management**
One of the issues in using an inline frame is the height management of the inline frame itself. If you get this wrong, you will see scroll bars appearing for the component on the page, which you may or may not want.

In order to manage the height of the inline frame, the component must tell the page how tall it wants the inline frame to be. With remote components, you may be dealing with cross-domain issues, so you must use Sites SDK messaging to ask the page to set the inline frame to the required height after the component has rendered on the page. This is done by using the `SitesSDK.setProperty('height', {value})` API. (See Oracle Content and Experience Sites SDK Reference.)

For example, create the `setHeight` function and a custom binding handler to call it when the component has rendered on the page.

- **Update height function:**

  ```javascript
  // set the height of the iFrame for this App
  self.setHeight = function () {
    // use the default calculation or supply your own height value as a second parameter
    SitesSDK.setProperty('height');
  };
  ```

- **Knockout custom binding handler to call `setHeight` whenever the component is rendered on the page or a property changes:**

  ```javascript
  ko.bindingHandlers.sampleAppSetAppHeight = {
    update: function (element, valueAccessor, allBindings, viewModel, bindingContext) {
      // create dependencies on any observables so this handler is called whenever it changes
      var imageWidth = viewModel.imageWidth(),
          imageUrl = viewModel.imageUrl(),
          titleText = viewModel.titleText(),
          userText = viewModel.userText();
      // re-size the iFrame in the Sites page now the template has rendered
      // Note: If you still see scrollbars in the iframe after this, it is likely that CSS styling in your app is the issue
      viewModel.setHeight();
    }
  };
  ```

- **Template update to call binding handler:**

  ```html
  <div data-bind="sampleAppSetAppHeight: true"></div>
  ```

**Trigger and Action Registration**

While the trigger/action registration for components that are not in inline frames is located in the `appinfo.json` file, for inline frame components, the component itself is responsible for providing this information. This is done using these two APIs:

```javascript
SitesSDK.subscribe('GET_ACTIONS', self.getAppActions);
SitesSDK.subscribe('GET_TRIGGERS', self.getAppTriggers);
```
Here's an example of using these APIs.

```javascript
// Register TRIGGERS meta-data
SampleAppViewModel.prototype.getAppTriggers = function (args) {
  var triggers = [{
    "triggerName": "imageClicked",
    "triggerDescription": "Image clicked",
    "triggerPayload": {
      "name": "payloadData",
      "displayName": "Trigger Payload Data"
    }
  }];

  return triggers;
};

// Register ACTIONS meta-data
SampleAppViewModel.prototype.getAppActions = function (args) {
  var actions = [{
    "actionName": "setImageWidth",
    "actionDescription": "Update the image width",
    "actionPayload": {
      "name": "imageWidth",
      "description": "Image Width in pixels",
      "type": {
        "ojComponent": {
          "component": "ojInputText"
        }
      },
      "value": ""
    }
  }];

  return actions;
};
```

Access to Theme Styles

Because the component is rendered in an inline frame, it doesn't have access to the styles available in the theme. The Sites SDK provides an API to retrieve these styles so they can be applied to elements within the inline frame.

This topic is explored more in Step 14: Use Custom Styles When the Component is Rendered in an Inline Frame.

Mixed HTTPS Versus HTTP Protocol

Because Oracle Content and Experience Cloud uses the HTTPS protocol, all resources referenced within the page also must use HTTPS. Resources include the base .html file that will be rendered in the inline frame along with all the files that it references.

This resource requirement applies mostly to remote components, however, you must be aware of this constraint. Resources for local components using inline frames are provided by the Oracle Content and Experience Cloud server, so these components already use a matching protocol.
Step 14: Use Custom Styles When the Component Is Rendered in an Inline Frame

Components rendered in an inline frame don't have direct access to the design.css file. Instead there is an additional step to get the URL for the design.css in your component and add it to the page. You then must update your component to reflect the user-selected style.

To include and use the design.css file in your component requires changes in the render.html file:

1. Locate and include the URL to the design.css file
2. Get the value of the select style class whenever it changes
3. Update the template to reflect the styleClass selected
4. Reflect changes to the selected style class in your component
5. Make sure the inline frame resizes when the style changes

Here are the detailed instructions for editing the render.html file:

1. Locate and include the URL to the design.css file.

Dynamically add the design.css file to the <head> section of the page. After it has been loaded, set the height of the inline frame because it may have been altered by applying the styles.

Add the following code into the viewModel object:

```javascript
// Dynamically add any theme design URL to the <head> of the page
self.loadStyleSheet = function (url) {
    var $style,
        stylesheetDeferred = new $.Deferred(),
        attempts = 100,
        numAttempts = 0,
        interval = 50,
        pollFunction = function () {
            // try to locate the style sheet
            for (var i = 0; i < document.styleSheets.length; i++) {
                try {
                    // locate the @import sheet that has an href based on our expected URL
                    var sheet = document.styleSheets[i],
                        rules = sheet && sheet.cssRules,
                        rule = rules && rules[0];
                    // check whether style sheet has been loaded
                    if (rule && (rule.href === url)) {
                        stylesheetDeferred.resolve();
                        return;
                    }
                } catch (e) {}  
            }
        }

        // try to locate the style sheet
        for (var i = 0; i < document.styleSheets.length; i++) {
            try {
                // locate the @import sheet that has an href based on our expected URL
                var sheet = document.styleSheets[i],
                    rules = sheet && sheet.cssRules,
                    rule = rules && rules[0];
                // check whether style sheet has been loaded
                if (rule && (rule.href === url)) {
                    stylesheetDeferred.resolve();
                    return;
                }
            } catch (e) {}  
        }
```

Chapter 19
if (numAttempts < attempts) {
    numAttempts++;
    setTimeout(pollFunction, interval);
} else {
    // didn't find style sheet so complete anyway
    styleSheetDeferred.resolve();
}

// add the themeDesign stylesheets to <head>
// use @import to avoid cross domain security issues when determining when the stylesheet is loaded
$style = $('<style type="text/css">@import url("' + url + '")</style>);
$style.appendTo('head');

// kickoff the polling
pollFunction();

// return the promise
return styleSheetDeferred.promise();

// update with the design.css from the Sites Page
SitesSDK.getSiteProperty('themeDesign', function (data) {
    if (data && data.themeDesign && typeof data.themeDesign === 'string') {
        // load the style sheet and then set the height
        self.loadStyleSheet(data.themeDesign).done(self.setHeight);
    }
});

2. Get the value of the select style class whenever it changes.

Create an observable to track when the value of the `styleClass` property changes:

```javascript
self.selectedStyleClass = ko.observable();
```

Note that we can't render until we have the style class. Change this code:

```javascript
self.customSettingsDataInitialized = ko.observable(false); 
self.initialized = ko.computed(function () {
    return self.customSettingsDataInitialized() &&
```

Use this code instead:

```javascript
self.customSettingsDataInitialized = ko.observable(false); 
self.styleClassInitialized = ko.observable(false); 
self.initialized = ko.computed(function () {
    return self.customSettingsDataInitialized() &&
```
self.styleClassInitialized();
}, self);

Get the initial value for the selected style class by adding:

self.updateStyleClass = function (styleClass) {
    self.selectedStyleClass((typeof styleClass === 'string') ? styleClass : 'hello-world-default-style'); // note that this 'hello-world' prefix is based on the app name
    self.styleClassInitialized(true);
};
SitesSDK.getProperty('styleClass', self.updateStyleClass);

3. Update the template to reflect the styleClass. Change this code:

<p data-bind="attr: {id: 'titleId'}, text: titleText"></p>

Use this code instead:

<p data-bind="attr: {id: 'titleId'}, text: titleText, css: selectedStyleClass"></p>

4. Reflect changes to the selected style class in your component. Change this code:

if (settings.property === 'customSettingsData') {
    self.updateCustomSettingsData(settings.value);
}

Use this code instead:

if (settings.property === 'customSettingsData') {
    self.updateCustomSettingsData(settings.value);
}
if (settings.property === 'styleClass') {
    self.updateStyleClass(settings.value);
}

5. Make sure the inline frame re-sizes when the style changes. Change this code:

    // create dependencies on any observables so this handler is called whenever it changes
    var imageWidth = viewModel.imageWidth(),
        imageUrl = viewModel.imageUrl(),
        titleText = viewModel.titleText(),
        userText = viewModel.userText();

Use this code instead:

    // create dependencies on any observables so this handler is called whenever it changes
    var imageWidth = viewModel.imageWidth(),
        imageUrl = viewModel.imageUrl(),
        titleText = viewModel.titleText(),
        userText = viewModel.userText();
titleText = viewModel.titleText(),
userText = viewModel.userText(),
selectedStyleClass = viewModel.selectedStyleClass();

6. Save and sync your files to the Oracle Content and Experience Cloud instance server.

Check the Results for Step 14

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component onto the page.
4. Bring up the Settings panel against your component.
5. Go to the Style tab.
6. Switch between Gothic and Plain styles defined in your design.json file.

You'll notice the font size changes within your component it switches between the applied CSS class for each selection. The inline frame also re-sizes as you choose a different style.

Continue to Step 15: Integration with the Page Undo and Redo Behavior.

Step 15: Integration with the Page Undo and Redo Behavior

Because Oracle Content and Experience Cloud stores properties on behalf of the custom component, changes to those properties are automatically part of the page's Undo and Redo behavior.

To ensure that it is clear what is happening when a user clicks Undo or Redo, these "undo events" should only happen when a user has actually done something to the page. For example, bringing up the custom component Settings panel should not update the properties within the page until the user actually makes a change to the property. Simply initializing the properties in the Settings panel should not cause an update event.

If care is not taken to ensure this behavior, then unexpected behavior may occur. The page will still run, but to the detriment of the user experience. For example, these behaviors may occur:

- The Save button will become active simply by bringing up the Settings panel.
- The user must click Undo multiple times before any effect is visible.
- The Redo stack is removed because the component wrote back an unexpected change and updated the Redo stack with the new value.

The sample code provided in this tutorial for the Settings panel gives an example of how to ensure you are only writing back when you're ready to actually call saveData and not on initialization. Similar care should be taken within the component itself to not update customSettingsData unless it involved a user interaction, though typically this is less of a concern.

Continue to Step 16: Asset Management.
Step 16: Asset Management

This step describes and explains how to manage the assets used by a component.

Assets include components and custom components that Oracle Content and Experience Cloud must know about to manage the life cycle of the assets.

Oracle Content and Experience Cloud content Folder

Each site that you create in Oracle Content and Experience Cloud comes with its own content folder. This is a hidden folder that you won't normally see. When the site is published, all the files in the content folder are also published to the file system.

For example, when you select an image using the Image component, Oracle Content and Experience Cloud makes a copy of the image you selected and places it in the content folder. Your URL always points to this copied version of the image so that if you delete the original image, your site won't break. This also applies to the other components provided by Oracle Content and Experience Cloud: Gallery, Gallery Grid, Document, Social Bar, File Download, as well as background images for slots and componentGroups.

For a custom component to take part in this asset lifecycle, the custom component must tell Oracle Content and Experience Cloud about any assets it wants the service to manage on its behalf. Because this involves making a copy of the asset, the custom component must also use Oracle Content and Experience Cloud APIs to select the asset so that we know how to manage it.

Manage URLs

The URL to an asset changes based on a number of criteria.

- The runtime URL to a component is different than the Site Builder URL to the component
- If you copy a page, Oracle Content and Experience Cloud also makes a copy of all the referenced assets in the content folder so you never have two components pointing to the same asset in the content folder
- Dropping a componentGroup onto the page makes new copies for any assets referenced by a component in the componentGroup

In addition, while a relative URL may be fine for a local component, remote components require the fully qualified URL to any asset that you want Oracle Content and Experience Cloud to manage on your behalf so they can render their inline frame content with the full URL.

Because you can't rely on the URL remaining static, you must only hold references to the ID to the asset in your code and retrieve the asset's URL when you want to render the asset.

Manage Assets

These Sites SDK APIs are available for managing assets.

`SitesSDK.getProperty('componentAssets', callback);`

- This gets the array of current assets
- Each asset entry consists of:
- **id**: Unique ID for the asset.
- **title**: Oracle Content and Experience Cloud title metadata.
- **description**: Oracle Content and Experience Cloud description metadata.
- **fileName**: Original name of the selected file. Useful for display in the Settings panel for your custom component so users know what file they selected. This is not the name of the file copied to the content folder.
- **source**: Macro enabled URL to the asset. This value will change over time and should not be referenced by your component, but it must be saved as part of the asset.
- **url**: Fully qualified URL to the asset based on the context in which `getProperty()` was called.

```
SitesSDK.setProperty('componentAssets', [assets]);
```

- Call this to save all the assets you want Oracle Content and Experience Cloud to manage on your behalf.
- If you don't call this, then no asset will be saved.
- Any assets not in this array will be deleted when the site is published.
- The `assets` parameter is an array of assets in the same format as you get returned by `getProperty` and is also returned by `filePicker`.

**Note:**

No `url` value is stored. That value is dynamically created when you ask for the assets.

```
SitesSDK.filePicker(options, callback);
```

- An API to bring up the file picker to select the list of assets.
- It calls the callback on successful selection of assets passing in the array of selected assets.
- Nothing is saved at this point and it's up to the component to call `setProperty('componentAssets', [assets]);` to save items from this selection combined with any other assets to be saved.

**Example of Select Asset**

This section shows you how to select an asset, store its ID, and re-fetch the actual values from the stored assets.

1. **Edit the** `settings.html` **file.**
2. **Change the template object to include an** Image selection.

```
<div>
    <!-- Image selection -->
    <label id="imageLabel" for="imageAsset" class="settings-heading" data-bind="text: 'Image'"></label>
    <input id="imageAsset" data-bind="value: imageName" readonly class="settings-text-box">
    <button id="imageSelect" type="button" class="selectbutton">
```

Chapter 19  
Step 16: Asset Management
3. Change the viewModel to add an observable to store the ID of the selected asset.

   self.imageID = ko.observable();

4. Change the viewModel to manage selection of the asset by bringing up the file picker and displaying the name of the selected asset.

   //
   // handle component assets
   //
   self.assets = []

   // bring up a file picker to select the assets
   self.showFilePicker = function () {
       // select an image
       SitesSDK.filePicker({
           'multiSelect': false,
           'supportedFileExtensions': ['jpg', 'png']
       }, function (result) {
           if (result.length === 1) {
               // update the array of assets
               self.assets = result;

               // update the image in customSettingsData
               self.imageID(result[0].id);
           }
       });
   }

   // update the display name based on the assets
   self.imageName = ko.computed(function () {
       var imageName = '', imageID = self.imageID();
       for (var i = 0; i < self.assets.length; i++) {
           if (self.assets[i].id === imageID) {
               imageName = self.assets[i].fileName;
               break;
           }
       }

       return imageName
   }, self);

5. Update the viewModel to first retrieve the assets before getting the customSettingsData. This code will also cause the self.imageName() to be invoked when the self.ImageID() observable changes.

   SitesSDK.getProperty('componentAssets', function (assets) {
       self.assets = assets;
       SitesSDK.getProperty('customSettingsData', function (data) {
           //update observable
Finally, update the `save` function to save the `imageID` and make sure to update the `componentAssets` with your list of referenced assets.

```javascript
self.save = ko.computed(function () {
    var saveconfig = {
        'imageWidth': isNaN(self.imageWidth()) ? self.imageWidth() : self.imageWidth() + 'px',
        'imageID': self.imageID(),
        'titleText': self.titleText(),
        'userText': self.userText()
    };

    // store the selected asset and update custom settings
    if (self.saveData) {
        SitesSDK.setProperty('componentAssets', self.assets);
        SitesSDK.setProperty('customSettingsData', saveconfig);
    }
}, self);
```

Check the Results for Select Asset

1. Refresh your page in your site so Site Builder can pick up changes to the component.
2. Take the page into Edit mode.
3. Drag and drop your component on the page.
4. Bring up the Settings panel.
5. Click the Select image button.
6. Browse (or upload) and select an image.
   
   Note that the name of the image is stored showing the selected image.
7. Close out of the Settings panel.
8. Bring up the Settings panel again.
   
   Note that the name of the image is again reflected.

Example of Render Asset

This section shows you how to retrieve the assets and render them in your component, and also have the component dynamically update whenever values are changed in your settings panel.
1. **Edit the** `render.html` **file.**

2. **Update the template to include your asset:**

   ```html
   <!-- ko if: imageURL -->
   <div style="flex-shrink:0;">
   </div>
   <!-- /ko -->
   ```

3. **In the viewModel, create two observables to get the** `imageID` **from the customSettingsData and store the** `imageURL` **retrieved from the stored list of assets.**

   ```javascript
   self.imageID = ko.observable();
   self.imageURL = ko.observable();
   ```

4. **Update the viewModel so that whenever the** `imageID` **changes, it gets the corresponding image asset URL.**

   ```javascript
   self.imageID.subscribe(function (imageID) {
     // whenever the image changes get the updated referenced asset
     SitesSDK.getProperty('componentAssets', function (assets) {
       for (var i = 0; i < assets.length; i++) {
         if (assets[i].id === imageID) {
           self.imageURL(assets[i].url);
           break;
         }
       }
     });
   });
   ```

5. **Update the viewModel to retrieve the ID from the** `customSettingsData`. **Check the Results for Render Asset**

   1. Refresh your page in your site so Site Builder can pick up changes to the component.
   2. Take the page into Edit mode.
   3. Drag and drop your component on the page.
   4. Bring up the Settings panel.
   5. Click the **Select image** button.
   6. Browse (or upload) and select an image.
Note that the name of the image is stored showing the selected image.

7. Close out of the Settings panel.

At this point you should see your selected image rendered in the component.

Continue to Tutorial Review.

Tutorial Review

This tutorial gives you an overview of how to create a customized component using a Knockout Component Factory.

The main purpose of this tutorial is that using this pattern, you can create any custom component by just updating the `SampleComponentViewModel` and `sampleComponentTemplate` JavaScript objects. The `sampleComponentFactory` and `SampleComponentImpl` objects haven't changed as you went through the tutorial. You were able to implement these changes without having to deal with communicating with the page, and were able to perform these tasks:

- Communicate changes from your Settings panel to your component, and have those changes persisted.
- Execute triggers and actions, and interact with other components on the page.
- Create layouts and leverage nested components.
- Define component-specific styles.

While this example split out the custom component into a number of files, this was for clarity of the tutorial. For optimization, you should consider appropriately packaging your files to avoid multiple downloads.

Finally, while this tutorial is suitable for Knockout based components, if you want to create custom components using another JavaScript technology stack such as AngularJS, you must re-implement the `SampleComponentImpl` object to create the corresponding communication with that framework along with a technology specific implementation of the actual component. This work is beyond the scope of this tutorial.
Troubleshoot

If you should run into problems, here are some solutions.

- The remote component is not responding
- The remote component settings button is greyed out
- I can’t remove a remote component from the catalog

The remote component is not responding

The remote component you created and configured is not responding.

- Get the endpoint URLs from the remote component registration and try them directly in the browser.
- The remote component might be using `X-Frame-Options=sameorigin`, which is not supported.

The remote component settings button is greyed out

The component settings button can be greyed out for one of the following reasons.

- The remote component is not registered.
- The remote component was added to the page, and then unregistered.

I can’t remove a remote component from the catalog

I want to remove a remote component from the catalog, but it’s not allowed.

Only the user who registered the remote component can remove it from the components catalog.